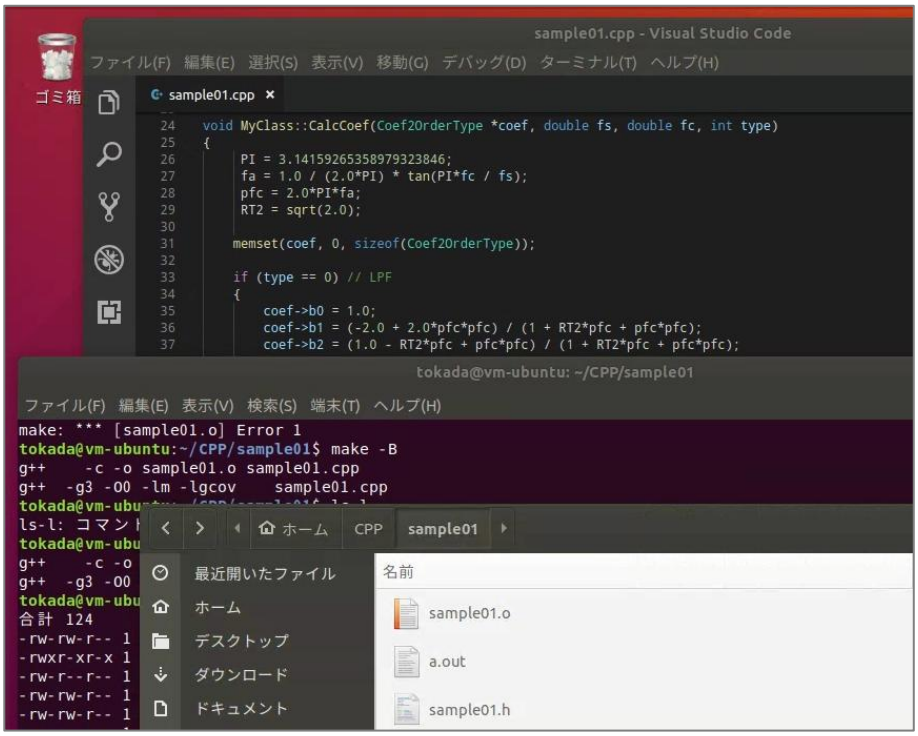


Quality Town for Embedded grade Tutorial for AWS MarketPlace (BYOL)

2nd April 2026

V3.0.0.0

Table of Contents

Introduction

Thank you for using “Quality Town for Embedded Grade” QTE, our Test Tool for C++/linux environments and Open-source software. QTE is a tool to efficiently design and run tests on the Software for Autonomous Driving and other Integrated ECUs. This Tutorial is for users. It starts with the installation process and covers the basic QTE features in a hands-on approach. The optional “Functional Safety” license is required for Chapter 5-6, and the optional “Integrated Report Output” license is required for Chapter 7.

- License Purchase

A license is required to use this product.
For inquiries regarding the purchase of a license, please contact us at:
https://info.gaio.co.jp/public/application/add/1187

- Supported Environments

For the latest and most comprehensive information on supported compilers and compatible Google Test versions, please refer to our official support page:
https://www.en.gaio.co.jp/eng/support/support_OpeEnv.html#qte

- Bundled Compiler

QTE (Operational Tool for GoogleTest in High-Reliability Software) BYOL is distributed with the following native compiler for host PCs: GCC Version 11.4
* A separate installation is required to use other supported compilers.

- Additional Compiler Support and Target Board Usage

For questions about extended compiler support or target boards, contact us at:
https://info.gaio.co.jp/public/application/add/1187

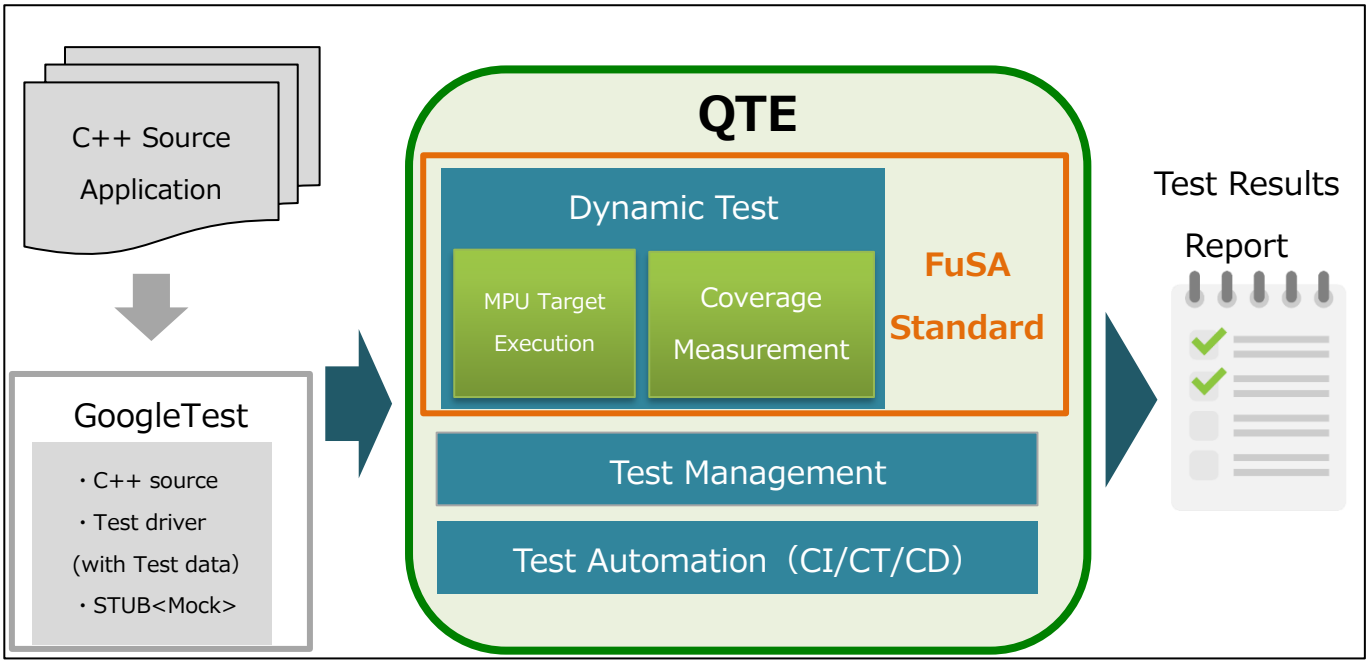
1 Overview of QTE

Before starting the tutorial, here is the required prior knowledge about the tool, its structure, and requirements.

A Test Tool for Embedded Application software running on a POSIX OS

QTE is an integrated test tool aimed at Autonomous Driving software running on a POSIX OS, providing the Test Evidence, Coverage Measurements required by the Functional Safety Standard ISO-26262.

It gathers results from Test Executions of the compiled Object and Statement, Branch, and MC/DC Coverage for the tests as well as Function and Fucntion Call coverages.



QTE's defining features are the reusability of GoogleTest assets and the efficient method to run Functional Safety (ISO26262)-compliant tests.

It allows for fast iterations of tests running the GCC-compiled object on a Linux PC, repeatingly checking the source and adding missing Test Data.

The same Test Data can then be used on the Real Target or a Simulation of it* running the Cross-compiled Object.

