

# ARM<sup>®</sup> DS-5

Version 5.9

## Using ARM Streamline

The ARM logo, consisting of the letters "ARM" in a bold, sans-serif font, with a registered trademark symbol (®) to the upper right.

# ARM DS-5

## Using ARM Streamline

Copyright © 2010-2012 ARM. All rights reserved.

### Release Information

The following changes have been made to this book.

#### Change History

Date	Issue	Confidentiality	Change
September 2010	A	Non-Confidential	ARM Streamline Performance Analyzer 1.0
January 2011	B	Non-Confidential	Update for DS-5 version 5.4
April 2011	C	Non-Confidential	Update for DS-5 version 5.5
July 2011	D	Non-Confidential	Update for DS-5 version 5.6
September 2011	E	Non-Confidential	Update for DS-5 version 5.7
November 2011	F	Non-Confidential	Update for DS-5 version 5.8
February 2012	G	Non-Confidential	Update for DS-5 version 5.9

### Proprietary Notice

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

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

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

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

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

The Visual Annotation screenshots in the documentation feature the DOOM software from the PrBoom project and contributors (<http://www.crowproductions.de/repos/prboom>).

### Confidentiality Status

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

### Product Status

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

### Web Address

<http://www.arm.com>

# Contents

## ARM DS-5 Using ARM Streamline

<b>Chapter 1</b>	<b>Conventions and Feedback</b>	
<b>Chapter 2</b>	<b>Setting Up Your Target</b>	
2.1	Setting up an ARM Linux target .....	2-2
2.2	Setting up an Android target .....	2-5
2.3	Recommended compiler options .....	2-8
2.4	Mali setup for Streamline .....	2-9
<b>Chapter 3</b>	<b>Using the ARM Streamline Data View</b>	
3.1	ARM Streamline Data view basics .....	3-2
3.2	ARM Streamline Data view toolbar options .....	3-4
3.3	The Analysis Data Locations dialog box .....	3-5
<b>Chapter 4</b>	<b>Setting Capture Options</b>	
4.1	Opening the Capture Options dialog box .....	4-2
4.2	Capture options .....	4-3
<b>Chapter 5</b>	<b>Configuring Counters</b>	
5.1	Opening the Counter Configuration dialog box .....	5-2
5.2	Using the Counters Configuration dialog box .....	5-4
<b>Chapter 6</b>	<b>The Timeline View</b>	
6.1	About the Timeline view .....	6-2
6.2	Timeline view charts .....	6-8
6.3	Filtering data and other Timeline view controls .....	6-10
6.4	Timeline view toolbar options, contextual menu options, and keyboard shortcuts .....	6-13
6.5	Visual Annotation in the Timeline view .....	6-16

<b>Chapter 7</b>	<b>The Table Views: Call Paths, Functions, and Stack</b>	
7.1	Table views toolbar options, contextual menu options and keyboard shortcuts .....	7-2
7.2	Sorting data in the table reports .....	7-5
7.3	Call Paths view column headers .....	7-6
7.4	Functions view column headers .....	7-7
7.5	Stack view column headers and the Maximum Stack Depth by Thread chart .....	7-8
<b>Chapter 8</b>	<b>The Code View</b>	
8.1	Code view basics .....	8-2
8.2	Code view toolbar options and keyboard shortcuts .....	8-5
<b>Chapter 9</b>	<b>The Call Graph View</b>	
9.1	Call Graph view basics .....	9-2
9.2	Contextual menu options .....	9-5
9.3	The toolbar and keyboard shortcuts .....	9-6
<b>Chapter 10</b>	<b>Annotate and the Log View</b>	
10.1	Customize reports using Annotate .....	10-2
10.2	Adding bookmarks using Annotate .....	10-4
10.3	Adding images to reports using Visual Annotate .....	10-6
10.4	The Log view .....	10-9
<b>Chapter 11</b>	<b>Advanced Customizations</b>	
11.1	Capturing data on your target .....	11-2
11.2	Creating custom performance counters .....	11-4
11.3	Using Stored Streamline Capture Data to create new Streamline Analysis Reports .....	11-7
11.4	Profiling the Linux kernel .....	11-8
<b>Chapter 12</b>	<b>Using the Energy Probe</b>	
12.1	Energy Probe overview .....	12-2
12.2	Energy Probe requirements .....	12-3
12.3	Energy Probe setup .....	12-5
12.4	Energy Probe operation .....	12-8
<b>Chapter 13</b>	<b>Using Streamline on the Command Line</b>	
13.1	Opening a Streamline-enabled command prompt or shell .....	13-2
13.2	The streamline command .....	13-3
<b>Chapter 14</b>	<b>Troubleshooting</b>	
14.1	Target connection issues .....	14-2
14.2	Report issues .....	14-3

# Chapter 1

## Conventions and Feedback

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

### Typographical conventions

The following typographical conventions are used:

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

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

*monospace* *italic*

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

**`monospace`** **bold**

Denotes language keywords when used outside example code.

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

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

### Feedback on this product

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

- your name and company

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

### Feedback on content

If you have comments on content then send an e-mail to [errata@arm.com](mailto:errata@arm.com). Give:

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

ARM also welcomes general suggestions for additions and improvements.

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

### Other information

- ARM Information Center, <http://infocenter.arm.com/help/index.jsp>
- ARM Technical Support Knowledge Articles, <http://infocenter.arm.com/help/topic/com.arm.doc.faqs/index.html>
- ARM Support and Maintenance, <http://www.arm.com/support/services/support-maintenance.php>.
- ARM Glossary, <http://infocenter.arm.com/help/topic/com.arm.doc.aeg0014-/index.html>

## Chapter 2

# Setting Up Your Target

ARM Streamline Performance Analyzer is a system-wide visualizer and profiler for targets running ARM Linux or Android native applications and libraries. Combining an ARM Linux kernel driver, target daemon, and a graphical user interface, it transforms system trace and sampling data into reports that present the data in both visual and statistical forms. Streamline uses hardware performance counters with kernel metrics to provide an accurate representation of system resources. Streamline supports Cortex™-A8, Cortex-A9, Cortex-A15, ARM9™, and ARM11™ processors running ARM Linux or Android.

The following topics describe how to set up your target to run Streamline:

- [\*Setting up an ARM Linux target on page 2-2\*](#)
- [\*Setting up an Android target on page 2-5\*](#)
- [\*Recommended compiler options on page 2-8\*](#)
- [\*Mali setup for Streamline on page 2-9\*](#)

## 2.1 Setting up an ARM Linux target

These instructions are specific to targets running ARM Linux.

---

### Note

---

You can locate all of the files provided by DS-5 by selecting **Help** → **ARM Extras...** from the main menu.

---

### 2.1.1 Prerequisites

You must have the following tools on your host to build the Linux kernel and the gator driver:

- Linux kernel source code for the target platform. Streamline supports only Linux kernel versions 2.6.32 and above.
- Either the cross compiler for building the Linux kernel or the ARM Linux GCC that comes with DS-5.

---

### Note

---

Streamline only supports hardware targets, not *Real Time System Models* (RTSMs). Streamline can run on an RTSM, but an RTSM does not provide the cycle and timing information to make the samples-generated data in the reports meaningful.

---

### 2.1.2 Load the gator daemon

The gator daemon and driver collect target metrics and then send them to your host machine.

To enable profiling, you must build and load the gator daemon on your target. Follow these steps to build the gator daemon from the source archive:

1. Move to the directory that contains the gator daemon source:

```
cd ../gator/daemon-src
```

2. Enter the following commands to unzip the archive:

```
tar -xzf gator-daemon.tar.gz
```

---

### Note

---

Make sure to untar this file in a directory in which you have write privileges.

---

3. Change directories to the newly created gator-daemon:

```
cd gator-daemon
```

4. Issue the make command to build gatord:

```
make
```

5. Move the newly created gatord to your desired directory on the host.

---

### Note

---

You must build the gator daemon on your gcc-enabled target or a Linux host. It is not possible to build the gator daemon on a Windows host.

---



### 2.1.3 Prepare and build your kernel

To prepare your kernel for use with Streamline, download the desired version of the Linux kernel, configure it to your target platform, customize it, and then build it. To do so, follow these steps:

1. Download your desired version of the Linux kernel. For example:  
**wget <http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.35.7.tar.bz2>**
2. Unzip the Linux kernel. For example, enter the following command:  
**tar xjf linux-2.6.35.7.tar.bz2**
3. Navigate to the root source directory of the Linux kernel. For example, enter the following command:  
**cd linux-2.6.35.7**
4. Enter the following command in your shell to export the cross compiler:  
**export CROSS\_COMPILE=\${CROSS\_TOOLS}/bin/arm-none-linux-gnueabi-**
5. To specify that this build is for an ARM architecture, enter the following command in your shell:  
**export ARCH=arm**
6. Enter the following to build the configuration file specific to your platform:  
**make platform\_defconfig**  
 Replace *platform\_defconfig* in the command with one of the configuration files located in the *your\_kernel/arch/arm/configs* directory appropriate for your platform or with a configuration file provided by a vendor.
7. To configure menus, enter the following in your shell:  
**make menuconfig**  
 You must enable certain kernel configuration options to run Streamline. The location of some of these options depends on your kernel version:  
**General Setup**  
 Enable the **Profiling Support** option.  
**Kernel Hacking**  
 Enable the **Trace process context switches** option.  
**Kernel Features**  
 Enable the **High Resolution Timer Support** option. Enable **Use local timer interrupts** if you are using a *Symmetric MultiProcessing* (SMP) target.  


---

**Note**  
 The **Trace context switches and events** option is not visible if you have other **Tracers** configuration options enabled. Enabling other **Tracers** configuration options is sufficient to turn on context switches and events and run Streamline.  


---
8. Use the following command to build the image:  
**make -j5 uImage**

### 2.1.4 Build the gator module

To use Streamline with your ARM target, you must build the gator driver on a Linux host and place it in the same directory as the gator daemon, *gator*, on the target file system.

DS-5 provides a `gator-driver.tar.gz` source archive. Assuming that you have unzipped the file and that you have all of the required tools for building kernel modules, enter the following command on your target to create the `gator.ko` module:

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

### 2.1.5 Run the gator daemon on your target

When all of the necessary files are in place, you can start the gator daemon.

To run `gatord`:

1. Load the kernel onto the target
2. Copy `gatord` and `gator.ko` into the file system on the target. `gatord` must be placed in the same directory as `gator.ko` on the target.
3. To ensure `gatord` has execute permission, enter the following command:  
**chmod +x gatord**
4. After making sure that you have root privileges, enter the following to execute the gator daemon:

```
./gatord &
```

By default, `gatord` uses port 8080 for communication with the host, but you can adjust this by launching `gatord` with the port number as a parameter and changing the **Port** option in the Capture Options dialog box. For example:

```
./gatord 5050 &
```

To open the Capture Options dialog box, click **Change capture options** in the ARM Streamline Data view.

### 2.1.6 See also

#### Tasks

- [Setting up an Android target on page 2-5.](#)

#### Reference

- [Target connection issues on page 14-2.](#)

## 2.2 Setting up an Android target

These instructions are specific to targets running Android.

---

### Note

You can locate all of the files provided by DS-5 by selecting **Help** → **ARM Extras...** from the main menu.

---

### 2.2.1 Prepare and build your kernel

You must enable certain kernel configuration options to run Streamline. In the kernel configuration menu, use the arrow keys to navigate to the desired submenu and press Enter. Each submenu is listed with the action you need to take within it.

#### General Setup

Make sure the Profiling Support option is enabled.

#### Kernel Hacking

In the Kernel Hacking submenu, navigate to the Tracers submenu and press Enter. Make sure the Trace process context switches option is enabled.

**Kernel** Make sure Profiling Support is enabled.

#### Kernel Features

Make sure the **High Resolution Timer Support** is enabled. Enable **Use local timer interrupts** if you are using a *Symmetric MultiProcessing* (SMP) target.

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.

### 2.2.2 Build the gator daemon

The gator daemon and driver collect target metrics and then send them to your host machine.

---

### Note

To build the gator daemon, you must install the Android NDK. For more information on how to do this, visit the Android NDK site, <http://developer.android.com/sdk/ndk>.

---

To enable profiling, you must build and load the gator daemon on your target. Follow these steps to build the gator daemon from the source archive:

1. Move to the directory that contains the gator daemon source:

```
cd ../gator/daemon-src
```

2. Enter the following commands to unzip the archive:

```
tar -xzf gator-daemon.tar.gz
```

---

### Note

You may need to use `sudo` in front of this command if you do not have privileges.

---

3. Issue the following command:

```
mv gator-daemon jni
```

4. Issue the following command to build gator:

**ndk-build**

———— **Note** ————

If ndk is not on your path, instead issue the following command:

**execute /path/to/ndk/ndk-build.**

5. The newly created gator is located in *build\_directory/libs/armeabi*

———— **Note** ————

You must build the gator daemon on your gcc-enabled target or a Linux host. It is not possible to build the gator daemon on a Windows host.

### 2.2.3 Build the gator module

To use Streamline with your Android target, you must build the gator driver on a Linux host and place it in the same directory as the gator daemon, gatord, on the target file system. Transfer the gator driver module sources from your host to the target. They are located on your host here:

`.../gator/driver-src/gator-driver.tar.gz`

Assuming that you have unzipped the file and that you have all of the required tools for building kernel modules, enter the following command on your target to create the gator.ko module:

**make -C kernel\_build\_dir M=`pwd` ARCH=arm CROSS\_COMPILE=<...> modules**

### 2.2.4 Connect to the target

Streamline supports connection to your target using ethernet. Using the *Android Debug Bridge* (ADB) utility that is part of the Android SDK, it is possible to forward a TCP port from the target to your localhost over a USB connection. ADB can also be configured to work over ethernet.

For more information about the Android SDK and ADB setup, see the Android website.

Once ADB is set up, forward the Streamline port to your localhost using the following command:

**adb forward tcp:8080 tcp:8080**

### 2.2.5 Run the gator daemon on your target

Copy gator and gator.ko into the file system on the target. . gator must be placed in the same directory as gator.ko on the target.

Enter the following command to instantiate the gator driver.

**insmod gator.ko**

Now run gator on the target:

**./gator &**

## 2.2.6 Set capture options to support your Android target

In the Capture Options dialog box, opened in the ARM Streamline Data view in Eclipse for DS-5 and enter the ethernet address. If you use the ADB to forward the port, enter localhost in the Address field.

---

### **Note**

With Android, make sure to add the shared libraries you are interested in profiling to the session using the Program Images section of the Capture Options dialog box.

---

## 2.2.7 See also

### Tasks

- [Opening the Capture Options dialog box on page 4-2](#)
- [Setting up an ARM Linux target on page 2-2.](#)

### Reference

- [Target connection issues on page 14-2](#)
- [Capture options on page 4-3](#)
- [ARM Streamline Data view basics on page 3-2.](#)

## 2.3 Recommended compiler options

When building executables for profiling using Streamline, it is best practice to use the following compiler options:

-g Turns on the debug symbols necessary for quality Analysis Reports.

-fno-inline Disables inlining. This compiler setting substantially improves the call path quality.

-fno-omit-frame-pointer

Compiles your EABI images and libraries with frame pointers. This enables Streamline to record the call stack with each sample taken.

### 2.3.1 See also

#### Tasks

- [Setting up an Android target on page 2-5.](#)

#### Reference

- [Target connection issues on page 14-2](#)
- [Capture options on page 4-3.](#)

## 2.4 Mali setup for Streamline

Streamline enables you to gather GPU-specific profiling data on a Mali-400 based device, adding a significant amount of data about the graphical performance of your target to the Analysis Reports.

### 2.4.1 Requirements

- A Mali-400 based device.
- A sufficiently recent version of the mali.ko driver. The driver must be capable of generating the tracepoints needed by the gator.ko driver. Consult your supplier to see if this version of the driver is available for your device. For more information, see the Mali developer site, <http://www.malideveloper.com>

### 2.4.2 Special instructions for building the gator driver

Follow normal installation and setup instructions for Streamline and the gator daemon. To build the gator driver, gator.ko with Mali support, add a build switch and an include to the make command. The build switch enables Mali support, and the include path allows the compiler to locate the mali\_linux\_trace.h header file containing the declaration of the Linux tracepoints necessary for Mali support. To build the gator module for Mali, follow these steps:

1. Add the following options to your gator.ko make command:  
`KCFLAGS="-IMali_driver_source_location/src/devicedrv/mali"`  
`GATOR_WITH_MALI_SUPPORT=MALI_400`
2. Insert the gator module as normal:  
`insmod gator.ko`
3. Verify that you built the module successfully:  
`ls -l /dev/gator/events/ARM_Mali-400*`  
 This command should produce a list of counters.

If you have successfully built the gator module with Mali support, you can run a capture session on a Mali target. Follow the normal instructions for setting capture options and a triggering a capture session. For more information on how to do this, see [ARM Streamline Data view basics on page 3-2](#).

### 2.4.3 See also

#### Tasks

- [Opening the Capture Options dialog box on page 4-2](#)
- [Setting up an ARM Linux target on page 2-2.](#)

#### Reference

- [Target connection issues on page 14-2](#)
- [Capture options on page 4-3](#)
- [ARM Streamline Data view basics on page 3-2](#)
- [Charts specific to Mali-400 targets on page 6-8](#)
- [Mali-specific events on page 5-6](#)

## Chapter 3

# Using the ARM Streamline Data View

The ARM Streamline Data view enables you to set target connection settings, start and stop capture sessions, and create new Streamline data files from stored capture sessions.

The following topics describe how to open and use the ARM Streamline Data view:

- [\*ARM Streamline Data view basics\* on page 3-2](#)
- [\*ARM Streamline Data view toolbar options\* on page 3-4](#)
- [\*The Analysis Data Locations dialog box\* on page 3-5.](#)



### 3.1 ARM Streamline Data view basics

Much of the functionality of ARM Streamline requires the ARM Streamline Data view. To open the ARM Streamline Data view:

1. Select **Window → Show View → Other...**
2. Expand the **ARM Streamline** group.
3. Select **ARM Streamline Data**.

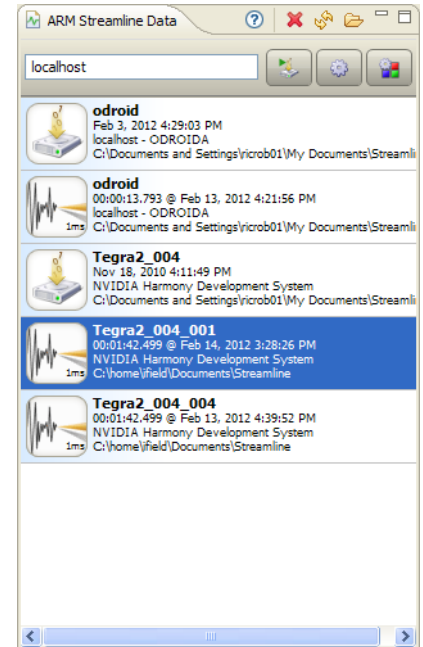


Figure 3-1 The ARM Streamline Data view

#### 3.1.1 Streamline Capture Data and Streamline Analysis Reports

If you have successfully captured data on your target, the ARM Streamline Data view contains two resource types. Streamline Capture Data has a .apc extension and contains all of the raw data collected on the target during a capture session. Streamline Analysis Reports have a .apd extension and are created on the host from the information contained in the Streamline Capture Data.

