Arm® Streamline

Version 7.1

Target Setup Guide for Android

Arm® Streamline

Target Setup Guide for Android

Release Information

Document History

Issue	Date	Confidentiality	Change
0701-00	25 September 2019	Non-Confidential	New document for v7.1.
0701-01	30 October 2019	Non-Confidential	Updated document for v7.1.

Confidentiality Status

Product Status

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

Web Address

www.arm.com

Contents

Arm® Streamline Target Setup Guide for Android

Preface

This preface introduces the *Arm® Streamline Target Setup Guide for Android*.

It contains the following:

- *About this book* on page 7.

About this book

This book describes how to set up Arm® Streamline on an Android target.

Using this book

This book is organized into the following chapters:

Chapter 1 Target Setup
Set up your target and host devices ready to use Arm Streamline for application or system profiling by following the instructions in this chapter.

Chapter 2 Application profiling on an Android device
Profile your application while it is running on a non-rooted Android device.

Chapter 3 System profiling on an Android device
Profile all applications and services that are running on a rooted Android device.

Chapter 4 Troubleshooting Common Issues
Troubleshoot common Arm Streamline issues.

Appendix A Advanced target setup information
This appendix provides extra configuration information beyond the standard setup.

Glossary

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

See the *Arm® Glossary* for more information.

Typographic conventions

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

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

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

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

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

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

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

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

SMALL CAPITALS

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

Feedback

Feedback on this product

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

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

Feedback on content

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

- The title *Arm Streamline Target Setup Guide for Android*.
- The number 101813_0701_01_en.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note

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

Other information

- *Arm® Developer*.
- *Arm® Information Center*.
- *Arm® Technical Support Knowledge Articles*.
- *Technical Support*.
- *Arm® Glossary*.

Chapter 1
Target Setup

Set up your target and host devices ready to use Arm Streamline for application or system profiling by following the instructions in this chapter.

It contains the following sections:

1.1 Application and system profiling

Arm Streamline supports two types of profiling. Application profiling is the most common use case, but system profiling is also supported.

Application profiling

Arm Streamline supports data capture on a non-rooted Android device. It collects CPU performance data and Mali GPU performance data so you can profile your game or app without device modification. Configuring Arm Streamline to collect the right data is easy – use the templates to select the most appropriate set of counters for your target device. Identify bottlenecks and optimize your application for mobile devices faster.

System profiling

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

Related references

Chapter 2 Application profiling on an Android device on page 2-15

Chapter 3 System profiling on an Android device on page 3-19

1.2 Compiling your application

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

When using GCC or Clang, use the following options:

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

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

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

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

Call-stack unwinding

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

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

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

Note

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

Android

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

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

To run `dex2oat` with the `--no-strip-symbols` option, run the following command on the device and then re-install the APK file:

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

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

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

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

```
readelf –S …/images/*.dex
```

Related information

readelf

1.3 Set up your host machine

To use Arm Streamline, install the necessary software and set up environment variables on your host machine.

Prerequisites

- Python 3.5 or higher installed and on the `PATH`.
- A target device that is correctly configured to generate performance data. You can use many Android devices off-the-shelf. A list of the recommended consumer devices that support Arm Streamline is on *https://developer.arm.com/tools-and-software/graphics-and-gaming/arm-mobile-studio/support/supported-devices*.

  If you are building your own device software, ensure that your kernel configuration includes the options that are described in *A.1 Kernel configuration menu options* on page Appx-A-28.

Procedure

1. Download a studio package appropriate to your requirements.
   - Download Arm Mobile Studio from *https://developer.arm.com/tools-and-software/graphics-and-gaming/arm-mobile-studio/downloads*.
   - Download Arm Development Studio from *https://developer.arm.com/tools-and-software/embedded/arm-development-studio/downloads*.
2. Install the studio package.
   - Install Arm Mobile Studio using the instructions at *https://developer.arm.com/tools-and-software/graphics-and-gaming/arm-mobile-studio/installation*.
   - Install Arm Development Studio using the instructions in the *Arm Development Studio Getting Started Guide*.
3. Install Android Debug Bridge (adb).

   adb is available with the Android SDK platform tools (*https://developer.android.com/studio/releases/platform-tools.html*).
4. Ensure that the Android SDK platform tools directory is in the PATH environment variable.

