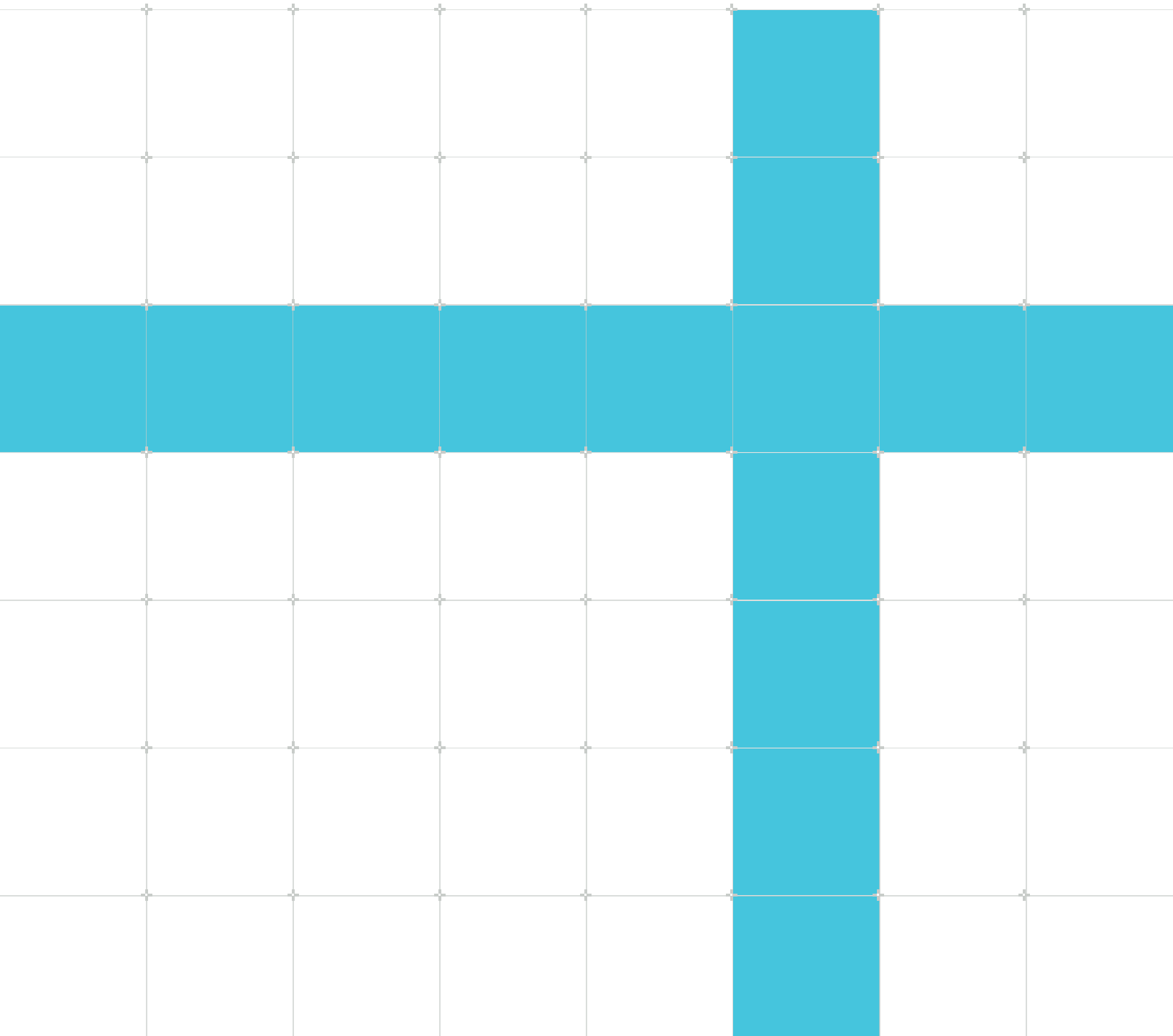
Arm® Streamline

Version 9.5

Target Setup Guide for Android

Issue 01
101813_9.5_01_en



Arm® Streamline Target Setup Guide for Android

This document (101813_9.5_01_en) was issued on 2025-03-20. There might be a later issue at https://developer.arm.com/documentation/101813

The product version is 9.5.

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

Start reading

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

Intended audience

This book is intended for users who need to use Arm® Streamline Performance Analyzer on Android targets. It describes how to prepare your Android target for either application or system profiling, and how to capture a profile with Streamline.

Inclusive language commitment

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

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

Feedback

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

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

Contents

1. Target Setup

This chapter explains how to get your application, device and host machine ready to use Streamline for application or system profiling.

1.1 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 Android devices. Streamline collects CPU performance data and Arm® Mali™ GPU, or Arm Immortalis™ GPU, performance data from a single application, so that you can profile your debuggable game or application without device modification. Streamline also supports non-debuggable application profiling on a rooted device. To configure Streamline to collect the right data, use the templates to select the most appropriate set of counters for your device.

System profiling

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

Related information

Application profiling on page 11
System profiling on page 22

1.2 Compile your application

Before you can profile your executable with Streamline, you must compile your executable. This topic describes the compiler options to use when you compile your application.

Profiling native code

When compiling with 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 `-marm` is required. Using `--with-mode=thumb` without `-marm` breaks call stack unwinding in Streamline.

Optional compiler options for call stack unwinding

To enable call stack unwinding in Streamline, you need to compile your executable with some additional compiler options:

- For AArch64 applications:
  - Compiling with GCC, use: `-fno-omit-frame-pointer` and `-mno-omit-leaf-frame-pointer`

    
    Note

    Arm recommends using `-mno-omit-leaf-frame-pointer` to prevent samples in leaf functions incorrectly listing their grand-parent function as their parent.