———— **Note** ————

Streamline Analysis Reports files are not compatible across different versions of Streamline, but the same is not true of Streamline Capture Data. If an upgrade to Streamline renders a Streamline Analysis Report incompatible, double-click on the Streamline Capture Data resource to re-run it and create a new, compatible Streamline Analysis Report.

#### 3.1.2 Stopping a capture session manually

You can use the Capture Options dialog box to set a specific capture session length, but if you want to control the length of the session yourself, you can use the Streamline data view to terminate it manually. To do so, click the **Stop** button that appears on the left in the new capture file.

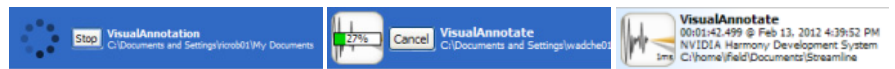


Figure 3-2 From working Streamline Capture Data to Streamline Analysis Report

### 3.1.3 See also

#### Tasks

- [Opening the Capture Options dialog box on page 4-2.](#)

#### Reference

- [Capture options on page 4-3.](#)

## 3.2 ARM Streamline Data view toolbar options

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

- Show Help** Opens a list of help topics for the ARM Streamline Data view in a Help view.
- Delete** Deletes a stored Capture Data or Streamline Analysis report from the file system.
- Refresh** Refreshes the contents of the of the ARM Streamline Data view. If you have added Streamline files to any of your defined locations outside of Eclipse, use **Refresh** to sync this view.
- Address** Enter the IP address of your target here. This can also be defined in the Capture Options dialog box.
- Edit Locations...**  
Opens the Analysis Data Locations dialog box that enables you define the folders on your file system that contain Streamline data.
- Start Capture**  
Starts a capture session. Streamline uses your settings from the Capture Options dialog box. If you have not defined settings using the Capture Options dialog box, Streamline connects to the target at the address that you entered in the **Address** field using default values for each of the capture options.
- Capture options**  
Opens the Capture Options dialog box. Use it to set the parameters of any future capture sessions.
- Counter configuration**  
Opens the Counter Configuration dialog box. Use it to modify the performance counters tracked during your capture session.

### 3.2.1 See also

#### Tasks

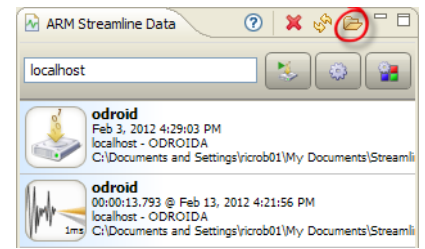
- [Opening the Capture Options dialog box on page 4-2.](#)
- [Opening the Counter Configuration dialog box on page 5-2](#)

#### Reference

- [ARM Streamline Data view basics on page 3-2.](#)
- [The Analysis Data Locations dialog box on page 3-5](#)
- [Using the Counters Configuration dialog box on page 5-4](#)

### 3.3 The Analysis Data Locations dialog box

To open the Analysis Data Locations dialog box, click the **Edit Locations...** button in the ARM Streamline Data view:



**Figure 3-3 The Edit Locations... button**

Use the Analysis Locations dialog box to define the folders on your file system that contain Streamline data. The ARM Streamline Data view shows any Capture Data and Streamline Analysis Reports contained in any of paths listed in this dialog box. To edit an existing path, double-click on it.

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

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

**Note**

There is a sample Streamline Analysis Report in the xaos sample directory if you have not already created one of your own. Use the **Add** button and choose the xaos project directory to make the example Analysis Report appear in the Streamline Data view.

**Remove** Deletes a folder from the list.

**OK** Closes the dialog box. All Capture Data and Streamline Analysis Reports contained in the newly defined folders now appear in the Streamline Data view.

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

**Reset** Returns the defined folders to the default list.

#### 3.3.1 See also

##### Tasks

- [Opening the Capture Options dialog box on page 4-2.](#)

##### Reference

- [Capture options on page 4-3.](#)

## Chapter 4

# Setting Capture Options

The following topics describe how to open the Capture Options dialog box and describe each of the settings it contains:

- [Opening the Capture Options dialog box on page 4-2](#)
- [Capture options on page 4-3](#)

## 4.1 Opening the Capture Options dialog box

The Capture Options dialog box enables you to change capture session settings such as duration, sample length, and buffer size.

To open the Capture Options dialog box:

1. Make sure that you have the ARM Streamline Data view open.

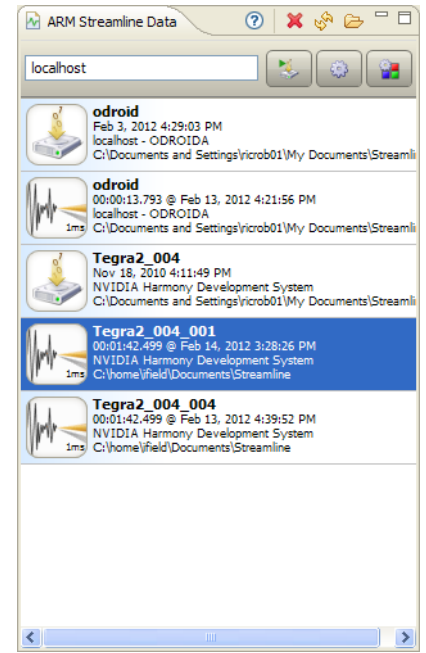


Figure 4-1 The ARM Streamline data view

To open the ARM Streamline Data view, select **Window** → **Show View** → **Other...**, and choose it from the ARM Streamline folder.

2. Click **Change capture options**, located in the upper right of the ARM Streamline Data view.

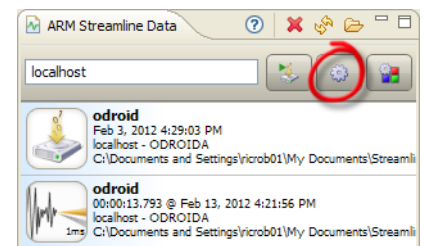


Figure 4-2 The Change capture options button

### 4.1.1 See also

#### Reference

- [Capture options on page 4-3](#)
- [ARM Streamline Data view basics on page 3-2.](#)

## 4.2 Capture options

The Capture Options dialog box presents various options that you can use to customize a capture session.

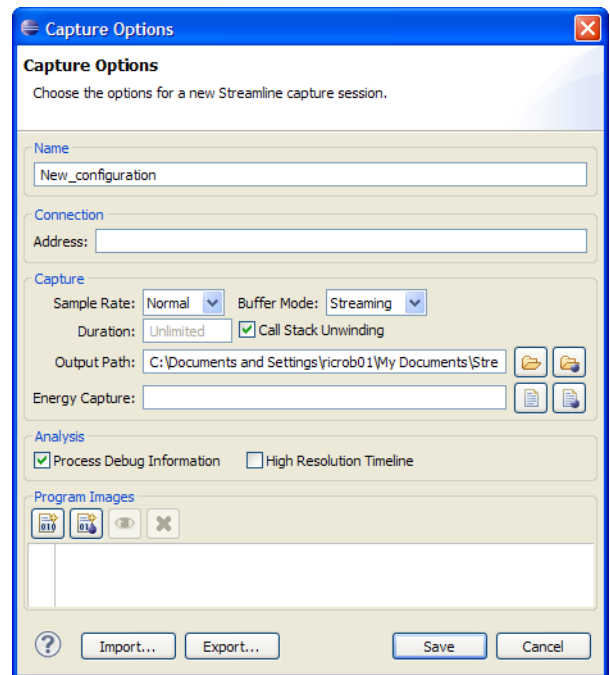


Figure 4-3 Configuration options

It contains the following settings:

**Name** A descriptive name for your capture options. By default, Streamline uses the contents of this field to name capture and analysis files.

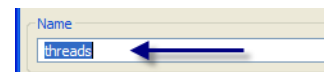


Figure 4-4 Naming the capture options

**Address** The network address of the target. You can also enter the network name of your target here. The value given in this field overwrites the value in the **Address** field of the ARM 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 ARM Streamline data view, it replaces the value entered here.

### Note

By default, Streamline uses port 8080 to connect to a target. To use a different port, specify one here by entering a colon and the desired port number after the IP address. For example, enter *Your\_IP\_address:1010* to use port 1010 to connect to the target.

### Note

If you use the port forwarding of ADB with USB, enter *localhost* in the **Address** field.

**Sample Rate**

The target generates periodic measurement interrupts according to the following settings: **Normal**=1kHz, **Low**=100Hz. The **Normal** setting works well for most instances. **Low** is recommended if you have a slow target, or if the target is heavily loaded, as the **Low** setting means less intrusion by Streamline. The **Low** setting does necessitate a longer capture to collect representative data.

**Buffer Mode**

The default setting is 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:

**Large** 16MB

**Medium** 4MB

**Small** 1MB

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

**Duration**

The length of the capture session, in 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.

**Call stack unwinding**

Select this checkbox to ensure that Streamline records call stacks. This greatly improves your visibility into the behavior of your target, but increases the amount of raw data Streamline sends from the target to the host. Make sure to compile your EABI images and libraries with frame pointers using the `-fno-omit-frame-pointer` compilation option.

———— **Note** ————

Streamline supports callstack unwinding for ARM binaries created using gcc, provided you compile them with frame pointers enabled. Streamline does not support Thumb or output from armcc.

**Output Path**

Use this field to define the directory location and name of the file generated by the capture and analysis session. By default, the file is saved to a results directory defined by an install variable and given the name `@F_@N.apd`. `@F` is a variable for the given configuration name, while `@N` is a sequential number.

**Energy Capture**

Use this field to define the path to the `caiman.exe` executable, necessary to use the ARM Energy Probe to gather power output statistics. Leave this field blank if you do not have Energy Probe or do not wish to capture power data during the capture session. For more information on setting up the Energy Probe, see [Energy Probe setup on page 12-5](#)

**Process Debug Information**

If you select this checkbox, Streamline processes dwarf debug information and line numbers. This provides a higher level of detail in your Analysis Reports, but results in higher memory usage.



---

**Note**

---

If you disable this option, the source section of the Code view does not display the source code or source code statistics. The disassembly code is still available with this option disabled, but the source section shows only a No source available message.

---

**High Resolution Timeline**

If you select this checkbox, Streamline processes more data, enabling you to zoom in three more levels in the Timeline view. By default, the highest resolution in the Timeline view is the millisecond, but with this option enabled, you can zoom into one hundred microsecond, ten microsecond, and one microsecond bin sizes.

**4.2.1 The Program Images section**

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

---

**Note**

---

When compiling images, make sure to set the `-g` compilation option to enable debug symbols. Disabling inlining with the `-fno-inline` compiler setting substantially improves the call path quality.

---

The following buttons are included in the Program Images section of the Capture Options dialog box:



**Figure 4-5 Program images toolbar**

**Add Program...**

Opens a file system dialog box that you can use to choose images to add. Select the image or executable and click **Open** to add the file to the list.

**Add Program from Workspace...**

Opens an image from your Eclipse workspace.

**Toggle Symbol Loading**

Activates or deactivates symbol loading for the selected elf image. An eye appears next to any image or library in the list with symbol loading turned on. Using this option instead of removing an image from the list makes it easy to toggle an image on and off over multiple runs, as it does not remove it from the list of program images.

**Remove**

Removes the selected images and libraries.

**4.2.2 See also****Tasks**

- [Opening the Capture Options dialog box on page 4-2](#)

**Reference**

- [ARM Streamline Data view basics on page 3-2.](#)

# Chapter 5

## Configuring Counters

The following topics describe how to open the Counter Configuration dialog box and describe each of the settings it contains:

- [\*Opening the Counter Configuration dialog box on page 5-2\*](#)
- [\*Using the Counters Configuration dialog box on page 5-4\*](#)

## 5.1 Opening the Counter Configuration dialog box

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

To open the Capture Options dialog box:

1. Make sure that you have the ARM Streamline Data view open.

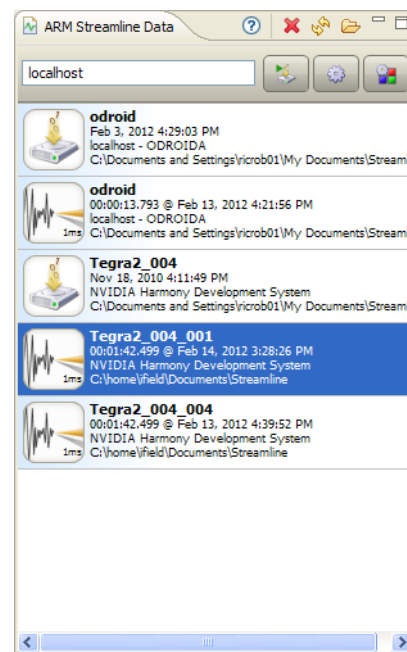


Figure 5-1 The ARM Streamline data view

To open the ARM Streamline Data view, select **Window** → **Show View** → **Other...**, and choose it from the ARM Streamline folder.

2. Click **Counter Configuration**, located in the upper right of the ARM Streamline Data view.

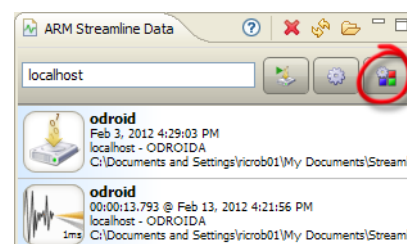


Figure 5-2 The Counter Configuration button

### Note

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

### 5.1.1 See also

#### Reference

- [Using the Counters Configuration dialog box on page 5-4](#)
- [ARM Streamline Data view basics on page 3-2.](#)

## 5.2 Using the Counters Configuration dialog box

The Counter Configuration dialog box has a drag and drop interface to facilitate adding new performance counters to your existing Events to Collect hierarchy.

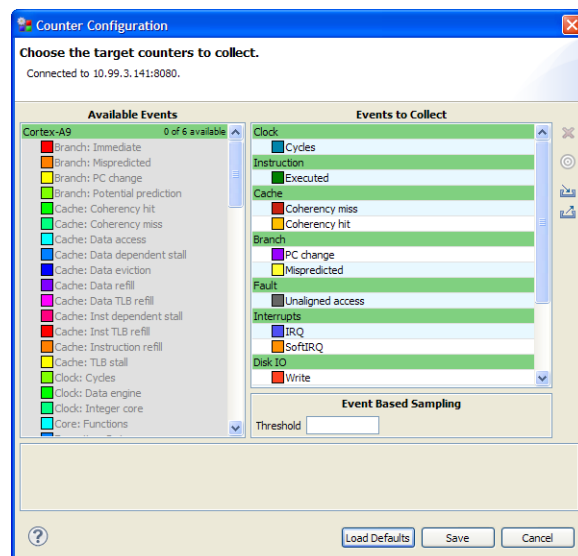


Figure 5-3 The Counters Configuration dialog box

It contains the following two areas:

### Available Events

The Available Events section contains a categorized list of events offered for each core on your current target as well as a list of OS-specific events. The events contained in the processor lists depends on the type of processor as does the number of events that you can add. Events that appear in gray are already in the Events to Collect hierarchy. The maximum number of available events and the amount remaining are in the upper right hand corner of the Available Events list. Once you have reached this maximum, all entries in the list are grayed out and you cannot add any additional events to the Events to Collect hierarchy.

### Events to Collect

This list of categories and events is what the Timeline view uses for its line graphs. Each category listed here appears in the Timeline view as a set of line graphs and each of its subordinate events appears as a graph within it.

### 5.2.1 Adding new events

To add events to the Events to Collect hierarchy, select and drag them from the list of available events and drop them where you want to place them in the hierarchy. A red arrow and line indicate where Streamline places the added events.

The line shows you the order of placement, but the arrow indicates whether the new event is added to an existing category or a new category.

An inset triangle that sits on the left edge of the box of an existing event in the Events to Collect hierarchy means the new event is added to the list. In the example, it is inserted below **L2 miss** in the **Cache** category.

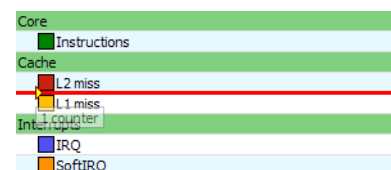


Figure 5-4 Add counter to an existing category

A triangle that sits a bit to the left of the existing events, in line with event categories, means the new event is inserted in a new category. The default name of the new category name is dependent on the type of the event.



Figure 5-5 Add counter to a new category

### 5.2.2 Removing events

To remove events from the Events to Collect hierarchy, select them and press **Delete**. If you select a category and press **Delete**, that category and all of the events it contains are removed from the Events to Collect hierarchy.

### 5.2.3 Customizing the Events to Collect hierarchy

The Events to Collect hierarchy dictates what appears in the Timeline view line graphs. The Timeline view includes a set of colored line graphs for each category that you dictate here. In addition to dictating which events appear, you can change how Streamline labels each event and the color of the events line in the Timeline view.

To change an the title of an event, double click on its text in the Events to Collect section. This opens a Change Counter Name dialog box.

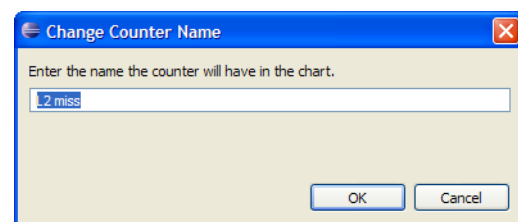


Figure 5-6 The Change Counter Name dialog box

Enter a new name for the counter in the provided field.

To change the color of the event on the Timeline view, double-click on the color box of the event in the Events to Collect section. Doing so opens a Color dialog box.

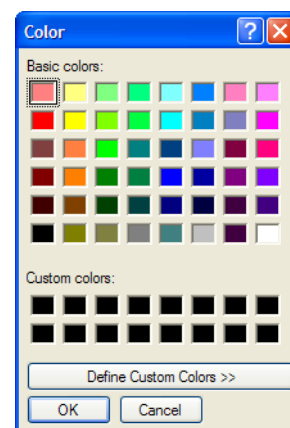


Figure 5-7 The Color dialog box

Pick a new color for the event from the existing options or define a custom color. Streamline uses this defined color in the line charts of the Timeline view.

#### 5.2.4 Mali-specific events

If you build the gator driver with Mali support and connect to a Mali-400 target, there are many additional options in the counter configuration dialog.

For information about the meaning of each of the Mali-specific counters and how to interpret the profiling data please read the Mali GPU Application Optimization Guide, available on the Mali developer site, <http://www.malideveloper.com>.

When choosing which Mali events to add to the Events to Collect hierarchy, consider the following:

- The Mali-400 GPU contains two counters per block, each of which can count one of many events. You can add either or both to the Events to Collect hierarchy.
- 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.

#### 5.2.5 Events-based sampling

By default, Streamline records a sample at an interval determined by the **Sample Rate** drop-down menu in the Capture Options dialog box. To override this, select an event from the Events to Collect list and click the Toggle events-based sampling button on the right side of the dialog box. This activates the **Threshold** field at the bottom of the Events to Collect list.

##### ————— **Note** —————

Events-based sampling is only possible when the PMU on the target hardware can generate interrupts.

