



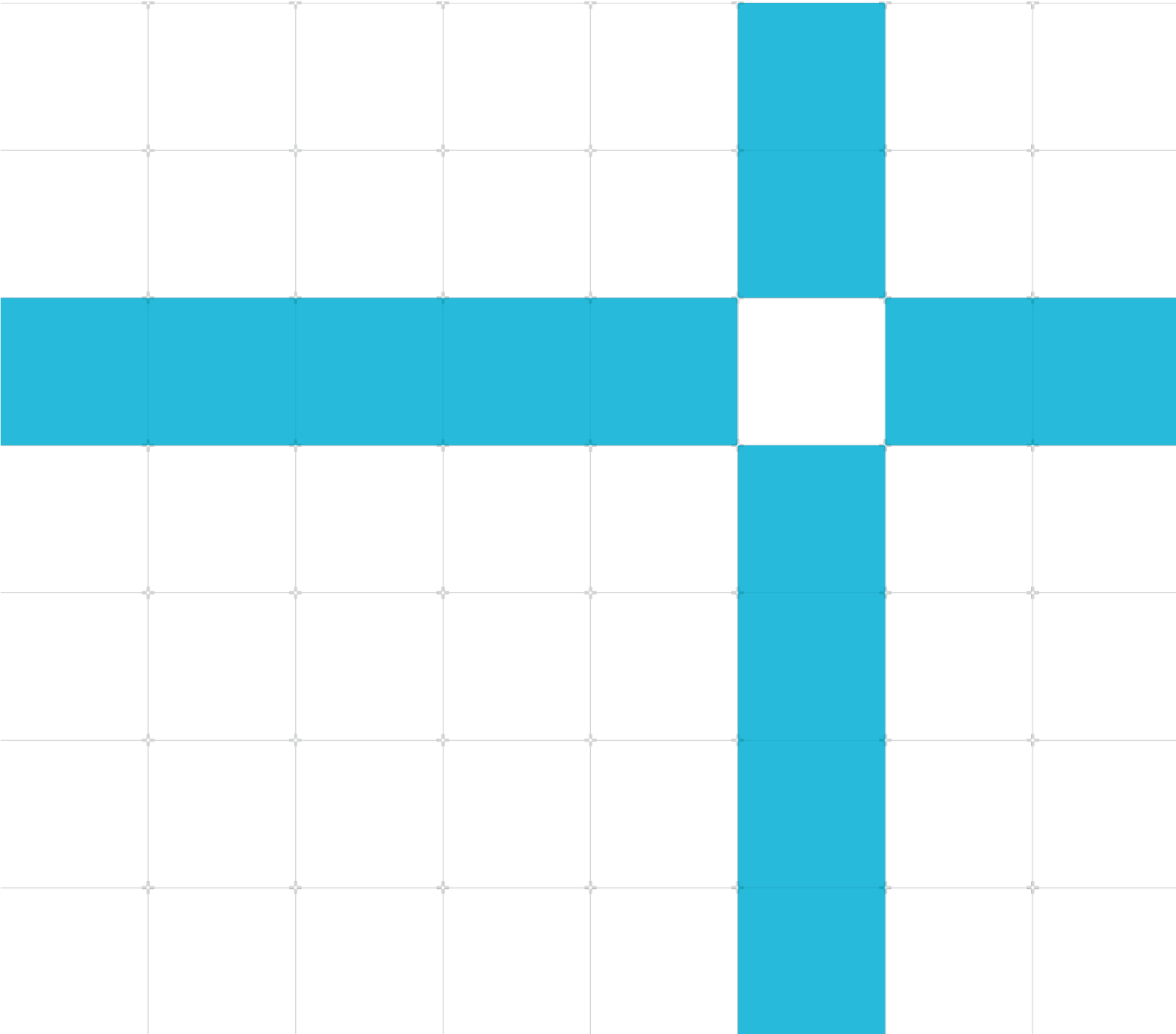
Analyze performance on the Raspberry Pi with Arm Streamline

Non-Confidential

Copyright © 2020 Arm Limited (or its affiliates).
All rights reserved.

Issue 1.0

102110_0001_00



Analyze performance on the Raspberry Pi with Arm Streamline

Copyright © 2020 Arm Limited (or its affiliates). All rights reserved.

Release information Non-Confidential Proprietary Notice

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

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

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

This document may include technical inaccuracies or typographical errors.

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

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

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

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

Copyright © 2020 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.

