Perfetto

Perfetto

Stable: 23.06.2020 - 13:54 / Revision: 02.06.2020 - 15:39

This article provides the basic information required to start using the **perfetto** [1] Android™ tool.

Contents

1 Introduction

The following table provides a brief description of the tool, as well as its availability depending on the software packages:

: this tool is either present (ready to use or to be activated), or can be integrated and activated on the software package.

: this tool is not present and cannot be integrated, or it is present but cannot be activated on the software package.

Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
		perfetto[1] is a performance instrumentation and				*	*	

Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
perfetto	Tracing tools	tracing tool for Android™. • open-source project for tracing	⊗	⊗	⊗			

Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
		• Trace processing and analysis • Web-based trace viewe				** Not available for Android 9.0.0 as only protobuf binary format is supported for configuration files. It's required to have a full baseline*		

Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
		v e r U l						

2 Getting started with perfetto

From Android 10.0.0, Perfetto instructions can be generated using the recording tool[2].

- Create record settings required
- Copy generated instructions
- Execute the copied instructions on a opened terminal with the device connected through USB, having ADB installed.

At the end of the trace execution, you can get back the trace:

```
adb pull /data/misc/perfetto-traces/trace
```

From this stage you can directly open the trace through the web viewer[3].

3 For Android 9.0.0 distribution

3.1 Installing perfetto trace and debug tool on your target board

3.1.1 Using the STM32MPU Embedded Software distribution for Android™

perfetto is installed by default on all STM32MPU Embedded Software Packages for Android. However **there is currently a restriction for using it with Starter and Developer Packages, so it is currently only supported on the Distribution Package**:

- With Android-P(9.0.0), perfetto version does not support the *<--txt>* option that provides a trace configuration file in protobuf text format, so a Distribution Package is required to get protobuf compiler (protoc or aprotoc), which enables building protobuf binary file from protobuf text string file.

perfetto is integrated in Android image distribution through Android base makefile: *build/make/target/product/base.mk*:

```
# Base modules (will move elsewhere, previously user tagged)
PRODUCT_PACKAGES += \
    20-dns.conf \
    95-configured \
...
    netd \
    perfetto \
    ping \
    ping6 \
```

3.2 Getting started with perfetto

To execute perfetto, it is mandatory to have root access rights on the board target and be allowed to create trace files.

Proceed as follows:

```
PC $> adb root
restarting adbd as root
```

Then start the required daemon and create the output trace directory (mandatory to avoid SELinux denials)

```
PC $> adb shell

Board $> start traced
Board $> start traced_probes
Board $> mkdir /data/misc/perfetto-traces
```

3.2.1 Using perfetto

To use the perfetto current version integrated in STM32MPU Embedded Software distribution for Android™, proceed as follows:

```
Board $> perfetto --help
Usage: perfetto
  --background     -b     : Exits immediately and continues tracing in background
  --config         -c     : /path/to/trace/config/file or - for stdin
  --out            -o     : /path/to/out/trace/file
  --dropbox        -d TAG : Upload trace into DropBox using tag TAG (default: perfetto)
  --no-guardrails  -n     : Ignore guardrails triggered when using --dropbox (for
testing).
  --help           -h

statsd-specific flags:
  --alert-id           : ID of the alert that triggered this trace.
  --config-id          : ID of the triggering config.
  --config-uid         : UID of app which registered the config.
```

3.2.2 Using basic test commands

It is possible to check that perfetto is working on the board by using the default test configuration:

```
PC $> adb shell perfetto --config :test --out /data/misc/perfetto-traces/trace
 perfetto_cmd.cc:255      Connected to the Perfetto traced service, starting tracing
for 2000 ms
 perfetto_cmd.cc:322      Wrote 40350 bytes into /data/misc/perfetto-traces/trace
```

Then trace viewer can be used to decode and display trace datas.

3.2.3 Using custom trace configurations in proto buffer binary format

Linux® environment with Distribution Package is required to be able to use the protobuf configuration templates available in the baseline

.

The configuration file to be passed in the perfetto command line is given in protobuf binary format[4]. When it is written in text format, it must be built to generate the corresponding binary.

3.2.3.1 Creating custom trace configuration files

You can create a custom trace configuration file in a text editor, or use a shell command line. Below an example:

```
PC $> cat > /tmp/config.txpb <<EOF
# This is a text-encoded protobuf for /protos/perfetto/config/trace_config.proto
duration_ms: 10000

# For long traces, set the following variables. It periodically drains the
# trace buffers into the output file, allowing to save a trace larger than the
# buffer size.
write_into_file: true
file_write_period_ms: 5000

buffers {
  size_kb: 10240
}

data_sources {
  config {
    name: "linux.ftrace"
    target_buffer: 0
    ftrace_config {
      buffer_size_kb: 40 # Kernel ftrace buffer size.
      ftrace_events: "sched_switch"
      ftrace_events: "print"
    }
  }
}

data_sources {
  config {
    name: "linux.process_stats"
    target_buffer: 0
  }
}
EOF
```

Other examples of custom configuration files are available in the perfetto source package: platform/external/perfetto/test/configs

3.2.3.2 Compiling custom trace configuration files

Follow the commands below to compile the custom proto configuration:

```
PC $> aprotoc --encode=perfetto.protos.TraceConfig \
        -I$(pwd)/external/perfetto/protos \
        $(pwd)/external/perfetto/protos/perfetto/config/perfetto_config.proto \
        < /tmp/config.txpb \
        > /tmp/config.pb
```

Note: perfetto_config.proto is the protbug template file entry.

The output is a protobuf binary format file, /tmp/config.pb.

3.2.3.3 Executing perfetto commands and getting a trace

- Use adb link if it is connected:

```
PC $> cat /tmp/config.pb | adb shell perfetto -c - -o /data/misc/perfetto-traces/trace.
pb
```

```
PC $> adb shell cat /data/misc/perfetto-traces/trace.pb > /tmp/trace.pb
```

Then */tmp/trace.pb* can be open by using the Trace viewer.

3.2.4 Using custom trace configurations in proto buffer text format

Not supported for current perfetto version

3.2.5 Trace viewer

Open the saved trace file through the web UI[6].

3.3 To go further

More information and documentation can be found on the perfetto web site[7].

For example, trace files can be converted in specific readable format.

Additional information are available within the perfetto source package[8].

4 References

- 1.01.1 https://perfetto.dev/
- https://ui.perfetto.dev/#!/record
- https://ui.perfetto.dev

- https://developers.google.com/protocol-buffers/
- https://ui.perfetto.dev/#!/record
- https://ui.perfetto.dev
- https://perfetto.dev/#/?id=perfetto-performance-instrumentation-and-tracing
- platform/external/perfetto/docs

User Interface