



Arm[®] Streamline

Version 8.0

Target Setup Guide for Android

Non-Confidential

Issue 00

Copyright © 2021–2022 Arm Limited (or its affiliates). All rights reserved. 101813_0800_00_en



Arm® Streamline Target Setup Guide for Android

Copyright © 2021–2022 Arm Limited (or its affiliates). All rights reserved.

Release information

Document history

Issue	Date	Confidentiality	Change
0708-00	20 August 2021	Non-Confidential	New document for v7.8
0709-00	18 November 2021	Non-Confidential	New document for v7.9
0800-00	18 February 2022	Non-Confidential	New document for v8.0

Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND

REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

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

Copyright © 2021–2022 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349)

Confidentiality Status

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

Unrestricted Access is an Arm internal classification.

Product Status

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

Feedback

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

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

Inclusive language commitment

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

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

Contents

1 Introduction.....	6
1.1 Conventions.....	6
1.2 Other information.....	7
2 Target Setup.....	8
2.1 Application and system profiling.....	8
2.2 Compiling your application.....	8
2.3 Set up your host machine.....	10
2.4 Set up your target device.....	11
3 Application profiling on an Android device.....	12
3.1 Profile your application.....	12
3.1.1 Connect Streamline to Android devices.....	12
3.1.2 Choose a counter template.....	13
3.1.3 Capture a profile.....	14
3.2 Generate a headless capture.....	15
4 System profiling on an Android device.....	17
4.1 Profile your system.....	17
4.2 Enabling atrace annotations.....	17
5 Troubleshooting known issues.....	18
5.1 Troubleshooting target connection issues.....	18
5.2 Troubleshooting Android issues.....	18
5.3 Troubleshooting gatord issues.....	19
6 Advanced target setup information.....	20
6.1 Kernel configuration menu options.....	20
6.2 Build gatord yourself.....	21
6.3 gatord command-line options.....	21
6.4 Connect Streamline to devices using TCP.....	25

1 Introduction

1.1 Conventions

The following subsections describe conventions used in Arm documents.




Glossary




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

See the Arm® Glossary for more information: developer.arm.com/glossary.

Typographic conventions

Arm documentation uses typographical conventions to convey specific meaning.

Convention	Use
<i>italic</i>	Citations.
bold	Interface elements, such as menu names. Signal names. Terms in descriptive lists, where appropriate.
monospace	Text that you can enter at the keyboard, such as commands, file and program names, and source code.
monospace bold	Language keywords when used outside example code.
monospace <u>underline</u>	A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: <pre>MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2></pre>
SMALL CAPITALS	Terms that have specific technical meanings as defined in the <i>Arm® Glossary</i> . For example, IMPLEMENTATION DEFINED , IMPLEMENTATION SPECIFIC , UNKNOWN , and UNPREDICTABLE .
 Caution	Recommendations. Not following these recommendations might lead to system failure or damage.
 Warning	Requirements for the system. Not following these requirements might result in system failure or damage.
 Danger	Requirements for the system. Not following these requirements will result in system failure or damage.

Convention	Use
 Note	An important piece of information that needs your attention.
 Tip	A useful tip that might make it easier, better or faster to perform a task.
 Remember	A reminder of something important that relates to the information you are reading.

1.2 Other information

See the Arm website for other relevant information.

- [Arm® Developer](#).
- [Arm® Documentation](#).
- [Technical Support](#).
- [Arm® Glossary](#).

2 Target Setup

This chapter explains how to set up your target and host devices ready to use Streamline for application or system profiling.

2.1 Application and system profiling

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

Application profiling

Streamline supports data capture on a non-rooted Android device. It collects CPU performance data and Arm® Mali™ GPU performance data from a single application, so you can profile your game or application without device modification. Configuring 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, Streamline also supports system-wide Android profiling when running on development devices with root access. System profiling enables manufacturers to simultaneously monitor all applications and services running on their device, allowing identification of problematic processes or scheduling behaviors.



Some Streamline features are license-managed. For more information, see [Licenses](#) in the *Arm Streamline User Guide*.