Document history

| Issue | Date | Confidentiality | Change |
|-------|--------------|------------------|---------------|
| 01 | 4th May 2020 | Non-confidential | First release |

Web Address

www.arm.com

Contents

| | |
|--|-----------|
| 1 Overview | 5 |
| 1.1 Before you begin | 5 |
| 2 Starting Streamline..... | 6 |
| 2.1 Preparing the target | 7 |
| 3 Configuring the Raspberry Pi..... | 8 |
| 4 Connect to a wireless network..... | 9 |
| 5 Installing the Gator daemon..... | 11 |
| 5.1 Installing the gatord executable | 11 |
| 5.2 Building gatord from source code..... | 11 |
| 6 Connecting Streamline | 12 |
| 7 Capturing some profiling data..... | 14 |
| 8 Profiling an example application..... | 15 |
| 9 Source level profiling | 16 |
| 10 Inserting markers..... | 18 |
| 11 Related information..... | 19 |

1 Overview

In this guide, we will explore Linux application and system performance analysis and learn how to find where a system is spending time. Annotating applications and finding performance bottlenecks helps focus software optimization efforts to improve system performance.

The Streamline Performance Analyzer provides system performance metrics, software tracing, and statistical profiling to help engineers get the most performance from hardware and find important bottlenecks in software.

The Raspberry Pi 3 and the Raspberry Pi 4 are low-cost boards with Cortex-A processors. This means that a Raspberry Pi is a useful tool for learning Linux profiling with Streamline. Because Raspberry Pi boards are designed for education, they do not require complex procedures to enable profiling features.

1.1 Before you begin

To work through this guide, you will need access to Streamline, which we will explain in [Starting Streamline](#). You will also need a Raspberry Pi 3 or Raspberry Pi 4 board. An HDMI monitor and USB keyboard and mouse is the easiest way to interact with the Raspberry Pi. Finally, an SD card is needed to hold the Raspbian operating system.

A basic understanding of working with terminals in Linux will also be helpful.

2 Starting Streamline

In this section of the guide, we will install and start Streamline on a Windows or Linux computer.

Streamline is included with Arm Development Studio. A [30-day free trial](#) is available, which you can use to work through this guide. If you already have Arm Development Studio installed, check the [downloads](#) area to make sure that you have the latest version.

Follow these steps to start Arm Development Studio. You can refer to the Getting Started Guide for more information about the requirements and install process.

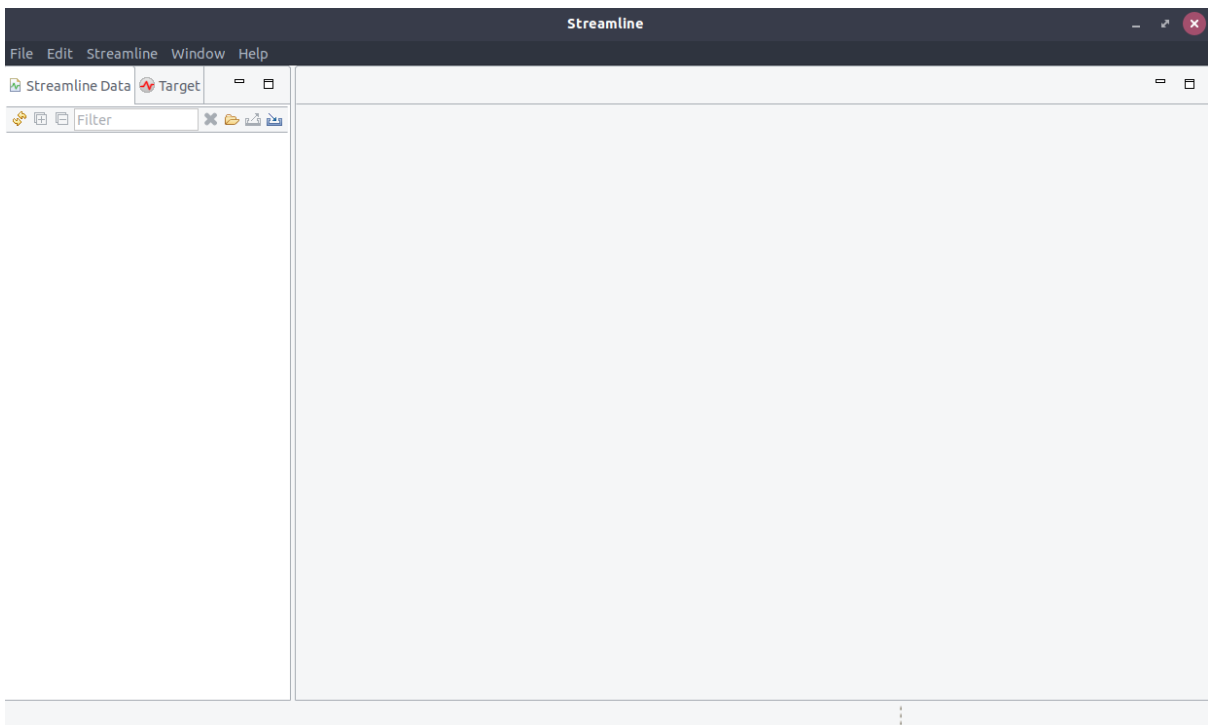
There is only one step required to start Streamline for Linux:

1. Start Streamline for Linux using the Streamline command

```
$ Streamline & .
```

There are two steps required to start Streamline for Windows:

1. Find Streamline for Windows on the start menu. If everything is installed correctly and the license is working, you will see the following screenshot:



2. Install `ssh` and `scp` for Windows if you want to, which enable connection to the Raspberry Pi. Possible options are PuTTY, Windows SSH, and Git Bash SSH.

