

# FAQAS Framework

## SUM - Software User Manual

O. Cornejo, F. Pastore

Interdisciplinary Centre for Security, Reliability and Trust

University of Luxembourg

ITT-1-9873-ESA-FAQAS-SSS

Issue 1, Rev. 1

March 22, 2021

---

# Revisions

Issue Number	Date	Authors	Description
ITT-1-9873-ESA-FAQAS-SUM Issue 1 Rev. 1	March 31th, 2021	Oscar Cornejo, Fabrizio Passtore	Initial release.

# Contents

<b>1</b>	<b>Scope and content</b>	<b>5</b>
1.1	Applicable and reference documents . . . . .	5
<b>2</b>	<b>Terms, definitions and abbreviated terms</b>	<b>7</b>
<b>3</b>	<b>External View of the Software</b>	<b>9</b>
<b>4</b>	<b>Operations Manual</b>	<b>11</b>
4.1	Set-up and Initialization . . . . .	11
4.1.1	Dependencies . . . . .	11
4.2	Getting started . . . . .	11
4.2.1	Initialization of the MASS workspace . . . . .	11
4.2.2	MASS Configuration . . . . .	13
4.2.3	Prepare the SUT Script Configuration . . . . .	14
4.2.4	Mutation Script Configuration . . . . .	15
4.2.5	Build Script for Compiler Optimizations . . . . .	15
4.2.6	Running MASS on single machines . . . . .	15
4.2.7	Running MASS on HPC infrastructures . . . . .	17
4.3	Mode selection and control . . . . .	18
4.4	Normal Operations . . . . .	18
4.5	Normal Termination . . . . .	18
4.6	Error Conditions . . . . .	18
4.7	Recover Runs . . . . .	18
<b>5</b>	<b>Tutorial</b>	<b>19</b>

5.1	Introduction . . . . .	19
5.2	Getting Started . . . . .	19
5.3	Using the software on a typical task . . . . .	19
5.3.1	Single Machine Example: Mathematical Library for Flight Software . . . . .	19
5.3.2	HPC Infrastructure Example: Mathematical Library for Flight Software . . . . .	22

# Chapter 1

## Scope and content

This document is the deliverable SUM of the ESA activity ITT-1-9873-ESA. It concerns the software user manual for the *FAQAS framework*. Following the structure described in the SoW *AO9873-ws00pe\_SOW.pdf*, it provides instructions for the users of the FAQAS framework according to ECSS-E-ST-40C Annex B.

### 1.1 Applicable and reference documents

- D1 - Mutation testing survey
- D2 - Study of mutation testing applicability to space software



## Chapter 2

# Terms, definitions and abbreviated terms

- FAQAS: activity ITT-1-9873-ESA
- FAQAS-framework: software system to be released at the end of WP4 of FAQAS
- D2: Deliverable D2 of FAQAS, *Study of mutation testing applicability to space software*
- KLEE: Third party test generation tool, details are provided in D2.
- SUT: Software under test, i.e, the software that should be mutated by means of mutation testing.
- WP: Work package



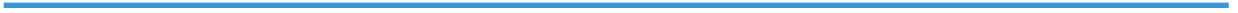


## Chapter 3

# External View of the Software

The FAQAS-framework is delivered as a compressed archive consisting of source files and an installer. The following bulletpoints provide a description of archive structure after it is uncompressed:

- `FAQASFramework/`
  - `SRCMutation/`: contains the source files of the component that performs code-driven mutations.
  - `llvm-build.sh`: build script that compiles the `SRCMutation` component
  - `PythonWrappers/`: contains Python script wrappers that facilitate code-driven mutations.
  - `MASS/`: contains all the executable files and scripts that implement the methodology for code-driven mutation testing supported by the FAQAS-Framework. They are listed below.
    - \* `FAQAS-Setup`: contains the Bash scripts necessary to install the FAQAS-Framework.
    - \* `FAQAS-GenerateCodeCoverageMatrixes`: contains the Bash scripts providing procedures to collect code coverage from the SUT.
    - \* `FAQAS-GenerateMutants`: contains a Bash script that invokes the `SRCMutation` component to generate mutants.
    - \* `FAQAS-CompileOptimizedMutants`: contains the scripts (in Python and Bash) that provide the procedures to compile mutants and filter equivalent and redundant mutants based on trivial compiler optimizations.
    - \* `FAQAS-CompileAndExecuteMutants`
      - `FAQAS-GeneratePrioritizedTestSuite`: contains the Python and Bash scripts that provide the procedures to generate prioritized and reduced test suites from the SUT.
      - `FAQAS-CompileAndExecute`: contains the Python and Bash scripts that provide the procedures to compile and execute the mutants against the SUT test suite. It also provides the procedures to determine the mutation stopping criterion (i.e., mutant sampling).
      - `FAQAS-IdentifyEquivalentAndRedundantMutants`: contains the Python and Bash scripts that provides the procedures to identify equivalent mutants based on code coverage.
    - \* `FAQAS-MutationScore`: contains the Python and Bash scripts that provide the procedures to compute the mutation score and provide summarized information about the code-driven mutation testing process.



## Chapter 4

# Operations Manual

### 4.1 Set-up and Initialization

MASS depends on LLVM for source code mutation. For this reason, the set-up procedure consists of installing LLVM-3.8.1 and the SRCMutation component. For this procedure, a Bash script is provided.

The following shell command installs the corresponding dependencies and the SRCMutation component.

```
1 $ ./llvm-build.sh
```

#### 4.1.1 Dependencies

**TODO:** should we add the LLVM version?

- Linux packages: r-base, jq, Python 3.7 or higher
- R packages: binom
- Python packages: numpy, scipy

### 4.2 Getting started

#### 4.2.1 Initialization of the MASS workspace

MASS shall create a workspace folder where all the steps from the methodology shall be stored.

An installation Bash script is provided for the creation of this workspace, the script can be found on \$FAQAS/MASS/FAQAS-Setup/install.sh

To use the installation script the shell variable INSTALL\_DIR has to be set:

```
1 $ export INSTALL_DIR=/opt/DIRECTORY
```

**TODO:** the meaning of the following is not clear. Add an example.

If the `INSTALL_DIR` directory must be binded inside a container. Then, also the shell variable `EXECUTION_DIR` has to be set. This step is optional.

After setting the corresponding environment variables, the following commands are necessary to create the *MASS* workspace folder:

```
1 $ cd $FAQAS/MASS/FAQAS-Setup
2 $ ./install.sh
```

Once the installation folder has been created, the folder shall contain the following structure and files:

- `Launcher.sh`: *MASS* single launcher; the script executes all the steps of the methodology in one command.

**TODO:** "has to be set" what does it mean?

- `mass_conf.sh`: *MASS* configuration file, has to be set before being able to execute *MASS*.
- `mutation_additional_functions.sh`: Bash script that must be filled by the application engineer before executing *MASS*.
- `MASS_STEPS_LAUNCHERS/`: folder containing all the single launchers for each step of the *MASS* methodology.
  - `MASS_STEPS_LAUNCHERS/PrepareSUT.sh`: launcher for the script that prepares the SUT and collects information about the SUT test suite.
  - `MASS_STEPS_LAUNCHERS/GenerateMutants.sh`: launcher for the generation of mutants.
  - `MASS_STEPS_LAUNCHERS/CompileOptimizedMutants.sh`: launcher for the trivial compiler optimization step.
  - `MASS_STEPS_LAUNCHERS/OptimizedPostProcessing.sh`: launcher for the post-processing of the trivial compiler optimization step.
  - `MASS_STEPS_LAUNCHERS/GeneratePTS.sh`: launcher for the generation of prioritized and reduced test suites.
  - `MASS_STEPS_LAUNCHERS/ExecuteMutants.sh`: launcher for the execution of mutants against the SUT test suite.
  - `MASS_STEPS_LAUNCHERS/IdentifyEquivalents.sh`: launcher for the identification of equivalent mutants based on code coverage.
  - `MASS_STEPS_LAUNCHERS/MutationScore.sh`: launcher for the computation of the mutation score and final reporting.
  - `MASS_STEPS_LAUNCHERS/PrepareMutants_HPC.sh`: launcher that prepares the mutants workspace for the execution on HPCs.
  - `MASS_STEPS_LAUNCHERS/ExecuteMutants_HPC.sh`: launcher that executes mutants on HPCs.
  - `MASS_STEPS_LAUNCHERS/PostMutation_HPC.sh`: launcher that assesses past mutant executions, and decides whether more mutant executions are needed.