  - Compiling with Clang, use: `-fno-omit-frame-pointer`

    
    Note

    `-mno-omit-leaf-frame-pointer` is not supported on Clang.

- For AArch32 applications, compiling with either GCC or Clang, use: `-fno-omit-frame-pointer`, `-marm`, and `-mapcs-frame`.

  
  Note

  Streamline supports call stack unwinding for code that has been generated by Arm® Compiler 6.

  Streamline does not support call stack unwinding for T32 (Thumb®) code.

Profiling Java or Kotlin code

In addition to profiling native code, Streamline can profile OAT files that Android runtime (ART) generates. The Streamline report for the application shows function names and disassembly in the **Code** view, but not source code.

To enable OAT files to be built with debug symbols, you must run `dex2oat` with the `--no-strip-symbols` option. To run `dex2oat` with the `--no-strip-symbols` option:

1. Run the following command on the device:

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

2. Re-install the APK file
3. To verify the options for `dex2oat` are set correctly, run the command:

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

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

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

Source code annotations

To enable Streamline to provide extra context when profiling your executable, you can add annotations to your source code. Streamline supports two types of annotations:

- User space annotations, for annotating your application
- Kernel annotations, to profile system calls

You can read more information about annotating your code in the Annotate your code chapter of the *Arm Streamline User Guide*.

Related information

readelf

1.3 Set up your host machine

Arm® Streamline Performance Analyzer is available for the Arm Performance 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 device that is correctly configured to generate performance data. You can use most consumer Android devices without modification.
- If you are building your own device software, ensure that your kernel configuration includes the options that are described in Kernel configuration menu options .
- Ensure you have installed Python 3.8, or later. Python is used to run the `streamline_me.py` script, which configures and collects data for headless captures on Android devices.
- Ensure you have installed Android Debug Bridge (adb). The adb utility is available with the Android SDK platform tools, which are installed as part of Android Studio. Alternatively, you can download the latest version of adb from the Android SDK platform tools site.


Note

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

Procedure

1. Download the studio package appropriate to your host platform (Windows, Linux, or macOS):
   - Download Arm Performance Studio from Arm Performance Studio Downloads
   - Download Arm Development Studio from Arm Development Studio Downloads
2. Install your studio package:
   - To install Arm Performance Studio, use the instructions in the Arm Streamline User Guide
   - To install Arm Development Studio, use the instructions in the Arm Development Studio Getting Started Guide
3. Add the path to the Android SDK and Python executables to your `PATH` environment variable.

Next steps

- Set up your device
- Profile your application

1.4 Set up your device

To use Streamline, set up an Android 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 device 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 device, and you can run `adb shell`.
3. To profile a debuggable application, build and install the application with the appropriate settings:
   - 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.

Note

This step is not required when building non-debuggable applications, for profiling on a development device with root access.

Next steps
Profile your application

Related information
Device connection issues

1.5 Launch Streamline

Learn how to open Streamline using different operating systems.

Before you begin
Before you can open and use Streamline, ensure you have followed the steps in Set up your host machine and Set up your device.

Procedure
Launch Streamline:

- On Windows:
  - Arm® Performance Studio users: From the **Start** menu, search for **Streamline** and select the **Streamline** shortcut.
  - Arm Development Studio users: Launch Arm Development Studio, navigate to the **Streamline data** view, and click **Launch Streamline**.

    Note

    If required, to open the **Streamline data** view, either:
    - Search for 'Streamline data' in the Arm Development Studio search function, and then select the **Streamline data** view.
    - Select **Window > Show view > Other...**, expand **Streamline**, select **Streamline Data**, and click **Open**.
- On macOS, go to the `<install_directory>/streamline` folder, and double-click the `Streamline.app` file.
- On Linux:
  - Arm Performance Studio users: Navigate to the `<install_directory>/streamline` folder and run the `Streamline` file:

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

- Arm Development Studio users: Launch Arm Development Studio, navigate to the **Streamline data** view, and click **Launch Streamline**.


Note

If required, to open the **Streamline data** view, either:

- Search for 'Streamline data' in the Arm Development Studio search function, and then select the **Streamline data** view.
- Select **Window > Show view > Other...**, expand **Streamline**, select **Streamline Data**, and click **Open**.

Next steps

- Connect Streamline to your device

2. Application profiling

Profile your debuggable application while it is running on an Android device. You can also profile non-debuggable applications when running on development devices with root access.

2.1 Profile your application

Use the Streamline GUI to capture a profile of your application.

2.1.1 Connect Streamline to your device

Use the **Start** view to connect Streamline to an Android device and application that you want to collect data from.

Before you begin

Before you can connect Streamline to a device, ensure you have followed the steps in Set up your device.

Procedure

1. Launch Streamline.
2. In the **Start** view, select **Android (adb)** as your 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 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.