2.1 Preparing the target

There are two important things to consider when you use Streamline on a new target system:

- Configuring the Linux kernel (if necessary)
- Installing the gator daemon application

The Linux kernel configuration involves ensuring that the profiling features are enabled in the kernel configuration. If the kernel has `perf_event` support, Arm Performance Monitoring Unit (PMU) counters are visible to Streamline. The required kernel support is enabled in the Raspberry Pi kernel, so no special configuration is required.

In the past, a Linux kernel driver facilitated data collection for Streamline, but this has been deprecated. Instead, standard Linux interfaces including `perf` and `trace` are used by Streamline to capture data.

Streamline relies on the Gator daemon application to collect profiling information from a target system. The separation between the data collection and data display makes the daemon approach ideal for embedded systems which have no user interface.

The Gator daemon can be copied from Streamline or compiled from source. Both methods are covered in [Installing the Gator daemon](#). For best results, you must run the Gator daemon as `root` on the target system. This is easy to do with a Raspberry Pi.

[Related information](#) includes a link to more information about target setup for Streamline.

3 Configuring the Raspberry Pi

In this section of the guide, we will configure the Raspberry Pi to run with Streamline.

Follow these steps:

1. Navigate to the [latest version of Raspbian](#) from Raspberry Pi. Follow the instructions to create an SD card, using whatever path is easiest.
2. Use the dd command on Linux, or the [Raspberry Pi Imager on Linux, macOS or Ubuntu](#), to copy the `.img` file to the SD card. The Raspberry Pi Imager is easy to use. We recommend that you use an SD card that is 16GB or larger.
3. Boot the system for the first time. The easiest way to boot the system is to connect an HDMI monitor and keyboard. The default username is `pi` and the default password is `raspberrypi`.
4. Connect the wireless or wired network for Internet connectivity. A wired connection is available by plugging an Ethernet cable into your router. Usually no additional steps are needed.

4 Connect to a wireless network

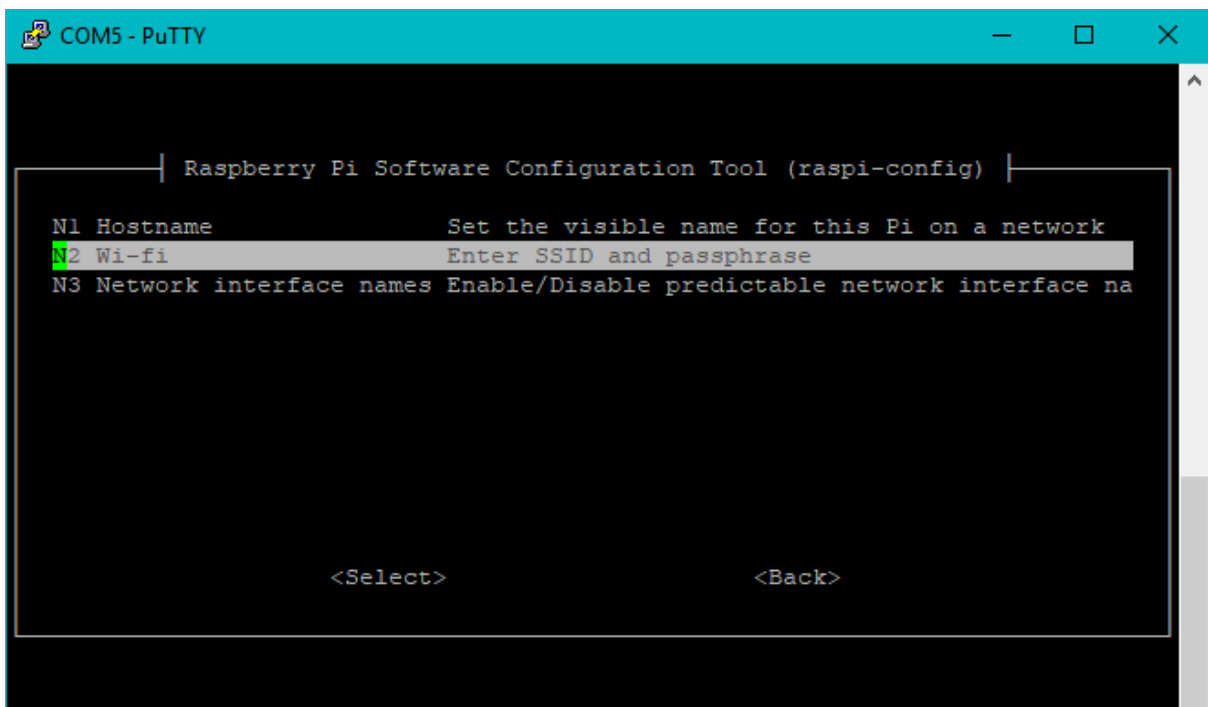
Let's explore how to connect the Raspberry Pi to a wireless network.