Related information

[Application profiling on an Android device](#) on page 12

[System profiling on an Android device](#) on page 17

2.2 Compiling your application

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

When using GCC 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 Streamline to record the call stack with each sample taken.

-mno-omit-leaf-frame-pointer

Keeps the frame pointer in leaf functions.

-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 Streamline.

Call stack unwinding

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.

For AArch32 applications, the flags `-fno-omit-frame-pointer`, `-marm`, and `-mapcs-frame` are required.



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, Streamline can profile OAT files that Android runtime (ART) generates, in addition to profiling native code.

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 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 can use the GNU tools `readelf` command, for example:

```
readelf -S ../images/*.dex
```

2.3 Set up your host machine

Streamline is available for the Arm® Mobile Studio or the Arm Development Studio product suites. To use Streamline, install the necessary software and set up environment variables on your host machine.

Before you begin

- Ensure you have 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 Streamline is available from <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 [Kernel configuration menu options](#).

Procedure

1. Download and install the studio package appropriate to your host platform (Windows, Linux, or macOS).
 - 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, which you can download from <https://developer.android.com/studio/releases/platform-tools>.
4. Add the path to the Android SDK platform tools directory to your `PATH` environment variable. The adb executable must be accessible to Streamline.

Next steps

- [Set up your target device](#)
- [Profile your application](#)

2.4 Set up your target device

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

Before you begin

[Set up your host machine](#)

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 and approve the debug connection when prompted. 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 and install the debuggable application for profiling:
 - For Unity applications, select the **Development Build** option in the **Build Settings**.
 - For applications that are not built with Unity, ensure it is marked as debuggable in the Android application manifest. See how to debug your application in the [Android Studio documentation](#).

Next steps

[Profile your application](#)

3 Application profiling on an Android device

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

3.1 Profile your application

Capture a profile of a debuggable application running on an unrooted Android target with a Mali™ GPU.

3.1.1 Connect Streamline to Android devices

Use the **Start** view to connect to Android devices and collect data from debuggable applications.

Before you begin

Before you can connect Streamline to a device, ensure you have followed the steps in [Set up your target device](#).

About this task

To profile non-debuggable applications on pre-release (eng or user debug) builds of Android, you must manually install `gator`, and connect to your device using the **TCP (Advanced)** view. See [Connect Streamline to devices using TCP](#) for more information.

Procedure

1. Launch Streamline:

- On Windows, from the **Start** menu, navigate to the **Arm Mobile Studio** folder, and select the **Streamline** shortcut.
- On macOS, go to the `<install_directory>/streamline` directory, and double-click the `streamline.app` file.
- On Linux, go to the `<install_directory>/streamline` directory, and run the `streamline` file:

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

2. In the **Start** view, select **Android (adb)** as your target device type, then select your device from the list of detected devices. If you do not see your device in the list, check that it is connected properly through USB. See [Set up your target device](#) for more information.
3. Wait a few moments for the list of available packages to populate, then select the package you want to profile from the list of packages available on the selected device.

Next steps

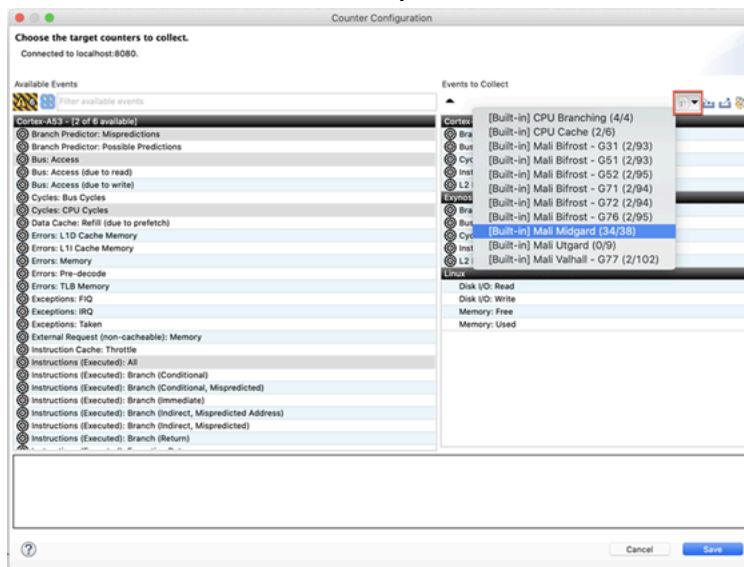
Choose a counter template. For more information about how to find and select a counter template, see [Choose a counter template](#).

