Caliper Developer Manual	Public
1.0	Total 10 pages

Caliper [Open source benchmarking framework] Developer Manual

Shyju PV (shyju.pv@gmail.com), Pinkesh shah (pinkesh.shh@gmail.com)

Table of Contents

1 Introduction

The user should go trhough the caliper user manual to understand the caliper flow and execute the integrated tools.
This document would further explain about how new tools to be added to the current framework and modifications to be done in various modules of caliper.

To add a new tool to Caliper one should define how to build, execute and parse the output of that tool. The configuration files should be updated to specify how the target, server, client to be configured and tools to be executed. One should be familiar with the directory structure of caliper before adding any tools.

2 Caliper folder organisation

Caliper folder structure is as follows.

benchmarks: This directory contains the actual source code for each tool.

client: The parser files for each tools are placed under client/parser directory.

config: This folder contains **client_config.cfg file**. This file is used to configure the host, target and client/server information.

server: This directory contains the scripts for dispatching the build, run and parse on the Host and remote login in the Target.

> **build:** This directory mainly contains the build.sh and build.py which are responsible for building of tools.
>
> **compute_model:** This directory includes the files to get the score from the parser output; the method of scoring is mainly in the scores_method.py.
>
> **parser_process:** This one contains the source code to process the yaml file for scoring and normalisations.
>
> **run:** This directory mainly includes the test_run.py, it is the main code to run the commands for execution of tools and parsing the output of each tools.
>
> **hosts:** This directory mainly contains the class of hosts and how to use hosts.

test_cases_cfg: This contains the config files to select the tools and test cases to be executed.

frontend: This folder contains all the files necessary for the report generation.

Frontend/polls: This folder contains the following files:

> **plot:** This folder contains the files for generating the graphs.
>
> **templates/polls**: This folder contains all template htmls.
>
> **static**: This folder contains all javascripts and css related files.
>
> **views.py:** This file is responsible to get the data from the yamls and jsons.

utils: contains **automation_scripts** that contains all the scripts for dependency checking and automation of caliper execution and **Documentation** folder having excel population scripts.

Note : *Caliper python django framework for report generation.*

3 Add Tool to benchmark directory

Please find the specific tools folders under the benchmark directory. You need to add your tool source code similarly in this folder.

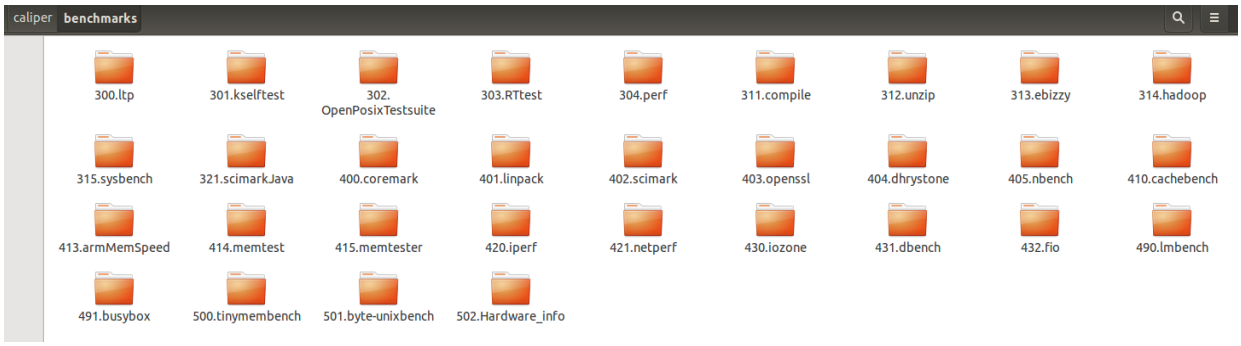

Illustration 1: Caliper benchmark directory

4 Test case Definition structure

The directory named test_cases_cfg is the key of how to build, run and parse the tool. The content of it is listed below.