Follow these steps to connect to a wireless network:

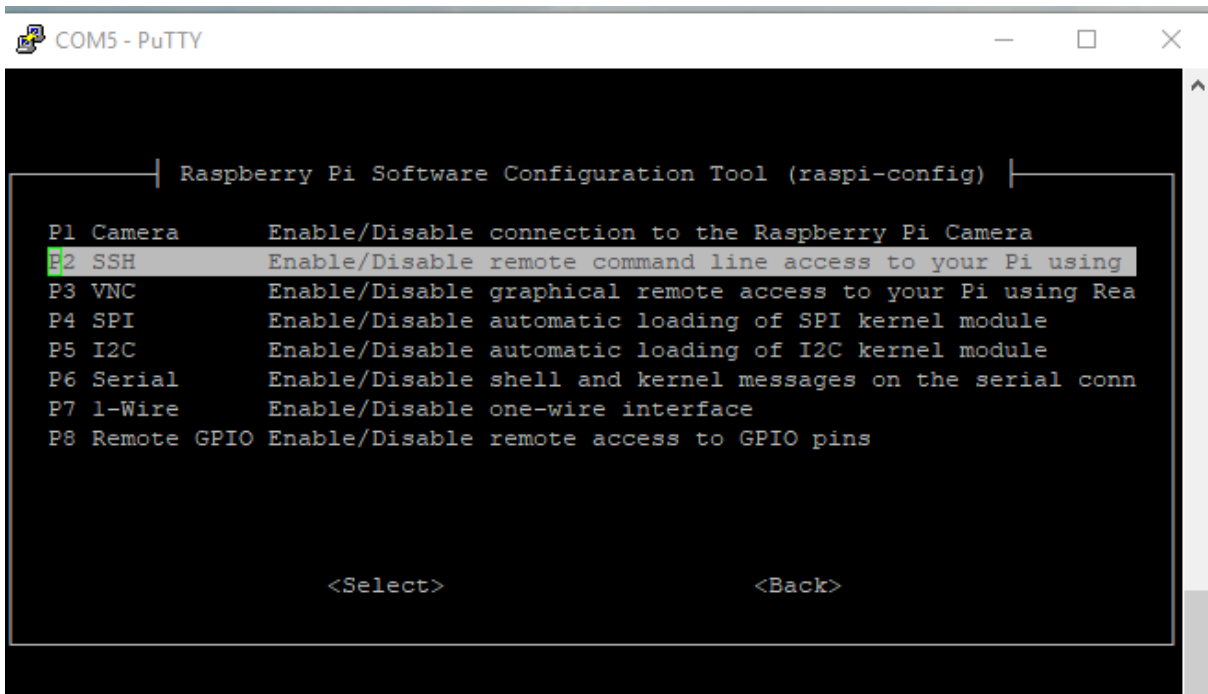
1. Start the console-based Raspberry Pi configuration tool:

```
$ sudo raspi-config.
```

2. Use option 2 to select Network Options, then option 2 again to select Wi-Fi. Enter the SSID and key, if needed, as you can see in this screenshot:



3. Enable `ssh`. Use option 5 **Interfacing Options** from the top menu, then **2 SSH**, as you can see in this screenshot:



If the Raspbian desktop is running, enable the `ssh` server using the configuration **Preferences -> Raspberry Pi**. Click the **Interfaces** tab.

4. Change the password to something other than the default value. The use of `sudo` is automatically enabled for user `pi`. Enter `passwd` on the command line and click **Enter**. You will be prompted to enter the current password and then asked for a new password. The new password will be requested a second time to confirm.

In [Installing the Gator daemon](#), we will set up the gator daemon to enable the connection to Streamline.

5 Installing the Gator daemon

In this section of the guide, we will install the Gator daemon. There are two ways to get the Gator daemon software:

- A pre-compiled executable that is provided with Arm Development Studio. This can be copied to the Raspberry Pi and run.
- An [open-source project on github](#). The source code can be downloaded, compiled on the Raspberry Pi, and run.