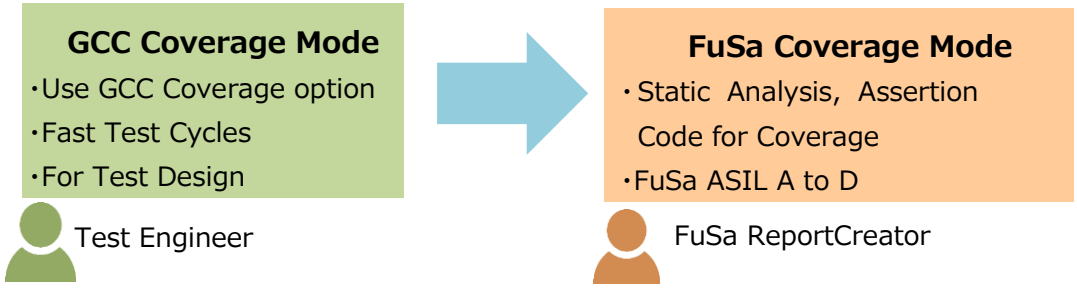
*: The simulator is ARM's Fixed Virtual Platform (FVP)

Test Modes foreach Test phase

QTE provides two Test modes depending on the Test phase.

When designing the Tests, the GCC Coverage mode is used for faster Test cycles.

Then the final test for the Functional Safety Standard, including Coverage Measurement, is done in the Functional Safety Coverage mode. Functional Safety Coverage mode can be run on either the Native or the Real Target environment.

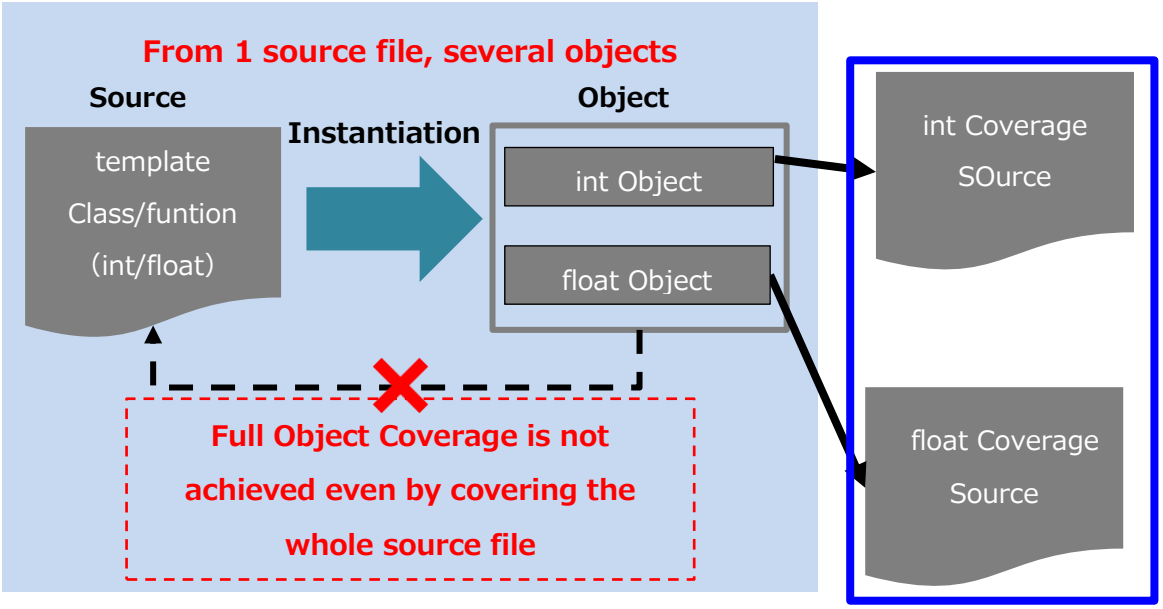


Template Classes and Methods are tested for each data type

Coverage Measurement for C++ templates is done for each declared type.

Several objects are generated from 1 template from 1 source, so full Source coverage does not imply full Object coverage.

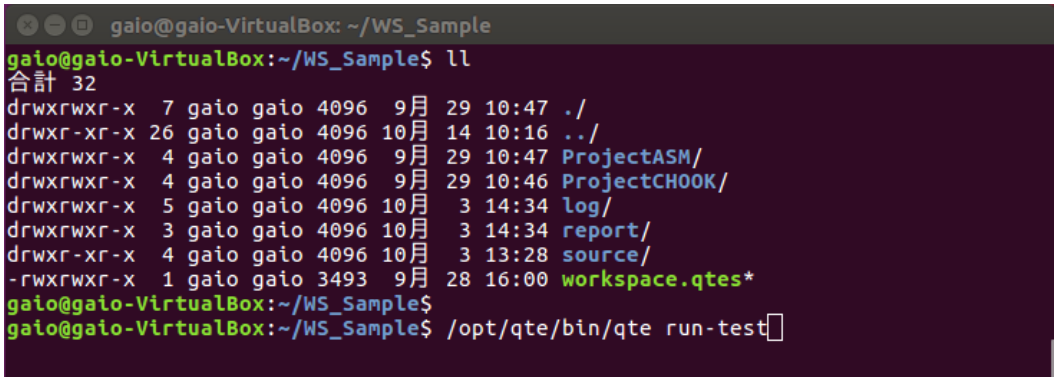
QTE tests for each declared type of a template, ensuring Object-level coverage.



Linux CLI (Command line Interface)

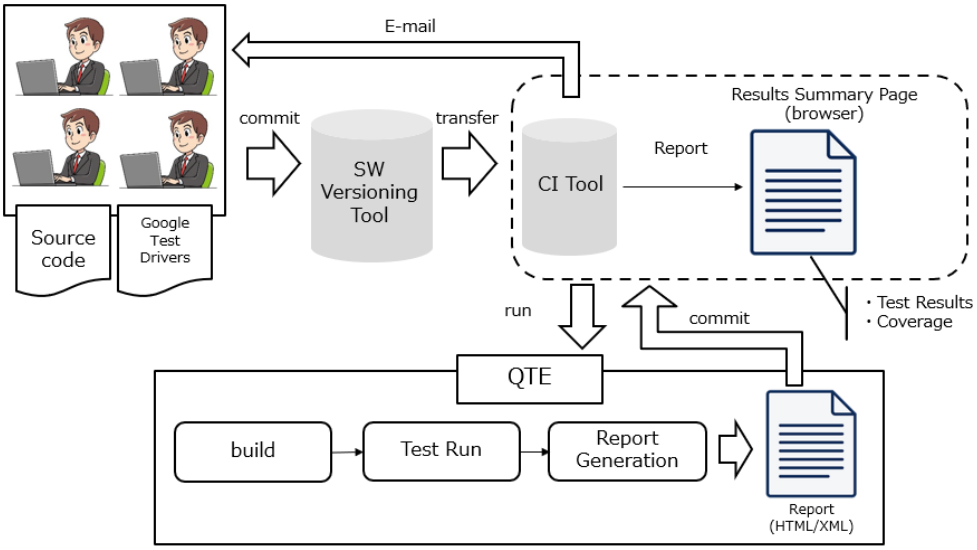
QTE is entirely controlled through a Linux Terminal via command lines.

QTE Settings are written (and edited) through a JSON format file. These descriptive specifications are open to users, allowing them to build highly flexible testing assets in the Linux environment.