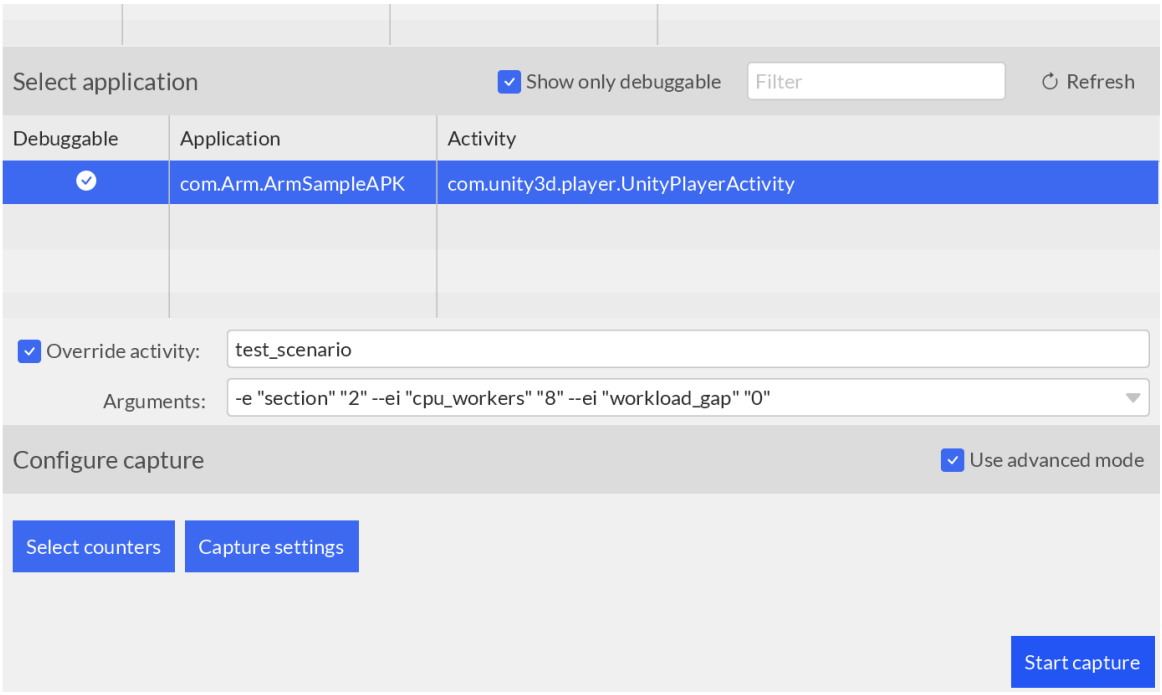

Note

- To enable the Start capture button when you select a non-debuggable package for profiling, you must be running a development device with root access.
- Applications must have at least one MAIN or launchable activity.

4. Optionally, enter arguments for activities that require additional configuration, or for activities that cannot be listed in the table:
   a) Select the **Override activity** check box, and enter a suitable name for the activity.
   b) In the **Arguments** text box, enter the arguments that you want to run on the selected package.
   You can enter multiple arguments passed on a single line. See valid argument options in https://developer.android.com/tools/adb#IntentSpec.

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

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



5. Select the counter template that you want to use to review performance of your CPU and GPU:
   - To use the most appropriate counter template for your Arm GPU, select the **Capture Arm GPU profile** checkbox.
   - To use the default counter template, which captures basic information such as CPU and memory usage, but no GPU data, clear the **Capture Arm GPU profile** checkbox.

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

   - To use a different counter template, see Choose a counter template.

Next steps

For more information about how to capture a profile, see Capture a profile.

2.1.2 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. The counter template you choose overrides the default counter template for the GPU in your device.