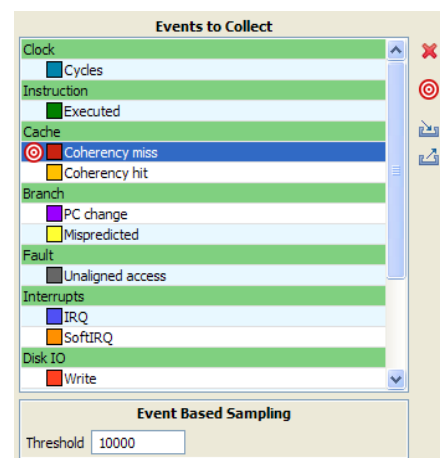


Figure 5-8 Setting events-based sampling

When you have enabled events-based sampling, Streamline samples only when the selected event is triggered a number of times equal to the value in the **Threshold** field, and does so for each core on your target. 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 in many of the Streamline reports now indicate which processes and functions are the potential cause of inefficient caching.

———— **Note** ————

Avoid setting a very low threshold for high frequency events. If you enter a Threshold value that generates too many samples, Streamline could fail, and you might have to restart your target. To find an appropriate value to enter in the **Threshold** field, turn off events-based sampling to run a standard, time-based profile with your desired event counter enabled. Look at the resulting Analysis Report and note the peak per-second value in the chart for your desired counter. Your target for the **Threshold** field is 1000 samples per second, so if the peak for that event is 2,000,000, a good value to insert in the **Threshold** field is 2,000.

### 5.2.6 Counter Configuration options

The Counter Configuration dialog box contains the following buttons:

- |                      |  |
|----------------------|--|
| <b>Help</b>          | Opens Counter Configuration help in the dialog box.  |
| <b>Load Defaults</b> |  |
|                      | Returns the Events to Collect hierarchy to the Streamline defaults.  |
| <b>Import...</b>     | Opens a dialog that enables you to search for a .xml file that you previously generated.   |
| <b>Export...</b>     | Exports the current counters configuration to a .xml file. You can create the .xml file and manually add as an option when running gator on the target, if you choose to capture data locally. |
| <b>Save</b>          | Saves your current counter configurations and exits the dialog box. The counter configuration file is saved to your target.  |
| <b>Cancel</b>        | Exits the Counters Configuration dialog box without saving the defined settings.   |



## 5.2.7 See also

### Tasks

- [Opening the Counter Configuration dialog box on page 5-2](#)

### Reference

- [ARM Streamline Data view basics on page 3-2.](#)

# Chapter 6

## The Timeline View

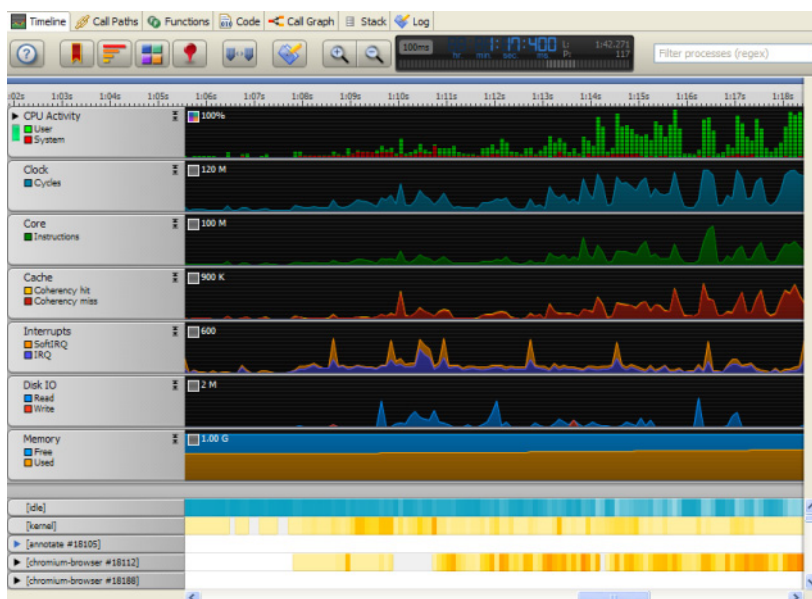
The following topics describe the Timeline view and how to use it:

- *About the Timeline view on page 6-2*
- *Timeline view charts on page 6-8*
- *Filtering data and other Timeline view controls on page 6-10*
- *Timeline view toolbar options, contextual menu options, and keyboard shortcuts on page 6-13*
- *Visual Annotation in the Timeline view on page 6-16*

## 6.1 About the Timeline view

The Timeline view is the first view that you see when ARM Streamline opens a report. It provides you with high level information about the performance of your target during the capture session.

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



**Figure 6-1 The Timeline view**

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

### 6.1.1 Charts

Streamline collects data for the charts from hardware and software performance counter resources. Use the chart expansion control on the right side of the chart handle to increase the size of any chart.



**Figure 6-2 The chart expansion control**

The data is dependent on how you have configured your counters and the type of system you use. For SMP systems, the chart per core disclosure control enables you to expand the data to show collection per core.

Unlike the other charts in the Timeline view, the CPU Activity, GPU Vertex, and GPU Fragments charts are bar charts and the current selection in the processes section affects how they are displayed. When there is an active selection in the processes section the green bars change to reflect CPU activity caused by the selected processes. Streamline still displays the total activity in dark gray so that you can visually compare the selected processes CPU Activity values to the total.

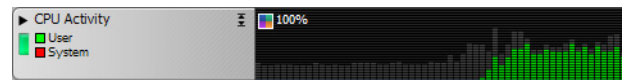


Figure 6-3 A bar chart with an active selection in the processes section

## 6.1.2 Processes

The Processes section of the Timeline view shows you the active processes in each bin. The entries are derived from process/thread trace data from the Linux kernel scheduler. Weighted colors reflect the number of samples in each process or thread.



Figure 6-4 Process bars

<b>White</b>	Not running
<b>Gray</b>	The process has started, but is dormant. It could be sleeping, waiting on user input, or waiting for some other process to finish.
<b>Yellow to red</b>	Responsible for some percentage of total instructions during this bin. Red indicates a higher percentage.

### ———— Note ————

The [idle] process is color-coded differently than 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 appears white.

If you select a process or multiple processes, any bar charts in the Timeline view update to reflect only activity caused by the selected processes. GPU Vertex and Fragment bar charts will display only activity initiated by the selected processes. This allows you to differentiate between GPU activity caused by your application and activity resulting from other applications or system services.

## 6.1.3 Detail bars

The detail bars show functions with the most samples in the currently selected cross-section. Selecting a bar jumps you into the relevant context in the Call Paths view. Double-click on the relevant Capture Data to load more applications and decrease the number of bracketed entries in this list.

The Samples HUD can be turned on and off using the **Samples HUD** button in the toolbar.



Figure 6-5 Timeline Detail Bar

#### 6.1.4 X-Ray mode

X-Ray mode changes the process trace from an intensity map of time, to a mode that highlights core affinity. In this mode, the bars show the mapping of software threads to processor cores. Streamline supports dual-core and quad-core targets. The colors in X-Ray mode are:

<b>Blue</b>	First core
<b>Turquoise</b>	Second core
<b>Amber</b>	Third core
<b>Purple</b>	Fourth core

Hovering the mouse cursor over a color-coded bin shows you which core the color identifies.

X-Ray mode is useful only in SMP systems. All entries in the processes section appear blue in a single core system report.



Figure 6-6 X-Ray mode

### 6.1.5 The process filter

The process filter is located on the right-hand side of the toolbar. Enter a regular expression in the field to filter the processes in the processes section of the Timeline view. For example, if you enter a standard string consisting only of letters, the processes section updates to include only [idle], [kernel], and any processes that contain the entered string. Regular expression strings are case sensitive unless you include (?i) in front of your search expression.

The bar charts in the Timeline view update to display only activity from the remaining processes.

### 6.1.6 Bookmarks

You can create bookmarks in the Timeline view, enabling you to label and quickly return to critical points in the Timeline view. To do so:

1. Double click in the timeline itself. The timeline is the numbered ruler above the charts in the Timeline view.
2. Give the new bookmark a title by entering it into the resulting field.

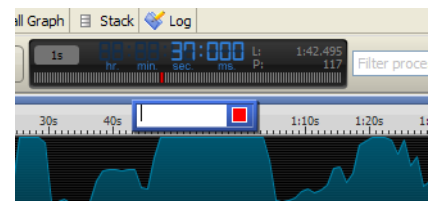


Figure 6-7 Adding a bookmark

3. Choose a color for the bookmark by clicking on the color selector to the right of the bookmark entry field.



Figure 6-8 Choosing a color

4. Use the Color dialog box to choose a color for the new bookmark.

After giving the bookmark a title and color, a bookmark now appears in the timeline. Hover over the bookmark to see an overlay that shows you the title of the bookmark and the time in the capture session. The overlay also provides two buttons:

#### Edit bookmark

Use this button to change the title or color of the bookmark. Double-clicking on a bookmark has the same effect.

### Delete bookmark

Use this button to remove the bookmark from the Timeline.

If you scroll away from the bookmark, you can easily return to it by clicking on the colored mark in the timeline overview.



Figure 6-9 The bookmark in the timeline overview

### 6.1.7 Value pins

In addition to bookmarking you can also place a pin in a particular point in any chart of the Timeline view. To do so, click the **Toggle Value Pin Mode** button in the toolbar and click on a point in any chart.

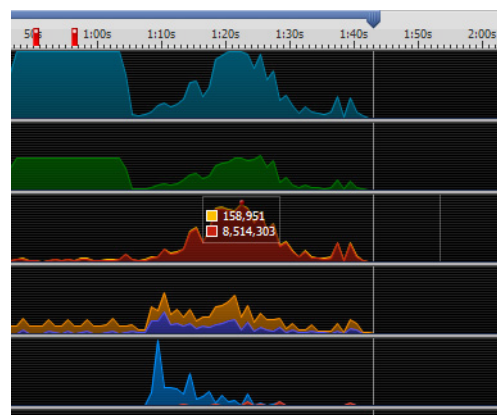


Figure 6-10 A value pin

In value pin mode, a gray line appears under your cursor, so that you can more precisely place each pin. Each value pin shows an overlay with the values of the chart at the location of the pin.

Click on a value pin to remove it.

### 6.1.8 Process focus buttons

The CPU activity, GPU Vertex, and GPU fragments are special charts types. They have a button on the left side of the chart handle that enables you to select that chart as the current focus of the processes section. By default, this is set to the CPU activity chart.

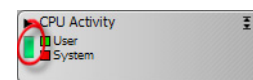


Figure 6-11 The Processes Focus button

If you click a Processes Focus button on one of the bar charts, the processes section updates to show a heat map of processes that contributed to activity in the newly selected bar chart. For example, if you click the Processes Focus button on the GPU Vertex chart, the processes section updates to show heat mapped activity for the GPU vertex processor only. The GPU charts are only available if you have captured data on a Mali-400 target and built the gator module to support Mali.





## 6.2 Timeline view charts

You can customize any of the charts in the Timeline view. Open the Configure Counters dialog box to define which counters Streamline tracks and under which charts those counters appear. Available counters depend on your hardware type. For more information on how to customize these charts, see [Using the Counters Configuration dialog box on page 5-4](#).

Here are some of the default charts in the Timeline view:

<b>Clock</b>	The number of cycles used by each core.
<b>Core</b>	The total number of instructions executed by each core.
<b>Cache</b>	The number of cache coherency hits and misses. A cache coherency miss occurs every time a processor tries to read from or write to the cache when it is in incoherent state. This is more common in SMP systems where multiple processors share resources. A coherency hit occurs when a processor reads from or writes to a cache that is in a valid state.

### CPU Activity

The percentage of the CPU time spent in system or user code, the remainder being idle time. Unlike most of the other charts in the Timeline view, the CPU Activity chart is a bar chart and the green bars depend on the active selection in the processes section.

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

<b>Disk IO</b>	Measures the number of times the core triggered reads and writes to disk.
----------------	---

<b>Memory</b>	Charts the available system memory over the time of the execution.
---------------	--

If you are using a target with more than one core, some graphs in the Timeline view have a disclosure control. Use this to break each graph into multiple sections, one for each core in your system.

### 6.2.1 Charts specific to Mali-400 targets

If you have built gator.ko to support Mali and have run a capture session on a Mali-400 target, the following charts are added to the default set of Timeline view charts:

- GPU Vertex chart - Streamline reports whether the status of the Mali-400 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 used to transform their coordinates.
- GPU Fragment chart - Streamline reports whether the status of the Mali-400 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 used to determine the final pixel color. Pixels rendered include on and off screen pixels and the additional pixels required for super sampling.

### 6.2.2 See also

#### Tasks

- [Customize reports using Annotate on page 10-2](#)
- [Creating custom performance counters on page 11-4](#).

## Reference

- [Filtering data and other Timeline view controls on page 6-10](#)
- [Timeline view toolbar options, contextual menu options, and keyboard shortcuts on page 6-13.](#)

## 6.3 Filtering data and other Timeline view controls

The Timeline view provides a number of controls that enable you to customize your perspective. The caliper controls at the top of the Timeline view control filtering. The gray title boxes on the left of each chart and process are handles that you can use to drag and drop the charts and processes to re-order them in your preferred order. The marker control can be dragged left and right and even expanded to include a range of units.

### 6.3.1 Filtering using the caliper controls

The Timeline view contains calipers that you can use to set the specific window of time on which you want to focus. Streamline updates each of the report views based on where the in and out markers are set.

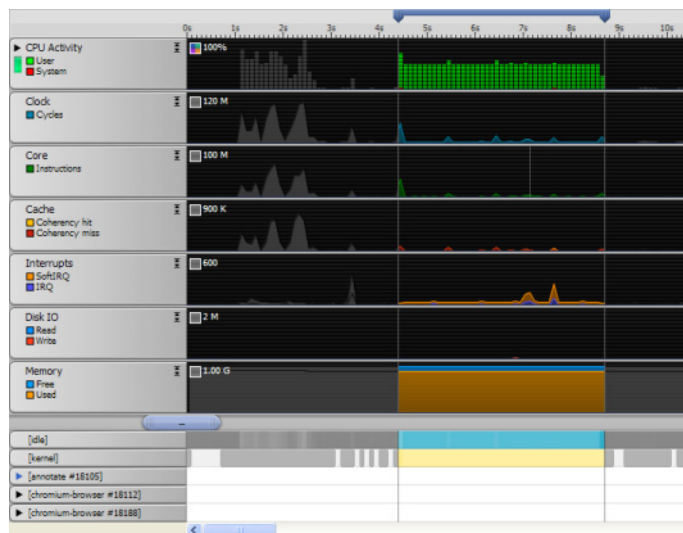


Figure 6-13 Using the calipers to filter

In the example shown in the figure, the caliper controls are used to narrow the focus of the report. Only data relevant to this interval appears in the other views, so the Code Paths, Functions, Code, Call Graph and Stack views update when you move the calipers.

### 6.3.2 Re-ordering charts and processes using the handle controls

Each chart and process has a rounded box on the left that contains the title, and in the case of charts, a color-coded key. You can also use these rounded bars, called handles, to drag and drop individual charts and processes into a preferred order.

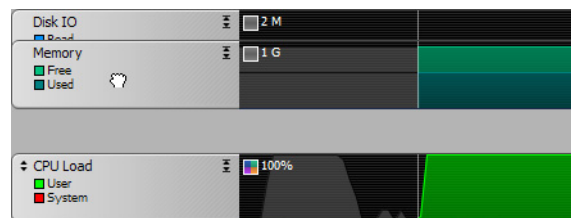
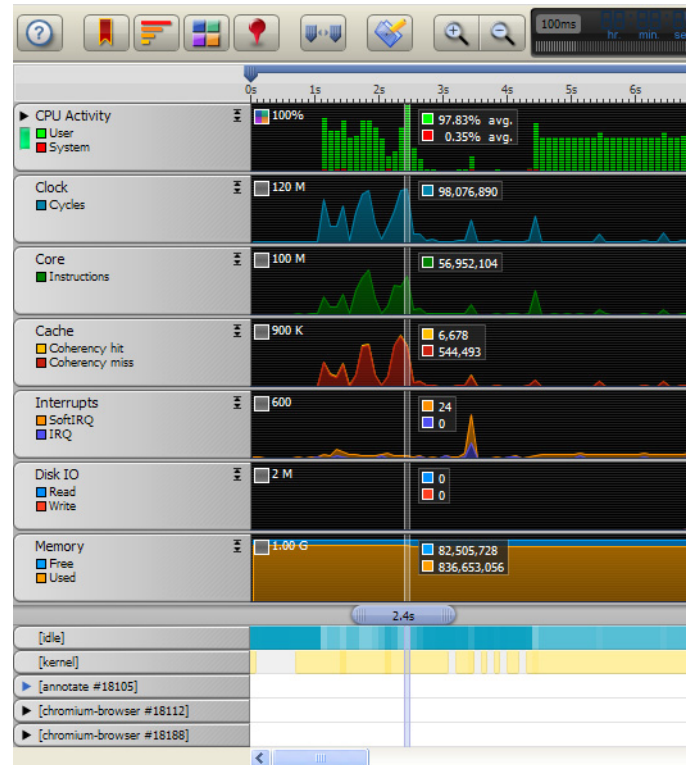


Figure 6-14 Moving a chart using the handle control

To re-order a chart or process, 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.

### 6.3.3 The cross-section marker

By default, the cross-section marker is inactive and the cross-section marker handle rests on the far left of the divider bar. To move the cross-section marker, click anywhere in charts or processes. The marker appears where you clicked and provides data specific to the bin where you placed it.



**Figure 6-15 The cross-section marker**

You can also stretch the marker using the handle in the divider bar that splits the charts and processes section. Click and drag on either that right or the left side of the handle to expand it. The information contained in the samples HUD relates only to the window of time defined by the current 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.

### 6.3.4 The divider

The divider bar that separates the charts from the processes section is also a control. Click and drag on the divider to change the proportion of the window that the charts and processes sections use. Drag the bar down to reveal more charts and drag the bar up to show more processes.

### 6.3.5 See also

#### Tasks

- [Customize reports using Annotate on page 10-2](#)
- [Creating custom performance counters on page 11-4](#)

## Reference

- *About the Timeline view on page 6-2*
- *Timeline view toolbar options, contextual menu options, and keyboard shortcuts on page 6-13*

## 6.4 Timeline view toolbar options, contextual menu options, and keyboard shortcuts

ARM Streamline provides easy ways to navigate and modify the Timeline view using the toolbar and a variety of contextual menu options and keyboard shortcuts.

### 6.4.1 Toolbar options

The toolbar buttons in the Timeline are:

**Help** Brings up contextual help in Eclipse.

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

**Toggle HUD Display**  
Toggles the Samples HUD on and off.

**X-Ray mode**  
Toggles x-ray mode on and off. When on, Streamline replaces the heat map in the Processes section with a color-coded map that shows which core spent more time on a process or thread in the given increment. Blue in the Processes section means the first processor spent the most cycles, while turquoise indicates that the second processor did. Amber represents the third core and purple the fourth. In a single core system, all active areas in the Processes section are blue. Hover over a color in the processes for a tooltip that tells you which core the color represents.

**Value pin mode**  
Toggles Value pin mode on and off. If you click anywhere in one of the charts of the Timeline view while this mode is active, you place a pin with an overlay that shows you the values for that chart at the placement of the pin. Clicking on a chart while this mode is inactive moves the cross-section marker.