Easy Automation/Integration for CI/CT

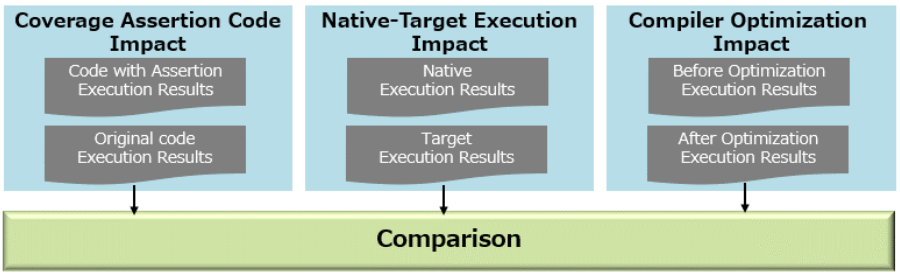
QTE has no GUI, and all its features are used through the standard Linux CLI. All settings, automation scripts, logs, reports and results are all in standard formats, so that Automation and Integration with CI tools like Jenkins can be done easily.



Verify the impact of Instrumentation Code or Compiler Optimizations

QTE can check for differences in execution caused by its own assertion code, or by compiler Optimizations or other gaps between Execution environments.

Comparing with the GoogleTest results it can dectect discrepancies introduced at later points.



Generate an integrated report for the whole project

QTE can generate several reports aimed at different stakeholders of the project, with varying amounts of information. It can be used to verify that no tests are missing or duplicated, and the overall progress of the Test Phase. It can also be used as Test Evidence.

2 Content of this Tutorial

This Tutorial uses a small sample program to demonstrate how to run tests using all the major QTE features.

2.1 Features used in this Tutorial

This Tutorial does the following:

<Chapter 5>

- Use the **Functional Safety Coverage mode** with the Native environment on a Linux machine

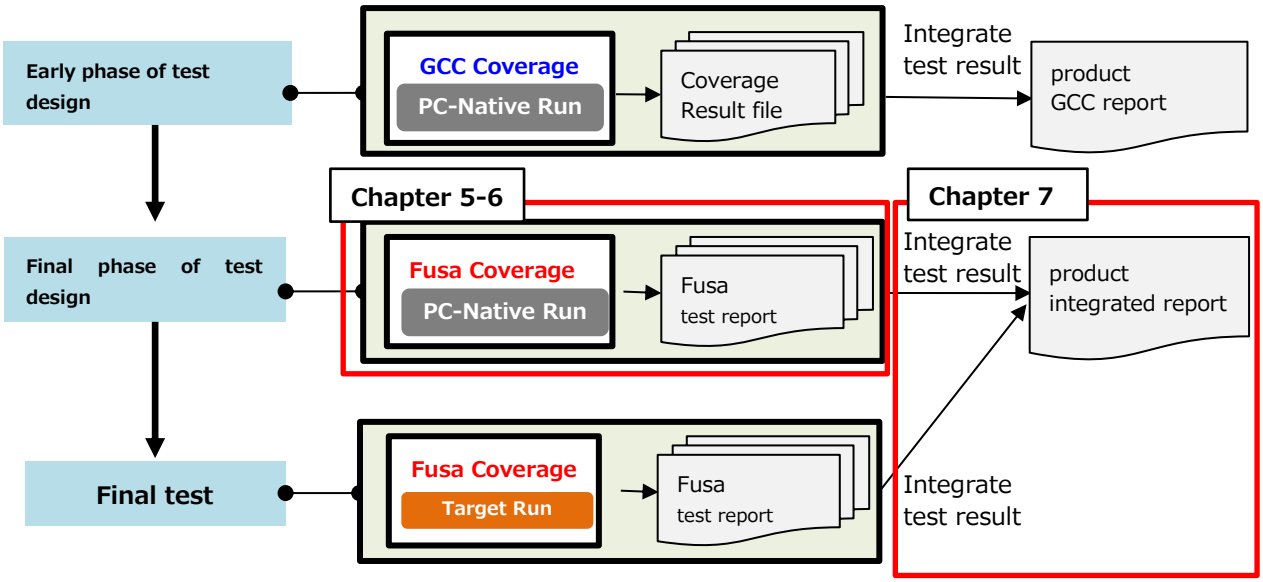
<Chapter 6>

- Use the **Tool Validation Check Feature** with the Native environment on a Linux machine

<Chapter 7>

- Check the **workspace integrated test report**
- Check the **product integrated test report**

Features used in this Tutorial



2.2 How to run this Tutorial

This Tutorial is run via commands executed on a terminal.

Commands to be executed are displayed as shown below.1

In bold are the QTE commands, the standard Linux commands are in non-bold.

```
$ cd ~/QTE_EvaluationKit/workspace    # Normal Linux command
$ qte-cov create-workspace            # QTE command
```

3 Preparation and Checks Before the Tutorial

Before explaining the operation of QTE, start the AWS environment for the tutorial.

For those who are already using AWS, a simplified procedure (Quick Start Guide) to prepare the tutorial environment is provided. Please refer to "3.1 Quick Start Guide" and set up the environment.

If you are not using AWS, please prepare the environment by following "3.2 AWS Tutorial Environment Setup Procedure."

3.1 Quick Start Guide

Please perform the following steps 1 through 3 in order.

1. Create a key pair for SSH connection.
2. Create an EC2 instance from the AMI starting with "ami-qte," and the key pair created in step 1.
3. Assign a public IP address to the instance created in step 3.

3.2 AWS Tutorial Environment Setup Procedure

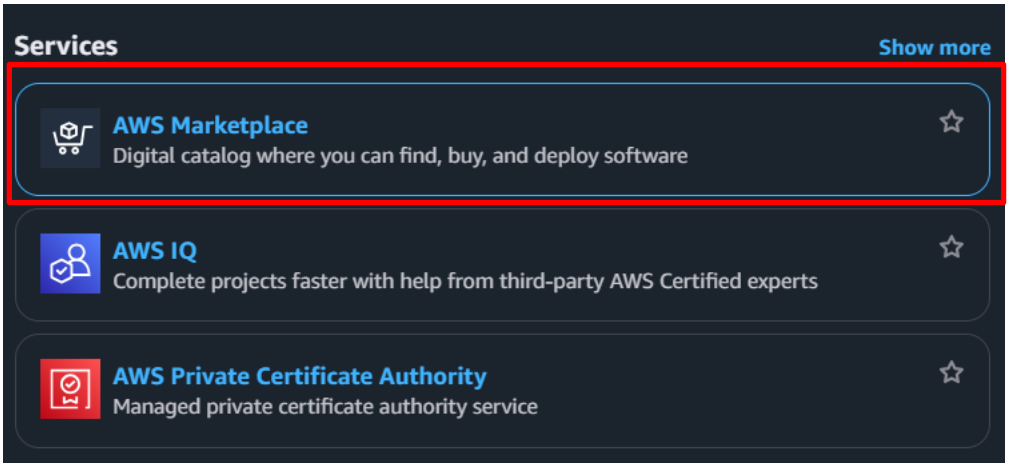
The following steps are based on the procedure as of the end of September 2025. Please note that the screen layout and displayed text may change. If there are any differences, follow the actual screen and displayed text accordingly.

Log in to the AWS Management Console.

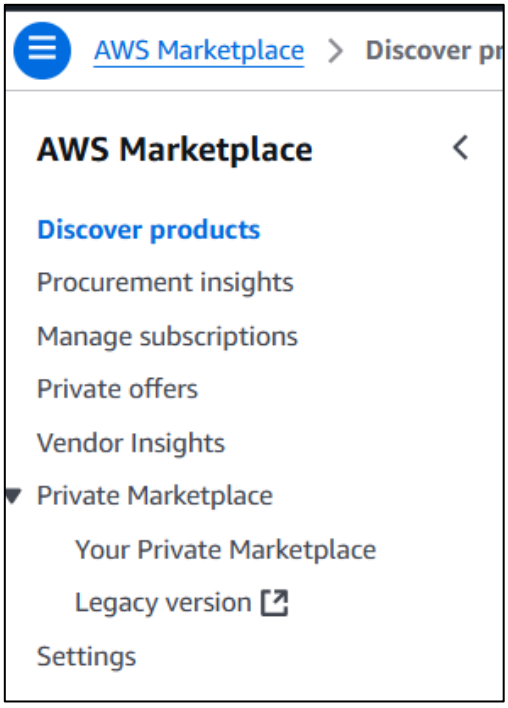
3.2.1 Creating an EC2 Instance

After logging in, enter "AWS Marketplace" in the search bar at the top left.