Let's look at each in turn.

5.1 Installing the gatord executable

To copy `gatord` from the Arm DS installation, follow these steps:

1. Use the Linux `scp` command and substitute the IP address of your Raspberry Pi.

For example:

```
$ scp $ARMDS_HOME/sw/streamline/bin/arm/gatord pi@192.168.68.121:~/
```

5.2 Building gatord from source code

To download the source code from github and compile it, follow these steps:

1. Download the software, using the following commands:

```
$ git clone https://github.com/ARM-software/gator.git  
$ cd daemon  
$ make
```

2. Start the Gator daemon with root privileges for system profiling. Starting it without root limits the visibility to only those processes that are owned by the user account. The `gator` command is:

```
$ sudo gator &
```

6 Connecting Streamline

To connect Streamline from a Windows or Linux machine, follow these steps:

1. Start the Streamline GUI using the menu or by running Streamline from the command line, as described in Installing Streamline.
2. Click the eyeball on the **Target** tab.
3. Click **Select** if the Raspberry Pi shows up immediately. Click **Setup Target** if the Raspberry Pi does not show up automatically.
4. Enter the IP address of the Raspberry Pi and the username `pi`.

If you enter the IP address of the Pi but Streamline still cannot find it, ensure that Streamline and the Raspberry Pi are on the same network. Only the last number of the IP addresses of the host and target machine should be different. Below is the target setup dialog:

Setup Target [Close]

Setup Target
Install user space gator onto an Android or Linux target.

Android

No Android devices detected

Linux

Address: 192.168.68.121

User Name: pi

User Password: []

Root Password: []

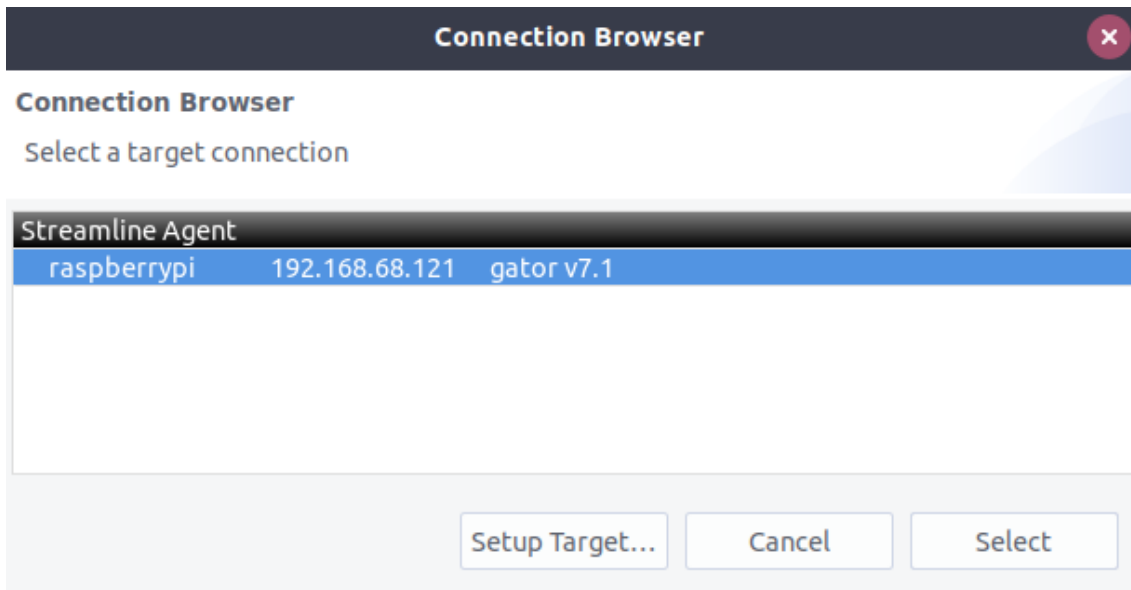
Script Path: /home/jasand01/ArmTools/ArmDS/sw/streamline/gator/setup/gator_setup [Eye]

Cancel Install

5. Click **Install**.

6. Click **No** in the dialog box that is displayed after you click **Install**. This dialog box is confusing, because it asks you to install the `gator` on the target system. You do not need to do this, because the `gator` is already running.