## 4.2.2 MASS Configuration

There are three Bash scripts that should be edited by the engineer to configure *MASS*. These three scripts enable *MASS* to correctly identify the SUT paths (e.g., source code folder, test suite folder), the SUT compilation commands, the SUT test suite execution commands, and the configuration of *MASS* itself (e.g., trivial compiler optimizations flags, mutant selection strategy, sampling rate).

**TODO:** We should have a table with the script name, the parameter to be configured and a description

A detailed description of the MASS Configuration File follows.

### 4.2.2.1 MASS Configuration File

The *MASS* configuration file is the Bash file `$INSTALL_DIR/mass_conf.sh`; within this file there are multiple environment variables that must be set; they are shown in Listing ??.

**TODO:** You should always refer to Figures/Listings in appropriate manner. However, this "style" is ok for the running example.

```

1 \label{listing:MASS:conf}
2 # set SRCIROR path
3 export SRCIROR=
4
5 # set workspace directory path where MASS files can be stored (i.e., $INSTALL_DIR)
6 export APP_RUN_DIR=
7
8 # specifies the building system, available options are "Makefile" and "waf"
9 export BUILD_SYSTEM=
10
11 # directory root path of the SUT
12 export PROJ=
13
14 # directory source path of the SUT
15 export PROJ_SRC=
16
17 # directory test path of the SUT
18 export PROJ_TST=
19
20 # directory coverage path of the SUT
21 export PROJ_COV=
22
23 # directory path where the compiled binary is stored
24 export PROJ_BUILD=
25
26 # list of folders not to be included during coverage analysis, name folders shall be separated by
27 # '\|'
28 export COVERAGE_NOT_INCLUDE=
29
30 # filename of the compiled file or library
31 export COMPILED=
32
33 # path to the original build script
34 export ORIGINAL_MAKEFILE=
35
36 # compilation command of the SUT, the command shall be specified as a Bash array, e.g., (). Special
37 # characters shall be escaped.
38 export COMPILATION_CMD=
39
40 # additional commands for compiling the SUT (e.g., setup of workspace), the command shall be
41 # specified as a Bash array, e.g., (). Special characters shall be escaped.
42 export ADDITIONAL_CMD=

```

```

41 # command to be executed after each test case execution (optional), the command shall be specified as
    a Bash array, e.g., (). Special characters shall be escaped.
42 export ADDITIONAL_CMD_AFTER=
43
44 # compilation command for TCE analysis, the command shall be specified as a Bash array, e.g., ().
    Special characters shall be escaped.
45 export TCE_COMPILE_CMD=
46
47 # command to clean installation of the SUT, the command shall be specified as a Bash array, e.g., ().
    Special characters shall be escaped.
48 export CLEAN_CMD=
49
50 # relative path to location of gcov files (i.e., gcda and gcno files)
51 export GC_FILES_RELATIVE_PATH=

```

Furthermore, the following specific *MASS* variables must be set:

```

1 # specify if MASS will be executed on a HPC, possible values are "true" or "false"
  export HPC=
2
3
4 # TCE flags to be tested, the flags shall be specified as a Bash array, e.g., ("-00" "-01").
  export FLAGS=
5
6
7 # set if MASS should be executed with a prioritized and reduced test suite, possible values are "true
  " or "false"
8 export PRIORITIZED=
9
10 # set sampling technique, possible values are "uniform", "stratified", and "fsci"
11 # note: if "uniform" or "stratified" is set, $PRIORITIZED must be "false"
12 export SAMPLING=
13
14 # set sampling rate if whether "uniform" or "stratified" sampling has been selected
15 export RATE=

```

### 4.2.3 Prepare the SUT Script Configuration

**TODO:** The title should be changed, probably it should become something like *Edit Prepare-SUT.sh*. My main problem is that I cannot map this section to something describe before, it comes out of the blue. Also, shouldn't it be a subsubsection? I would expect to have in 4.2.2 the list of all the files to be editing and in the following subsubsections a description for each of them.