   The adb executable must be accessible to Arm Streamline.

Next Steps

- *1.4 Set up your target device* on page 1-14
- *2.1 Profile your application* on page 2-16

1.4 Set up your target device

To use Arm Streamline, set up a target device with the application to profile.

Prerequisites

1.3 Set up your host machine on page 1-13

Procedure

1. Ensure that Developer Mode is enabled, then enable USB Debugging by selecting **Settings** > **Developer options**.
2. Connect the target to the host through USB. If the connection is successful, running the `adb devices` command on the host returns the ID of your target, and you can run `adb shell`.
3. Build a debuggable application, where the application is marked as debuggable in the Android application manifest.
4. Install the application to be profiled.

Next Steps

- *2.1 Profile your application* on page 2-16

Chapter 2
Application profiling on an Android device

Profile your application while it is running on a non-rooted Android device.

It contains the following sections:

2.1 Profile your application

Set up and run Arm Streamline with an unrooted Android target with a Mali GPU.

Prerequisites

- *1.3 Set up your host machine* on page 1-13
- *1.4 Set up your target device* on page 1-14

Procedure

1. On the host, run the `gator_me.py` Python script to set up the target device so that Arm Streamline can connect to it.

   The script is in the following directory:

   `<install_directory>/streamline/gator/`

   Use the following command-line arguments:
   - The Android package name of the application that you want to profile.
   - The path on the host to the `gatord` binary to install on the device. By default, this path is the current working directory. Your installation provides two versions of `gatord`, in the following directories:
     — `<install_directory>/streamline/bin/arm/` for 32-bit architectures.
     — `<install_directory>/streamline/bin/arm64/` for Armv8 64-bit architectures.

   Example:

   ```
   python3 gator_me.py --package <your_app_package> --daemon <path_to_gatord>
   ```

   The `gator_me.py` script does the following:
   - Kills and removes `gatord` and removes any counter configuration file that was previously created.
   - Enables perf profiling.
   - Copies `gatord` to the target.
   - Runs `gatord` inside your Android application sandbox.
   - Configures port forwarding.
   - Waits for you to configure and perform the capture in Arm Streamline.
   - When the capture is complete, it kills and removes `gatord`.
2. Launch Arm Streamline and connect to the target.

   In the **Target** view, enter `localhost:8080` in the address field. This value is the local TCP port that is specified in the `gator_me` script.
3. Configure the counters to collect.

   Click **Counter Configuration**, then **Add counters from a template**. Select a counter template appropriate for the target GPU from the drop-down list.

   Counter templates are pre-defined sets of counters that have been chosen to enable you to perform an initial performance review of both CPU and GPU behavior. Arm Streamline notifies you if the target device does not support all the counters that are defined in the selected template.
4. Optionally, click **Capture & analysis options** to set more capture options, including the sample rate and the capture duration (by default unlimited).
5. Click **Start Capture** to start the capture.

   Specify the name and location on the host of the capture file that Arm Streamline will create when the capture is complete. Arm Streamline then switches to **Live** view.
6. Start the application that you want to profile.

   Live view shows charts for each counter that you selected in step 3. Below the charts is a list of running processes in your application with their CPU usage. The charts update in real time to show the data that `gatord` captures from your running application.

7. Stop the capture.

   Unless you specified a capture duration, click **Stop capture** to end the capture. Arm Streamline stores the capture file in the location that you specified in step 5 and then prepares the capture for analysis. When complete, the capture appears in the **Timeline** view.
8. Select the template in the **Timeline** view for visualization.

   Click **Switch and manage templates** to select the same template that you used to create the capture in step 3.