**Reset filtering**  
Reset the calipers. The calipers are the blue arrow controls at the top of the Timeline view that you can use to filter the statistics in the reports to an area of interest.

**Select Annotation log...**  
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. For more information on how to use the Annotate feature, see [Customize reports using Annotate on page 10-2](#).

**Zoom in one level, Zoom out one level**  
Enable you to cycle through the levels of zoom. Use the plus and minus buttons to increase or decrease the units of time used to represent one bin in the Timeline view.

**Time index marker**  
Shows the time index of the current location of the mouse cursor.

## 6.4.2 Contextual menu options

Right-click anywhere in the charts or processes sections of the Timeline view to open a contextual help menu that enables you to change the current position of the caliper filtering controls. Right-clicking on a chart or process handle does not produce a contextual menu.

The basic contextual menu options are:

### Set left caliper

Sets the left caliper control to the current location. Streamline filters all data before the left caliper from all views.

### Set right caliper

Sets the right caliper control to the current location. Streamline filters all data after the right caliper location from all views.

### Reset calipers

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

If you right-click on a bookmark to open a contextual menu, there are two additional options:

### Rename bookmark

Opens the small interface that enables you to change the title and color of your bookmark.

### Delete bookmark

Removes the bookmark from the Timeline view.

## 6.4.3 Keyboard shortcuts

While you can navigate every report in ARM Streamline using the mouse, you can also use keyboard shortcuts. The keyboard shortcuts available for the Timeline view 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 cross section marker if it is wider than one bin.

### Shift + right arrow

Expands the width of cross section marker.

<b>L</b>	Opens the Log tab and highlights the annotation indicated by the cross-section marker
<b>P</b>	Turns value pin mode on and off.
<b>S</b>	Reveals or hides the Samples HUD.
<b>X</b>	Turns X-Ray mode on and off.
<b>I, Ctrl +</b>	Zooms the Timeline view in one level.
<b>O, Ctrl -</b>	Zooms the Timeline view out one level.

- Ctrl+Z** Triggers the undo command. It steps back one change to the filtering in the Timeline View.
- Ctrl+Y** Triggers the redo command. It steps forward one change to the filtering in the Timeline view, if you have previously used undo.

#### 6.4.4 See also

##### Tasks

- [Customize reports using Annotate on page 10-2](#)
- [Creating custom performance counters on page 11-4](#)

##### Reference

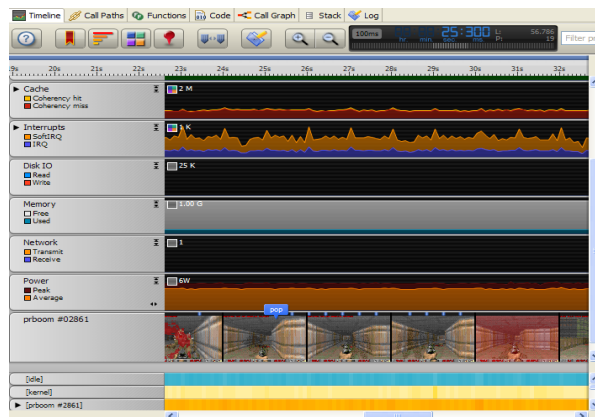
- [About the Timeline view on page 6-2](#)
- [Filtering data and other Timeline view controls on page 6-10.](#)



## 6.5 Visual Annotation in the Timeline view

If you used the Visual Annotate feature to add images to the capture data, they appear in the charts section of the Timeline view.

For instructions on how to use the Visual Annotate feature, see [Adding images to reports using Visual Annotate on page 10-6](#).



**Figure 6-16 Visual annotation in the Timeline view**

Rows in the charts view that contain images can be re-ordered in the same way as the other charts. But rows containing images have a few unique properties:

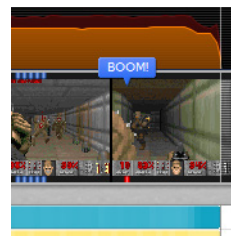
- Hovering your mouse over the images in a visual annotation chart shows you the image for the current position of the mouse. If there is more than one image in the twenty bin range covered by each thumbnail pictured in a visual annotation chart, Streamline uses the first image from the range. Moving the mouse over that thumbnail reveals the other annotated images in that twenty bin range.
- Hover over an image in the Timeline view and two red markers appear above and below the image.



**Figure 6-17 Hovering over an image in the Timeline view**

The red markers show you in which bin your code produced the current image. Move the mouse right and left and these markers move each time you move to a new image.

- Blue markers above and below images mark images annotated with text.



**Figure 6-18 Text with visual annotation**

Hover over a blue marker and the text associated with that image appears above the upper marker.

# Chapter 7

## The Table Views: Call Paths, Functions, and Stack

Although the information contained in the Call Paths, Functions, and Stack views differ, their basic functionality is similar. Each one is a table report, with rows for each function and columns for each statistical category.

The following topics describe the common functionality of the table views and then breaks down the statistics available in each of them.

- [\*Table views toolbar options, contextual menu options and keyboard shortcuts on page 7-2\*](#)
- [\*Sorting data in the table reports on page 7-5\*](#)
- [\*Call Paths view column headers on page 7-6\*](#)
- [\*Functions view column headers on page 7-7.\*](#)
- [\*Stack view column headers and the Maximum Stack Depth by Thread chart on page 7-8\*](#)

## 7.1 Table views toolbar options, contextual menu options and keyboard shortcuts

Streamline provides a variety of ways to interact with the table views. You can use the toolbar, contextual menu options, or keyboard shortcuts to move through the data or change options.

### 7.1.1 Toolbar options

The following options are available from the toolbar:

**Help** Opens contextual help in Eclipse.

**Percentages**

If this button is active, the table view shows Self and Process as percentages.

**Totals** Displays both time fields as total units of time. The time tooltips still displays both percentages and time units, regardless of whether you have activated Totals or Percentages.

**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.

**Edit Source**

Opens the source file in your preferred editor. Without any rows selected in the table view, the **Edit Source** option is disabled.

**Fully expand all rows**

*Call Paths view only* - Shows the entire hierarchy. It has the effect of opening disclosure controls to reveal all processes, thread, 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.

### 7.1.2 Contextual menus

To open a contextual menu, right click in any table. The following options are available from contextual menus in the table views:

**Select Process/Thread in Timeline**

Opens the Timeline view with the currently selected processes, threads, and functions selected in the Processes section. Handles for selected processes in the Timeline view appear partially blue.

**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.

**Select in Call Graph**

Opens the Call Graph view. Any function in the current selection is selected in the Call Graph view.

**Select in Stack**

Opens the Stack view. All functions related to the selection in the current report are selected in the Stack view.

**Edit Source**

Opens the file that contains the process, thread, or function in your default code editor.

**Fully Expand Rows**

*Call Paths view only* - Opens the hierarchy to reveal every descendent of the selected functions.

**Expand Selection to All Matching Functions**

*Call Paths view only* - 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.

**Collapse Unselected Rows**

*Call Paths view only* - Collapses every row in the hierarchy that is not part of the current selection.

**7.1.3 Keyboard shortcuts**

While you can navigate every report in ARM Streamline using the mouse, you can also use keyboard shortcuts. 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**

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.

**7.1.4 See also****Reference**

- [Filtering data and other Timeline view controls on page 6-10](#)
- [Sorting data in the table reports on page 7-5](#)
- [Call Paths view column headers on page 7-6](#)
- [Functions view column headers on page 7-7](#)
- [Stack view column headers and the Maximum Stack Depth by Thread chart on page 7-8.](#)

## 7.2 Sorting data in the table reports

To change the sort order, click once on any of the column headers. The data in the table views is reordered based on the data contained in that column. To reverse the sort order, click in the same column header again. 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.

You are not limited to a simple one-level sort. You can specify as many chained sort criteria as there are columns. To specify more levels in the sort hierarchy, hold down the shift key and click on other columns until you have achieved your desired sequence. Such a sort is best illustrated in an example. To first sort by Self and then by the Process/Thread/Function name, click twice on the Self column header, and then shift-click on the Process/Thread/Function Name header. Your Call Paths view now looks like the one pictured:

Self	Process	Total	Stack	Process/Thread/Function Name	Location
0.00%	100.00%	1.77%	0	[Xorg #3663]	-
0.00%	100.00%	0.08%	0	[wnck-applet #8386]	-
0.00%	100.00%	< 0.01%	0	[upstart-udev-bridge #3063]	-
0.00%	100.00%	0.04%	0	[upowerd #3946]	-
0.00%	0.00%	0.00%	0	[update-notifier #8467]	-
0.00%	100.00%	< 0.01%	0	[unix_chkpwd #18071]	-
0.00%	100.00%	0.08%	0	[unix_chkpwd #18069]	-
0.00%	100.00%	0.08%	0	[unix_chkpwd #18068]	-
0.00%	0.00%	0.00%	0	[udev #3229]	-
0.00%	100.00%	< 0.01%	0	[udev #3228]	-
0.00%	100.00%	< 0.01%	0	[udev #3074]	-
0.00%	0.00%	0.00%	0	[trashapplet #8387]	-
0.00%	100.00%	1.16%	0	[tfft.xcf #18097]	-
0.00%	24.90%	0.29%	0	[thread #18101]	-
0.00%	24.90%	0.29%	32	runFFT	tfft.c:127
16.03%	24.90%	0.29%	64	FFT	tfft.c:86
5.54%	8.88%	0.10%	96	BitReverse	tfft.c:72
3.34%	3.34%	0.04%	112	Rot	tfft.c:67
0.00%	24.82%	0.29%	0	[thread #18100]	-
0.00%	24.82%	0.29%	0	[thread #18099]	-
0.00%	25.41%	0.30%	0	[thread #18098]	-
0.00%	0.04%	< 0.01%	0	[thread #18097]	-

Figure 7-1 A multi-level sort

Sorting in the Call Paths view works differently than in the other report types. You can still click various columns to add sort criteria and change the direction of the sorts, but the sort criteria does not break the integrity of the call paths so the hierarchy of the call paths is not rearranged. The order in which the children of a particular function appear depends on the sort criteria.

The sort triangles show the direction of sort for each field, and the dots in the lower right of the column headers indicate ordering. In this case, Self has one dot, indicating that it is the primary sort criteria, and Process/Thread/Function Name has two dots because it is the secondary sort criteria.

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

### 7.2.1 See also

#### Reference

- [Filtering data and other Timeline view controls](#) on page 6-10
- [Table views toolbar options, contextual menu options and keyboard shortcuts](#) on page 7-2
- [Call Paths view column headers](#) on page 7-6
- [Functions view column headers](#) on page 7-7
- [Stack view column headers and the Maximum Stack Depth by Thread chart](#) on page 7-8.

## 7.3 Call Paths view column headers

Here is a list of all of the column headers contained in the Call Paths view:

<b>Self</b>	Self time is a measure of time spent in a function, without including time spent in descendant functions. It reports these amounts here, either as a percentage of total samples, if the <b>Percentages</b> button is active, or as a total of samples, if the <b>Totals</b> button is active.
<b>Process</b>	The total time spent executing this function measured against the time spent executing the process as a whole.
<b>Total</b>	Total time represents the amount of time used by the function and all of the functions it calls.
<b>Stack</b>	The number of bytes used by the stack in this function. A question mark is presented if the stack usage of the function is unknown.

### Process/Thread/Function Name

The name of the process, thread, or function.

#### ———— Note ————

If you disabled call stack unwinding in the Capture Options dialog box, the sampled functions all appear directly under the threads in the Call Paths view. For more information on call stack unwinding, see [Capture options on page 4-3](#).

<b>Location</b>	This column reports the location of the function, listing both the file name and line of the declaration.
-----------------	---

#### ———— Note ————

All data in the Call Paths 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 data in the Call Paths reflects this selection.

### 7.3.1 See also

#### Reference

- [Filtering data and other Timeline view controls on page 6-10](#)
- [Table views toolbar options, contextual menu options and keyboard shortcuts on page 7-2](#)
- [Sorting data in the table reports on page 7-5](#)
- [Functions view column headers on page 7-7](#)
- [Stack view column headers and the Maximum Stack Depth by Thread chart on page 7-8.](#)



## 7.4 Functions view column headers

Here is a list of all of the column headers contained in the Functions view:

<b>Self</b>	Self time is a measure of time spent in a function, without including time spent in descendant functions. It reports these amounts here, either as a percentage of total samples, if the <b>Percentages</b> button is active, or as a total of samples, if the <b>Totals</b> button is active.
<b>Instances</b>	The number of times the functions appears in the Call Paths view.
<b>Function Name</b>	The name of the function.
<b>Location</b>	This column reports the location of the function, listing both the file name and line of the declaration.
<b>Image</b>	The image file that contains the function.

---

### Note

---

All data in the Functions 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 data in the Functions view reflects this selection.

---

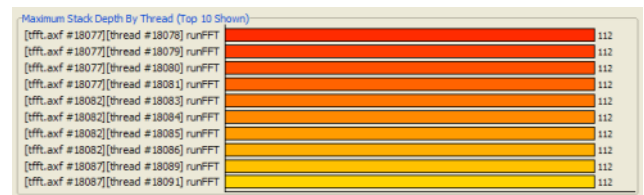
### 7.4.1 See also

#### Reference

- [Filtering data and other Timeline view controls on page 6-10](#)
- [Table views toolbar options, contextual menu options and keyboard shortcuts on page 7-2](#)
- [Sorting data in the table reports on page 7-5](#)
- [Call Paths view column headers on page 7-6](#)
- [Stack view column headers and the Maximum Stack Depth by Thread chart on page 7-8.](#)

## 7.5 Stack view column headers and the Maximum Stack Depth by Thread chart

At the top of the Stack view is a chart that lists best candidate for stack optimization for each thread.



**Figure 7-2 Maximum Stack Depth by Thread chart**

Beside each bar is the name of the process that called the function, then the thread, then finally the function itself. This function exists in the call path with the highest stack depth for the thread, and, out of all the functions that exist in this call path, it used the most bytes.

Clicking on one of the bars filters the stack view to feature only functions that contributed to the peak stack for that thread. Click the **Show All** button to return the Stack view to its default state.

Here is a list of all of the column headers available in the Stack view:

**Stack** The number of bytes used by the stack in this function. A question mark appears next to the total here if the function's stack usage is unknown. Try turning on Call stack unwinding in the Capture Options dialog box for best results in this view.

**Size** The total size of the function, in bytes.

**Function Name**

The name of the function, as specified in the source code.

**Location** This column reports the location of the function, listing both the file name and line of the declaration.

———— **Note** —————

The Stack view depends on successful call stack unwinding. For more information on setting the **Call Stack Unwinding** option, see [Capture options on page 4-3](#).

———— **Note** —————

All data in the Stack 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 data in the Stack view reflects this selection.

### 7.5.1 See also

#### Reference

- [Filtering data and other Timeline view controls on page 6-10](#)
- [Table views toolbar options, contextual menu options and keyboard shortcuts on page 7-2](#)
- [Sorting data in the table reports on page 7-5](#)
- [Call Paths view column headers on page 7-6](#)
- [Functions view column headers on page 7-7](#).

# Chapter 8

## The Code View

Of all of the views available in ARM Streamline, the Code view provides the highest level of detail. It breaks statistics down by individual line of code and disassembly instruction.

The following topics describe the use of the Code view:

- [Code view basics on page 8-2](#)
- [Code view toolbar options and keyboard shortcuts on page 8-5](#)

## 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 see both code and disassembly instructions, click the Disassembly view button to display the Disassembly view.

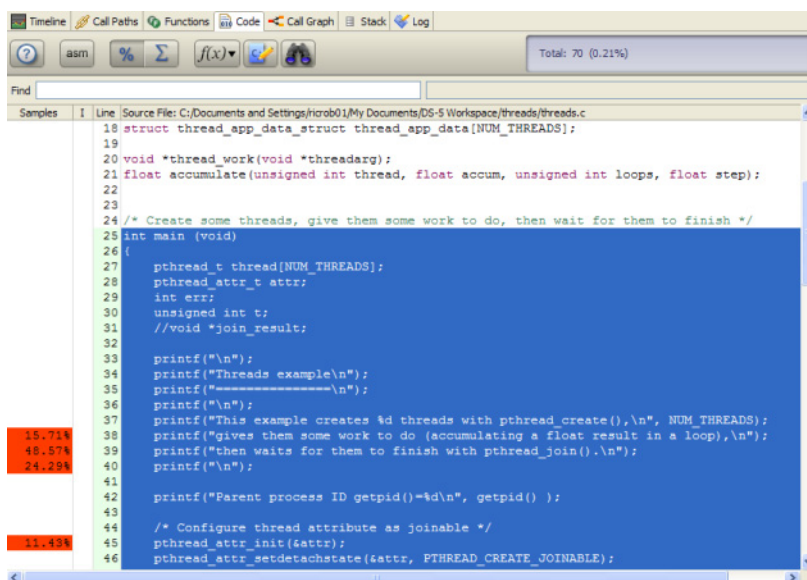


Figure 8-1 The Code view

The Code view presents the percentage each source line or disassembly entry contributed to the total samples collected for the function.

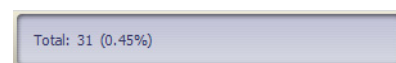


Figure 8-2 The Totals panel

### ———— Note ————

All 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 data in the Code view reflects this selection.

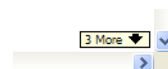
### 8.1.1 Code view selection behavior

Selecting code in the Code view highlights related instructions in the disassembly panel. This feature ignores coding comments.

Click an instruction in the disassembly panel to select all of the instructions that relate to a single line of source code. Click on a function label in the disassembly view to select all of the instructions and lines of code that make up that function.

To select multiple rows, hold down the mouse button and drag it across a range of rows. Selection behavior available in other applications is also present here. Hold down the shift key and select the first and 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.

If selected source code lines or disassembly instructions contain too many rows to fit in the bounds of the current window, small selection indicators appear 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.

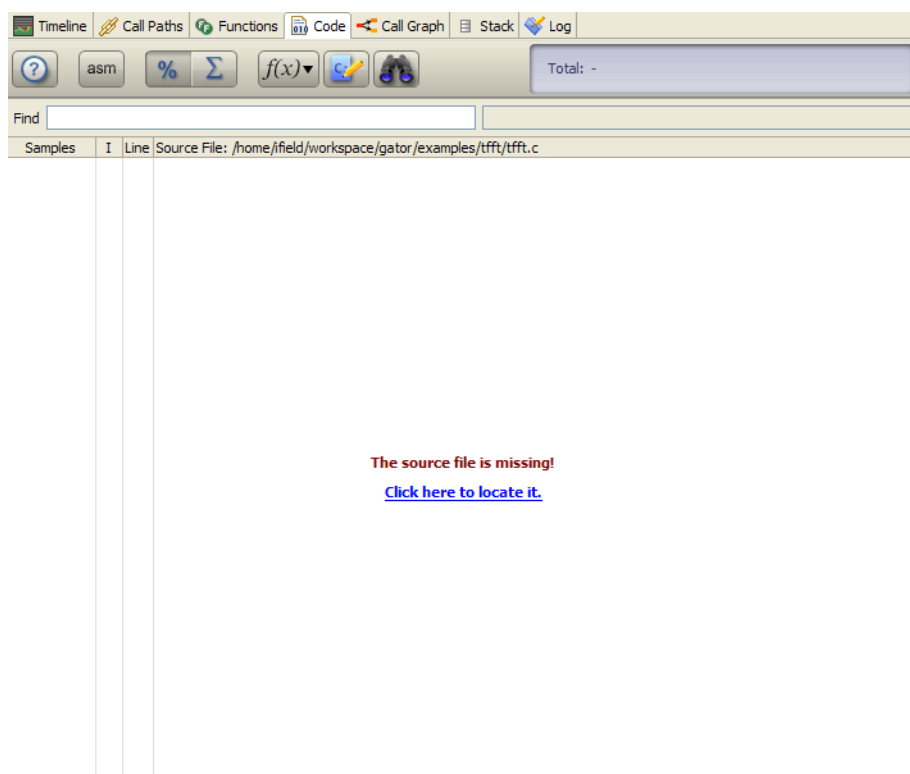


**Figure 8-3 The More indicator**

Click the **More** indicator to see additional selected rows.