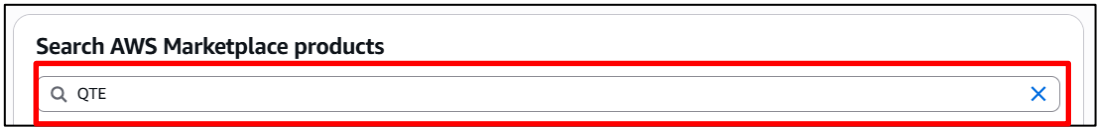
From the search results, select "AWS Marketplace" and click it.



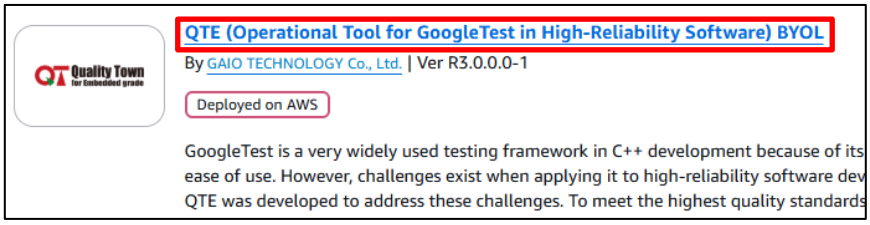
Click "Discover products" from the menu on the left-hand side.



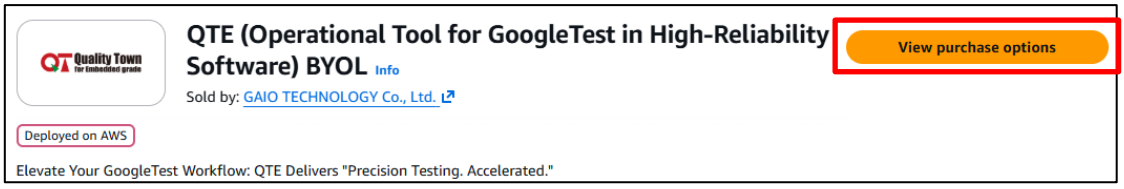
Enter "QTE" in the search bar at 'Search AWS Marketplace products'.



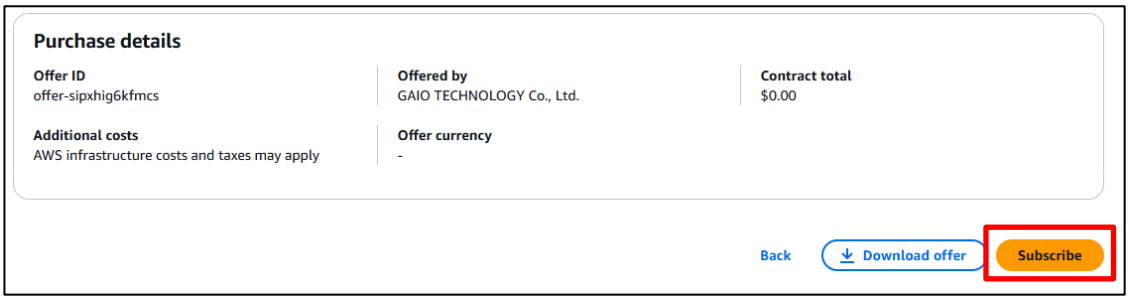
Click 'QTE (Operational Tool for GoogleTest in High-Reliability Software) BYOL'.



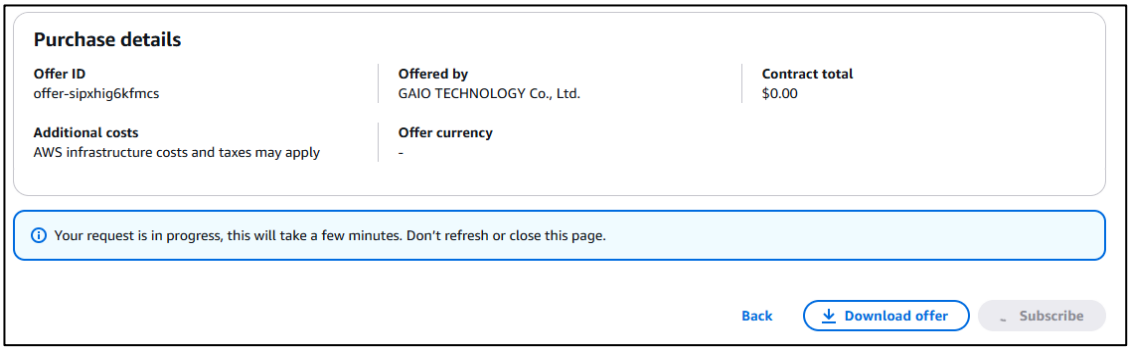
Access the QTE page from the marketplace. Click 'View purchase options'.



Click 'Subscribe' in the lower right of the page.

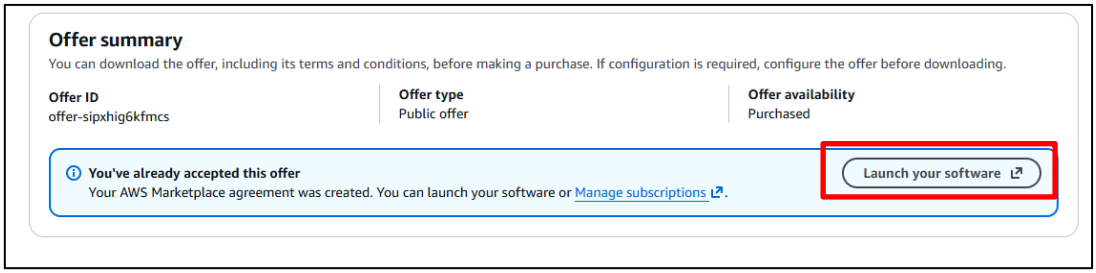


The following will be displayed on the screen.

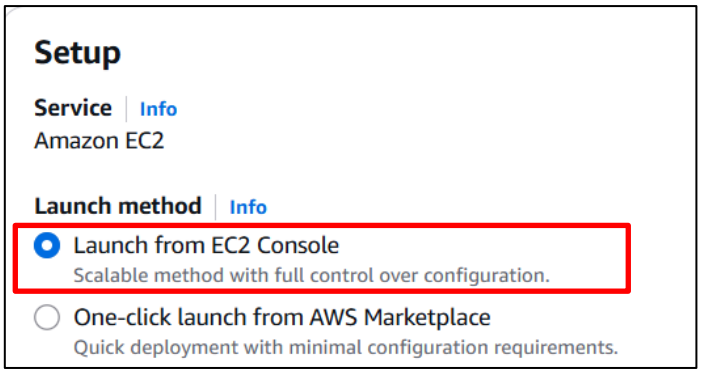


Please wait a few minutes for your subscription to complete.