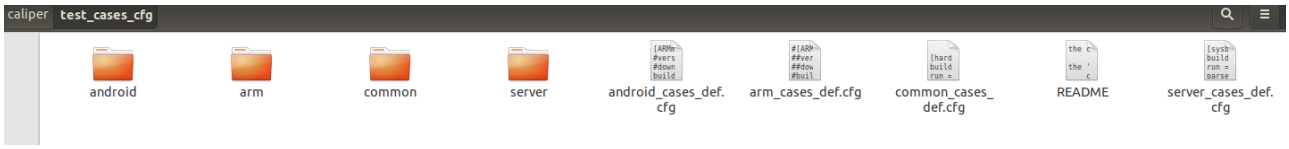

Illustration 2: Test_cases_cfg directory structure

The contents of **common** directory are given below. Same follows for the **server** directory.

Depending on the tool execution requirements, tool files (tool_build.sh and tool_run.cfg) should be placed in server or common directory.

Each tool has the tool_build.sh and tool_run.cfg defined in the directory named as tool_name inside **calliper/test_cases_cfg/XXXX/** directory. (XXXX can be common / server / application)

Tool_run.cfg will be in following format:

```
[lmbench lat]
category = Performance
scores_way =  exp_score_compute 1 -0.5
command = "cd lmbench; ./test.sh latency "
parser = lmbench_lat_parser

#[lmbench bandwidth]
#category = Performance
#scores_way = compute_speed_score 3
#command = "cd lmbench; ./test.sh bandwidth "
#parser = lmbench_bandwidth_parser
```

Illustration 3: Tool_run.cfg format

Section 6 contains the detailed information of tool_run.cfg file structure.

5 Tool Building

- ***tool*_build.sh** file contains the procudure to build the binary image of the tool for x86_64 and arm_64 target

platforms.

- Sample script file for scimark tool is as follows:

```
build_scimark() {
    set -e
    SrcPath=${BENCH_PATH}"402.scimark"
    myOBJPATH=${INSTALL_DIR}/bin
    pushd $SrcPath
    if [ $ARCH = "x86_32" -o $ARCH = "x86_64" ]; then
    make CC=$GCC CFLAGS="-msse4"
        cp scimark2 $myOBJPATH/
                make CC=$GCC clean
                    fi
    if [ $ARCH = "arm_32" ]; then
        # -mfloat-abi=hard -mfpu=vfpv4 -mcpu=cortex-a15
        make CC=$GCC CFLAGS=" -mfloat-abi=hard -mfpu=vfpv4 -mcpu=cortex-a15 "
        cp scimark2 $myOBJPATH/
        make CC=$GCC clean
    fi
    if [ $ARCH = "arm_64" ]; then
        make CC=$GCC      #CFLAGS="-mabi=lp64"
        cp scimark2 $myOBJPATH
        make CC=$GCC clean
    fi

    if [ $ARCH = "android" ]; then
        ndk-build
        cp libs/armeabi-v7a/scimark2 $myOBJPATH/
        ndk-build clean
        rm -rf libs/ obj/
    fi
    popd
}
```

Illustration 4: Tool_build.sh format

6 Tool Execution

Tool execution procedure has been described in **tool_name_run.cfg** file.
The content of the configuration file is like this:

[*test_case_name*]
category = category scenario sub_scenario test_case_name
scores_way = compute_speed_score 1
command = ./bin/*tool_name*
parser = *tool_name*_parser

- The **category** key defines the categorization of the tool in the resultant yaml. Category key can be **functional** or **performance**. Scenario can be network, application, memory, etc. Sub-scenario can be specific to the test case category.

- The **scores_way** defines the way to compute the score for the parsed values.
  Two scoring methods are available for computing the score.

  For **Higher The Better** values use
  scores_way = compute_speed_score index
  where the score is calculated as: score_way = value / (10^index)

  For **Lower The Better** values use
  scores_way = exp_score_compute *base index*
  Where the score is calculated as: *score_way = (value/(10^base))^ index*

  Example:
  Bandwidth values can be categorized in **Higher The Better** scenario.
  Latency values can be categorized in **Lower The Better** scenario.

  Note:
  1. For redis tool, 2 score way method has been used. Because the single test case of redis contains both

bandwidth and latency.

2. For redis tool, "**scores_way**" is for latency and "**scores_way1**" is for bandwidth.
3. If user wants to add such type of tool, which contain bandwidth and latency in single tets case, please refer redis tool configuration file and parser file.

- The **command** is the actual test case which will run on target.
- The **parser** key defines the method that will parse the output of the command executed above. Refer section 7 for more details.