To configure *MASS* to work with the SUT, the engineer should edit the Bash file `$INSTALL_DIR/MASS_STEPS`. The following actions shall be performed by the engineer:

1. Provide commands to generate a compilation database file `compile_commands.json` of the SUT. Note that the paths defined within the database file must be full paths;

**TODO:** Not clear why the engineer should do what written above

2. Provide commands to compile the SUT;
3. Provide additional commands to prepare the SUT workspace (optional);
4. Provide commands to execute the SUT test suite iteratively over each test case; more precisely, the engineer should do the following:
  - After the command executing a test case add a call to `$MASS/FAQAS-GenerateCodeCoverageMatrixes/FAQAS-CollectCodeCoverage.sh` script;

- the script `FAQAS-CollectCodeCoverage.sh` shall be invoked with three arguments: (i) the test case name, (ii) the time taken to run the test case in seconds, and (iii) the root folder where all the coverage files are being stored.

#### 4.2.4 Mutation Script Configuration

**TODO:** subsubsection?

The mutation script configuration file is the Bash file `mutation_additional_functions.sh`. In it, the engineer is expected to implement the Bash function `run_tst_case`. This function shall receive as argument the name of the test case to be executed. It should execute the command for running the specified test case. The function shall return 0, if the test case passes; it shall return 1 if the test case fails.

#### 4.2.5 Build Script for Compiler Optimizations

**TODO:** subsubsection?

The SUT engineer shall provide a build script for the SUT. Such script shall be placed in the same folder where the original build script resides (a different name shall be used). The script shall have the following characteristics:

- The script shall not contain any debugging flag within compilation/linking commands;
- The script shall not contain any code coverage flag within compilation/linking commands;
- The script shall contain a placeholder for the compiler optimization option, specifically the placeholder `TCE`;
- The script shall contain a 'sort' command in the source dependency list to ensure that source files are always compiled in the same order;

**TODO:** Not clear what a source dependency list is, example?

- The script shall be named the same as the original build script, but with an ending `'.template'`.

#### 4.2.6 Running MASS on single machines

**TODO:** Add a paragraph that explains that you are going to describe two ways of running MASS

**TODO:** You should either (a) specify the files generated for each step, or (b) put a sentence saying that the output/inputs of each step are described in the specification document XY (and you have to draft it by recycling the old SSS).

#### 4.2.6.1 One Step Launcher

One possible way to run *MASS* is to execute all the eight steps of the framework with one command. Such action will execute the following steps sequentially:

1. PrepareSUT
2. GenerateMutants
3. CompileOptimizedMutants
4. OptimizedPostProcessing
5. GeneratePTS
6. ExecuteMutants
7. IdentifyEquivalents
8. MutationScore

To execute the one step launcher, the following command shall be provided:

```
$ ./Launcher.sh
```

#### 4.2.6.2 Multiple Step Launchers

Another possible way to run *MASS* is to execute all the eight steps of the framework through independent commands. The multiple steps of the methodology and its command are described in the following.