7. Select the target, as in the screenshot below:

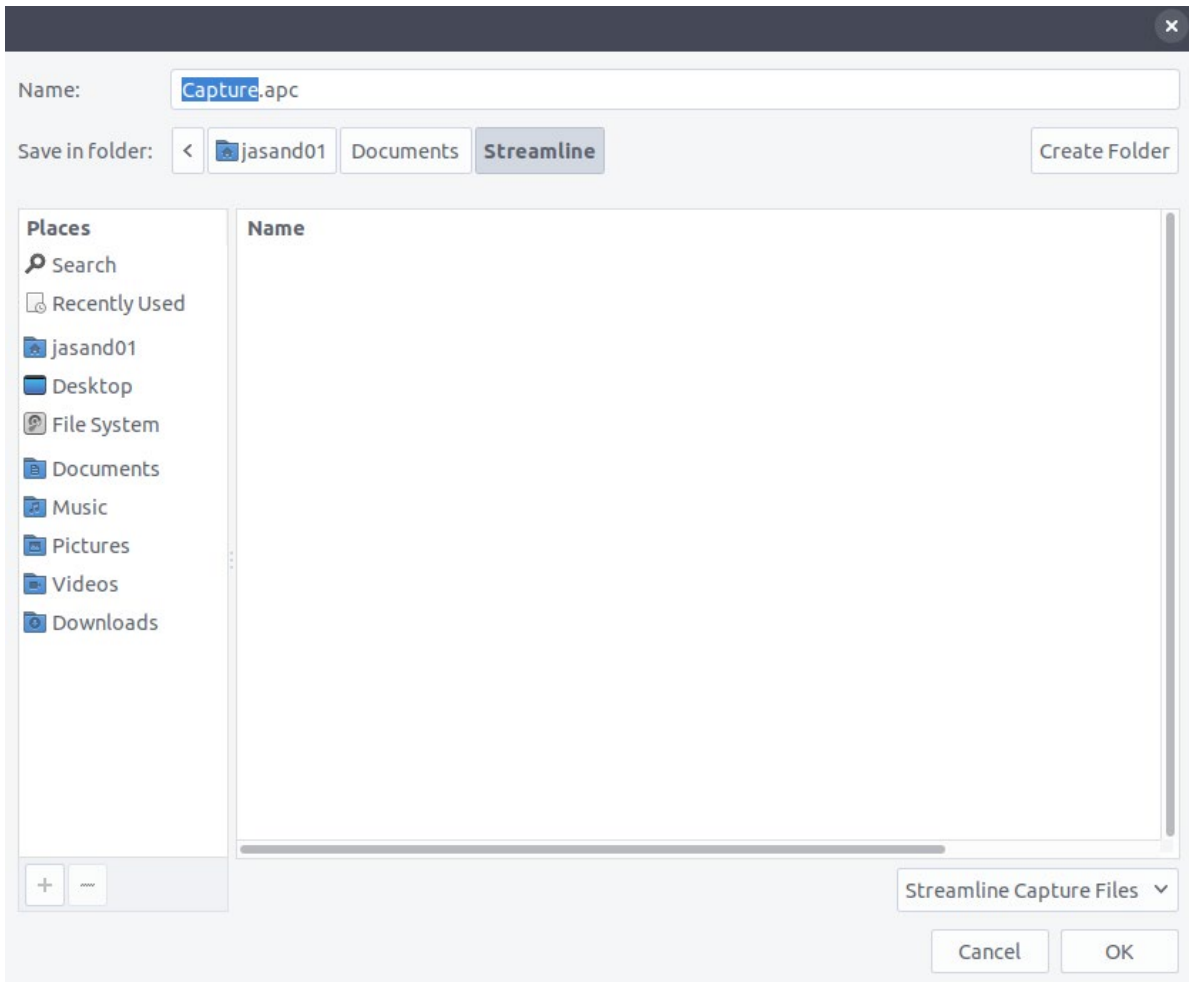


7 Capturing some profiling data

In this section of the guide, we will start a capture session, run a program, and end the capture session. Obtaining a capture confirms that data collection is working.

Follow these steps:

1. Select the name and location of the capture file using the red circle. Click **OK**. You will be promoted for a location and filename to save the capture as shown in this screenshot:



2. Clicking the red stop sign icon ends the capture.

Without any source code or software images, the **Call Paths**, **Functions**, and **Code** tabs do not provide much information. Instead, these tabs show process names and process ID values with some blank screens and **unknown code** messages. This will improve when Streamline has the software images and source. In [Profiling an example application](#), we will download an example application to profile, and learn how to insert markers in the code for Streamline to visualize.

8 Profiling an example application

In this section of the guide, we use the LMBench applications to demonstrate profiling with Streamline. These applications are easy to **download** and build with `-g`. With these applications, Streamline can map the source code of the applications. In **Profiling an example application**, we will download an example application to profile, and learn how to insert markers in the code for Streamline to visualize.

Follow these steps:

1. Download an example Linux application:

```
$ wget  
https://downloads.sourceforge.net/project/lmbench/development/lmbench-3.0-a9/lmbench-  
3.0-a9.tgz  
$ tar xvfz lmbench-3.0-a9.tgz  
$ cd lmbench-3.0-a9/src  
$ make debug
```

This generates executables in the `bin/armv7l-linux-gnu` directory.