### 8.1.2 Locating missing source files

ARM Streamline automatically locates and displays the source code in the source view. If, however, the source files are not located in the same directory location they were in during compilation, the source view is not populated.



**Figure 8-4 Missing source file**

To populate the source view, you must locate the version of the source file used to create the analysis report. Click the link in the source view to open the Locate dialog box, for example:

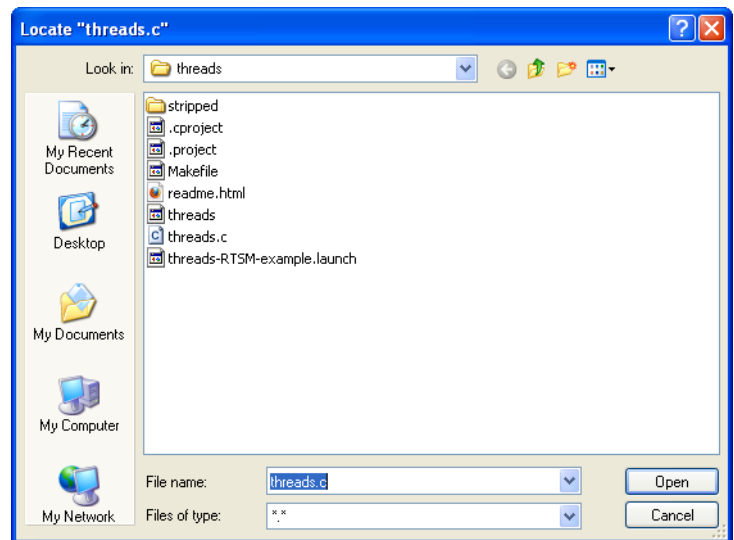


Figure 8-5 Locate source dialog box

Locate the source file, select it, and click the **Open** button. Streamline now populates the view with your source code and the statistical overlay. The Locate source dialog box is a standard file navigation window, and varies depending on your operating system.

### 8.1.3 The find command

To search your code and instructions for a function name or a hexadecimal instruction address, use the Find field, located just below the toolbar.

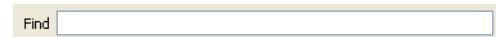


Figure 8-6 The Find field

Enter a string and the field on the right hand side updates to show the current match, if there is one. Pressing the Enter key takes you to the first match in the code and subsequent presses of the Enter key cycles through all of the available matches.

### 8.1.4 See also

#### Tasks

- [Opening the Capture Options dialog box on page 4-2.](#)

#### Reference

- [Filtering data and other Timeline view controls on page 6-10](#)
- [Code view toolbar options and keyboard shortcuts on page 8-5](#)
- [ARM Streamline Data view basics on page 3-2](#)
- [Capture options on page 4-3.](#)

## 8.2 Code view toolbar options and keyboard shortcuts

Streamline provides both toolbar buttons and keyboard shortcuts that enable you to interact with the code view.

### 8.2.1 Toolbar options

Here are the toolbar buttons available in the Code view:

**Help** Opens contextual help in Eclipse.

#### **Disassembly View**

Opens the disassembly panel. The disassembly panel takes up the bottom section of the Code view and shows the instructions associated with the source code.

#### **Percentages**

If this button is active, the code view shows Self and Process as percentages.

**Totals** Both time fields are shown as total units of time. The time tooltips still displays both percentages and time units, regardless of which of these two options are selected.

#### **Edit Source**

Opens the source file in your preferred editor.

**Find** Opens a dialog box that enables you to search your file system to find the source file that matches the source code used during profiling. Use this button if the source file is missing or you've previously selected an out of sync source file.

### 8.2.2 Keyboard shortcuts

While you can navigate every report in ARM Streamline using the mouse, you can also use keyboard shortcuts. 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** The home key 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** The end key 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.

### 8.2.3 See also

#### Tasks

- [Opening the Capture Options dialog box on page 4-2.](#)

#### Reference

- [Code view basics on page 8-2](#)
- [ARM Streamline Data view basics on page 3-2.](#)
- [Capture options on page 4-3.](#)



# Chapter 9

## The Call Graph View

The Call Graph view is a visual representation of your code hierarchy. Streamline places every called function in a tree and connects calling and called functions with arrows. It includes many controls that aid you in navigation.

The following topics describe the Call Graph view and how to use it:

- [\*Call Graph view basics\*](#) on page 9-2
- [\*Contextual menu options\*](#) on page 9-5
- [\*The toolbar and keyboard shortcuts\*](#) on page 9-6

## 9.1 Call Graph view basics

The Call Graph view provides you with a visual representation of your code hierarchy, laying out each function according to where it is called and using arrows to connect calling functions. The direction of the arrow indicates which function was the calling function. An arrow pointing to a function tells you that function is the callee and the function from which the line originates is the calling function. This topic describes the layout of the call graph in more depth, providing a quick overview of how the hierarchy is built and what the bullets to the right and left of the functions represent.

### 9.1.1 How the hierarchy is built

The hierarchy of functions, as presented in the Call Graph view, is built based on the call chains sampled during execution.

#### ———— Note ————

The Call Graph view depends on call stack unwinding, an option you can set using the Capture Options dialog box. For more information on call stack unwinding, see [Capture options on page 4-3](#).

The originating function is placed in the far left column and functions it calls are placed in the column to its right. Functions that these functions call are placed in a column to the right of that and so on down the line, until all of the functions have been placed. There is a caveat to this placing behavior. If a function is called at multiple levels of the hierarchy, it is placed as far left as possible in the Call Graph view.

To illustrate, if the function `main` calls function `a` which in turn calls function `b`, it looks like this:



**Figure 9-1 A Simple Call Hierarchy**

If, in addition to function `a`, `main` also calls function `b`, function `b` is put in a higher place in the hierarchy, nearer to `main`, it might look like this:



**Figure 9-2 A Call Hierarchy with Multiple Call**

The Call Graph view presents a simple call hierarchy, but real-world algorithms describe hierarchies far more complex than those shown in the figures. Rather than present the Call Graph view with all of these connections visually represented as a spider web of call arrows, the Call Graph view uses a simple method to determine whether or not to draw a call line.

### 9.1.2 Caller and callee bullets

In cases where the calling function is in the same column or in a column to the left or right of the called function, a call arrow is drawn from the caller to the callee. If, however, the called function appears in a column more than one column to the left of the calling function, a bullet is added to the left of the calling function and to the right of the called function. The number contained in the bullet represents how many calling or called functions the bullet represents.

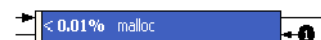


Figure 9-3 Caller Bullets

Right-click on a caller or callee bullet to see all of the functions contained in it. Choose a function in the contextual menu to select and center that function in the Call Graph view. In this way, all of the calling and called functions are still easily accessible, but call arrows are not used to cross many layers of the hierarchy.

### 9.1.3 The mini-map

In the bottom left hand corner of the Call Graph view is a mini-map that can be used to easily navigate around the Call Graph view when the hierarchy is too large to fit in the editor section of Eclipse.

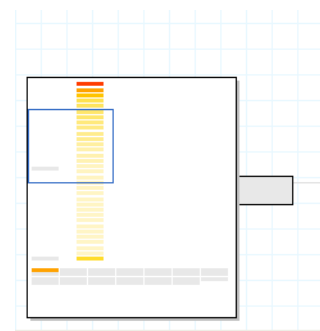


Figure 9-4 The Call Graph Mini-map

When inside the mini-map, but outside the draggable view area, the cursor changes into a crosshair. Click on a location in the mini-map to center on that section. If you hover over the draggable area box within the mini-map, the cursor changes into a hand that enables you to click and drag the view area. Panning the view area in this way enables you to quickly scan sections of the hierarchy without using the scroll bars.

The objects in the mini-map have the same color coding as the functions in the Call Graph view itself. The bright red function in the hierarchy appears as bright red in the mini-map so that you can use the mini-map to quickly zoom to a bottleneck. Selected functions appear dark blue in the mini-map.

You can hide the mini-map by using the **Hide Mini-map** button, located to the left of the drop-down menu in the toolbar.

### 9.1.4 Color coding

The Call Graph view color codes the functions according to total samples so the critical functions are quickly identifiable in the Call Graph view. These colors range from bright red to light yellow, red being the highest value, light yellow the lowest. These colors are easily identifiable in the mini-map so you can scroll quickly to the critical functions.

### 9.1.5 Selection behavior

Left-clicking on any function in the hierarchy selects it. In addition to coloring the rectangle dark blue, it changes the color of all of the arrows from gray to black, clearly showing you to what functions the selection function is connected.

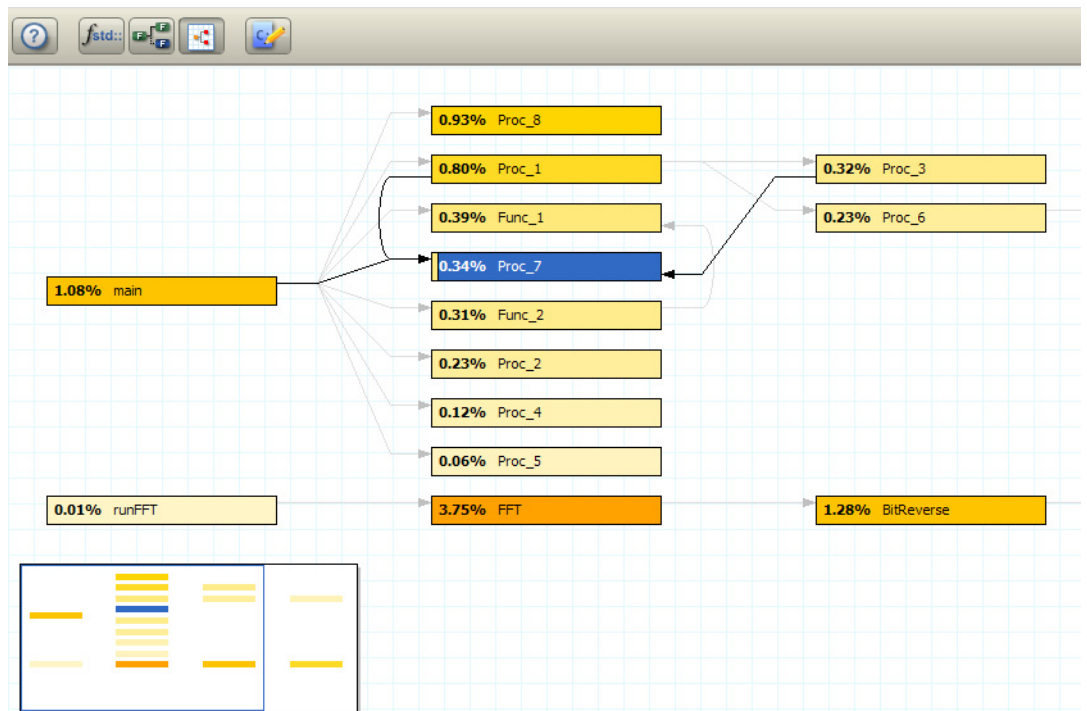


Figure 9-5 A Selected Function

Selecting a function also changes any connected caller or callee bullets from gray to black.

### 9.1.6 See also

#### Reference

- [The toolbar and keyboard shortcuts on page 9-6](#)
- [Contextual menu options on page 9-5.](#)

## 9.2 Contextual menu options

Right-click on any function in the call graph to open a contextual menu, which presents you with a list of selection and navigation options so that you have easy access to that function's calling and called functions, in addition to detailed statistics for that function via the table reports.

Right-clicking on the Call Graph view without a function selected opens a contextual menu that contains only two options, **Show System Functions** and **Show Uncalled Functions**.

The contextual menu options for functions in the Call Graph view are:

**Callers** Use the arrow to the right of this contextual menu option for a complete list of all of the functions that called the selected function. Choosing a function from this list selects it in the hierarchy. If the function is the root function, the Call Graph view grays out this menu option.

**Callees** This menu option works identically to the **Callers** menu option, only it contains a selectable list of functions that are called by the selected function. If the selected function does not call any functions, the Call Graph view grays out this menu option.

### Select Callers

Selects every function that called the selected functions.

### Select Caller Tree

Selects the chain of functions that led to the call of the selected functions.

### Select Callers and Callees

Selects any function that called or was called by the selected functions.

### Select Caller Tree and Callee Tree

Selects the chain of functions that led to call of the selected functions and every descendent of the selected functions.

**Select in...** There are four **Select in...** contextual menu options. Each opens a report view with the current function selected in the new view.

**Edit Source** Opens the file that contains the selected function in your default code editor.

### 9.2.1 See also

#### Reference

- [Call Graph view basics on page 9-2](#)
- [The toolbar and keyboard shortcuts on page 9-6.](#)

## 9.3 The toolbar and keyboard shortcuts

The toolbar buttons available in the Call Graph view are:

**Show Help** Opens the help view with a list of topics relevant to the Call Graph view.

### Toggle System Functions

Streamline classifies all functions that begin with either an underscore or `std::` as system functions, and hides them by default. You can, however, show them by choosing the **Show System Functions** drop-down menu option. All orphaned functions that are no longer connected to the tree when the system functions are hidden are shown as unconnected boxes at the bottom of the Call Graph view. If this option is active, the contextual menu option changes to **Hide System Functions**. This button is highlighted if system function visibility is currently off.

### Toggle Uncalled Functions

This option works in much the same way as the **Show/Hide System Functions** option. Select **Show Uncalled Functions** and all of the functions contained in your code that were not called during the captured execution appear as disconnected boxes in the bottom of the hierarchy.

### Toggle Mini-map

Toggle mini-map visibility on and off.

### Edit the source

Edit the source file that contains the selected function.

You can also use the following keyboard shortcuts to navigate the Call Graph view:

**Up arrow** Moves the current selection up one function box.

### Down arrow

Moves the current selection down one function box. The down arrow does not move the selection to the uncalled and disconnected functions. These must be selected using the mouse.

### Right arrow

Moves the current selection to the right. If no function is to the immediate right of the current function box, ARM Streamline chooses the closest available function in the row to the right of the currently selected function.

**Left arrow** Moves the current selection to a function box to the left of the currently selected function. Works in the same manner as the right arrow command.

**Home** Selects the top function box in the current Call Graph view row.

**End** Selects the bottom function box in the current Call Graph view row.

### SPACEBAR

Holding down the spacebar turns the mouse cursor into a hand and enables you to click and drag the viewable area.

**TAB** Cycles the selection to the next highest self time value.

### Shift + TAB

Cycles the selection to the function one above the current selection in terms of its self time value. For example, if the function with the third highest self time value is selected, pressing shift + TAB selects the second highest function.

### 9.3.1 See also

#### Reference

- [Call Graph view basics](#) on page 9-2
- [Contextual menu options](#) on page 9-5.

# Chapter 10

## Annotate and the Log View

Use the Annotate feature to customize the data in the Streamline Analysis Reports. In addition to the color-coded overlays it adds to the Timeline view, the Annotate feature records each message generated by the inserted Annotate code and lists it in the Log view.

The following topics describe the Log view and how to use the Annotate feature:

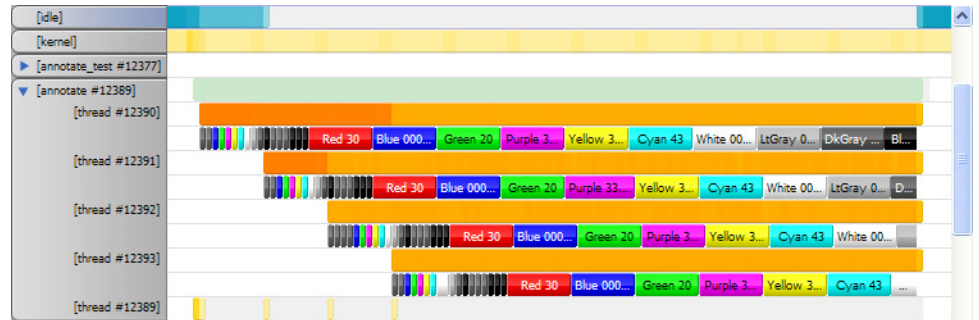
- [\*Customize reports using Annotate on page 10-2\*](#)
- [\*Adding images to reports using Visual Annotate on page 10-6\*](#)
- [\*The Log view on page 10-9\*](#)



## 10.1 Customize reports using Annotate

While ARM Streamline provides a large variety of target information, sometimes you require extra context to decipher exactly what the target is doing at certain instances. Streamline Annotate provides a facility for you to add this context to Streamline.

The Streamline Annotate feature works in a similar way to `printf`, but instead of console output, annotate statements populate the Log view and place framing overlays right in the Streamline Timeline view:



**Figure 10-1 Annotate overlays**

When the user space application writes to the `/dev/gator/annotate` file, the gator driver marks the recorded annotate-driven output with a timestamp and integrates the recorded data into the Streamline sample and trace capture report.

The annotated text is marked with a thread identifier that keeps the data uncluttered and eliminates the necessity of user mutexes. Writing to the annotate file is handled by the standard C-library functions.

The application code accesses the virtual annotate file using the standard c-library functions: `fopen`, `fwrite`, and `fprintf`. To start using the annotate feature, do the following:

1. Ensure gatord is running. gatord creates the `/dev/gator/annotate` file.
2. Open `/dev/gator/annotate` with write permissions
3. Write null-terminated strings to the file from any thread
4. Optionally set the color of the annotation by sending the ASCII escape code followed by a 3-byte RGB value
5. Disable buffering on the annotate file, or manually flush the file after each write
6. Write an empty string to clear the annotation message for the thread

Unless you are running out of file handles, closing the annotate file on completion is optional.

### ————— Note —————

You can locate all of the files provided by DS-5 by selecting **Help** → **ARM Extras...** from the main menu.

For an example of how to use annotate, see the `Streamline_annotate` example in the `Bare-metal_examples.zip` archive, located in `.../examples/`.

---

**Note**

---

To annotate from within the kernel or a module, use the `annotate_kernel.h` file instead of `streamline_annotate.h`. The annotate statements in `annotate_kernel.h` duplicate the functionality of the standard annotate statements defined in `streamline_annotate.h`, but they are titled differently. Kernel annotate statements all have the `KERNEL_` prefix. For example, to add a color-coded annotate statement to your kernel module code, include `annotate_kernel.h` and add the `KERNEL_ANNOTATE_COLOR(color, string)` statement to your code.