##### 1. Prepare the SUT

```
$ ./MASS_STEPS_LAUNCHER/PrepareSUT.sh
```

##### 2. Generate Mutants

```
$ ./MASS_STEPS_LAUNCHER/GenerateMutants.sh
```

##### 3. Compile Optimized Mutants

```
$ ./MASS_STEPS_LAUNCHER/CompileOptimizedMutants.sh
```

##### 4. Compile Optimized Mutants Post-Processing

```
$ ./MASS_STEPS_LAUNCHER/OptimizedPostProcessing.sh
```

##### 5. Generate Prioritized and Reduced Test Suites

```
$ ./MASS_STEPS_LAUNCHER/GeneratePTS.sh
```

##### 6. Execute mutants

```
$ ./MASS_STEPS_LAUNCHER/ExecuteMutants.sh
```



## 7. Identify Equivalent Mutants based on Code Coverage

```
1 $ ./MASS_STEPS_LAUNCHER/IdentifyEquivalents.sh
```

## 8. Computer Mutation Score

```
1 $ ./MASS_STEPS_LAUNCHER/MutationScore.sh
```

### 4.2.7 Running MASS on HPC infrastructures

Given that resources from HPC infrastructures has to be requested for every performed tasks, it is not possible to run all the steps from *MASS* in one step. However, since resources can be requested accordingly, *MASS* can perform multiple steps simultaneously, enhancing the capabilities of the toolset. With a HPC, for example, *MASS* could analyze more mutants than if *MASS* is executed on a single machine.

The multiple steps of the methodology and its command are described in the following.

#### 1. Prepare the SUT

```
1 $ ./MASS_STEPS_LAUNCHER/PrepareSUT.sh
```

#### 2. Generate Mutants

```
1 $ ./MASS_STEPS_LAUNCHER/GenerateMutants.sh
```

3. **Compile Optimized Mutants:** If the environment variable `HPC` is set to `true`. Then, a parameter can be passed to the launcher script indicating the optimization level to be processed. If six levels of optimizations are defined, then numbers between zero and five can be provided.

```
1 $ level=0
2 $ ./MASS_STEPS_LAUNCHER/CompileOptimizedMutants.sh $level
```

#### 4. Compile Optimized Mutants Post-Processing

```
1 $ ./MASS_STEPS_LAUNCHER/OptimizedPostProcessing.sh
```

#### 5. Generate Prioritized and Reduced Test Suites

```
1 $ ./MASS_STEPS_LAUNCHER/GeneratePTS.sh
```

#### 6. Prepare mutants

```
1 $ ./MASS_STEPS_LAUNCHER/PrepareMutants_HPC.sh
```

7. **Execute mutants:** The launcher script receives two parameters: the first parameter is the number of the mutant to be executed, the number of mutant is defined as  $1..M$ , being  $M$  the total number of mutants. The total number of mutants can be derived from the folder `$INSTALL_DIR/hpc-src-mutants`. The second parameter defines if the test suite has to be executed in a reduced fashion or not. The possible values are “true” and “false”.

```
1 $ nr_mutant=1
2 $ reduced="false"
3 $ ./MASS_STEPS_LAUNCHER/ExecuteMutants_HPC.sh $nr_mutant $reduced
```

8. **Post-mutation execution:** The launcher script receives two numbers as parameters, a minimum and a maximum value, that represent the range of mutants to assess. The assessment consists of evaluating if more mutant executions are needed.

```
1 $ min=1
2 $ max=700
3 $ ./MASS_STEPS_LAUNCHER/PostMutation_HPC.sh $min $max
```

9. **Identify Equivalent Mutants based on Code Coverage**

```
1 $ ./MASS_STEPS_LAUNCHER/IdentifyEquivalents.sh
```

10. **Computer Mutation Score**

```
1 $ ./MASS_STEPS_LAUNCHER/MutationScore.sh
```

## 4.3 Mode selection and control

## 4.4 Normal Operations

## 4.5 Normal Termination

**TODO:** "The SUM shall describe how the user can cease or interrupt use of the software and how to determine whether normal termination or cessation has occurred."

## 4.6 Error Conditions

**TODO:** Can 't we say anything about each program not returning any error message?

## 4.7 Recover Runs

**TODO:** Shall we say that an engineer can restart the process form a task if all the preconditions are met?

# Chapter 5

## Tutorial

### 5.1 Introduction

This tutorial presents how to use *MASS* on a typical case. Since *MASS* provides two modes of execution (i.e., running *MASS* on a single machine and running *MASS* on HPC infrastructures), we provide an example for both.

Both examples use the Mathematical Library for Flight Software as case study to exemplify the steps to follow for the use of FAQAS-framework.

### 5.2 Getting Started

TODO: ?

### 5.3 Using the software on a typical task

#### 5.3.1 Single Machine Example: Mathematical Library for Flight Software

The first step regards installing the *MASS* framework, please refer to Section 4.1.