The category should contain maximum 4 levels depending on the value returned by parser method.
Based on the type of value returned by the parser the length of category varies.

- If the returned value is a dictionary of depth 3 only **category** should be mentioned.
- If the returned value is a dictionary of depth 2 **category [scenario]** should be mentioned.
- If the returned value is a dictionary of depth 1 **category [scenario] [sub_scenario]** should be mentioned.
- If the returned value is of int, float or any other primitive data type then **category [scenario] [sub_scenario] [test_cases]** should be mentioned.

Example - lmbench_run.cfg:

[lmbench bandwidth stream v1]
category = Performance misc lmbench_bandwidth_stream_v1
scores_way = compute_speed_score 3
command = "if [-f lmbench_tar.gz]; then tar -xvf lmbench_tar.gz; rm lmbench_tar.gz; fi;cd lmbench; numactl --cpunodebind=0 --localalloc ./stream -v 1 -M 200M -P 16"
parser = lmbench_bandwidth_stream_v1

- Here, **category** contains 3 levels - Performance, misc and lmbench_bandwidth_stream_v1.
- The parser method **lmbench_bandwidth_stream_v1** returns the value in dictionary format like,

```
dic['copy'] = {}
dic['scale'] = {}
dic['add'] = {}
dic['triad'] = {}
```

7 Tool Parsing

The output of the execution needs to be parsed for the relevant information from the output log of the tool execution. Parser file of each tool is present in **client/parser** directory. The parser method name defined in tool_name_parser.py should be same as defined in "**parser**" line in the tool_name_run.cfg file.

There are following sections in the tool_name_parser.py file:

- Import python modules.
- Parser method definition.

Below example shows the parser logic for **stressng** tool:

```
1. import re
2. def stress_ng_parser(content,outfp):
3.     result = 0
4.     if re.search(r'stress-ng: info:\s+\[[0-9]+\]\scpu\s+\d+\s+(\d+\.\d+).*',content):
5.         real_time = re.search(r'stress-ng: info:\s+\[[0-     9]+\]\scpu\s+\d+\s+(\d+\.\d+).*',content)
6.         result =  real_time.group(1)
7.     outfp.write(content)
```

8. return result

- The method must have two arguments: First argument contains the **content** of output log and the second argument is the **file pointer** which points to the parser log file.
- Here line number 4 to 6 contains the parsing logic – re.search(). Depending on the requirements to parse the output log, user can write this logic.
- In addition the parser must return a value or dictionary which is needed later for raw yaml generation and score computing.

8 Tool Definition

According to the categorization of tool under common / server or application category, update common_cases_def.cfg / server_cases_def.cfg or application_cases_def.cfg file with following information.

```
[tinymembench]
build = tinymembench_build.sh
run = tinymembench_run.cfg
parser = tinymembench_parser.py

[openssl]                 #this is the name of the tool
build = openssl_build.sh  #tool_name_build.sh
run = openssl_run.cfg     #tool_name_run.cfg
parser = openssl_parser.py#tool_name_parser.py
```

Illustration 5: Tool Definition

New tool can be categorized in following way:

1. If the tool will executes on the target platform, then place the tool directory into test_cases_def/common/ directory. Example: openblas, Dhrystone, etc.
2. If the tool execution required client-server configuration and the "**Target**" platform acts as **client** machine, then place the tool directory into **test_cases_def/server** directory. Example: iperf, netperf, etc.
    1. Write the test cases into tool_run.cfg file. These test cases will executes on "**Target**".
    2. Write the test cases into tool_server_run.cfg file. These test cases will executes on "**TestNode**".
3. If the tool execution required client-server configuration and the "**Target**" platform acts as **server** machine, then place the tool directory into **test_cases_def/application** directory. Example: nginx, redis etc.
    1. Write the test cases into tool_run.cfg file. These test cases will executes on "**Target**".
    2. Write the test cases into tool_application_run.cfg file. These test cases will executes on "**TestNode**".

9 Graph Generation

- Once the execution of caliper is completed the resultant yamls are generated in the **caliper_output/<workspace_name>/output/results/yaml** folder.

  There are 3 files inside this directory: **<platform_hw_info>.yaml**, <p**latform_name>.yaml and** <p**latform_name>_score.yaml.**
- <p**latform_name>_score.yaml** is taken as an input for report generation.
- The values in this yaml file are normalized and the graphs are generated with these values.
- There is an html page for each scenario as **caliper/frontend/polls/templates/polls/<*scenario_name*>.html**.