2. Run the application, using these commands:

```
$ cd ..  
$ bin/armv7l-linux-gnu/bw_mem 512M rd
```

3. Enable the Streamline capture and run the test.

4. Stop the test to see the results.

9 Source level profiling

To see the application source code in Streamline, the files must be on the host machine where Streamline is running. To do this, follow these steps:

1. Copy the `lmbench-3.0-a9` directory from the Raspberry Pi to the host machine using `scp`. Again, substitute the IP address of the Pi in the following command:

```
$ scp -r pi@192.168.68.121:~/lmbench-3.0-a9 .
```

The **Code** tab in Streamline will not be able to find the source code. This is because the path on the target is different from the path on the host.

2. Use **Click here to locate source** and navigate to the file that is being referenced, in this case `bw_mem.c`. The following screenshot shows the missing source code and how to fix it:

The screenshot shows the Streamline Code window with a search bar and a toolbar. The main area displays a red error message: "The source file is missing!" with a blue link "Click here to locate the file." Below the error message is a disassembly table with the following columns: Self: Samples (#/%), I, Address, Opcode, Disassembly, and File.

| Self: Samples (#/%) | I | Address | Opcode | Disassembly | File |
|---------------------|---|------------|----------|------------------------------|-----------------|
| | | | | rd | |
| | | 0x00011A20 | E92D4070 | PUSH {r4-r6, lr} | bw mem.c:201 |
| | | 0x00011A24 | E1A04000 | MOV r4, r0 | bw mem.c:201 |
| | | 0x00011A28 | E1A06001 | MOV r6, r1 | bw mem.c:201 |
| | | 0x00011A2C | E5915020 | LDR r5, [r1, #0x20] | bw mem.c:202 |
| | | 0x00011A30 | E59F014C | LDR r0, {pc}+0x154 ; 0x11b84 | bw mem.c:204 |
| | | 0x00011A34 | E8000B24 | BL {pc}+0x2c98 ; 0x146cc | bw mem.c:206 |
| | | 0x00011A38 | E3A00000 | MOV r0, #0 | bw mem.c:208 |
| | | 0x00011A3C | E2444001 | SUB r4, r4, #1 | bw mem.c:208 |
| | | 0x00011A40 | E3740001 | CMN r4, #1 | bw mem.c:208 |
| | | 0x00011A44 | 0A00004C | BEQ {pc}+0x138 ; 0x11b7c | bw mem.c:208 |
| | | 0x00011A48 | E5963014 | LDR r3, [r6, #0x14] | bw mem.c:209 |
| | | 0x00011A4C | E1550003 | CMP r5, r3 | bw mem.c:210 |
| | | 0x00011A50 | 3AFFFFF9 | BCC {pc}-0x14 ; 0x11a3c | bw mem.c:210 |
| | | 0x00011A54 | E2832C02 | ADD r2, r3, #0x200 | bw mem.c:210 |
| | | 0x00011A58 | E0451003 | SUB r1, r5, r3 | bw mem.c:210 |
| | | 0x00011A5C | E3C11F7F | BIC r1, r1, #0x1fc | bw mem.c:210 |
| | | 0x00011A60 | E3C11003 | BIC r1, r1, #3 | bw mem.c:210 |
| | | 0x00011A64 | E2833B01 | ADD r3, r3, #0x400 | bw mem.c:210 |
| | | 0x00011A68 | E0811003 | ADD r1, r1, r3 | bw m... 71 More |

When the source code is found, the window looks like what you can see in the following screenshot:

Capture ☰ ☐

Timeline 🔗 Call Paths 🔗 Functions 🔗 Code 🔗 Log

Find 🔍 2,195 (13.13%) 🔗

Self: Samples (#/%) | Line Source File: /home/jasand01/lmbench-3.0-a9/src/bw_mem.c

| | |
|-----|--------|
| 331 | 15.08% |
| 352 | 16.04% |
| 432 | 19.68% |
| 423 | 19.27% |
| 352 | 16.04% |
| 305 | 13.90% |

```

201 {
202     state t *state = (state t *) cookie;
203     register TYPE *lastone = state->lastone;
204     register int sum = 0;
205
206     ANNOTATE MARKER STR("start of rd");
207
208     while (iterations-- > 0) {
209         register TYPE *p = state->buf;
210         while (p <= lastone) {
211             sum +=
212 #define DOIT(i) p[i]+
213 DOIT(0) DOIT(4) DOIT(8) DOIT(12) DOIT(16) DOIT(20) DOIT(24)
214 DOIT(28) DOIT(32) DOIT(36) DOIT(40) DOIT(44) DOIT(48) DOIT(52)
215 DOIT(56) DOIT(60) DOIT(64) DOIT(68) DOIT(72) DOIT(76)
216 DOIT(80) DOIT(84) DOIT(88) DOIT(92) DOIT(96) DOIT(100)
217 DOIT(104) DOIT(108) DOIT(112) DOIT(116) DOIT(120)
218         p[124];
219         p += 128;
220     }