The second step, consists of creating and installing a workspace folder for running *MASS* on the MLFS example. For this case, the workspace folder will be created on `/opt/MLFS`.

```
1 $ cd $FAQAS/MASS/FAQAS-Setup
2 $ export INSTALL_DIR=/opt/MLFS
3 $ ./install.sh
```

The third step consists of configuring the *MASS* configuration file `mass_conf.sh`. In the following, we provide excerpts of the file that require intervention from the engineer. The following excerpt contains the necessary configuration for the MLFS case study.

```
1 # set FAQAS path
2 export SRCIROR=/opt/srcirorfaqas
3
4 ...
```

```

5
6 # set directory path where MASS files can be stored
7 export APP_RUN_DIR=/opt/MLFS
8
9 # specifies the building system, available options are "Makefile" and "waf"
10 export BUILD_SYSTEM="Makefile"
11
12 # directory root path of the software under test
13 export PROJ=$HOME/mlfs
14
15 # directory src path of the SUT
16 export PROJ_SRC=$PROJ/libm
17
18 # directory test path of the SUT
19 export PROJ_TST=$HOME/unit-test-suite
20
21 # directory coverage path of the SUT
22 export PROJ_COV=$HOME/blts_workspace
23
24 # directory path of the compiled binary
25 export PROJ_BUILD=$PROJ/build-host/bin
26
27 # filename of the compiled file/library
28 export COMPILED=libmlfs.a
29
30 # path to original Makefile
31 export ORIGINAL_MAKEFILE=$PROJ/Makefile
32
33 # compilation command of the SUT
34 export COMPILE_CMD=(make all ARCH=host EXTRA_CFLAGS="-DNDEBUG" \&\& make all COVERAGE="true" ARCH=
    host_cov EXTRA_CFLAGS="-DNDEBUG")
35
36 # compilation additional commands of the SUT (e.g., setup of workspace)
37 export ADDITIONAL_CMD=(cd $HOME/blts/BLTSConfig \&\& make clean install INSTALL_PATH="$HOME/
    blts_install" \&\& cd $HOME/blts_workspace \&\& $HOME/blts_install/bin/blts_app --init)
38
39 # command to be executed after each test case (optional)
40 export ADDITIONAL_CMD_AFTER=(rm -rf $HOME/blts_workspace/*)
41
42 # compilation command for TCE analysis
43 export TCE_COMPILE_CMD=(make all ARCH=host EXTRA_CFLAGS="-DNDEBUG")
44
45 # command to clean installation of the SUT
46 export CLEAN_CMD=(make cleanall)
47
48 # relative path to location of gcov files (i.e., gcda and gcno files)
49 export GC_FILES_RELATIVE_PATH=Reports/Coverage/Data

```

Also MASS variables shall be configured within the same file. Particularly, we will run MASS with the following setup.

```

1 ### MASS variables
2
3 # TCE flags to be tested
4 export FLAGS=("-O0" "-O1" "-O2" "-O3" "-Ofast" "-Os")
5
6 # specify if MASS will be executed on a HPC, possible values are "true" or "false"
7 export HPC="false"
8
9 # set if MASS should be executed with a prioritized and reduced test suite
10 export PRIORITIZED="true"
11
12 # set sampling technique, possible values are "uniform", "stratified", and "fsci"
13 # note: if "uniform" or "stratified" is set, $PRIORITIZED must be "false"
14 export SAMPLING="fsci"
15
16 # set sampling rate if whether "uniform" or "stratified" sampling has been selected
17 export RATE=""

```

The fourth step consists of configuring the prepare SUT configuration file (/opt/MLFS/MASS\_STEPS\_LAUNCHER/PrepareSUT.sh; within this file the following actions must