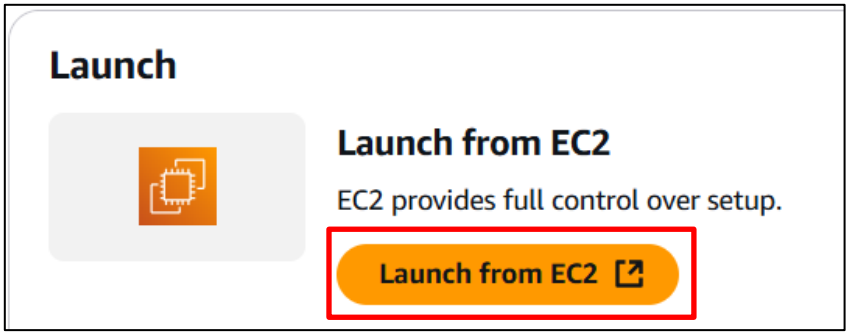
Once your subscription is complete, the screen will change.
Click 'Launch your software'.



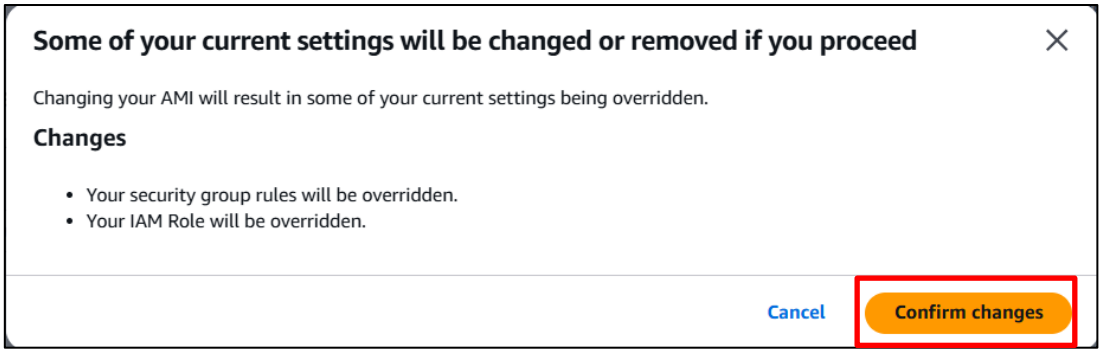
From 'Launch method', select 'Launch from EC2 Console'



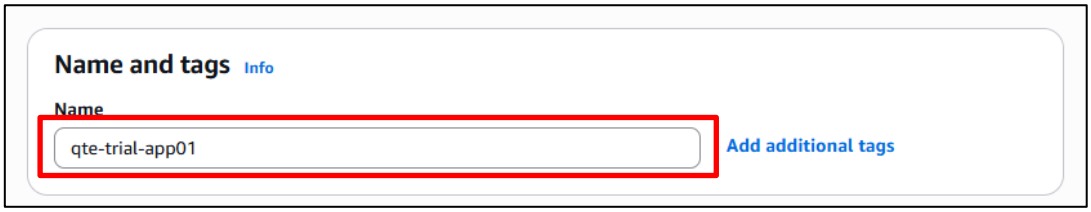
From 'Launch' in the middle of the page, click 'Launch from EC2'.



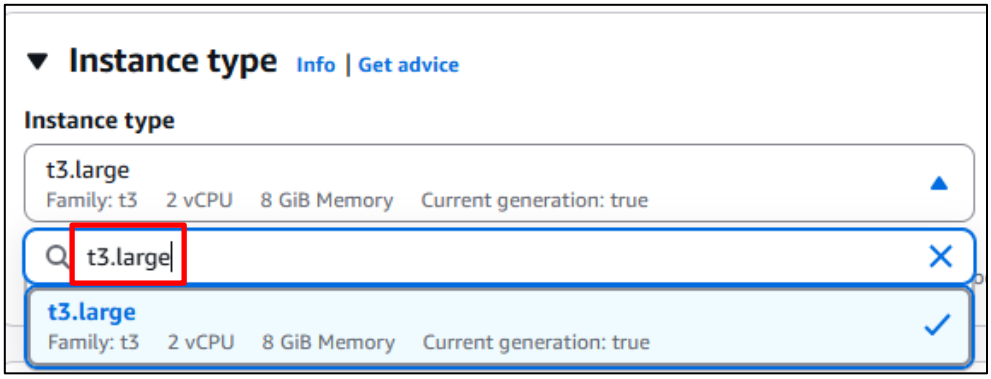
If the following dialog appears, click 'Confirm changes'.

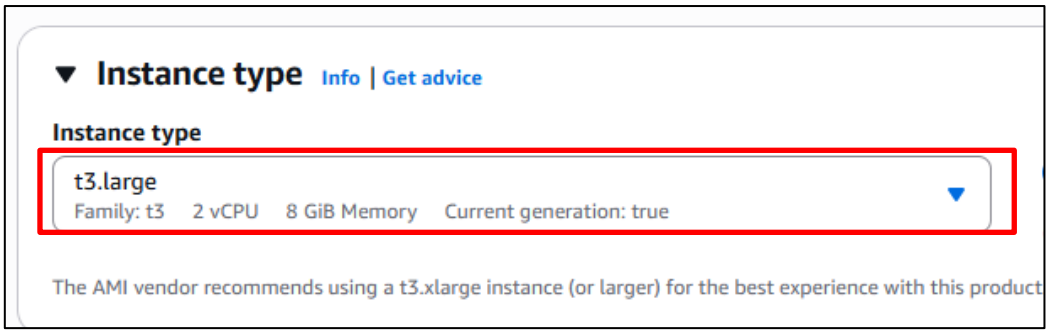


In the "Name and tags" section, enter “qte-trial-app01” in the "Name" field.

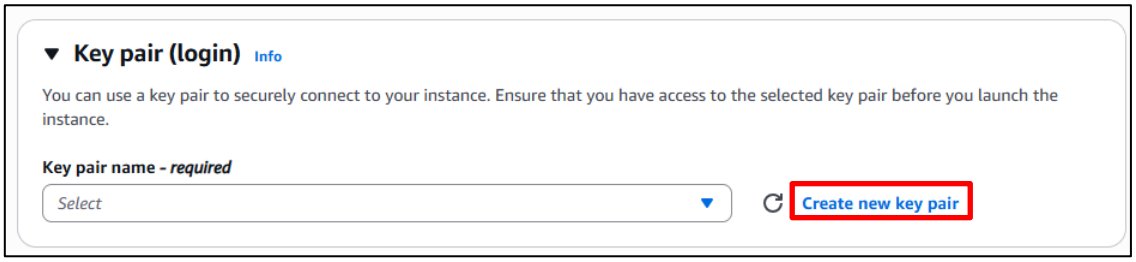


From the "Instance type" dropdown, select "t3.large."