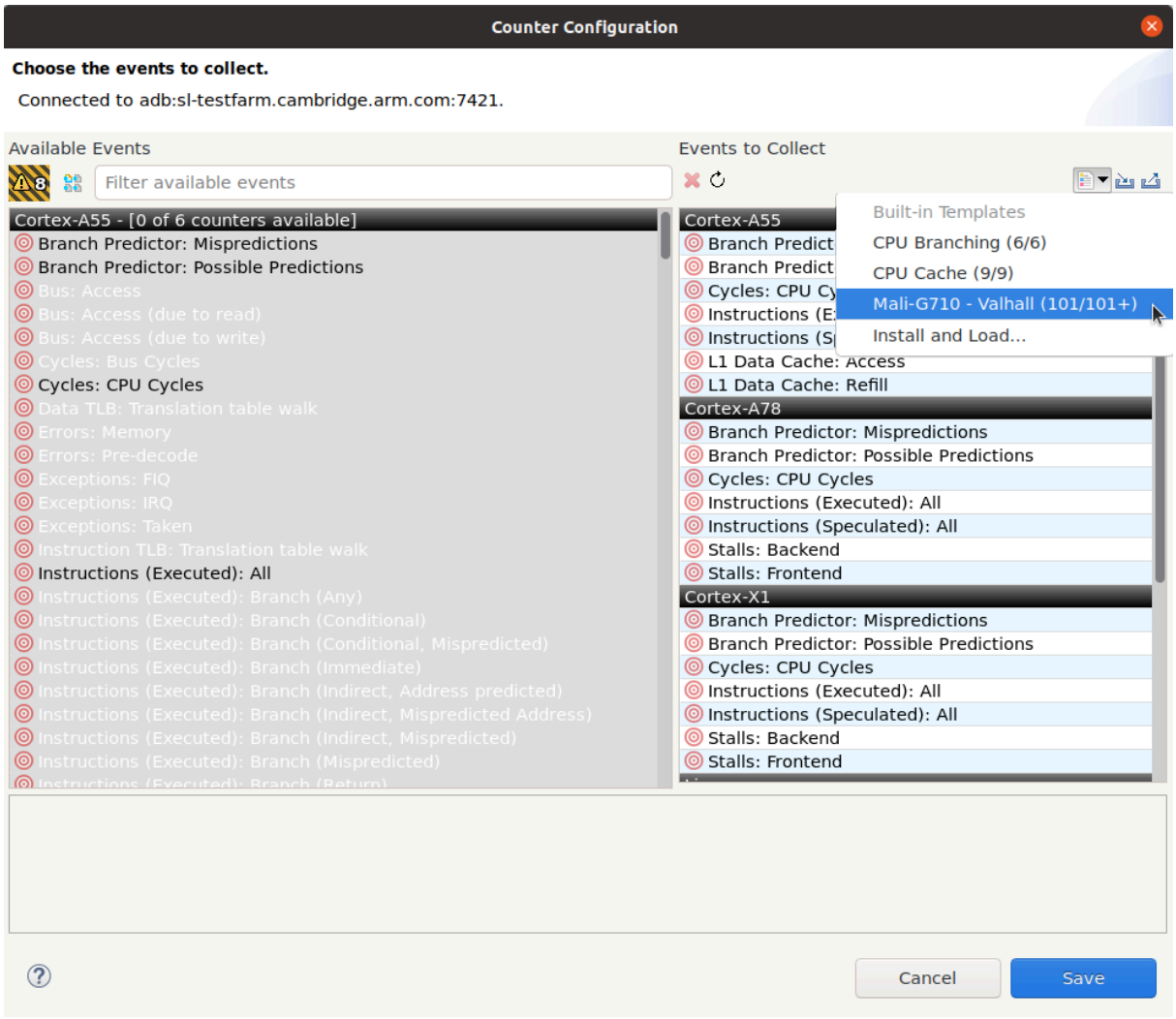
Before you begin

Follow the instructions in Connect Streamline to your device before you choose your counter template.

Procedure

1. In the **Start** view, select the **Use advanced mode** checkbox.
2. Click **Select counters**.
3. Click **Add counters from a template** to see a list of available templates.

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



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

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

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

5. Optionally, in the **Start** view, click **Capture Settings** to change settings related to your capture session, such as the sample rate and the capture duration (by default unlimited). See 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.

2.1.3 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 device
- Connect Streamline to your device

Procedure

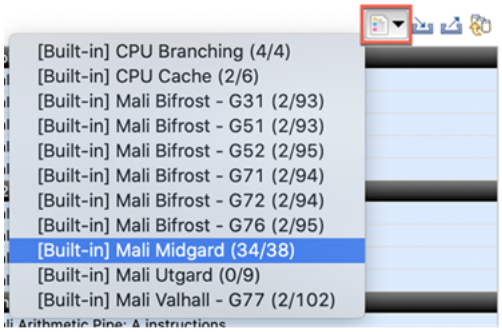
1. In the **Start** view, click **Start capture** to start capturing data from your device.
   Specify the name and location on the host of the capture file that Streamline creates when the capture is complete. Streamline then switches to the **Live** view and starts the application automatically.

   The **Live** view shows charts for each counter that you selected. Below the charts is a list of running processes in your application with their CPU usage. The charts now start updating in real time to show the data that Streamline captures from your running application.

Figure 2-3: Live view shows charts capturing data from your running application.

2. 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.
3. Click **Switch and manage templates** and select the same counter configuration template that you chose to create the capture.

   Figure 2-4: Choose a counter template appropriate to the target GPU in your device.



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*.

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

Before you begin

- Install Python 3.8, or later, to run the provided `streamline_me.py` script.
- Add the path to the `Python3` directory to your `PATH` environment variable.
- To export a configuration containing the counters that you want to capture, follow the instructions in Configure counters and export a counter configuration file. Repeat the configuration export for each class of device you need to profile.

Procedure

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

   ```
   python3 streamline_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:

   `--package`

   The Android package name of the application that you want to profile.

   > **Note**
   > To profile a non-debuggable package, you must be running a development device with root access.

   `--daemon`

   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.

   `--config`

   The path to the configuration file that you saved in the **Before you begin** steps.

   `--headless`

   The path to store the saved output file to.

If you specify `--package-activity`, then the script automatically starts the activity when gator is ready to capture data. If you do not specify an activity, then you must start the activity manually in the next step. If you specify an activity, you can also specify command line arguments using `--package-activity` to pass an argument string. You must quote this string carefully to ensure it is correctly handled, for example:

```
# Argument value with no spaces
--package-arguments "--es fileName /sdcard/example/file.txt"

# Argument value with spaces and Bash quoting
--package-arguments "--es fileName \"/sdcard/example/file with spaces.txt\""

# Argument value with spaces and PowerShell quoting
--package-arguments "--es fileName '/sdcard/example/file with spaces.txt'"
```

2. If you did not use the `--package-activity` option you must start the application manually. Wait for the script to prepare the device, and then start the application on the target device. For example:
   ```
   adb shell am start -n <app.package.name>
   ```
3. Wait for the script to download the data from the device, 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

2.3 Profile executable and linkable format binaries on Android

You can profile Executable and Linkable Format (ELF) binaries using the command-line, or the Streamline GUI.

Before you begin

- Ensure your application is compiled with symbols and debug information.
- Ensure your Android device is connected to your host computer.
- Ensure *gatord* and the application binary are located in `/data/local/tmp` on the Android device. To install *gatord*, see Build gatord yourself.

About this task

This task explains how to profile ELF binaries using the command-line. If you want to profile ELF binaries using the Streamline GUI, see Run a command on the target in the *Arm Streamline User Guide*.

Procedure

1. To profile your ELF binaries, use adb to run *gatord* in local capture mode, and instruct it to run the application that you want to profile on the Android device:

```
adb shell
cd /data/local/tmp
./gatord -o capture.apc -A <elf_application>
```


Note

To change the counter configuration, use either the `-c` argument or the `--config-xml` argument. See gatord command-line options.

`gatord` generates a profile on your Android device at `/data/local/tmp/capture.apc`.

2. Exit the shell on the Android device, then move the profile to the Streamline data directory:

```
exit
adb pull /data/local/tmp/capture.apc <streamline-data-directory>
```


Note

The Streamline data directory is configured in **Window > Preferences > Streamline > Data Locations**.

3. Open Streamline on your host computer.
   Your profile is available in the **Streamline Data** tab.
4. Add the ELF binary file before you analyze a profile of your application in Streamline.
   a) Open Streamline.
   b) In the **Streamline Data** tab, right-click the profile that you want to analyze.
   c) Click **Analyze ...**.
   d) In the **Program Images** tab, click **Add image**.
   e) Select the ELF files that contain the debug symbols for the application and libraries.
   f) Click **Analyze**.

   In the **Call Paths** and **Functions** views, the **Processes** and **Functions** are named appropriately.

Next steps

You can now analyze your profile in Streamline. See Analyze your capture in the *Arm Streamline User Guide*.

2.4 Add debug symbols from Unity

To help you to identify expensive parts of your application, add debug symbols to show function names in the **Functions** and **Call Paths** views in Streamline. This task explains how to add debug symbols from Unity.

Before you begin

1. Ensure your application is debuggable.
2. Ensure your Unity project supports Android. See https://docs.unity3d.com/Manual/android-sdksetup.html.
3. Set your scripting backend to `IL2CPP`, which converts the scripts into a format that is supported in Streamline.

Procedure

1. In Unity, open the **Build Settings** window for your Unity project, click **File > Build Settings**.
2. In **Platform**, select **Android**.
3. Set **Create symbols.zip** to **Debugging**.
4. Select the **Development Build** check box.
5. Compile the application in either **Debug** mode or **Release** mode:
   In **Debug** mode:
   - The Android APK contains the required debug information for Streamline.
   - If the application is built for ARM64 only, you can attach the Android APK in your Streamline capture.
   - Alternatively, you can attach the `.zip` file that contains the debug symbols from Unity to your Streamline capture.

   In **Release** mode:
   - Set **Create symbols.zip** to **Debugging**.
   - Attach the contents of the `.zip` file to your Streamline capture.
6. Select any other options that are relevant to your build.
7. Select **Build**.

Results

The build process generates the Android APK and a `.zip` file that contains the debug symbols files. Add either the APK or the contents of the debug symbols `.zip` file to Streamline.

Next steps

You can now add the debug symbols file when you analyze a capture of your application in Streamline. See Select images and libraries for profiling in the *Arm Streamline User Guide*.

2.5 Add debug symbols from Android Studio

To help you to identify expensive parts of your application, add debug symbols to show function names in the **Functions** and **Call Paths** views in Streamline. This task explains how to add debug symbols from Android Studio.

Before you begin

Ensure your application is debuggable.

Procedure

1. Set the active build variant in Android Studio, click **Build Variants**, then choose **Debug** to include debug symbols in the APK.
2. Click **Build**. Android Studio generates build artifacts in the `project-name/module-name/build/outputs/apk/debug` directory.
3. Unzip the `app-debug.apk` file.
4. Locate the `.so` files, which contain the debug symbols.
5. Add the relevant files for your target architecture. For example, if your target architecture is arm64-v8a and the unzipped APK contains:

   ```
   lib/arm64-v8a/libdemo.so
   lib/x86/libdemo.so
   lib/x86_64/libdemo.so
   lib/armeabi-v7a/libdemo.so
   ```

   instruct Streamline to use `lib/arm64-v8a/libdemo.so`

Next steps

You can now add the debug symbols file when you analyze a capture of your application in Streamline. See Select images and libraries for profiling in the *Arm Streamline User Guide*.

2.6 Add debug symbols from Unreal Engine

To help you to identify expensive parts of your application, add debug symbols to show function names in the **Functions** and **Call Paths** views in Streamline. This task explains how to add debug symbols from Unreal Engine.

Before you begin

Ensure your application is debuggable.

Procedure

1. In Unreal Engine, ensure your build configuration state keyword is `Debug` or `DebugGame`.
2. Open the **Library** tab, then locate the card for the relevant engine version.
3. Click the drop-down arrow next to the **Launch** button on the card, then select **Options**.

4. Select **Editor symbols for debugging**, then click **Apply**.

Results

The debug symbols are generated when you compile the build.


Note

Adding debug symbols to your Unreal Engine build can significantly increase the file size of your application.

Next steps

You can now add the debug symbols file when you analyze a capture of your application in Streamline. See Select images and libraries for profiling in the *Arm Streamline User Guide*.

3. System profiling

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

3.1 Profile your system

Set up and run Streamline with Android device root user access.

Before you begin

- Set up your host machine
- Set up your device

Procedure

1. In a command terminal, run `gatord` as root:

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

2. Launch Streamline.
3. Open the **Start** view, and select **TCP** as your device type.
4. Select your device by entering the address or by using `adb <serial-number>`. Alternatively, select your device from the list of detected devices.

Next steps

Choose a counter template. For more information about how to find and select a counter template, see Choose a counter template.

3.2 Enable atrace annotations

Streamline can capture Android trace points that atrace generates. Application-generated atrace macros are converted into either string annotations or counter charts. Select Android `ATRACE_TAG_*` macros as events for your capture.

Before you begin

- To collect atrace, your Android device must be rooted and use a Linux kernel version of 3.10 or later.
- You must have set up and run Streamline with Android device root user access. See Profile your system for instructions on how to do this.
- Streamline uses a Dalvik executable file called `notify.dex` to alert applications that atrace annotations are enabled. If you use the **TCP** device type when capturing from the **Start** view, you must first manually install `notify.dex`. Locate the file in `<install_directory>/streamline/`