---

**10.1.1 See also****Tasks**

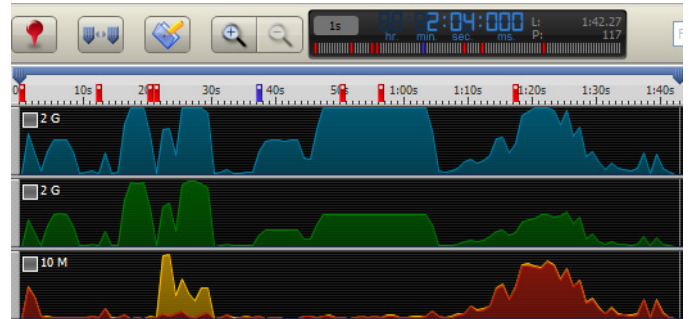
- [Adding bookmarks using Annotate on page 10-4](#)
- [Capturing data on your target on page 11-2](#)
- [Creating custom performance counters on page 11-4.](#)

**Reference**

- [About the Timeline view on page 6-2](#)
- [Filtering data and other Timeline view controls on page 6-10.](#)

## 10.2 Adding bookmarks using Annotate

You can use `annotate` to add bookmarks to the Timeline automatically. The process is similar to adding standard `annotate` overlays, but requires the use of a different set of `annotate` functions. For general instructions on how to set up annotation, see [Customize reports using Annotate on page 10-2](#).



**Figure 10-2 Automatically adding bookmarks during capture**

The file `streamline_annotate.h`, located in `.../gator/annotate/`, contains all of the functions necessary to automatically add bookmarks. You must include this header file in your source code to use them.

### ———— Note ————

You can locate all of the files provided by DS-5 by selecting **Help** → **ARM Extras...** from the main menu.

Use the following functions to add bookmarks:

`ANNOTATE_MARKER()`

Use this function to add a red bookmark to the Timeline view without a title. It has no parameters.

`ANNOTATE_MARKER_STR(...)`

This function adds a marker to the Timeline view with a title. Pass a string as a parameter to `ANNOTATE_MARKER_STR(...)` and the Timeline view displays it when you hover over the bookmark.

`ANNOTATE_MARKER_COLOR(setColor)`

This function adds a bookmark and assigns it a color. Pass a color through as a parameter to automatically assign it to the bookmark.

`ANNOTATE_MARKER_COLOR_STR(setColor, ...)`

This function creates a bookmark with a title, and a color. Set the color and the title string using the function parameters and the created bookmark appears in the Timeline view with the defined properties.

### ———— Note ————

To annotate from within the kernel or a module, use the `annotate_kernel.h` file instead of `streamline_annotate.h`. The `annotate` statements in `annotate_kernel.h` duplicate the functionality of the standard `annotate` statements defined in `streamline_annotate.h`, but they are

titled differently. Kernel annotate statements all have the `KERNEL_` prefix. For example, to add a titled bookmark using Annotate, include `annotate_kernel.h` and add the `KERNEL_ANNOTATE_MARKER_STR(string)` statement to your kernel or module code.

---

### 10.2.1 See also

#### Tasks

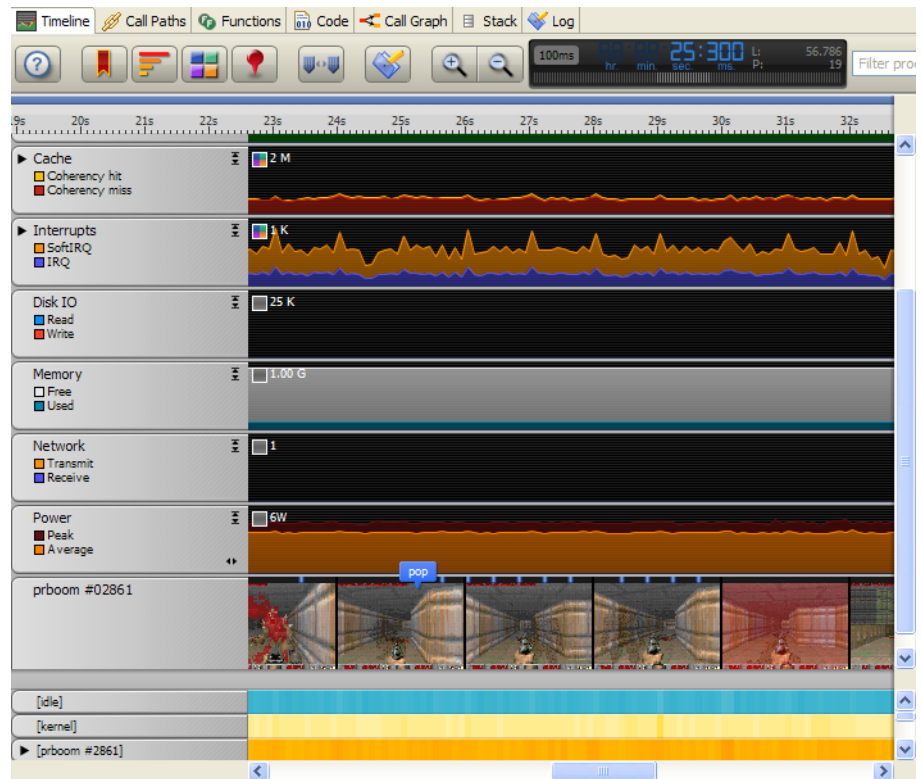
- [Customize reports using Annotate on page 10-2](#)
- [Capturing data on your target on page 11-2](#)
- [Creating custom performance counters on page 11-4.](#)

#### Reference

- [About the Timeline view on page 6-2](#)
- [Filtering data and other Timeline view controls on page 6-10.](#)

## 10.3 Adding images to reports using Visual Annotate

In addition to simple text overlays using Annotate, Streamline supports the annotation of images, providing further application-level context to the Timeline view.



**Figure 10-3 Visual Annotation in the Timeline view**

Just like textual annotation, the application writes to `/dev/gator/annotate` virtual file using standard c-library functions. The gator driver outputs this data with a timestamp to Streamline and integrates it with the trace and sample report.

The mechanics of instrumenting your source code to provide visual annotation is similar to the process of adding standard annotation. For more information on text-only annotation, see [Customize reports using Annotate on page 10-2](#).

To include images in the data sent to the host during a capture session, use the `ANNOTATE_VISUAL` macro in your source code instead of the `ANNOTATE` and `ANNOTATE_COLOR` macros used in standard annotation. `ANNOTATE_VISUAL` provides a parameter for image data.

### ———— Note ————

You can locate all of the files provided by DS-5 by selecting **Help** → **ARM Extras...** from the main menu.

To use visual annotation, you must:

1. Include the `streamline_annotate.h` header file located in `.../gator/annotate/` in your source code using the following line of code:

```
#include "streamline_annotate.h"
```

---

**Note**

---

You can customize the example `streamline_annotate.h` file provided by DS-5 or create your own. Use it as a template if you want to create your own customized annotate functions.

---

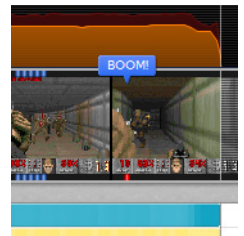
2. You must call the following before using the `ANNOTATE_VISUAL` macro:

```
ANNOTATE_DEFINE;
```

3. Insert the `ANNOTATE_VISUAL` macro into your code:

```
ANNOTATE_VISUAL(data, length, str);
```

Replace *data* with your image, *length* with the size of the data being written to the *annotate* file, and *str* with a descriptive string to be included with the image. Including a string is optional.



**Figure 10-4 Visual annotation with optional text**

---

**Note**

---

Visual Annotation supports images in the following formats: GIF, PNG, JPEG, TIFF, ICO, and BMP +RLE. There is no limit to the image size but the larger the image, the greater impact on the system performance. Increasing the amount of data sent to the host in this way increases the probe effect for the applications you are profiling.

---

For an example of how to use Visual Annotate, see the `annotate.c` example source file, located in `.../gator/annotate/example/`.

---

**Note**

---

To annotate from within the kernel or a module, use the `annotate_kernel.h` file instead of `streamline_annotate.h`. The annotate statements in `annotate_kernel.h` duplicate the functionality of the standard annotate statements defined in `streamline_annotate.h`, but they are titled differently. Kernel annotate statements all have the `KERNEL_` prefix. For example, to annotate your kernel module code to add images to report data, include `annotate_kernel.h` and add the `KERNEL_ANNOTATE_VISUAL(data, length, string)` statement to your code.

---

You can see the effects of visual annotation in the Timeline and Log views of your Streamline Analysis Reports. With visual annotation, the Timeline view data includes a chart that contains the annotated images. For more information about how visual annotation changes the Timeline view and how to interact with it, see [Timeline view charts on page 6-8](#).

Any annotation event that includes an image has an icon in the message field of the Log view. Hover over the icon to see the image.

### 10.3.1 See also

#### Tasks

- [Customize reports using Annotate on page 10-2](#)
- [Capturing data on your target on page 11-2](#)
- [Creating custom performance counters on page 11-4.](#)

#### Reference

- [About the Timeline view on page 6-2](#)

## 10.4 The Log view

The Log view lists every message generated by the ANNOTATE, ANNOTATE\_COLOR and ANNOTATE\_VISUAL statements in your code along with information related to the message.

When	Delta	Message	Core	Where
00:37.428088	+00.000000	Red 0	0	[annotate #18104]   [thread #18104]
00:37.429088	+00.001000	Blue 00001	0	[annotate #18104]   [thread #18104]
00:37.430087	+00.000999	Green 2	0	[annotate #18104]   [thread #18104]
00:37.431087	+00.001000	Purple 3.00	0	[annotate #18104]   [thread #18104]
00:37.432087	+00.001000	Yellow 4.000000e+00	0	[annotate #18104]   [thread #18104]
00:37.433088	+00.001001	Cyan 5	0	[annotate #18104]   [thread #18104]
00:37.434087	+00.000999	White 00006	0	[annotate #18104]   [thread #18104]
00:37.435087	+00.001000	LtGray 00007	0	[annotate #18104]   [thread #18104]
00:37.436088	+00.001001	DkGray 00008	0	[annotate #18104]   [thread #18104]
00:37.437091	+00.001003	Black 00009	0	[annotate #18104]   [thread #18104]
00:37.438091	+00.001000	Repeat	0	[annotate #18104]   [thread #18104]
00:37.439093	+00.001002	Repeat	0	[annotate #18104]   [thread #18104]
00:37.440091	+00.000998	Repeat	0	[annotate #18104]   [thread #18104]
00:37.441091	+00.001000	Repeat	0	[annotate #18104]   [thread #18104]
00:37.442090	+00.000999	Repeat	0	[annotate #18104]   [thread #18104]
00:37.443089	+00.000999	Repeat	0	[annotate #18104]   [thread #18104]
00:37.444090	+00.001001	Red 10	0	[annotate #18104]   [thread #18104]
00:37.445090	+00.010000	Blue 00011	0	[annotate #18104]   [thread #18104]
00:37.446090	+00.010000	Green c	0	[annotate #18104]   [thread #18104]
00:37.447088	+00.009998	Purple 13.00	0	[annotate #18104]   [thread #18104]
00:37.448093	+00.010005	Yellow 1.400000e+01	0	[annotate #18104]   [thread #18104]
00:37.449090	+00.009997	Cyan 17	0	[annotate #18104]   [thread #18104]
00:37.504090	+00.010000	White 00016	0	[annotate #18104]   [thread #18104]
00:37.514088	+00.009998	LtGray 00017	0	[annotate #18104]   [thread #18104]
00:37.524087	+00.009999	DkGray 00018	0	[annotate #18104]   [thread #18104]
00:37.534087	+00.010000	Black 00019	0	[annotate #18104]   [thread #18104]
00:37.544088	+00.010001	Repeat	0	[annotate #18104]   [thread #18104]
00:37.554087	+00.009999	Repeat	0	[annotate #18104]   [thread #18104]
00:37.564087	+00.010000	Repeat	0	[annotate #18104]   [thread #18104]
00:37.574087	+00.010000	Repeat	0	[annotate #18104]   [thread #18104]

Figure 10-5 The Log view

### Note

To populate the Log view, insert ANNOTATE statements in your code. Absent ANNOTATE statements in the source, the view is empty.

For instructions on how to use Annotate, see [Customize reports using Annotate](#) on page 10-2.

Right-click on any message in the list and use the **Select in Call Paths** contextual menu to open the Call Paths view with the calling Process/Thread/Function highlighted.

### 10.4.1 Log view search fields

The Log view provides three search fields above the table data that enable you to find particular messages based on the field type you use and the regular expression that you enter. The regular expression in the search field acts as a filter. Only messages that contain the matching pattern appear in the list, sorted in the chronological order.

**Message** Search the message field for a string. Only messages that match the given pattern appear in the log view until the search is modified or cleared. Regular expression strings are case sensitive unless you include (?i) in front of your search expression.

**Core** Find messages called by a particular core. Enter a core number here and the Log view displays only messages triggered by that core.



**Where** Search based on the location that triggered the annotate message. To narrow down messages to those called by a particular function, enter that function name in the **Where** field.

### 10.4.2 Log view table headers

Here is a list of all of the column headers available in the Log view:

**When** This value, given in seconds, tells you when message was generated during the captured execution. All messages appear in the Log view in chronological order.

**Delta** The difference in time between when this message and the one previous to it were generated. Filtering affects the values in this column. If your searches narrow this list down, the delta values reflect only the filtered messages.

**Message** The contents of the message. For example, if your inserted ANNOTATE statement was ANNOTATE("Total Value= (%d)", value) and the value of (%d) was 12, the Message field would contain the string "Total Value = 12". If an annotation message contains an image, a camera icon appears in the Message column. Select a row with a camera icon to see the image.

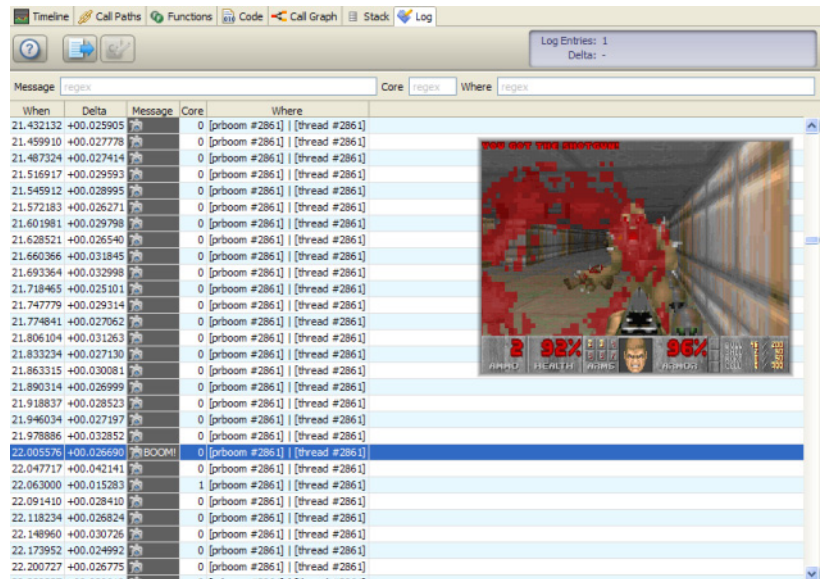


Figure 10-6 Visual annotation in the Log view

**Core** The core that generated the message.

**Where** The process and thread that generated the message.

**Location** The file location and line of code that generated the message.

### 10.4.3 Log view totals panel

The totals panel provides additional information when you select multiple messages in the Log view.

**Log Entries** The total number of messages you have selected in the Log view.

**Delta** The total difference in time between the function called first and the function called last amongst the selected functions. Use the totals panel to easily see the time gap between two messages by clicking on one and holding **Ctrl** and clicking on a second. The value in the Totals Panel **Delta** field updates to show you the time difference between the two selected messages.

#### 10.4.4 Log view contextual menu options

Right-click anywhere in the table to open a contextual menu. The menu contains the following options:

##### 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.

#### 10.4.5 See also

##### Tasks

- [Customize reports using Annotate on page 10-2](#)
- [Creating custom performance counters on page 11-4.](#)

##### Reference

- [About the Timeline view on page 6-2](#)
- [Filtering data and other Timeline view controls on page 6-10.](#)

# Chapter 11

## Advanced Customizations

ARM Streamline enables you to perform tasks such as creating a report and examining the data available in the report. It enables you to perform these tasks without going to the command line or modifying XML files. However, if you want to customize the data that ARM Streamline collects and change how it is presented to you, or you want to capture data and store it on your target, advanced customizations are required.

The following topics describe how capture data locally, use the Annotate feature, and customize performance counters:

- [\*Capturing data on your target on page 11-2\*](#)
- [\*Creating custom performance counters on page 11-4\*](#)
- [\*Using Stored Streamline Capture Data to create new Streamline Analysis Reports on page 11-7.\*](#)

## 11.1 Capturing data on your target

Typically, ARM Streamline uses an active network connection to send captured data from the target to the host. If this is not possible because of limitations with your target, you can save the data to local target storage for manual transfer to your host.

To capture data locally:

1. Create a session.xml file. For an example session.xml file, see the end of this topic.
2. Make sure to replace the output\_path variable in session.xml with target\_path. This ensures that Streamline saves the capture data locally. The session.xml example at the end of this topic includes the target\_path declaration.
3. To pass the new session.xml file as a parameter to gatord on the command line, enter:  
`./gatord session.xml &`

---

### Note

---

The target\_path attribute accepts the @F and @N codes. @F adds the value of the title attribute to the file name and @N adds a sequential number. In the following example, the first Streamline Capture Data generated would be called local-test\_001.apc.

---

Normally, you can click the **Stop** button in the ARM Streamline Data view to stop a capture. Capturing data locally prohibits this method because you are not connecting to your target using the interface of Streamline. Use one of the following options to terminate the capture:

- Specify a duration using the duration attribute in session.xml.
- In session.xml, set the value of the buffer\_mode variable to something other than streaming. Use one of the following values: Large, Normal, or Small. A Large store-and-forward buffer is 16MB, while Normal is 4MB and Small is 1MB. The profiling session terminates automatically when it reaches the set buffer size.
- Press Ctrl+C on the console to interrupt the gator daemon. The daemon must be running in the foreground.
- Determine the process id of gatord and enter the kill command: `kill process_ID`.

When the capture stops, Streamline creates a .apc directory on the target with the data and .xml files. To view your Capture Data in Eclipse for DS-5, do the following:

1. Transfer the directory to your host.
2. Open Eclipse for DS-5.
3. If the ARM Streamline Data view is not already open, select **Window** → **Show View** → **Other** and choose ARM Streamline Data from the list of available views.
4. Click the **Edit Locations...** button in the upper right of the ARM Streamline Data View.
5. Choose the directory that contains the .apc directory that you transferred from your target.
6. Double-click on the .apc directory in the ARM Streamline Data view to open the Analyze dialog box.
7. Define your image files using the Analyze dialog box. Image files must match the images that you ran on your target during the local capture session.

---

**Note**

---

For more information on using a stored capture session to generate a new Streamline Analysis Report, see [Using Stored Streamline Capture Data to create new Streamline Analysis Reports](#) on page 11-7.

---

8. Click **Analyze**.
9. Double-click on the Streamline Analysis Report in the ARM Streamline Data View.

---

**Note**

---

You can also use the .apc capture directory to create a new analysis report. Double-click on the .apc directory in the ARM Streamline Data view to open the Analyze dialog box.