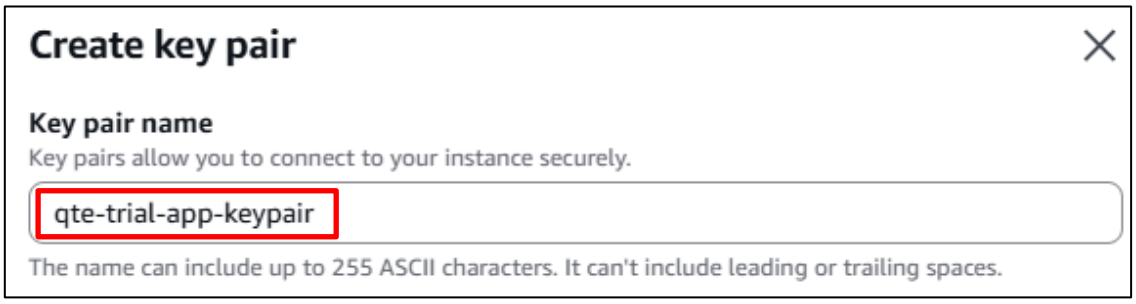




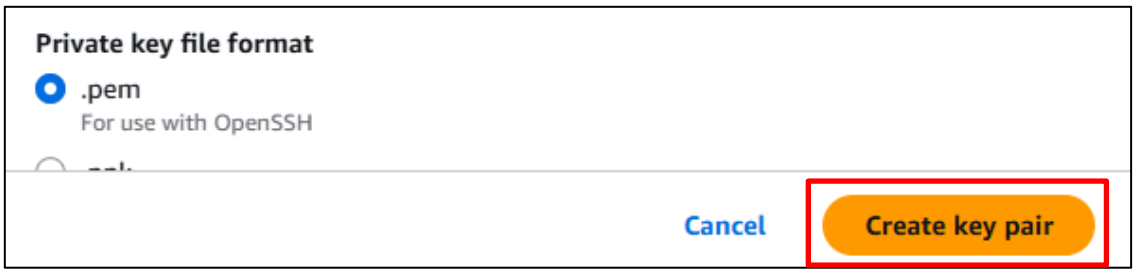
Under "Key pair (login)," click "Create new key pair."



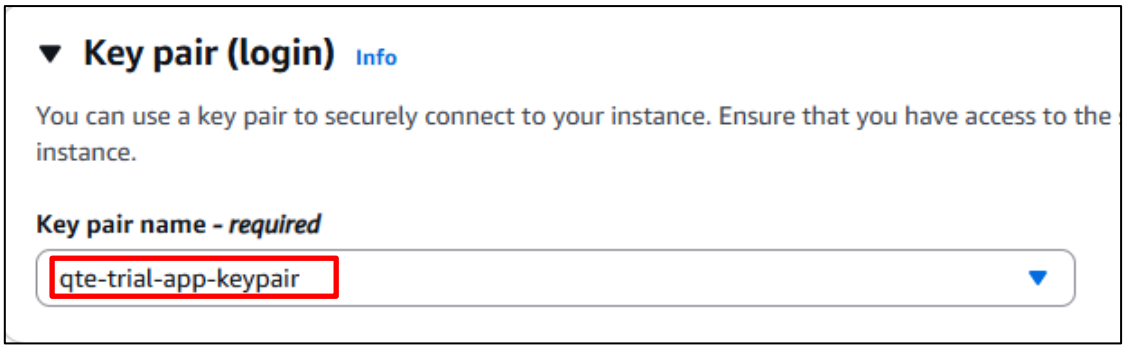
Enter any name in the "Key pair name" field. (If there are no specific requirements, please enter "qte-trial-app-keypair".)



Leave the other fields as they are, and click "Create key pair."

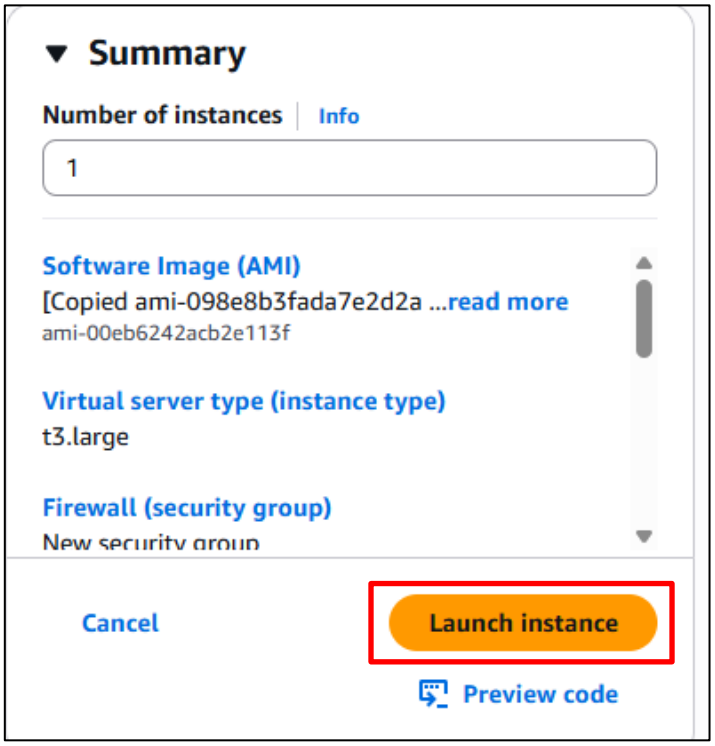


After clicking, confirm that "qte-trial-app-keypair" is selected in the "Key pair name" field under "Key pair (login)" on the original screen, and that a file with the ".pem" extension has been downloaded.

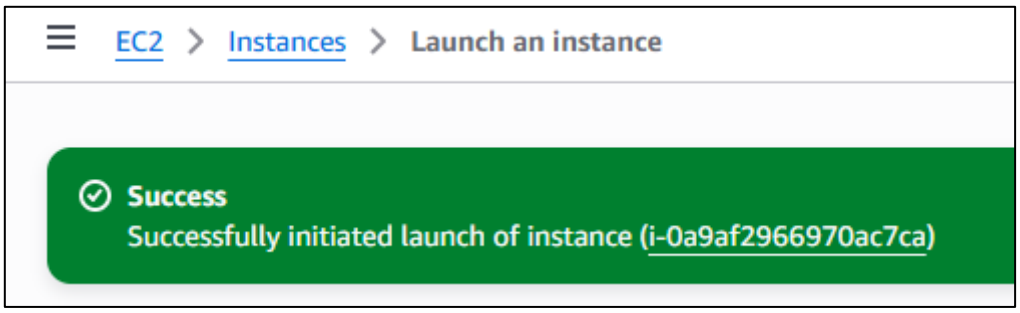


This completes the EC2 instance configuration.

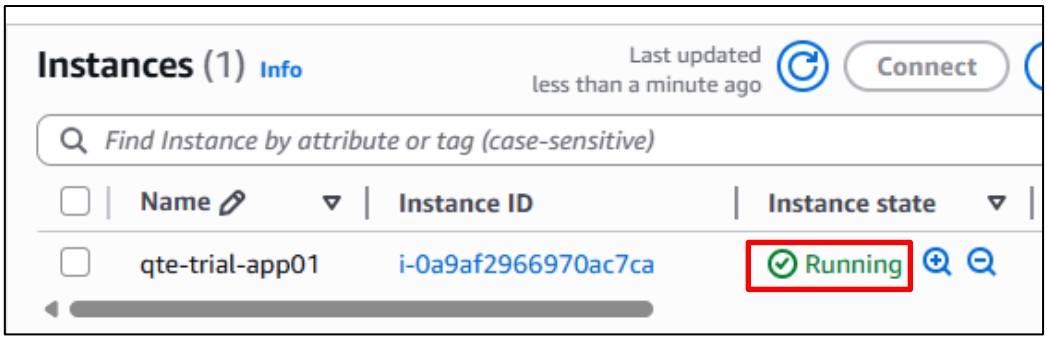
Click "Launch instance" in the "Summary" section on the right side of the screen to start the EC2 instance.



Once "Success" is displayed, click "Instances".