3.1.2 Choose a counter template

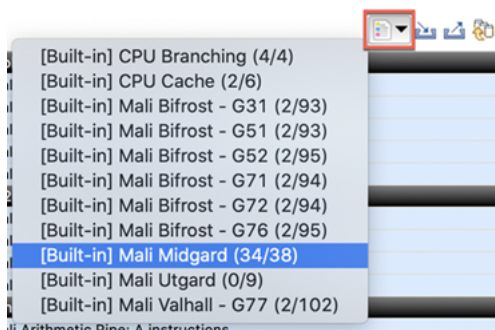
Counter templates are pre-defined sets of counters that enable you to review the performance of both CPU and GPU behavior. Choose the most appropriate template for the GPU in your target device.

Procedure

1. In the **Start** view, click **Configure Counters**.
2. Click **Add counters from a template**  to see a list of available templates.



3. Select a counter template appropriate for the GPU in your target device, then **Save** your changes.
The number of counters in the template that your target device supports is shown next to each template. For example, here, 34 of the 38 available counters in the Mali™ Midgard template are supported in the connected device.



4. Optionally, in the **Start** view, click **Advanced Settings** to set more capture options, including the sample rate and the capture duration (by default unlimited). Refer to [Set capture options](#) in the *Arm Streamline User Guide*.

Next steps

Capture a profile using Streamline. For more information about how to capture the behavior of your CPU and GPU performance using Streamline, see [Capture a profile](#).

3.1.3 Capture a profile

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

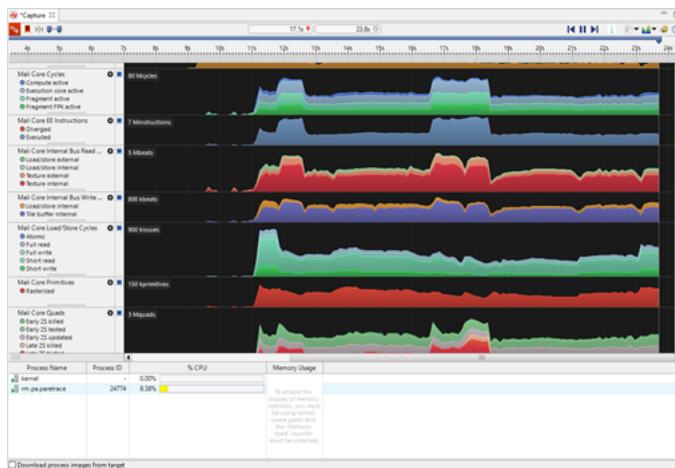
Before you begin


Before you capture a profile, ensure you have followed the steps in:


- [Set up your target device](#)
- [Connect Streamline to Android devices](#)

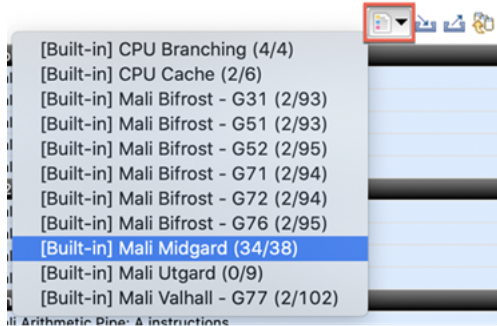
Procedure

1. In the **Start** view, click **Start Capture** to start capturing data from the target device. Specify the name and location on the host of the capture file that Streamline will create when the capture is complete. Streamline then switches to the **Live** view and waits for you to start the application on the device.
2. Start the application that you want to profile. The **Live** view shows charts for each counter that you selected. Below the charts is a list of running processes in your application with their CPU usage. The charts now start updating in real time to show the data that Streamline captures from your running application.



3. Unless you specified a capture duration, in the **Capture Control** view, click **Stop capture and analyze**  to end the capture. Streamline stores the capture file in the location that you specified previously, and then prepares the capture for analysis. When complete, the capture appears in the **Timeline** view.