---

**11.1.1 Example session.xml**

```
<?xml version="1.0" encoding="US-ASCII" ?>
<session version="1" title="local-test" call_stack_unwinding="yes"
buffer_mode="streaming" sample_rate="normal" target_path="@F_@N"
duration="6"/>
```

**11.1.2 See also****Tasks**

- [Customize reports using Annotate](#) on page 10-2
- [Creating custom performance counters](#) on page 11-4
- [Using Stored Streamline Capture Data to create new Streamline Analysis Reports](#) on page 11-7.

**Reference**

- [Capture options](#) on page 4-3.

## 11.2 Creating custom performance counters

In addition to the hardware-specific and Linux performance counters that you can configure using the Counter Configuration dialog box, the gator daemon and driver provide hooks that enable you to customize the counters collected during a capture.

Streamline derives its default set of counters from the performance monitoring unit, linux hooks, and memory-mapped peripherals. You can add your own counters to this list if there is a hardware metric that you want to track which Streamline does not provide by default.

### 11.2.1 The `gator_events_mmaped.c` example file

In the `gator-driver.tar.gz` archive distributed with your copy of Streamline is a gator source file called `gator_events_mmaped.c`. The events contained in this file are not included in the `events.xml` file and do not appear in the list of available counters in the Counter Configuration dialog box. This file was provided as an example of how to add custom counters.

---

#### Note

---

You can locate all of the files provided by DS-5 by selecting **Help** → **ARM Extras...** from the main menu.

---

To familiarize yourself with the process of adding your own counters, incorporate the simulated examples from `gator_events_mmaped.c` into gator.

To do so, follow these steps:

1. Open the `gator_events_mmaped.c` sample file in the editor of your choice.
2. Copy the xml from the comments section of `gator_events_mmaped.c`.
3. Open `events.xml`. This file is only available after you have built the gator daemon. If you built your gator daemon for a Linux target, it is located in the `build_location/daemon-src` directory. If you built it for android, it is located in the `build_location/jni` directory.
4. Add the copied xml from the comments section of `gator_events_mmaped.c` to `events.xml`, just after the closing tag of the Linux category.
5. Remove any \* comment markers from the copied xml.
6. Save `events.xml`.
7. Copy `events.xml` to your target. You must place it in the same directory as `gatord`.
8. Enter `./gatord &` on the command line of your target to restart `gatord`.
9. Open the Counter Configuration dialog in Streamline. Note that a new category, **mmaped**, appears with the **Sine**, **Triangle**, and **PWM** simulated counters.
10. Add **Sine** to list of counters.
11. Run a capture session.

If successful, the waveform generated by the simulated Sine counter appears in the charts section of the Timeline view.

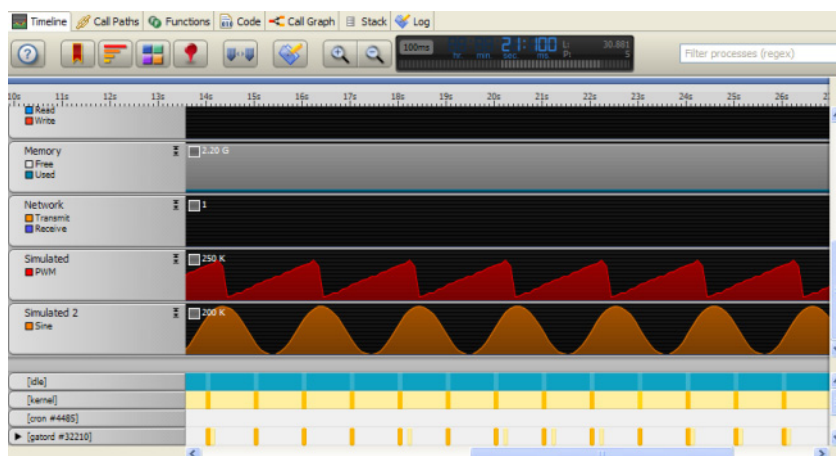


Figure 11-1 The custom Sine counter in the Timeline view

## 11.2.2 Creating your own counters

To create your own counters, mimic the methods used in the `gator_events_mmaped.c` or any of the other `gator_events_x` files included with the gator source.

Make sure to do the following to ensure that gator interacts with your custom source:

1. Create an empty `gator_events_your_custom.c` file or duplicate `gator_events_mmaped.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_mmaped.c` as a template.
4. Include the `gator_events_init` macro.
5. 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`.
6. All of your new counters must be added to the `events.xml` file.

## 11.2.3 gator\_events functions

Here is a brief description of each of the gator event functions:

`gator_events_your_custom_init`

gator calls this function startup.

`gator_events_your_custom_interface`

Tells gator what triggers calls to your custom events file.

`gator_events_your_custom_create_files`

Adds custom directories and enabled, event, and key files to `/dev/gator/events`

`gator_events_your_custom_start`

gator calls this at the start or execution.

`gator_events_your_custom_read`

gator calls this function at every sample.

`gator_events_your_custom_stop`

gator calls this at the termination of a capture session.

### 11.2.4 Adding your events to the `events.xml` file

Use the following attributes in the counter elements in your `events.xml` file:

<b>type</b>	The identifier of the counter resource.
<b>event</b>	Architecture specific or implementation-specific event number, in decimal, determined from either the architecture specification document or the Technical Reference Manual of the processor.
<b>title</b>	The title of the chart that Streamline displays. Counters with matching titles stack in the charts.
<b>name</b>	The name that Streamline displays.
<b>per_cpu</b>	Defines whether or not Streamline collects data on a per cpu basis. Set this value to yes and Streamline collects data from each cpu separately for this counter.

### 11.2.5 L2C-310 memory-mapped peripherals

To get L2C-310 memory-mapped peripherals working with Streamline, do the following:

1. Open the `gator_events_l2c-310.c` file located in the archive in `.../gator/driver-src`
2. Find the `init` function in `gator_events_L2C-310.c`
3. Add the following `#if` statement to the existing `#if` statements in the `init` function:

```
#if defined(Your_Architecture)
    gator_events_l2c310_probe(L2C_310_Address);
#endif
```

#### ———— Note ————

Replace *Your\_Architecture* with the name of your architecture and *L2C\_310\_Address* with the physical address of your L2C-310 in hexadecimal.

4. Rebuild the gator driver.
5. Connect to your target.

### 11.2.6 See also

#### Tasks

- [Capturing data on your target on page 11-2](#)
- [Customize reports using Annotate on page 10-2.](#)

#### Reference

- [About the Timeline view on page 6-2.](#)



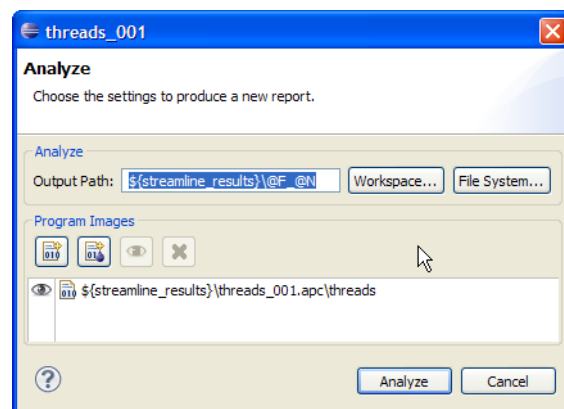
## 11.3 Using Stored Streamline Capture Data to create new Streamline Analysis Reports

You might want to use stored capture data to create new Streamline Analysis reports for the following reasons:

- You captured data locally and transferred the Streamline Capture Data to the host.
- Sources were not available when the data was captured.
- Current Streamline Analysis files have been rendered incompatible by a new version of Streamline.

To create a new Streamline Analysis File:

1. Open the ARM Streamline Data view.
2. Double-click on a stored Streamline Capture Data resource in the list.
3. In the resulting dialog box, make the required changes to the settings.



**Figure 11-2 The Analyze dialog box**

The options here are a subset of the options available in the Capture Options dialog box and they work the same way. Use **Output Path** to change how the new file is named and the Program Images section to add and remove images.

4. Click **Analyze**.

### 11.3.1 See also

#### Tasks

- [Capturing data on your target on page 11-2](#)
- [Customize reports using Annotate on page 10-2](#)
- [Creating custom performance counters on page 11-4.](#)

#### Reference

- [Capture options on page 4-3](#)
- [ARM Streamline Data view basics on page 3-2.](#)

## 11.4 Profiling the Linux kernel

If you do not include the kernel in the images in the capture options dialog box, the statistics generated by the kernel are not aligned with source code in the Analysis Reports. To build a version of the kernel that you use in profiling, follow these steps:

1. Enter the following command to change a menu setting:  

```
make ARCH=arm CROSS_COMPILE=${CROSS_TOOLS}/bin/arm-none-linux-gnueabi- menuconfig
```
2. Under Kernel Hacking, enable the **Compile the kernel with debug info** option.
3. Insert the following command to build the image:  

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

In the capture options dialog box, define the newly created `vmLinux` image in the Images section.

You can profile drivers using this method. Statically link the driver into the kernel image or add the module as an image in the capture options dialog box.

### 11.4.1 Kernel stack unwinding

Unless you build `gator.ko` with kernel stack unwinding specifically turned on, the data related to the kernel in the Call paths view appears flat. It does not represent an accurate call hierarchy.

To perform kernel stack unwinding and module unwinding, edit the Makefile to enable `GATOR_KERNEL_STACK_UNWINDING` and rebuild `gator.ko`. To do this, open the makefile in the editor of your choice and remove the comment tags that surround the `GATOR_KERNEL_STACK_UNWINDING` command.

---

#### Note

---

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.

---

### 11.4.2 See also

#### Tasks

- [Capturing data on your target on page 11-2](#)
- [Customize reports using Annotate on page 10-2](#)
- [Creating custom performance counters on page 11-4.](#)

#### Reference

- [Capture options on page 4-3](#)
- [ARM Streamline Data view basics on page 3-2.](#)

# Chapter 12

## Using the Energy Probe

The Energy Probe is a compact probe designed for application and system software developers to measure any combination of power, voltage, and current in up to three channels. The combination of ARM Streamline and Energy Probe enables the visualization of power metrics against the software behavior of your target hardware. It provides a better understanding of the static and dynamic behavior of your target system for the purposes of debugging, profiling, and analysis.

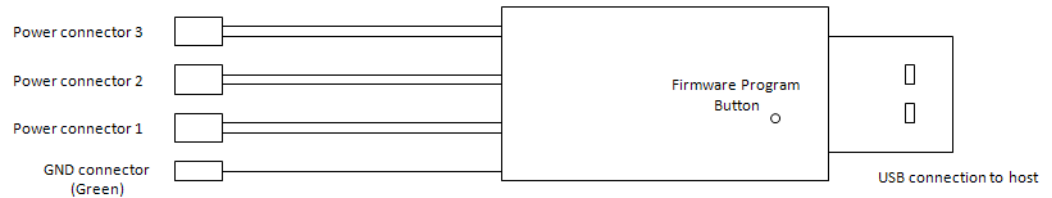
The following topics describe how to use the Energy Probe:

- [Energy Probe overview on page 12-2](#)
- [Energy Probe requirements on page 12-3](#)
- [Energy Probe setup on page 12-5](#)
- [Energy Probe operation on page 12-8.](#)

## 12.1 Energy Probe overview

The Energy Probe is designed to be a low impact, inexpensive solution to give you quick feedback on the impact of your code on the system energy footprint. It is not intended to be a high-precision data acquisition instrument for power benchmarking.

It 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 3 independent power, current, or voltage measurements.

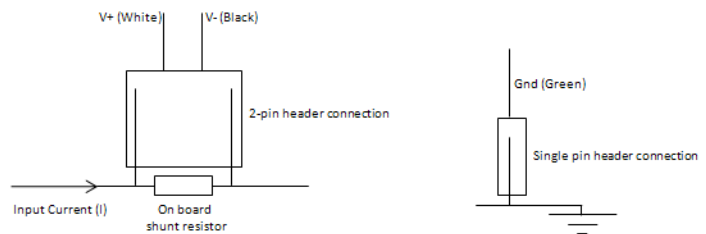


**Figure 12-1 Energy Probe schematic**

In addition to the three power connectors, the Energy Probe has a single pin GND connector that must be connected to ground on your target board. It provides a ground connection for the Energy Probe.

Each of the three power connectors measures:

- The current flow through a shunt resistor of a known value on your target system
- The voltage at the positive terminals of the Energy Probe.



**Figure 12-2 Energy Probe electrical connection example**

## 12.2 Energy Probe requirements

The Energy Probe has the following requirements:

- An installation of ARM DS-5 Basic or Professional Edition, version 5.9 or later.
- A suitable DS-5 license.
- A Streamline-enabled target. For Linux and Android targets, this requires the gator driver and daemon to be installed on the target.
- 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 32mV.

---

### 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, consult the documentation for your target.

---

### 12.2.1 Precautions

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.

### 12.2.2 Determining the correct shunt resistor value

With 100x amplification, Energy Probe requires 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 too low reduces measurement dynamic range, resulting in less resolution in the power data. A shunt resistor with a value too high can cause the input stage of the Energy Probe to saturate, which causes a flat line in the charts related to Energy Probe in the Streamline Timeline view.

To avoid input saturation, the drop across to the Rsense resistor must never be more than 32mV. You can also use the following equation to determine if your shunt resistor is appropriate:

$$R_{\text{shunt}}(\text{max}) = 3.2 \times V_{\text{supply}} / (100 \times \text{Power}(\text{max}))$$

$V_{\text{supply}}$  is the input/core voltage.  $\text{Power}(\text{max})$  is the maximum power that the Energy Probe measures.  $R_{\text{shunt}}(\text{max})$  is the maximum value of the shunt resistor. A shunt resistor value greater than  $R_{\text{shunt}}(\text{max})$  might cause input saturation.

This equation provides the absolute maximum value for  $R_{\text{shunt}}$ . Use something that is more than five percent lower than this value to allow for component tolerances.

#### Examples

- 5V power supply, 1.5W(max),  $R_{\text{shunt}} = 100\text{m}\Omega$
- 14V power supply, 5W(max),  $R_{\text{shunt}} = 50\text{m}\Omega$

- 5V power supply, 8W(max), Rshunt = 20mOhm
- 1V core voltage, 2.5W(max), Rshunt = 10mOhm
- 1.5V core voltage, 0.4W(max), Rshunt = 100mOhm.

### 12.2.3 Notes

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-.

## 12.3 Energy Probe setup

When first plugging in the Energy Probe on Windows, the LED flashes RED on and off at roughly one second intervals, indicating USB enumeration failed and that you must install the driver. To install, use the standard Windows driver installation wizard, pointing to the `lpc134x-vcom.inf` configuration file. This file is located in `.../sw/energy probe/`. There is no driver DLL, as Energy Probe uses the built-in USB CDC virtual serial port functionality on both Windows and Linux.

To setup on Linux, enter the following command:

```
sudo apt-get install libudev-dev
```

Once you have installed the driver and it is operating correctly, the LED turns solid GREEN. Attaching a probe backwards to Energy Probe or the target causes the LED to turn RED. Backwards connections do not damage your target or Energy Probe but this kind of connection does not provide useful data. Disconnect all Energy Probe probes and wait until the LED turns GREEN, then reattach one by one making sure to reverse the polarity on any probe that results in the LED turning RED. When you have attached the Energy Probe correctly, the LED remains GREEN.

The Energy Probe requires a common ground between the probe and the target. Use special care when first attaching the ground pin and the probes. The following symptoms are an indication that you have attached to the wrong testpoint:

- the LEDs on your target suddenly turn off
- the Energy Probe LED turns off
- you hear a buzzing sound from the board or the power supply.

Quickly remove the probe from the target and check you have not damaged your target.

### 12.3.1 caiman Files

Add the following files to a common folder on either Linux or Windows.

- `resistors.xml`  
Streamline uses this file to tell the energy probe the value of the shunt resistor that connects to each of the three power connectors. The units of measurements in this file are mOhms. Edit this file to change these values.
- `configuration.xml`  
Streamline uses this file to configure what each channel measures and displays in the Analysis Reports. Each channel can measure any combination of power, current and voltage and also a combination of average or peak value for each of these measurements.
- `events.xml`  
This file contains all the valid measurement options that can be used in the `configuration.xml`.

The location of these files is dependent on your operating system:

#### Windows XP

`Documents and Settings\user_name\My Documents\ARM\Energy Probe\`

**Windows 7** `Documents and Settings\user_name\My Documents\ARM\Energy Probe\`

**Linux** `$(HOME)/ARM/Energy Probe`

On Linux, add the following file to the same folder:

- caiman binary

---

**Note**

Make sure to enter the `sudo apt-get install libudev-dev` command to install the Energy Probe driver.

---

On Windows, add these files to a the same folder:

- caiman.exe binary
- caiman.exe.manifest manifest file
- lpc134x-vcom.inf driver configuration. This file is located in `.../sw/energy probe/`.

### 12.3.2 A configuration.xml file example

The Energy Probe caiman application is configured to measure Channel 0 power out of the box. You can configure it to support voltage, current, and power for all three channels.

Modify the configuration.xml file that caiman loads to change the configuration. The events.xml contains all possible entries for the configuration.xml file. To add a counter to configuration.xml, copy it over from events.xml and add the event tag and the color attribute to the declaration.

In this example, Streamline would measure the average and peak of power, current and voltage during the capture session and then display them in the Timeline view.

**configuration.xml** example:

```
<?xml version="1.0" encoding='UTF-8'?>

<configurations>
  <configuration counter="emeter_ch0_power1" title="Channel0 Power"
name="Average" color="0x80ff7e00" description="Channel power" operation="average"/>
  <event counter="emeter_ch0_power1" title="Channel0 Power" name="Peak"
description="Channel peak power" operation="maximum"/>
  <event counter="emeter_ch0_current0" title="Channel0 Current" name="Average"
description="Channel current" operation="average"/>
  <event counter="emeter_ch0_current1" title="Channel0 Current" name="Peak"
description="Channel peak current" operation="maximum"/>
  <event counter="emeter_ch0_voltage0" title="Channel0 Voltage" name="Average"
description="Channel voltage" operation="average"/>
  <event counter="emeter_ch0_voltage1" title="Channel0 Voltage" name="Peak"
description="Channel peak voltage" operation="maximum"/>
</configurations>
```

### 12.3.3 Attaching the Energy Probe to a target

If your target has headers available for power or energy metering, first attach the ground wire, then attach the probes. Make sure the probe polarity is correct by observing if the LED changes from GREEN to RED.

---

**Note**

The probe white wire is V+. The black wire is V-.