- Following code should be added to the **html file**. Here we have taken network_bandwidth test scenario from "network" category as an example.

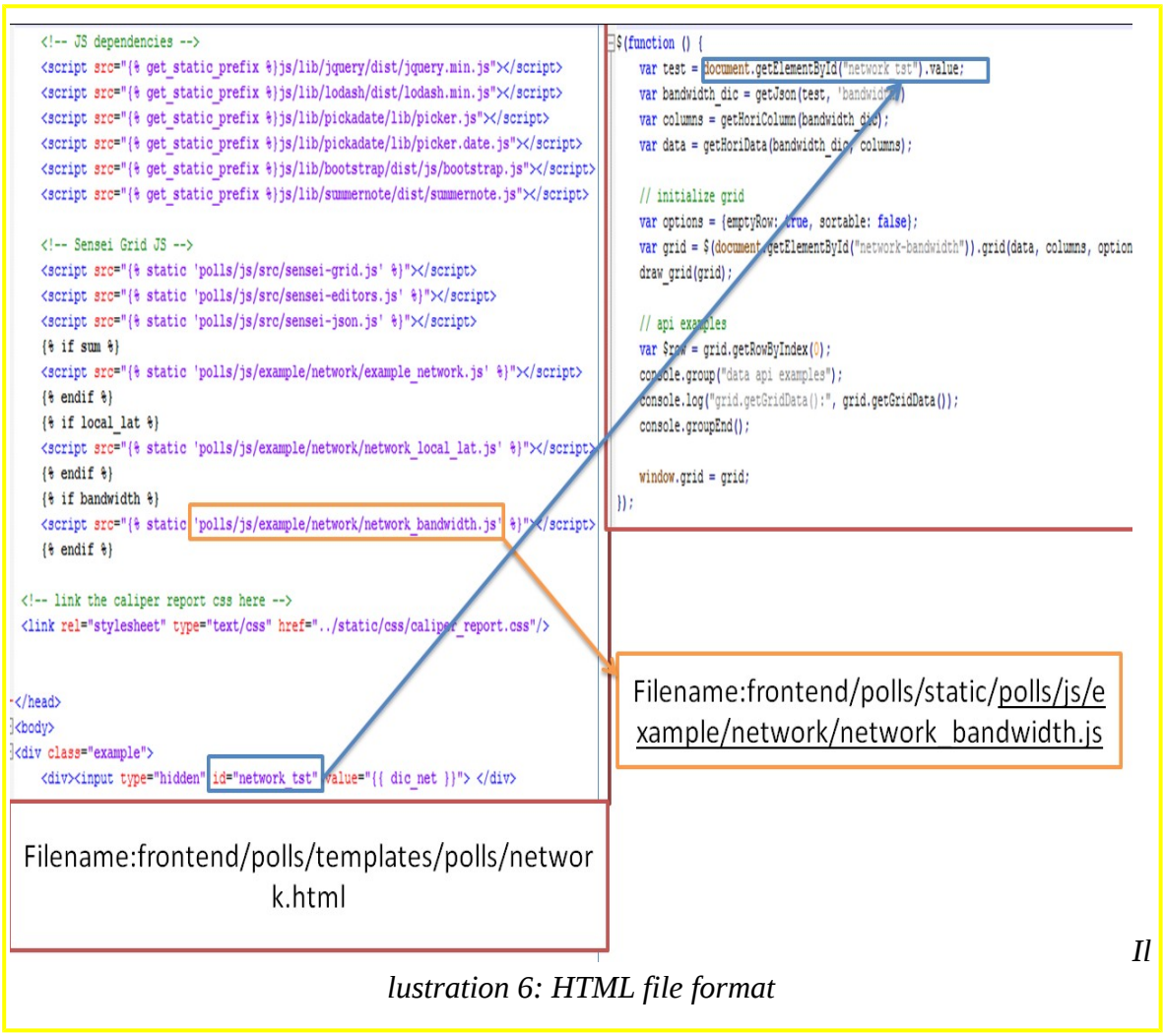

Illustration 6: HTML file format

- network_bandwidth.js filename explanation: "network" keyword will be constant for network.html file. "bandwidth" keyword is from test case category.