Next Steps

- Analyze the data. For more information about how to analyze performance with Arm Streamline, see *Analyze your capture* in the *Arm Streamline User Guide*.
- On the host, switch back to the terminal running the `gator_me.py` script and press any key to terminate the script. The script kills all processes that it started and removes `gatord` from the target.

Related information

Capture a Streamline profile in the Arm Streamline User Guide

2.2 Generate a headless capture

When integrating performance analysis into continuous integration, capturing data without having the host tool connected or a user manually controlling the GUI is often required. Use the `gator_me.py` script in headless mode to capture data without the Arm Streamline host tool connected.

Prerequisites

Complete the first three steps in *2.1 Profile your application* on page 2-16. In step three, in the **Counter Configuration** dialog, export the counter configuration that you want to capture for your target device to a `configuration.xml` file. Create one configuration file for each device class.

Procedure

1. On the host, run the `gator_me.py` Python script to set up the target device for a headless data capture.

   The script is in the following directory:

   `<install_directory>/streamline/gator/`

   Use the following command-line arguments:
   - The Android package name of the application that you want to profile.
   - The path on the host to the `gatord` binary to install on the device. By default, this path is the current working directory. Your installation provides two versions of `gatord`, in the following directories:
     — `<install_directory>/streamline/bin/arm/` for 32-bit architectures.
     — `<install_directory>/streamline/bin/arm64/` for Armv8 64-bit architectures.
   - The path to the configuration file that you saved in the Prerequisites.
   - The path to store the saved output file to.

   Example:

   ```
   python3 gator_me.py --package <your_app_package> --daemon <path_to_gatord> --config
   <path_to_your_configuration.xml> --headless <output.apc.zip>
   ```
2. Run your test scenario and exit the application when it has completed.
3. Wait for the script to download the data from the target, and write out the `output.apc.zip` file.

   The script stops automatically when it detects that the application is no longer running.
4. To view the data in the Arm Streamline GUI, start the host application and import the APC file into the **Streamline Data** view.

Next Steps

- Analyze the data. For more information about how to analyze performance with Arm Streamline, see *Analyze your capture* in the *Arm Streamline User Guide*.

Related information

Capture a Streamline profile in the Arm Streamline User Guide

Chapter 3
System profiling on an Android device

Profile all applications and services that are running on a rooted Android device.

It contains the following sections:

3.1 Installing user space gator

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

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

Two pre-built `gatord` binaries are available:
- For Armv7 targets, and Armv8 targets that support AArch32 execution state.
- For Armv8 AArch64 targets.

The source code for `gatord` is available from `<install_directory>/sw/streamline/gator/daemon/`.

3.2 Profile your system

Set up and run Arm Streamline with a rooted Android target with a Mali GPU.

Prerequisites

- *1.3 Set up your host machine* on page 1-13
- *1.4 Set up your target device* on page 1-14

Procedure

1. Run `gatord` as root using the following commands:

```
adb push gatord /data/local/tmp
adb shell /data/local/tmp/gatord
```

2. Continue from step two of *2.1 Profile your application* on page 2-16.

3.3 Enabling atrace annotations

Arm Streamline can capture Android trace points that atrace generates. It supports atrace annotations on Android targets that are running Linux kernel versions 3.10 and later.

Arm Streamline converts application-generated atrace macros into either string annotations or counter charts. It also lists any Android `ATRACE_TAG_*` macros that you enable as available events in an **Atrace** section in the **Counter Configuration** dialog. If you expect to see atrace events in this dialog but none are displayed, click the Warnings tag in the **Counter Configuration** dialog to see why atrace support is not enabled.

To notify running applications that atrace annotation tags have been enabled, the file `notify.dex` must be installed on the target in the same directory as `gatord`. You can install a pre-built version of `notify.dex` as part of target setup, by clicking the **Setup target...** button in the **Connection Browser** dialog. The Java source code for `notify.dex` is available in the following locations:

- `<install_directory>/sw/streamline/gator/notify/`
- *https://github.com/ARM-software/gator/tree/master/notify*

Chapter 4
Troubleshooting Common Issues

Troubleshoot common Arm Streamline issues.