---



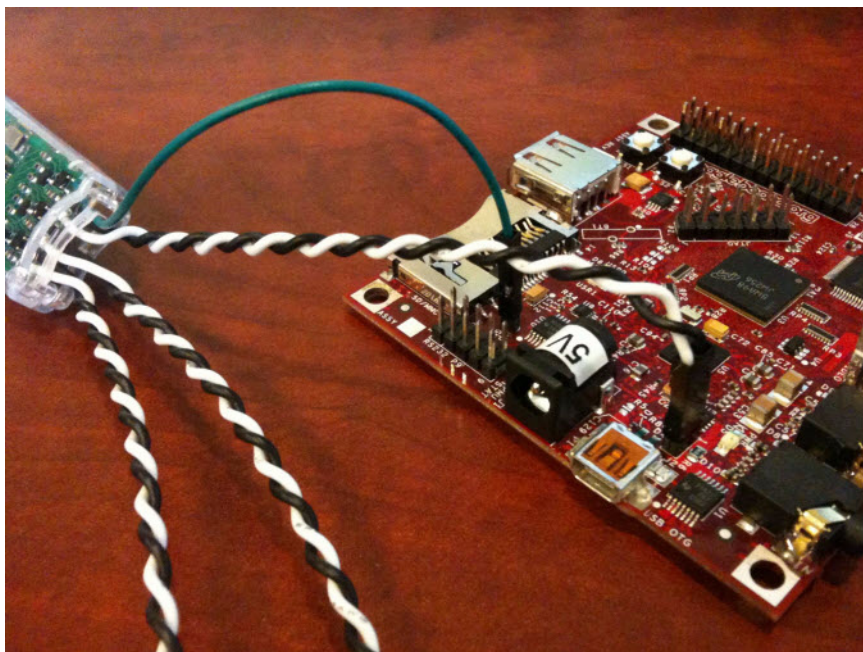


Figure 12-3 Connection to target

### 12.3.4 Updating your firmware

If necessary updates become available for caiman or the firmware, follow these instructions to update your software:

1. Detach the Energy Probe probes and ground from the target.
2. Copy the file `eprobe_firmware.bin` to a local folder on your Windows machine.

———— **Note** ————

Due to a limitation of the MCU vendor's USB bootloader, updating firmware on Linux is not supported.

————

3. Plug the Energy Probe into a USB port.
4. To enable firmware programming mode, insert a paperclip into the little hole on the top near the LED.
5. Press and hold down the button for 3 seconds.
6. Wait for LED to turn red, and then OFF. Firmware programming mode is now active.
7. Once the drive folder has opened, delete the `firmware.bin` file. This file is a placeholder.
8. Drag and drop the new `eprobe_firmware.bin` file onto the drive folder.
9. Safely remove the Energy Probe drive from the machine using the Windows driver helper.
10. Unplug the Energy Probe from the machine.
11. After waiting a second, plug it back in.

## 12.4 Energy Probe operation

In the Timeline view of Streamline, you can see the Energy Probe data aligned to the software activity. If your target meets the requirements, and you have attached a power probe point that closely resembles the activity of the CPU cores using Channel 0, the Streamline correlation algorithm aligns the data for you. Notes in the warnings tab communicate any limitations or failures in this area.

Once you have set up everything correctly, run a report capture and analysis as you normally would. 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 now appears in the Timeline view.

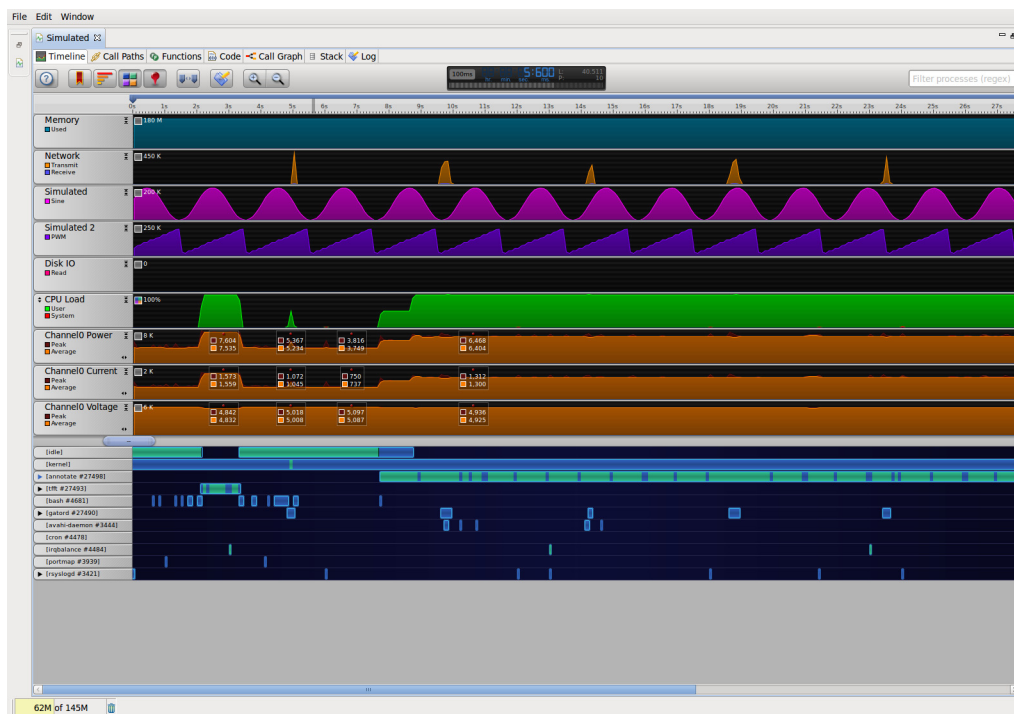


Figure 12-4 Energy Probe in the Timeline view

### 12.4.1 Adding the caiman application to Streamline

To configure Energy Probe, add the path to the caiman application in the Streamline options panel of the run configuration dialog box, under **Energy Capture**. Enter the path name or use the **Browse** button to locate the file.

# Chapter 13

## Using Streamline on the Command Line

---

**Note**

---

The feature is only available in the full DS-5 installation.

---

The following topic describes how to use Streamline on the command line:

- [Opening a Streamline-enabled command prompt or shell on page 13-2](#)
- [The streamline command on page 13-3](#)

## 13.1 Opening a Streamline-enabled command prompt or shell

To use Streamline on the command line, open a DS-5 command prompt. On Windows, select **Start All Programs** → **ARM DS-5** → **DS-5 Command Prompt**. On Linux, add the `.../bin` location to your PATH environment variable then open a UNIX bash shell.

## 13.2 The streamline command

The streamline command has three different modes that enable you to use most features of Streamline outside of the DS-5 for Eclipse user interface.

### 13.2.1 Syntax

```
streamline <mode> [options] <file...>
```

### 13.2.2 streamline command modes

Use any of the following modes directly after streamline on the command line:

**-capture** This mode initiates a capture session. You must enter a valid session.xml file with the capture mode declaration. 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 desired settings and then use the **Export...** option in the Capture Options dialog box within DS-5 for Eclipse. For instructions on how to create a session.xml file manually, see [Capturing data on your target on page 11-2](#).

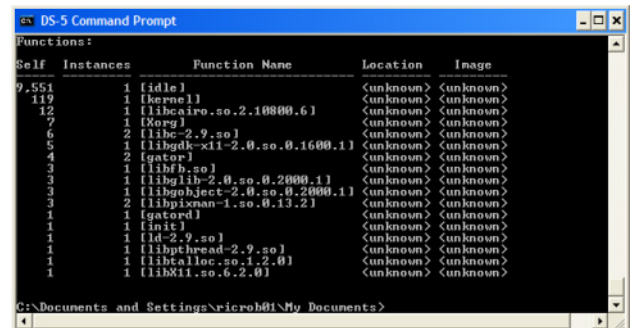
**-analyze** Use this mode to analyze existing Capture Data. You must enter a valid .apc capture file after -analyze. For example:

```
streamline -analyze threads_001.apc
```

For more information on the analyze feature, see [Using Stored Streamline Capture Data to create new Streamline Analysis Reports on page 11-7](#).

**-report** This mode reads data from an analysis report and outputs it your console. You can use options to define how that data appears in the console or define an output file to store the data to disk. You must enter a valid .apd file after the declaration of report mode.

```
streamline -report threads_001_001.apd
```



The screenshot shows a Windows Command Prompt window titled "DS-5 Command Prompt". It displays a table of functions. The table has four columns: "Self", "Instances", "Function Name", "Location", and "Image". The data is as follows:

Self	Instances	Function Name	Location	Image
0.551	1	[idle]	<unknown>	<unknown>
119	1	[kernel]	<unknown>	<unknown>
112	1	[libcairo.so.2.10000.6]	<unknown>	<unknown>
12	1	[Xorg]	<unknown>	<unknown>
6	2	[libc-2.9.so]	<unknown>	<unknown>
5	1	[libgtk-x11-2.0.so.0.1600.1]	<unknown>	<unknown>
4	2	[gator]	<unknown>	<unknown>
3	1	[libfb.so]	<unknown>	<unknown>
3	1	[libgl1h-2.0.so.0.2000.1]	<unknown>	<unknown>
3	1	[libpobject-2.0.so.0.2000.1]	<unknown>	<unknown>
3	2	[libpixman-1.so.0.13.2]	<unknown>	<unknown>
1	1	[gatorcl]	<unknown>	<unknown>
1	1	[init]	<unknown>	<unknown>
1	1	[ld-2.9.so]	<unknown>	<unknown>
1	1	[libpthread-2.9.so]	<unknown>	<unknown>
1	1	[libtalloc.so.1.2.0]	<unknown>	<unknown>
1	1	[libX11.so.6.2.0]	<unknown>	<unknown>

Figure 13-1 A functions report generated using report mode

### 13.2.3 streamline command options

Enter any option between your desired mode and file. The following options are available in any mode:

**-h, -?, -help**

Outputs help information to the console, listing each mode and option as well as the required syntax.

```

C:\Program Files\Microsoft Visual Studio\VC98\bin>tasklist /v /fi "idletime > 10" /s /u
Call Paths:

Self  Process Total Stack Process/Thread/Function Name Location
-----
0 9.553 98.26% 0 {idle} -
0 9.553 98.26% 0 {entry} -
9.551 9.551 98.24% 0 {idle} <unknown>
2 2 0.02% 0 {gator} <unknown>
0 121 1.24% 0 {kernel} -
0 121 1.24% 0 {entry} -
119 119 1.22% 0 {kernel} <unknown>
2 2 0.02% 0 {gator} <unknown>
0 27 0.28% 0 {gdngraster #16741} -
0 27 0.28% 0 {thread #16741} -
12 12 0.12% 0 {libcairo.so.2.10800.61} <unknown>
3 3 0.03% 0 {libgdi32-2.0.so.0.1600.1} <unknown>
3 3 0.03% 0 {libglib-2.0.so.0.2000.1} <unknown>
3 3 0.03% 0 {libgobject-2.0.so.0.2000.1} <unknown>
2 2 0.02% 0 {libpixmap-1.so.0.13.21} <unknown>
1 1 0.01% 0 {libpthread-2.9.so} <unknown>
1 1 0.01% 0 {libx11.so.6.2.0} <unknown>
0 16 0.16% 0 {Xorg #1581} -
16 16 0.16% 0 {thread #1581} -
7 7 0.07% 0 {Xorg} <unknown>
5 5 0.05% 0 {libc-2.9.so} <unknown>
3 3 0.03% 0 {libc.so} <unknown>
1 1 0.01% 0 {libpixmap-1.so.0.13.21} <unknown>
0 2 0.02% 0 {gatty #11221} -
0 2 0.02% 0 {thread #11221} -
1 1 0.01% 0 {libc-2.9.so} <unknown>
1 1 0.01% 0 {libc-2.9.so} <unknown>
0 1 0.01% 0 {gator #11218} -
0 1 0.01% 0 {thread #11220} <unknown>
1 1 0.01% 0 {gator #1} -
0 0 0.00% 0 {thread #11218} -
0 0 0.00% 0 {thread #11219} -
0 1 0.01% 0 {init #1} -
1 1 0.01% 0 {thread #1} <unknown>
1 1 0.01% 0 {init #1} -
0 1 0.01% 0 {nmdh #1421} -
0 1 0.01% 0 {thread #1421} -
1 1 0.01% 0 {libcaltloc.so.1.2.0} <unknown>
0 0 0.00% 0 {syslogd #13451} -
0 0 0.00% 0 {thread #13451} -

```

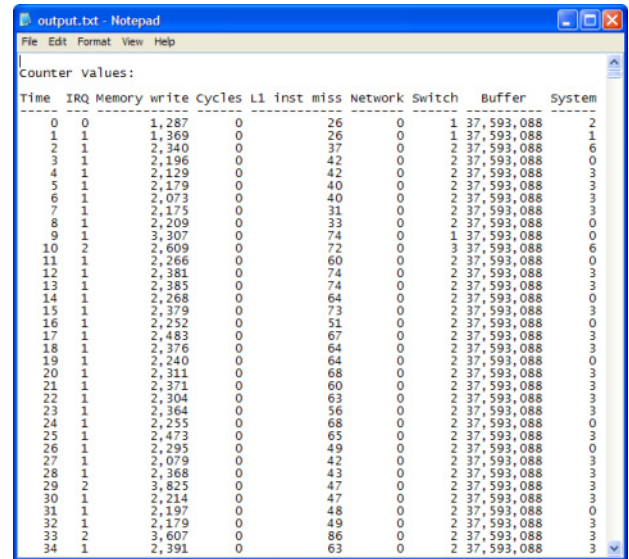
Copyright © 2010-2012 ARM. All rights reserved.  
Non-Confidential

### 13.2.4 Outputting data to a file

The reports generated by Streamline can be very large. You can add an output file to the command to send the large quantities of data to a .txt file. For example:

```
streamline -report -timeline capture_001_001.apd > output.txt
```

This creates a file, output.txt, if it does not already exist, and outputs the data to the new file instead of the command window.



Time	IRQ	Memory	write	Cycles	L1 inst miss	Network	Switch	Buffer	System
0	0	1,287	0	26	0	1	37,593,088	2	1
1	1	1,369	0	26	0	1	37,593,088	2	1
2	1	2,340	0	37	0	2	37,593,088	6	0
3	1	2,196	0	42	0	2	37,593,088	3	0
4	1	2,129	0	42	0	2	37,593,088	3	0
5	1	2,179	0	40	0	2	37,593,088	3	0
6	1	2,073	0	40	0	2	37,593,088	3	0
7	1	2,175	0	31	0	2	37,593,088	3	0
8	1	2,209	0	33	0	2	37,593,088	3	0
9	1	3,307	0	74	0	1	37,593,088	0	0
10	2	2,609	0	72	0	3	37,593,088	6	0
11	1	2,266	0	60	0	2	37,593,088	0	0
12	1	2,381	0	74	0	2	37,593,088	3	0
13	1	2,385	0	74	0	2	37,593,088	3	0
14	1	2,268	0	64	0	2	37,593,088	0	0
15	1	2,379	0	73	0	2	37,593,088	3	0
16	1	2,252	0	51	0	2	37,593,088	0	0
17	1	2,483	0	67	0	2	37,593,088	3	0
18	1	2,376	0	64	0	2	37,593,088	3	0
19	1	2,340	0	64	0	2	37,593,088	0	0
20	1	2,311	0	68	0	2	37,593,088	3	0
21	1	2,371	0	60	0	2	37,593,088	3	0
22	1	2,304	0	63	0	2	37,593,088	3	0
23	1	2,364	0	56	0	2	37,593,088	3	0
24	1	2,255	0	68	0	2	37,593,088	0	0
25	1	2,473	0	65	0	2	37,593,088	3	0
26	1	2,295	0	49	0	2	37,593,088	0	0
27	1	2,079	0	42	0	2	37,593,088	3	0
28	1	2,368	0	43	0	2	37,593,088	3	0
29	2	3,825	0	47	0	2	37,593,088	3	0
30	1	2,214	0	47	0	2	37,593,088	3	0
31	1	2,197	0	48	0	2	37,593,088	0	0
32	1	2,179	0	49	0	2	37,593,088	3	0
33	2	3,607	0	86	0	2	37,593,088	3	0
34	1	2,391	0	63	0	2	37,593,088	3	0

Figure 13-3 The Timeline view in a text file

### 13.2.5 See also

#### Tasks

- [Using Stored Streamline Capture Data to create new Streamline Analysis Reports on page 11-7.](#)

#### Reference

- [Call Paths view column headers on page 7-6.](#)
- [Functions view column headers on page 7-7](#)
- [Stack view column headers and the Maximum Stack Depth by Thread chart on page 7-8](#)
- [The Log view on page 10-9.](#)

# Chapter 14

## Troubleshooting

The following topics describe how to troubleshoot common Streamline issues:

- [\*Target connection issues on page 14-2\*](#)
- [\*Report issues on page 14-3\*](#)



## 14.1 Target connection issues

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

- Symptom** You receive the following error message: Unable to connect to the gator daemon at *your\_target\_address*. Please verify you have installed the gator daemon on your target and it is running. Installation instructions can be found in: `.../README_Streamline.txt`
- Solutions:* Make sure the gator daemon is running on your target. Enter the following command in the shell of your target:
- ```
ps -d | grep gatord
```
- If this command returns no results, gatord is not active. Start it by navigating to the directory that contains gatord and entering the following command:
- ```
sudo ./gatord &
```
- Try connecting to the target again.
- If gatord is active and you still receive this error message, try disabling any firewalls on your host machine that might be interfering with communication between it and the target.
- Symptom** You receive the following error message: Unknown host
- Solution:* Make sure that you have correctly entered the name or IP address of the target in **Address** field. If you have entered a name, try an IP address instead.
- Symptom** You receive the following error message: Unable to launch. Only one instance of Streamline may be running on this machine. Please close all instances of Eclipse and try again.
- Solution:* Stop any other running Streamline session and try connecting to the target again. If you cannot find another session, try closing Eclipse for DS-5, then re-starting it.
- Symptom** When using event-based sampling, Streamline fails to find the PMU.
- Solution:* The PMU on your hardware might not be correctly configured to allow the processor interrupts necessary for for Streamline to use event-based sampling. Test on alternate hardware or disable event-based sampling in the counter configuration dialog box.

### 14.1.1 See also

#### Tasks

- [Setting up an ARM Linux target on page 2-2.](#)

#### Reference

- [Report issues on page 14-3.](#)

## 14.2 Report issues

If the data in your reports seems incomplete, you might not have included a compilation option essential to Streamline. Common report problems and solutions are:

- |   |  |
|---|--|
| <b>Symptom</b>  | Streamline does not show any source code in the Code view.<br><i>Solution:</i> Make sure that you used the <code>-g</code> option during compilation. Streamline must have debug symbols turned on to match instructions to source code.   |
| <b>Symptom</b>  | Streamline does not show source code for shared libraries.<br><i>Solution:</i> Add the libraries using the Capture options dialog box. Click <b>Add Image...</b> in the images section, navigate to your shared library, and then add it.  |
| <b>Symptom</b>  | The data in the call paths view is flat. The presented table is a list, rather than a hierarchy.<br><i>Solution:</i> Use the <code>-fno-omit-frame-pointer</code> option during compilation and be sure to check the <b>Call Stack Unwinding</b> option in the capture options dialog box.                         |
| <hr style="width: 20%; margin: 0 auto;"/> <b>Note</b> <hr style="width: 20%; margin: 0 auto;"/> |  |
| <b>Symptom</b>  | Functions that you know are highly used are missing from the reports. Other functions might seem artificially large.<br><i>Solution:</i> This can be because of code inlining done by the compiler. To turn inlining off, add <code>-fno-inline</code> as an option during compilation.                            |
| <b>Symptom</b>  | A newly-generated analysis report has no data.<br>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. |

### 14.2.1 See also

#### Reference

- [Target connection issues on page 14-2.](#)