`bin/android/arm64` and copy it onto your device into the same directory as `gatord`. Use the same executable file for both 64-bit and 32-bit architectures. If required, you can find the java source code for `notify.dex` in two locations:

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

Procedure

1. In the **Start** view, select your device and application that you want to capture.
2. In the **Configure capture** section, select **Use advanced mode**.
3. Click **Select counters**.

You can also enable atrace counters in the equivalent **Select Counters** dialog for a **TCP** device capture. You must have installed `notify.dex` to enable atrace, as described in **Before you begin**.

4. Drag **Android: Atrace** into **Events to collect**. You can find **Android Atrace** in **Available events > Atrace control**. Capture events from the **Atrace** profile do not work without this event.

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. Ensure your device is compatible and configured as described in the **Before you begin** section of this topic.

5. Locate the **Atrace** profile in **Available events > Atrace**. Drag each event that you want to collect into **Events to collect**.
6. Click **Save**.

Next steps

Start the capture. In the Streamline GUI, click **Start capture**.

In your capture, string annotations or counter charts are created from your atrace macros. You can learn more about annotations and where they appear in the **Timeline** and **Log** views in the Annotate your code section of the the Streamline User Guide.

Related information

Annotate your code

4. Advanced device setup information

This appendix provides extra configuration information beyond the standard setup.

4.1 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:


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 Arm® 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. `gatord` 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.

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.

4.2 Build gatord yourself

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

About this task

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.

4.3 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 in all modes

The following gatord arguments are available for all modes:

`-h, --help`

Lists all 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 the counters that Streamline collects during the capture session.

-E, --append-events-xml <events_xml>
Specify the path and filename of the `events.xml` file to append.

-P, --pmus-xml <pmu_xml>
Specify the path and filename of the `pmu.xml` 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 `gatord`. 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, `no` is only applicable when `--allow-command` is specified. In this mode, you must enter a command in the **Start** view.

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 <none|low|normal|high>
Specify the sample rate for the capture. The frequencies for each sample rate are:

- `high` = 10kHz
- `normal` = 1kHz
- `low` = 100Hz
- `none` = the lowest possible rate

-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 `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 the capture when the launched application exits. Defaults to `no`, unless `--app`, `--pid`, or `--wait-process` 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.

-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.

-O, --disable-cpu-onlining <yes|no>

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 `no`.

-F, --spe-sample-rate <n>

Specify the SPE periodic sampling rate. The rate, `<n>`, 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.

-L, --capture-log

Enable to generate a log file for the capture in the capture's directory, as well as sending the logs to `stderr`. If you are using the streamline_me.py script, gator is launched with `--capture-log` enabled by default.

--smmuv3-model <model_id>|<iidr>

Specify the SMMUv3 model. You can specify the model ID string directly, such as `mmu-600`, or the hex value representation for the model's IIDR number either fully, such as `4832243b`, or partially, such as `483_43b`.

-Y, --off-cpu-time <yes|no>

Collect Off-CPU time statistics. Detailed statistics require `root` permission.

-I, --inherit <yes|no|poll>

When profiling an application, gatord monitors all subsequent threads and child processes. Specify `no` to monitor only the initial thread of the application. Specify `poll` to periodically look for new processes or threads.


Note

Per-function metrics are only supported in system-wide mode, or when `--inherit` is set to `no` or `poll`. The default is `yes`.

-N, --num-pmu-counters <n>

Override the number of programmable PMU counters that are available.

This option reduces the number of programmable PMU counters available for profiling. Use this option when the default is incorrect, or because some programmable counters are unavailable because they are consumed by the OS, or other processes, or by a hypervisor.

The Arm PMU typically exposes 6 programmable counters, and one fixed function cycle counter. This argument assumes the fixed cycle counter is not part of the reduced set of counters. If your target exposes 2 programmable counters and the fixed cycle counter, then `2` is passed for the value of `<n>`. However, if your target exposes 2 programmable counters and no fixed cycle counter, then `1` is passed for the value of `<n>`.

Arguments for Android devices

The following arguments are available on Android devices only:

l, --android-pkg <pkg>

Profiles the specified android package. Waits for the package app to launch before starting a capture unless `--android-activity` is specified.

m, --android-activity <activity>

Launch the specified activity of a package and profile its process. You must also specify `--android-pkg`.

n, --activity-args <arguments>

Launch the specified activity of a package and profile its process with the supplied activity manager (am) arguments. You must also specify `--android-pkg` and `--android-activity`.

Arguments available in daemon mode

The following arguments are available in daemon mode only:

-p, --port <port_number>|uds

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 usually prevented from creating a TCP server socket.

Alternatively, you can connect to `localhost:<local_port>` in Streamline using:

```
adb forward tcp:<local_port> localabstract:streamline-data
```

-a, --allow-command

Allows you to run a command on the device during profiling. The command is specified in the **Start** view.


Caution

If you use this option, an unauthenticated user could run arbitrary commands on the device using Streamline.

Arguments available in local capture mode

The following arguments are available in local capture mode only:

`-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>`

Specify the path and filename of the output directory for a local capture. The directory path will be appended with the extension `.apc` if it is not already the case.

`-i, --pid <pids...>`

A comma-separated list of process IDs to profile.

`-C, --counters <counters>`

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:

```
--counters <device>_cnt<s>:<e>
```

Where:

- `<device>` is the prefix that identifies the device type.
- `<s>` is the slot number.
- `<e>` is the event code.

`-X, --spe <id>[:events=<indexes>][:ops=<types>][:min_latency=<lat>][:inv]`

Enable the 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 is only recorded if all events are present.
- `<types>` is a comma-separated list of operation types to filter the sampling by. If a sample is any of the types in `<types>`, it is recorded. Valid types are `LD` for load, `ST` for store and `B` for branch.
- `<lat>` is the minimum latency. A sample is only recorded if its latency is greater than or equal to this value. The valid range is [0,4096].
- `:inv` include this flag to invert the SPE event filter. This value is ignored if the device does not support SPE 1.2. Default is disabled.

Example: Add support for a new PMU without rebuilding `gatord`

`--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 :file: *pmus.xml*, which is in the `gatord` 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 `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" description="loginuid"/>
    </category>
</events>
```