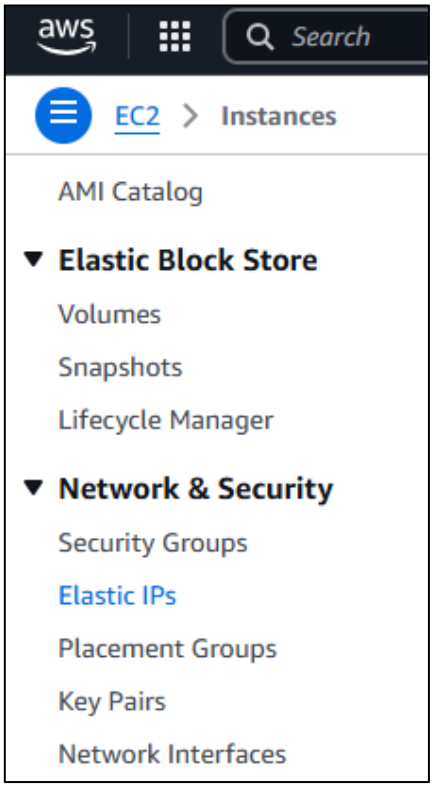


Confirm that the "qte-trial-app01" instance is in the "Running" state.

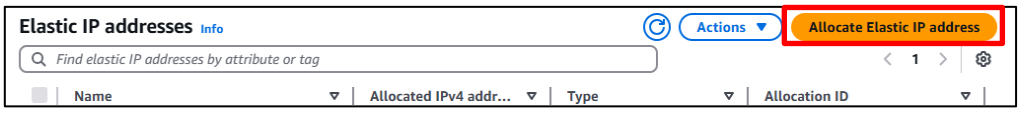


3.2.2 Configuring the Public IP

Click "Elastic IPs" from the menu on the left-hand side.

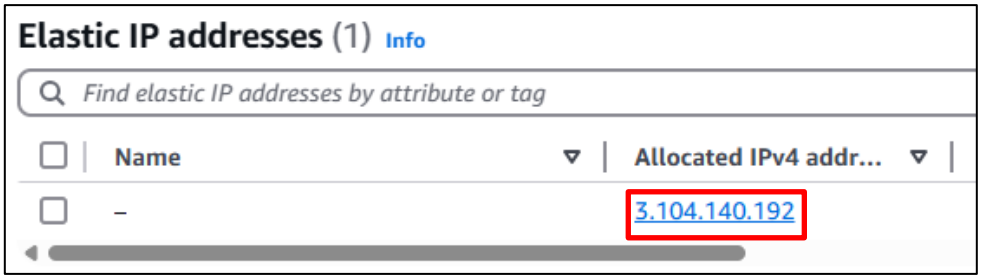


Click "Allocate Elastic IP address" in the top right.

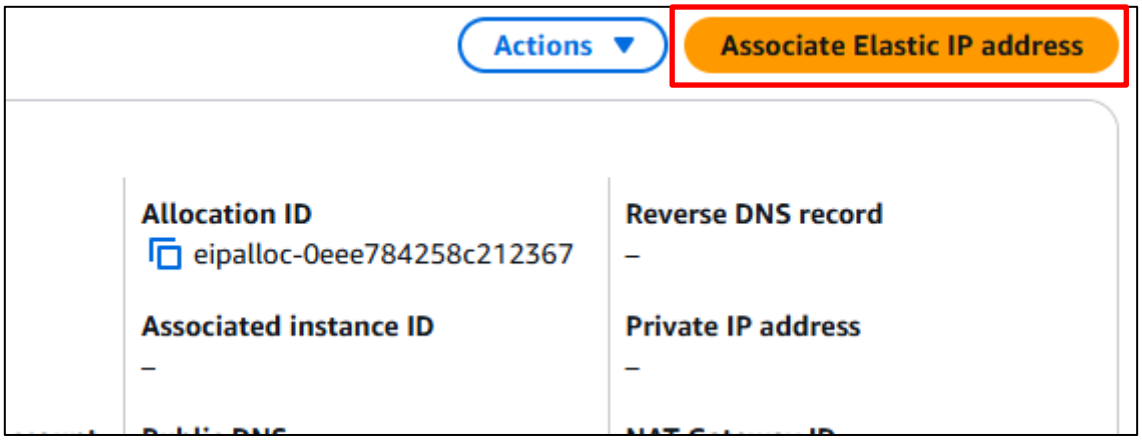


Leave the settings as they are, and click "Allocate" at the bottom of the screen.

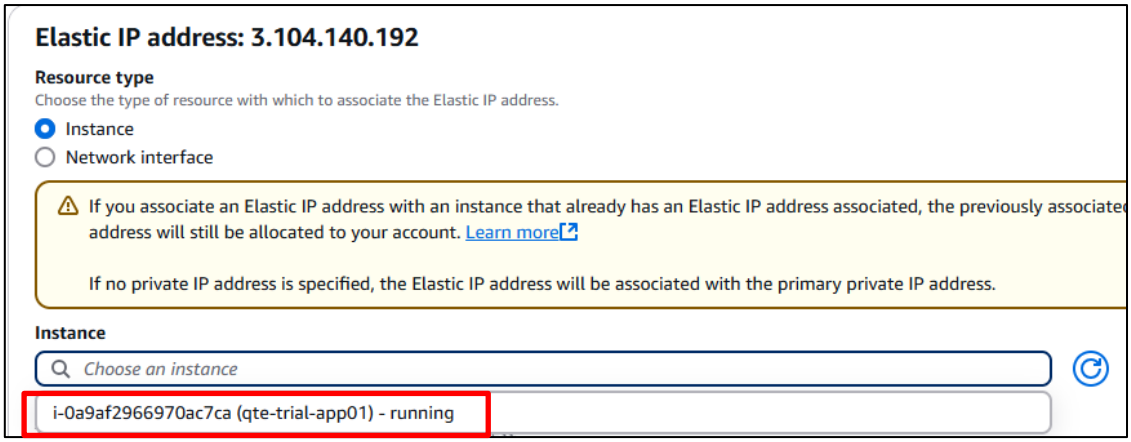
After confirming that a new IP address has been allocated, click on that IP address.



Click "Associate Elastic IP address" in the top right corner.

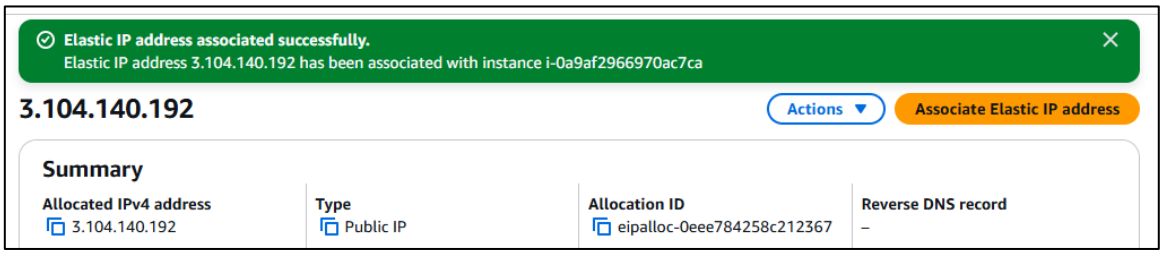


From the "Instance" dropdown, select the instance you created earlier.



Leave the other fields unchanged, and click "Associate" at the bottom of the screen.

If a message like the one below appears, the configuration is complete.



3.2.3 Connection Verification

Finally, verify the connection.

First, set the permissions for the downloaded key file.

Follow the instructions appropriate for your environment to configure the permissions correctly.