4. Click **Switch and manage templates**  and select the same counter configuration template that you chose to create the capture.



Next steps

Analyze the data. For more information about how to analyze performance with Streamline, see [Analyze your capture](#) in the *Arm Streamline User Guide*.

3.2 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. To capture data without the Streamline host tool connected, use the `gator_me.py` script in headless mode.

Before you begin

- Install Python 3.5 (or higher). You need Python 3.5 or higher to run the provided `gator_me.py` script, which uses the `gator` agent to connect Streamline to your Android target.
- Add the path to the `python3` directory to your `PATH` environment variable.
- [Connect Streamline to Android devices](#).
- [Choose a counter template](#).
- [Export the counter configuration to a configuration.xml file](#). Create one configuration file for each device class.

About this task

For Arm® Mobile Studio Professional users, please refer to the dedicated tutorial [Integrate Arm Mobile Studio into a CI workflow](#).

Procedure

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

```
python3 gator_me.py --package <your_app_package> --daemon <path_to_gatord> --  
config <path_to_your_configuration.xml> --headless <output.apc.zip>
```

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/android/arm/` for 32-bit architectures.
 - `<install_directory>/streamline/bin/android/arm64/` for 64-bit architectures.
 - The path to the configuration file that you saved in the Prerequisites.
 - The path to store the saved output file to.
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 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 Streamline, see [Analyze your capture](#) in the *Arm Streamline User Guide*.

Related information

[Capture a Streamline profile](#)

4 System profiling on an Android device

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

4.1 Profile your system

Set up and run Streamline with Android target root user access with a Mali™ GPU.

Before you begin

- [Set up your host machine](#)
- [Set up your target device](#)

Procedure

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

```
adb push gator /data/local/tmp
adb shell
cd /data/local/tmp
su
./gator --system-wide=yes
```

2. Continue from step four of [Connect Streamline to devices using TCP](#).

4.2 Enabling atrace annotations

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.

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 box. If you expect to see atrace events in this dialog box but none are displayed, click the Warnings tag to see why atrace support is disabled.

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 `gator`. The Java source code for `notify.dex` is available in the following locations:

- `<install_directory>/streamline/gator/notify/`
- The `notify` directory in <https://github.com/ARM-software/gator>

5 Troubleshooting known issues

Troubleshoot known Streamline issues.

5.1 Troubleshooting target connection issues

Use these solutions for common target connection issues.

Problem	Solution
Streamline cannot connect to the gator daemon on the target device.	Reboot the target device and try again.
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, Streamline fails to find the CPU PMU.	The PMU on your hardware might not be correctly configured to allow the processor interrupts necessary for Streamline to use event-based sampling. Test on alternate hardware or disable event-based sampling in the Counter Configuration dialog box.

5.2 Troubleshooting Android issues

Android has the following known issues:

Problem	Solution
<code>run-as</code> 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.	<p>This problem usually indicates a failure to configure the <code>perf</code> API. Run the following command:</p> <pre>adb shell setprop security.perf_harden 0</pre> <p>Reduce the set of events that are captured or try capturing only a limited set of <code>perf</code> software events. The following issues can cause this error:</p> <ul style="list-style-type: none"> 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.

5.3 Troubleshooting gatord issues

Consult the following table for solutions to issues related to `gatord`.

Problem	Solution
Annotations do not work when using system profiling.	Use the application profiling connection method. If you must use system profiling, disable SELinux by running # <code>setenforce 0</code> .

6 Advanced target setup information

This appendix provides extra configuration information beyond the standard setup.

6.1 Kernel configuration menu options

You must enable certain kernel configuration options to run Streamline.

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



- 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`.

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 chart in the **Timeline** view. `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 due to the kernel PMU driver not saving state when the processor powers down, or not restoring state when it powers 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 require the patch at <https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=cbcc72e037b8a3eb1fad3c1ae22021df21c97a51>.

6.2 Build gatord yourself

Build `gatord` yourself to apply patches for bug fixes or add support for new features.