It contains the following sections:

4.1 Troubleshooting target connection issues

You might have problems when trying to start a capture session, for instance by pressing the **Start capture** button. Use these solutions to solve common target connection issues.

Problem	Solution

Error message generated:
`Unable to connect to the gator daemon at <target_address>.`
`Please verify that the target is reachable and that you are running gator daemon v17 or later. Installation instructions can be found in: streamline/gator/README.md.`
`If connecting over WiFi, please try again or use a wired connection.`
	Make sure `gatord` is running on your target. Enter the following command in the shell of your target:
	`ps ax | grep gatord`
	If this command returns no results, `gatord` is not active. Start it by navigating to the directory that contains `gatord` and entering the following command:
	`sudo ./gatord &`
	Try connecting to the target again.
	If `gatord` is active and you still receive this error message, try disabling any firewalls on your host machine that might be interfering with communication between it and the target.
	In addition, if you are running Android on your target, make sure that the ports are accessible by using the `adb forward` command. For example:
	`adb forward tcp:8080 tcp:8080`

Error message generated: `Unknown host`	Make sure that you have correctly entered the name or IP address of the target in the **Address** field. If you have entered a name, try entering an IP address instead.
When using event-based sampling, Arm Streamline fails to find the PMU.	The PMU on your hardware might not be correctly configured to allow the processor interrupts necessary for Arm Streamline to use event-based sampling. Test on alternate hardware or disable event-based sampling in the **Counter Configuration** dialog box.

The target is running a firewall, which prevents Arm Streamline from connecting to `gatord`.	There are several possible ways to resolve this issue:
	• Update the firewall to allow connections to `gatord`, which defaults to using port 8080.
	• Use local captures.
	• If the target accepts SSH connections, you can establish an SSH tunnel by using the `ssh` command on the host. For example:
	`ssh <user>@<target> -L 8080:localhost:8080 -N`
	In this example, replace <user> with the username to log in as and <target> with the hostname of the target. On the target, use `localhost` as the hostname.
	——— **Note** ———
	An SSH tunnel requires extra processing on the target.
	———————
	• Reverse SSH tunnels are also possible by running `ssh` from the target to the host. For example:
	`ssh <user>@<host> -R 8080:localhost:8080 -N`

4.2 Troubleshooting Android issues

Android has the following known issues:

Problem	**Solution**
`run-as` command fails on Android.	Make sure that the application is debuggable and the target device is running the latest software update.
Capture fails on startup, usually showing no events captured.	This problem usually indicates a failure to configure the `perf` API. Run the following command: `adb shell setprop security.perf_harden 0`
	Reduce the set of events that are captured or try capturing only a limited set of `perf` software events. This error can be caused by: • Exceeding the limit for the number of open file descriptors. • The target device does not have a correctly configured PMU driver.
Hardware counters read as zero.	This error is usually a sign of misconfigured PMU. It is not usually possible to work around.
When running non-root on Android, `gatord` exits with the message: `Error creating server TCP socket`	• Run `gatord -p uds` … to enable use of UDS socket instead of TCP socket. • Execute `adb forward tcp:<some-port> localabstract:streamline-data` on your host to configure port forwarding. • Set the target address as `localhost:<some-port>` in the **Target View Address Field** text box.

4.3 Troubleshooting gator issues

Consult the following table for solutions to issues related to gator.

Problem	Solution
Annotations do not work on Android.	Use application profiling. If you must use system profiling, disable SELinux by running `# setenforce 0`.

Appendix A
Advanced target setup information

This appendix provides extra configuration information beyond the standard setup.

It contains the following sections:

A.1 Kernel configuration menu options

You must enable certain kernel configuration options to run Arm Streamline.

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

Note

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

General Setup

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

Note

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

From the module directory, enter the following command:

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

This command produces a `.ko` file.