```

| Self: Samples (#/%) | I | Address | Opcode | Disassembly | File |
|---------------------|---|------------|----------|-----------------------------|--------------|
| | | | | rd | |
| | | 0x00011A20 | E92D4070 | PUSH {r4-r6,lr} | bw mem.c:201 |
| | | 0x00011A24 | E1A04000 | MOV r4,r0 | bw mem.c:201 |
| | | 0x00011A28 | E1A06001 | MOV r6,r1 | bw mem.c:201 |
| | | 0x00011A2C | E5915020 | LDR r5,[r1,#0x20] | bw mem.c:202 |
| | | 0x00011A30 | E59F014C | LDR r0,{pc}+0x154 ; 0x11b84 | bw mem.c:204 |
| | | 0x00011A34 | EB000B24 | BL {pc}+0x2c98 ; 0x146cc | bw mem.c:206 |
| | | 0x00011A38 | E3A00000 | MOV r0,#0 | bw mem.c:208 |
| | | 0x00011A3C | E2444001 | SUB r4,r4,#1 | bw mem.c:208 |
| | | 0x00011A40 | E3740001 | CMN r4,#1 | bw mem.c:208 |
| | | 0x00011A44 | 0A00004C | BEQ {pc}+0x138 ; 0x11b7c | bw mem.c:208 |
| | | 0x00011A48 | E5963014 | LDR r3,[r6,#0x14] | bw mem.c:209 |
| | | 0x00011A4C | E1550003 | CMP r5,r3 | bw mem.c:210 |
| | | 0x00011A50 | 3AFFFFF9 | BCC {pc}-0x14 ; 0x11a3c | bw mem.c:210 |
| | | 0x00011A54 | E2832C02 | ADD r2,r3,#0x200 | bw mem.c:210 |
| | | 0x00011A58 | E0451003 | SUB r1,r5,r3 | bw mem.c:210 |
| | | 0x00011A5C | E3C11F7F | BIC r1,r1,#0x1fc | bw mem.c:210 |
| | | 0x00011A60 | E3C11003 | BIC r1,r1,#3 | bw mem.c:210 |
| | | 0x00011A64 | E2833B01 | ADD r3,r3,#0x400 | bw mem.c:210 |
| | | 0x00011A68 | E0811003 | ADD r1,r1,r3 | bw m 70 More |

10 Inserting markers

You can insert markers into the source code of any application, to make it easier to track progress on the Streamline timeline. You can find everything that you need in the `annotate/` directory of the gator software.

To insert a marker, follow these steps:

1. Add the following include file at the top of the source file, to add a marker to the LMBench source file `bw_mem.c`:

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

2. Add `ANNOTATE_SETUP`; during the setup, somewhere in the `main()` function before the test is run.
3. Put the markers where they are needed, to track the test on the Streamline timeline, as seen in the following code:

```
void  
rd(iter_t iterations, void *cookie)  
{  
    state_t *state = (state_t *) cookie;  
    register TYPE *lastone = state->lastone;  
    register int sum = 0;  
    ANNOTATE_MARKER_STR("start of rd");  
    while (iterations-- > 0) {  
        register TYPE *p = state->buf;  
        while (p <= lastone) {  
            sum +=  
#define DOIT(i) p[i]+  
            DOIT(0) DOIT(4) DOIT(8) DOIT(12) DOIT(16) DOIT(20) DOIT(24)  
            DOIT(28) DOIT(32) DOIT(36) DOIT(40) DOIT(44) DOIT(48) DOIT(52)  
            DOIT(56) DOIT(60) DOIT(64) DOIT(68) DOIT(72) DOIT(76)  
            DOIT(80) DOIT(84) DOIT(88) DOIT(92) DOIT(96) DOIT(100)  
            DOIT(104) DOIT(108) DOIT(112) DOIT(116) DOIT(120)  
            p[124];  
            p += 128;  
        }  
    }  
}
```

4. Edit the Makefile to:
 - o Add the directory containing `streamline_annotate.h` to the compiler include path.
 - o Add `streamline_annotate.c` to the list of source files to compile.

11 Related information

Here are some resources related to material in this guide:

- [Arm Development Studio](#)
- [Raspberry Pi](#)
 - [Raspberry Pi documentation](#)
- [Streamline:](#)
 - [Documentation](#) on target setup for Streamline
 - [Streamline Performance Analyzer](#)