About this task



Note

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

Procedure

1. Either download the `gatord` source from the daemon directory in <https://github.com/ARM-software/gator>, or copy the source that is supplied in `<DS_install_directory>/sw/streamline/gator/daemon/`.
2. Follow the instructions in the `README.md` file in the `gator` directory.

6.3 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 Streamline running on your host.

`gatord` has two modes of operation:

Daemon mode (the default mode)

Sends captured events to a host running 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 the available <code>gator</code> command-line options.
-c, --config-xml <config_xml>	Specify the path and filename of the <code>configuration.xml</code> 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 <code>events.xml</code> file. <code>events.xml</code> defines all the counters that Streamline collects during the capture session.
-E, --append-events-xml <events_xml>	Specify the path and filename of the <code>events.xml</code> file to append.
-P, --pmu-xml <pmu_xml>	Specify the path and filename of the <code>pmu.xml</code> file to append.
-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 <code>gator</code> . All subsequent arguments are passed to the launched application.
-k, --exclude-kernel <yes no>	Specify whether to filter out kernel events from the perf results.
-S, --system-wide <yes no>	Specify whether to capture the whole system. In daemon mode, <code>no</code> is only applicable when <code>--allow-command</code> is specified. In this mode, you must enter a command in the Start view. Requires kernel events to be enabled in Capture and Analysis Options of Streamline, or by setting <code>--exclude-kernel no</code> . Defaults to <code>yes</code> , unless <code>--app</code> , <code>--pid</code> , or <code>--wait-process</code> is specified.
-u, --call-stack-unwinding <yes no>	Enable or disable call stack unwinding. Defaults to <code>yes</code> .
-r, --sample-rate <none low normal high>	Specify the sample rate for the capture. The frequencies for each sample rate are: <ul style="list-style-type: none">• <code>high</code> = 10kHz• <code>normal</code> = 1kHz• <code>low</code> = 100Hz• <code>none</code> = the lowest possible rate Defaults to <code>normal</code> .
-t, --max-duration <s>	Specify the maximum duration that the capture can run for in seconds. Defaults to 0, which means unlimited.
-f, --use-efficient-ftrace <yes no>	Enable efficient ftrace data collection mode. Defaults to <code>yes</code> .
-w, --app-cwd <path>	Specify the working directory for the application that <code>gator</code> launches. Defaults to the current directory.
-x, --stop-on-exit <yes no>	Stop the capture when the launched application exits. Defaults to <code>no</code> , unless <code>--app</code> , <code>--pid</code> , or <code>--wait-process</code> is specified.
-Q, --wait-process <command>	Wait for a process that matches the specified command to launch before starting the capture. Attach to the specified process and profile it.

Option	Description
<code>-Z, --mmap-pages <n></code>	The maximum number of pages to map per mmaped perf buffer is equal to <code><n+1></code> . <code>n</code> must be a power of two.
<code>-O, --disable-cpu-onlining <yes no></code>	To not switch on CPU cores that are offline to read their information. This option is useful for kernels that fail to handle this action correctly, for example they reboot the system. Defaults to <code>no</code> .
<code>-F, --spe-sample-rate <n></code>	Specify the SPE periodic sampling rate. The rate, <code><n></code> , is the number of operations between each sample, and must be a nonzero positive integer. The hardware specifies the minimum rate. Values below this threshold are ignored and the hardware minimum is used instead.

Arguments available on Android targets only:

Option	Description
<ul style="list-style-type: none"> <code>l, --android-pkg <pkg></code> 	Profiles the specified android package. Waits for the package app to launch before starting a capture unless <code>--android-activity</code> is specified.
<ul style="list-style-type: none"> <code>m, --android-activity <activity></code> 	Launch the specified activity of a package and profile its process. You must also specify <code>--android-pkg</code> .

Arguments available in daemon mode only:

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

Arguments available to local capture mode only:

Option	Description
<code>-s, --session-xml <session_xml></code>	Specify the <code>session.xml</code> file that the configuration is taken from. Any additional arguments override values that are specified in this file.