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

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

Kernel Features

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

CPU Power Management

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

Kernel hacking

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

Caution

Kernel versions before 4.6, with `CONFIG_CPU_PM` enabled, produce invalid results. For example, counters not showing any data, large spikes, and non-sensible values for counters. This issue is a result of the kernel PMU driver not saving state when the processor is powered down, or not restoring state when it is powered up. To avoid this issue, upgrade to the latest version of the kernel, or apply the patch found at *https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=da4e4f18afe0f3729d68f3785c5802f786d36e34*. This patch applies cleanly to version 4.4, and it might also be possible to back port it to other versions. If you apply the patch, you might also need to apply the patch at *https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=cbcc72e037b8a3eb1fad3c1ae22021df21c97a51*.

A.2 Building the gator daemon yourself

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

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

——— **Note** ———

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

———————

To build `gatord`, follow these steps:

Procedure

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

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

     ```
     mv daemon jni
     ```
3. Issue the commands to build `gatord`.

   ```
   <NDK_install_directory>/ndk-build
   ```

   Results: `gatord` is now located in `libs/armeabi`.

   ——— **Note** ———

   To build `gatord` for AArch64, edit `jni/Application.mk` and replace `armeabi-v7a` with `arm64-v8a`.

   ———————

4. Make `gatord` executable by entering the following command:

   ```
   chmod +x gatord
   ```

A.3 gatord command-line options

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

`gatord` has two modes of operation:

Daemon mode (the default mode)
Sends captured events to a host running Arm Streamline.

Local capture mode
Writes the capture to a file then exits.
To enable this mode, specify an output directory with the `--output` flag.

Arguments available to all modes:

Option	Description
`-h, --help`	Lists all of the available `gatord` command-line options.
`-c, --config-xml <config_xml>`	Specify the path and filename of the `configuration.xml` file that defines the capture options. In daemon mode, the list of counters is written to this file. In local capture mode, the list of counters is read from this file.
`-e, --events-xml <events_xml>`	Specify the path and filename of the `events.xml` file. `events.xml` defines all of the counters that Arm Streamline collects during the capture session.
`-E, --append-events-xml <events_xml>`	Specify the path and filename of `events.xml` to append.
`-P, --pmus-xml <pmu_xml>`	Specify path and filename of `pmu.xml` to append.
`-m, --module-path <module>`	Specify path and filename of `gator.ko`.
`-v, --version`	Print version information.
`-d, --debug`	Enable debug messages.
`-A, --app <cmd> <args...>`	Specify the command to execute when the capture starts. This argument must be the last argument that is passed to `gatord`. All subsequent arguments are passed to the launched application.
`-S, --system-wide <yes|no>`	Specify whether to capture the whole system. In daemon mode, `no` is only applicable when `--allow-command` is specified. In this mode, you must enter a command in the **Capture & Analysis Options** dialog box. Defaults to `yes`, unless `--app`, `--pid`, or `--wait-process` is specified.
`-u, --call-stack-unwinding <yes|no>`	Enable or disable call stack unwinding. Defaults to `yes`.
`-r, --sample-rate <low|normal>`	Specify sample rate for capture. Defaults to `normal`.
`-t, --max-duration <s>`	Specify the maximum duration that the capture can run for in seconds. Defaults to `0`, meaning unlimited.
`-f, --use-efficient-ftrace <yes|no>`	Enable efficient ftrace data collection mode. Defaults to `yes`.
`-w, --app-cwd <path>`	Specify the working directory for the application that `gatord` launches. Defaults to the current directory.
`-x, --stop-on-exit <yes|no>`	Stop capture when launched application exits. Defaults to `no`, unless `--app`, `--pid`, or `--wait-process` is specified.

(continued)

Option	Description
`-Q, --wait-process <command>`	Wait for a process matching the specified command to launch before starting capture. Attach to the specified process and profile it.
`-Z, --mmap-pages <n>`	The maximum number of pages to map per mmaped perf buffer is equal to `<n+1>`. `n` must be a power of two.

Arguments available in daemon mode only:

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

Arguments available to local capture mode only:

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

Argument usage examples

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

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

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

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

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