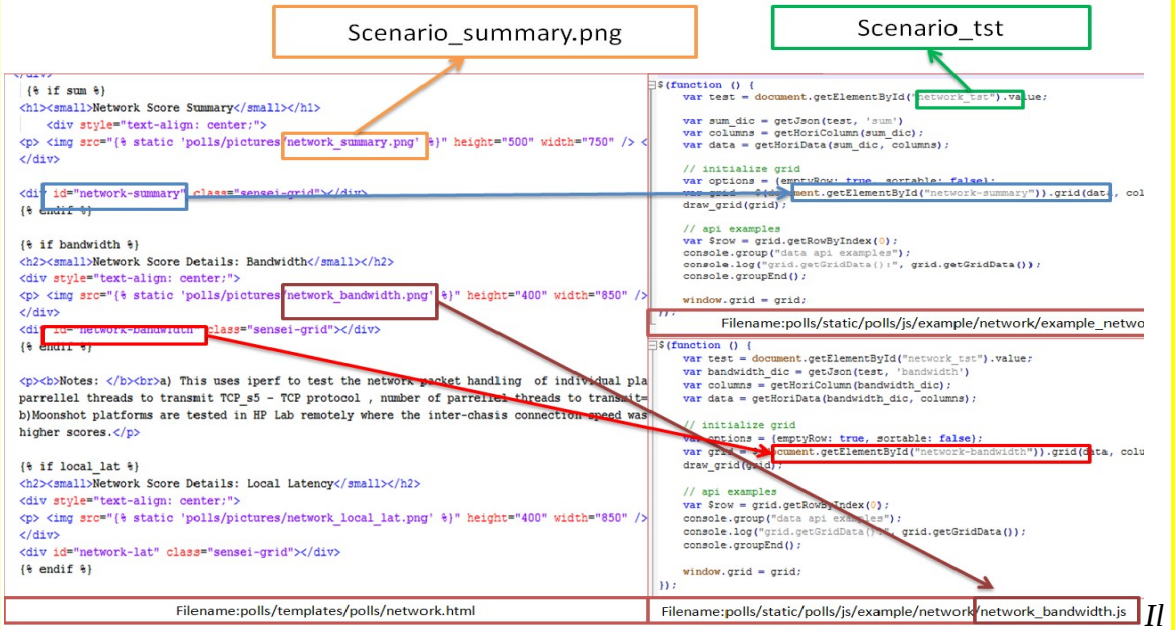

Illustration 7: Relation between HTML and javascript

- Java-script named as ***scenario_testcase.js (eg. network_bandwidth.js),*** needs to be added in the **caliper/frontend/polls/static/polls/js/example/<*scenario_name*>** directory.

The template for the ***.js** file is given below:

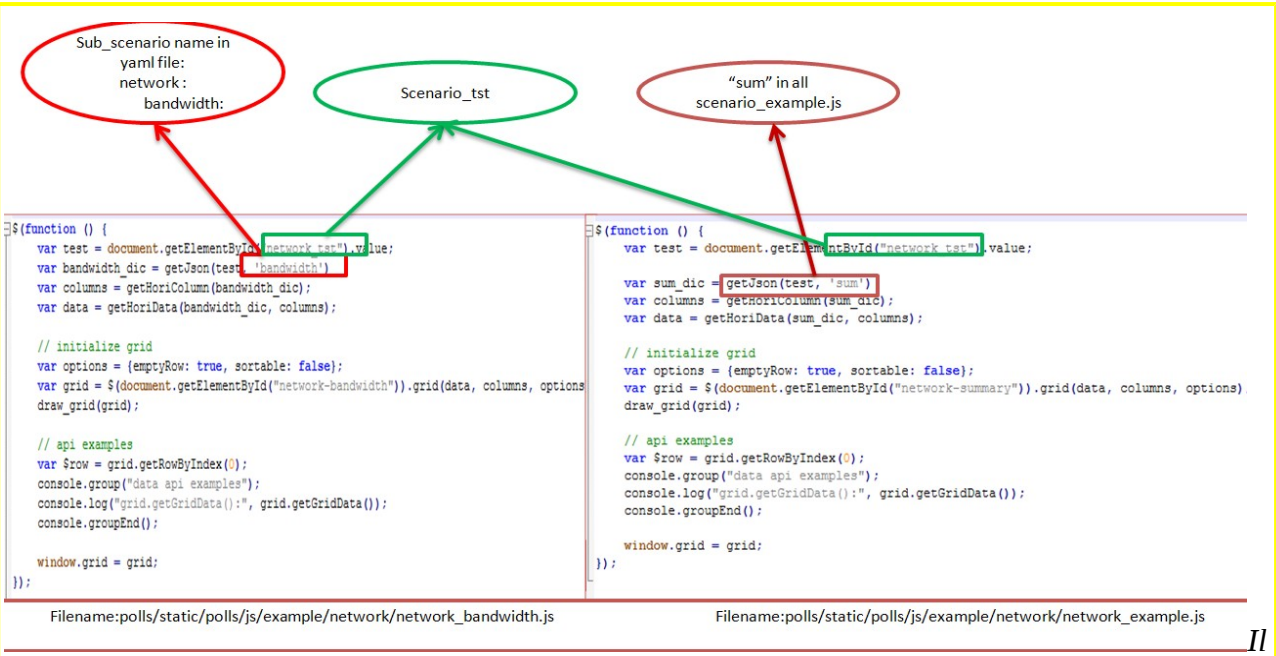

Illustration 8: java script file format

- When you are adding a new tool in an already existing scenario (e.g. application) you need to change the values as specified in images.
- network_tst is replaced by the id defined in **caliper/frontend/polls/templates/polls/<*scenario_name*>.html.**
- Bandwidth name to be replaced with the sub-scenario name which your tool writes in the yaml.

- After updating all the necessary files in frontend directory, copy **caliper/frontend/** directory into **~/caliper_output/** directory.

10 Excel sheet update for new test cases

- When new tool has been added into caliper framework, following excel file needs to update inside **caliper/frontend/polls/templates/polls /** directory:

  Performance_Template.xlsx or **Functional_Template.xlsx**

- Based on tool category, Performance_Template.xlsx or Functional_Template.xlsx can be update.
- After updating the xlsx file, copy it into **~/caliper_output/frontend/polls/templates/polls/** directory.
- For example, iperf and neterf tools test cases to be added in "network" category:

	A	B	C	D	E	F	G	H	I	J	K	L
1	Caliper Performance Tests: Network											
2	SL NO	TOOLS	Scenario	Testcase	Units							
3				TCP RR	MB/s							
4				TCP stream r	MB/s							
5	1	netperf		TCP CRR	MB/s							
6				TCP stream	MB/s							
7				UDP RR	MB/s							
8				TCP s01 RX	Mbits/sec							
9				TCP s01 TX	Mbits/sec							
10				TCP s03 RX	Mbits/sec							
11				TCP s03 TX	Mbits/sec							
12				TCP s04 RX	Mbits/sec							
13				TCP s04 TX	Mbits/sec							
14				TCP s05 RX	Mbits/sec							
15				TCP s05 TX	Mbits/sec							
16	2	iperf		TCP s10 RX	Mbits/sec							
17				TCP s10 TX	Mbits/sec							
18				UDP s01 TX	Mbits/sec							
19				UDP s01 RX	Mbits/sec							
20				UDP s03 RX	Mbits/sec							
21				UDP s03 TX	Mbits/sec							
22			bandwidth	UDP s05 RX	Mbits/sec							
23				UDP s05 TX	Mbits/sec							
24				UDP s10 RX	Mbits/sec							
25				UDP s10 TX	Mbits/sec							
26				tcp bw 000001B	Mbytes/sec							
27				tcp bw 000002B	Mbytes/sec							

Illustration 9: Excel file format

- Above figure shows the screen-shot of **Performance_Template.xlsx** excel sheet for iperf and netperf tools test cases.
- If user wants to merge **TOOLS** name or **Scenario** name as shown in above figure, then make sure that hidden cells should contains the proper contents. Otherwise values will be not populating for that test case.
- For example, in above figure B3 to B7 cell should contain "netperf" tool name.

Note:

1. Update **~/caliper_output/frontend/polls** directory for javascript, html and Performance_Template.xlsx changes manually.
2. After updating all the necessary files, install the caliper with ***sudo python setup.py install*** command.