Example: Configure various counters

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
```

4.4 Connect Streamline to devices using TCP

The Android connection mode, which is accessed from the **Start** view, allows the Streamline daemon, `gatord`, to connect and capture data of installed applications running on Android devices. For users profiling executables running on Android shell, `gatord` runs manually using `adb shell`.

Procedure

1. On your host machine, navigate to the Streamline installation directory, `<install_directory>/streamline/`.
2. Push the suitable `gatord` binary for the Android device in use.
   Your installation directory contains two versions of `gatord` , one version for 32-bit architecture, and one version for 64-bit architecture:

- `<install_directory>/streamline/bin/android/arm/` for 32-bit architectures.
- `<install_directory>/streamline/bin/android/arm64/` for 64-bit architectures.

3. Run `gatord` with the `--allow-command` option.
   For example:

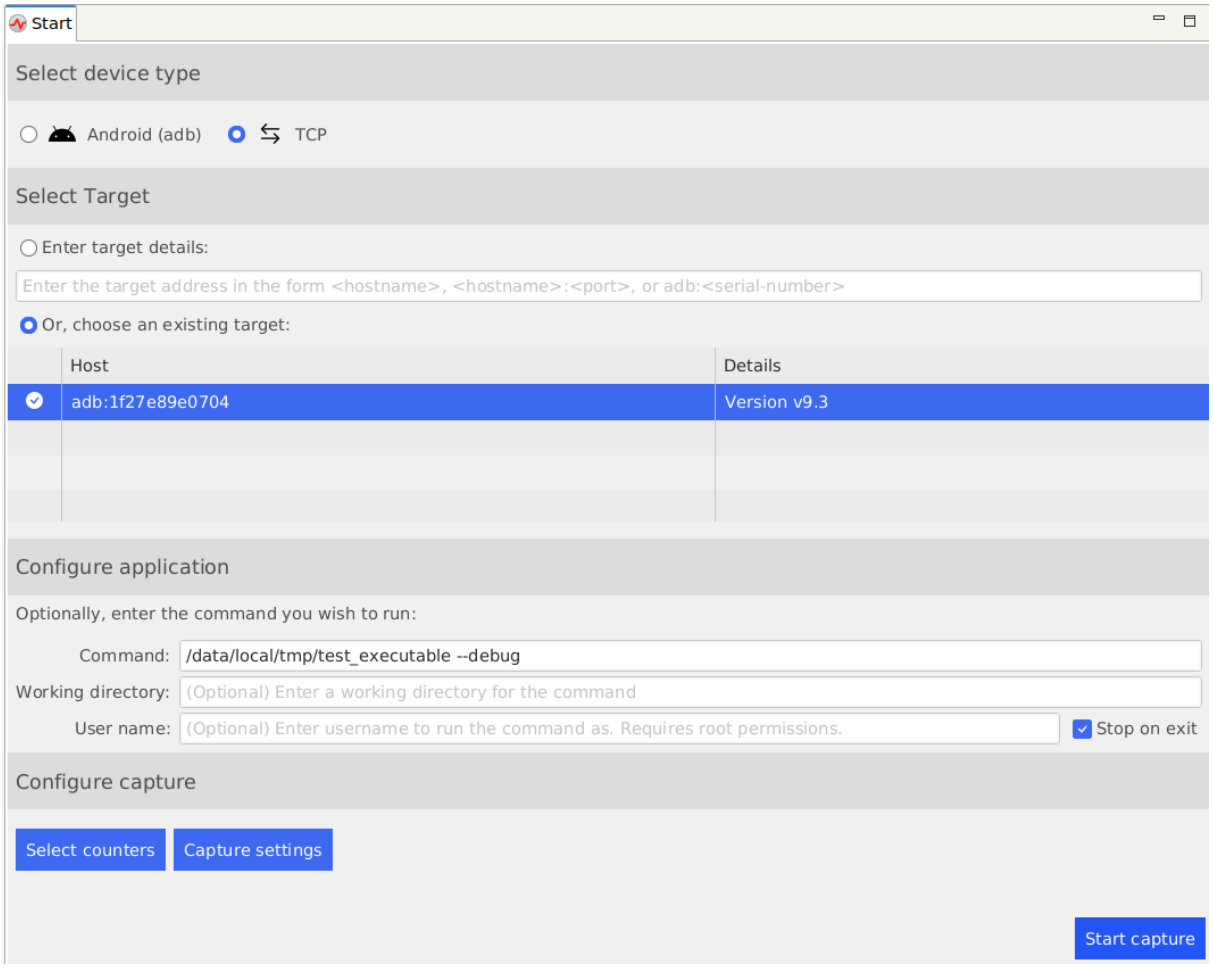
```
adb push bin/android/arm64/gatord /data/local/tmp
adb shell
cd /data/local/tmp
./gatord --allow-command
```

4. Launch Streamline:
   - On Windows, go to the **Start** menu, search for **Streamline**, and then select the **Streamline** shortcut.
   - On macOS, go to the `<install_directory>/streamline` folder, and double-click the `Streamline.app` file.
   - On Linux, navigate to the `<install_directory>/streamline` folder and run the `Streamline` file:

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

5. Open **Start > Select device type** , and choose **TCP**.
6. To select your target, enter the TCP address or the `adb <serial-number>`. Alternatively, select your target from the list of detected targets.
7. Enter the details for any command you want to run on the application.

Figure 4-1: Image from TCP view with example command



Next steps

Choose a counter template. For more information about how to find and select a counter template, see Choose a counter template.

Proprietary Notice

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

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

The content of this document is informational only. Any solutions presented herein are subject to changing conditions, information, scope, and data. This document was produced using reasonable efforts based on information available as of the date of issue of this document. The scope of information in this document may exceed that which Arm is required to provide, and such additional information is merely intended to further assist the recipient and does not represent Arm's view of the scope of its obligations. You acknowledge and agree that you possess the necessary expertise in system security and functional safety and that you shall be solely responsible for compliance with all legal, regulatory, safety and security related requirements concerning your products, notwithstanding any information or support that may be provided by Arm herein. In addition, you are responsible for any applications which are used in conjunction with any Arm technology described in this document, and to minimize risks, adequate design and operating safeguards should be provided for by you.

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

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Reference by Arm to any third party's products or services within this document is not an express or implied approval or endorsement of the use thereof.

This document consists solely of commercial items. You shall be responsible for ensuring that any permitted use, duplication, or disclosure of this document complies fully with any relevant

export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

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

The validity, construction and performance of this notice shall be governed by English Law.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

PRE-1121-V1.0

Product and document information

Read the information in these sections to understand the release status of the product and documentation, and the conventions used in Arm documents.

Product status

All products and services provided by Arm require deliverables to be prepared and made available at different levels of completeness. The information in this document indicates the appropriate level of completeness for the associated deliverables.

Product completeness status

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

Revision history

These sections can help you understand how the document has changed over time.

Document release information

The Document history table gives the issue number and the released date for each released issue of this document.

Document history

Issue	Date	Confidentiality	Change
0905-01	20 March 2025	Non-Confidential	Updated document for v9.5
0905-00	6 February 2025	Non-Confidential	New document for v9.5
0904-00	28 November 2024	Non-Confidential	New document for v9.4
0903-01	17 October 2024	Non-Confidential	Updated document for v9.3
0903-00	5 September 2024	Non-Confidential	New document for v9.3
0902-00	7 June 2024	Non-Confidential	New document for v9.2
0901-00	12 April 2024	Non-Confidential	New document for v9.1
0900-00	15 February 2024	Non-Confidential	New document for v9.0
0809-00	23 November 2023	Non-Confidential	New document for v8.9
0808-00	28 September 2023	Non-Confidential	New document for v8.8

Issue	Date	Confidentiality	Change
0807-00	3 August 2023	Non-Confidential	New document for v8.7
0806-00	8 June 2023	Non-Confidential	New document for v8.6
0805-00	20 April 2023	Non-Confidential	New document for v8.5
0804-00	14 February 2023	Non-Confidential	New document for v8.4
0803-00	17 November 2022	Non-Confidential	New document for v8.3
0802-00	19 August 2022	Non-Confidential	New document for v8.2
0801-00	20 May 2022	Non-Confidential	New document for v8.1
0800-00	18 February 2022	Non-Confidential	New document for v8.0
0709-00	18 November 2021	Non-Confidential	New document for v7.9
0708-00	20 August 2021	Non-Confidential	New document for v7.8

Change history

For information about the functional changes to Arm® Streamline, see the Arm Performance Studio Release Notes.

Conventions

The following subsections describe conventions used in Arm documents.

Glossary

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

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

Typographic conventions

Arm documentation uses typographical conventions to convey specific meaning.

Convention	Use
italic	Citations.
bold	Interface elements, such as menu names. Terms in descriptive lists, where appropriate.
`monospace`	Text that you can enter at the keyboard, such as commands, file and program names, and source code.

Convention	Use
`monospace underline`	A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`<and>`	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Terms that have specific technical meanings as defined in the *Arm® Glossary*. For example, **IMPLEMENTATION DEFINED**, **IMPLEMENTATION SPECIFIC**, **UNKNOWN**, and **UNPREDICTABLE**.

We recommend the following. If you do not follow these recommendations your system might not work.

Your system requires the following. If you do not follow these requirements your system will not work.

You are at risk of causing permanent damage to your system or your equipment, or harming yourself.

This information is important and needs your attention.

A useful tip that might make it easier, better or faster to perform a task.

A reminder of something important that relates to the information you are reading.

Useful resources

This document contains information that is specific to this product. See the following resources for other useful information.

Access to Arm documents depends on their confidentiality:

- Non-Confidential documents are available at developer.arm.com/documentation. Each document link in the following tables goes to the online version of the document.
- Confidential documents are available to licensees only through the product package.

Arm product resources	Document ID	Confidentiality
Arm Streamline User Guide	101816	Non-Confidential
Arm Development Studio Getting Started Guide	101469	Non-Confidential

Non-Arm resources	Document ID	Organization
Debug your app	-	Android Studio
Specification for intent arguments	-	Android Developers
Setup Android support	-	Android environment setup