Option	Description
<code>-o, --output <apc_dir></code>	Specify the path and filename of the output directory for a local capture. The directory path will be appended with the extension <code>.apc</code> if it is not already the case.
<code>-i, --pid <pids...></code>	A comma-separated list of process IDs to profile.
<code>-C, --counters <counters></code>	<p>A comma-separated list of counters to enable. You can specify this option multiple times. An event code and a slot identify most hardware counters. To specify the counter for a particular slot, pass:</p> <pre>--counters <device>_cnt<s>:<e></pre> <p>Where:</p> <ul style="list-style-type: none"> <code><device></code> is the prefix that identifies the device type. <code><s></code> is the slot number. <code><e></code> is the event code.
<code>-X, --spe <id>[:events=<indexes>][:ops=<types>][:min_latency=<lat>]</code>	<p>Enable the Statistical Profiling Extension (SPE).</p> <p>Where:</p> <ul style="list-style-type: none"> <code><id></code> is the name of the SPE properties that are specified in the <code>events.xml</code> or <code>pmus.xml</code> file. It uniquely identifies the available events and counters for the SPE hardware. <code><indexes></code> is a comma-separated list of event indexes to filter the sampling by. A sample is only recorded if all events are present. <code><types></code> is a comma-separated list of operation types to filter the sampling by. If a sample is any of the types in <code><types></code>, it is recorded. Valid types are <code>LD</code> for load, <code>ST</code> for store and <code>B</code> for branch. <code><lat></code> is the minimum latency. A sample is only recorded if its latency is greater than or equal to this value. The valid range is <code>[0,4096)</code>.

Argument usage examples

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

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

`--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 `gator` without having to rebuild `gator` 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" description="loginuid"/>
</category>
</events>
```

- The Instructions Executed counter is configured in slot 0 as:

```
--counters ARMv8_Cortex_A53_cnt0:0x08
```

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

```
--counters ARMv8_Cortex_A53_ccnt
```

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

```
--counters PERF_COUNT_SW_PAGE_FAULTS
```

6.4 Connect Streamline to devices using TCP

The Android connection mode, which is accessed from the **Start** view, requires the application to be debuggable. This mode allows the Streamline daemon, `gator`, to connect and capture data on released consumer devices. For users profiling non-debuggable applications on pre-release (`eng` or `userdebug`) builds of Android, Arm provides a Python script, `gator_me.py`, to set up the device connection.

Procedure

1. On your host machine, navigate to the Streamline installation directory, `<install_directory>/streamline/gator/`.
2. To supply the path to the `gator` binary to install on the device, run the `gator_me.py` Python script with the `--daemon` option.
Your installation directory contains two versions of `gator`, for 32-bit or 64-bit architectures:
 - `<install_directory>/streamline/bin/android/arm/` for 32-bit architectures.
 - `<install_directory>/streamline/bin/android/arm64/` for 64-bit architectures.

For example:

```
python3 gator_me.py --daemon <install_directory>/streamline/bin/android/arm64/
gator
```

3. The script returns a numbered list of the Android package names for the debuggable applications that are installed on your device. Enter the number of the package you want to profile.

The `gator_me.py` script does the following:

- Kills and removes `gator` and removes any counter configuration file that was previously created.
- Enables perf profiling.
- Copies `gator` to the target.
- Runs `gator` inside your Android application sandbox.
- Configures port forwarding.
- Waits for you to configure and perform the capture in Streamline.
- When the capture is complete, it kills and removes `gator`.



Alternatively, if you know the Android package name of the application you want to profile, you can specify it when running the script, using the `--package` option.

```
python3 gator_me.py --package com.mycompany.myapp --daemon  
<install_directory>/streamline/bin/android/arm64/gator
```

4. Launch Streamline:

- On Windows, from the **Start** menu, navigate to the **Arm Mobile Studio folder**, and select the **Streamline** shortcut.
- On macOS, go to the `<install_directory>/streamline` folder, and double-click the `streamline.app` file.
- On Linux, go to the `<install_directory>/streamline` folder, and run the `streamline` file:

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

5. Open the **Start** view, and select **TCP (Advanced)** as your target device type.
6. Select your device by entering the address or by using `adb <serial-number>`. Alternatively, select your target device from the list of detected devices.
7. Optionally enter the details for any command you want to run on the application.

Next steps

Choose a counter template. For more information about how to find and select a counter template, see [Choose a counter template](#).