be provided by the engineer:

```

1  #!/bin/bash
2
3  # This file should be prepared by the engineer!
4  cd /opt/MLFS
5  . ./mass_conf.sh
6
7  # 1. Compile SUT
8  ## example
9
10 cd $PROJ
11
12 # # generate compile_commands.json and delete build
13 bear make all && rm -rf build* && sed -i 's: libm: /home/mlfs/mlfs/libm:' compile_commands.json && mv
   compile_commands.json $MUTANTS_DIR
14 eval "${COMPILE_CMD[@]}"
15
16 # 2. Prepare test scripts
17 # example
18
19 cd $HOME/blts/BLTSConfig
20 make clean install INSTALL_PATH="$HOME/blts_install"
21
22 # Preparing MLFS workspace (e.g., where test cases data is stored)
23 cd $HOME/blts_workspace
24 $HOME/blts_install/bin/blts_app --init
25
26 # 3. Execute test cases
27 # Note: execution time for each test case should be measured and passed as argument to FAQAS-
   CollectCodeCoverage.sh
28
29 # example
30 for tst in $(find $HOME/unit-test-suite -name '*.xml');do
31     cd $HOME/blts_workspace
32
33     tst_filename_wo_xml=$(basename -- $tst .xml)
34
35     start=$(date +%s)
36     $HOME/blts_install/bin/blts_app -gcrx $tst_filename_wo_xml -b coverage --nocsv -s $tst
37     end=$(date +%s)
38
39     # call to FAQAS-CollectCodeCoverage.sh
40     # parameter should be test case name and the execution time
41     source $MASS/FAQAS-GenerateCodeCoverageMatrixes/FAQAS-CollectCodeCoverage.sh $tst_filename_wo_xml "
   $((end-$start))" $PROJ_COV
42 done

```

The fifth step consists of defining the function `run_tst_case` within the file `mutation_additional_functions.sh`:

```

1  run_tst_case() {
2
3     tst_name=$1
4     tst=$PROJ_TST/$tst_name.xml
5
6     echo $tst_name $tst
7
8     # run the test case
9     cd $PROJ_COV
10    $HOME/blts_install/bin/blts_app -gcrx $tst_name -b coverage --nocsv -s $tst
11
12    # define if test case execution passed or failed
13    summaryreport=$tst_name/Reports/SessionSummaryReport.xml
14    originalreport=$HOME/unit-reports/$summaryreport
15
16    test_cases_failed='xmlLint --xpath "//report_summary/test_set_summary/test_cases_failed/text()"
   $summaryreport'
17    o_test_cases_failed='xmlLint --xpath "//report_summary/test_set_summary/test_cases_failed/text()"
   $originalreport'
18
19    echo "comparing with original execution"
20    echo $test_cases_failed $o_test_cases_failed
21

```

```

22     if [ "$test_cases_failed" != "$o_test_cases_failed" ]; then
23         return 1
24     else
25         return 0
26     fi
27 }

```

The sixth step consists of providing a template for the build script for the trivial compiler optimizations step. In particular, we replaced the following command:

```

1 CFLAGS = -c -Wall -std=gnu99 -pedantic -Wextra -frounding-math -fsignaling-nans -g 02 -fno-builtin $(
    EXTRA_CFLAGS)

```

with the following one:

```

1 CFLAGS = -c -Wall -std=gnu99 -pedantic -Wextra -frounding-math -fsignaling-nans TCE -fno-builtin $(
    EXTRA_CFLAGS)

```

The seventh step consists of launching the one step launcher (see Section 4.2.6:

```

1 $ /opt/MLFS/Launcher.sh

```

The following results shall be reported at the end of the execution:

```

1 ##### MASS Output #####
2 ## Total mutants generated: 28071
3 ## Total mutants filtered by TCE: 6918
4 ## Sampling type: fsci
5 ## Total mutants analyzed: 461
6 ## Total killed mutants: 369
7 ## Total live mutants: 92
8 ## Total likely equivalent mutants: 53
9 ## MASS mutation score (%): 90.44
10 ## List A of useful undetected mutants: /opt/MLFS/RESULTS/useful_list_a
11 ## List B of useful undetected mutants: /opt/MLFS/RESULTS/useful_list_b
12 ## Number of statements covered: 1973
13 ## Statement coverage (%): 100
14 ## Minimum lines covered per source file: 2
15 ## Maximum lines covered per source file: 138

```

### 5.3.2 HPC Infrastructure Example: Mathematical Library for Flight Software

The first step regards installing the MASS framework, please refer to Section 4.1.

**TODO:** To complete?