3.2.3.1 For Windows

Open PowerShell and run the following command to set the access permissions for the key file.

```
> icacls (the path to the downloaded .pem file) /inheritance:r
(omitted)
> icacls (the path to the downloaded .pem file) /grant "$($env:USERNAME):R"
(omitted)
```

Here is an example:

```
> icacls .¥qte-trial-app-keypair.pem /inheritance:r
(omitted)
> icacls .¥qte-trial-app-keypair.pem /grant "$($env:USERNAME):R"
(omitted)
```

3.2.3.2 For Linux

Open PowerShell and run the following command to set the access permissions for the key file.

```
$ chmod 600 (the path to the downloaded .pem file)
```

Here is an example:

```
$ chmod 600 ./qte-trial-app-keypair.pem
```

3.2.3.3 Connection Test

Open the terminal on the host machine (Command Prompt if using Windows) and execute the following command.

```
$ ssh ubuntu@(The address set using the Elastic IP) -i (the path to the downloaded .pem file)
```

Here is an example:

```
$ ssh ubuntu@3.104.140.192 -i ./qte-trial-app-keypair.pem
```

When connecting for the first time, the following message will appear. Type “yes” to continue.

```
The authenticity of host '3.104.140.192 (3.104.140.192)' can't be established.
ED25519 key fingerprint is SHA256:ANA/GBE4IwP4pTa12ucnIaQIH5XRF785zNNIeGS4ge0.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

If the prompt appears as shown below, the connection was successful.

```
ubuntu@ip-(The address set using the Elastic IP):~$
```

3.2.3.4 License Settings

To use QTE, follow the steps below to configure your license.

- Setting up the license service connection destination

Open /etc/qte/qte.conf with a text editor as root.

Locate the line starting with License_url and specify the license server URL in the following format:

```
License_url=http://<IP address>:<port number>/fne/bin/capability
```

✧ Adjust <IP address> and <port number> according to your environment.

- Start/Restart the license service

Start/Restart the license service using the Linux "systemctl" command.

Execute the following command:

```
$ sudo systemctl restart qted
```

- Verify the license settings

Run the ‘qte-cov create-workspace’ command to verify the license settings.

```
$ qte-cov create-workspace ws-test
**** Workspace created. Please edit the workspace.json file. ****
2026-03-05 07:22:33 INFO [SC-0000] qte-cov exit: 0
```

If successful, run the following command to delete the created workspace:

```
$ rm -rf ws-test
```

In case of failure, verify and redo the license service connection settings.

- Auto start the license service on EC2 instance startup

To configure the license service to start automatically at instance startup, run the following command:

```
$ sudo systemctl enable qted
```

This completes the setup, and you can now start the tutorial.

4 About the sample program

This tutorial uses the learning program located in the home directory (/home/gaio) in QTE_EvaluationKit/QTE_Tutorial/source directory.

The learning program have the following structure:

```
source/
    sample1                          ・・・ Program and build environment
    |                                       for learning QTE execution procedures
    |                                        (Used in Chapter 5, 6, 7)
    └── src
        ├── Makefile                 ・・・ build file(makefile)
        ├── sample1.cpp              ・・・ Test Target
        ├── sample1.h                ・・・ Test Target
        └── sample1Test.cpp          ・・・ Test Driver
    sample2                          ・・・ Program and build environment
    |                                       for checking test results of template classes
    |                                        (Used in Chapter 5, 7)
    └── src
        ├── Makefile                 ・・・ build file(makefile)
        ├── sample2.cpp              ・・・ Test Target
        ├── sample2.h                ・・・ Test Target
        └── sample2Test.cpp          ・・・ Test Driver
    sample3                          ・・・ Program and build environment
    |                                       for learning integrated reporting procedures
    |                                        (Used in Chapter 77)
    ├── BaseClass.cpp                ・・・ Test Target
    ├── InheriClass.cpp              ・・・ Test Target
    ├── Makefile                     ・・・ build file(makefile)
    ├── include                      ・・・ header file storage directory
    └── test                         ・・・ Test Driver storage directory
    sample4                      ・・・ Program and build environment
    |                                       for learning integrated reporting procedures
    |                                        (Used in Chapter 7)
    ├── Makefile                     ・・・ build file(makefile)
    ├── include                      ・・・ header file storage directory
    ├── sample4_1.cpp                ・・・ Test Target
    ├── sample4_2.cpp                ・・・ Test Target
    ├── sample4_3.cpp                ・・・ Test Target
    ├── sample4_3_1.cpp              ・・・ Test Target
    └── test                         ・・・ Test Driver storage directory
```

You can check that nothing is mssing with the following:

```
$ cd ~/QTE_EvaluationKit/QTE_Tutorial
$ ls -F source
googletest/             sample2_module/         sample4_module/
sample1_module/         sample3_module/
```

QTE_EvaluationKit directory contains the QTE_Tutorial_comp.tar.gz archive which mirrors the final project files, Test cases and makefiles you should have after completing the turorial.

5 Checking Test Results with Functional Safety Coverage mode

This chapter shows how to set up and run tests with QTE's Functional Safety Coverage mode in PC native environment.

This section explains how to set up and run tests, using the QTE commands on a terminal.

If an error occurs when executing QTE during the following procedures, please refer to the QTE User Manual, section '10.2.3 Error Messages'.

5.1 Creating the Execution environment

We will create projects for sample1 and sample2, a Functional Safety Coverage mode one. Before running tests, a workspace must be created.

5.1.1 Creating a Workspace

Tests for a Module/Functionality are grouped in a workspace, which contains the QTE projects for the different parts of the Module.

The workspace directory is created with the command below:

```
$ cd ~/QTE_EvaluationKit/QTE_Tutorial/test
$ qte-cov create-workspace workspace
**** Workspace created. Please edit the workspace.json file. ****
(omitted)
```

The create-workspace command "creates" the workspace by creating the following files and directories:

```
$ cd workspace
$ ls -F
log/  report/  source/  workspace.json
```

The workspace.json file contains all the workspace-related settings that are described in Section 5.1.2. This completes the workspace creation, next we will set the Workspace.

5.1.2 Setting the workspace

Configure the workspace.json file. The options and values to be set are listed below.

Option	Value
workspace-name	SampleWS
gtest-version	1.10.0
external-source-dir	../../source

Open workspace.json with a Text Editor and set up as follows. For options other than the following use the default values in the workspace.json file.

```
"workspace-name": {
    "value": "SampleWS"
},

"gtest-version": {
    "value": "1.10.0"
},

"external-source-dir": {
    "value": "../../source"
},
```

Next we will run a Test in Functional Safety Coverage mode.

5.2 Exercise 1 : Test in Functional Safety Coverage mode

In Exercise 1, we will run the tests in Functional Safety Coverage mode.
The Sample1 program is shown below.

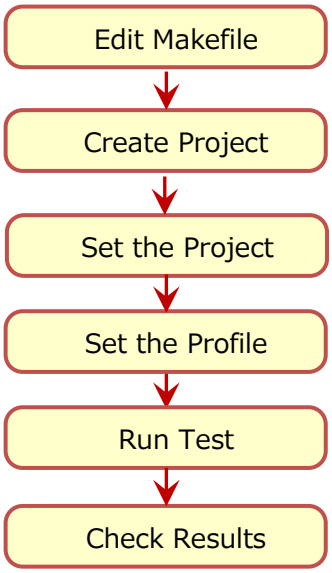
```
class Sample1 {
public:
    void func1(bool enable, int mode, int input);
    bool get_ret_code() const noexcept { return ret_code_; }
    int get_data() const noexcept { return data_; }

private:
    bool ret_code_;
    int data_;
};
```

func1 is defined as follows:

```
void Sample1::func1(bool enable, int mode, int input)
{
    if (enable) {
        switch (mode) {
        case 0:
            data_ = input;
            break;
        case 1:
            data_ = input * 10;
            break;
        case 2:
            data_ = input * 100;
            break;
        case 3:
            if ((input > 100) && (input < 300))
                data_ = 10000;
            else
                data_ = input * 100;
            break;
        default:
            data_ = -1;
        }
        ret_code_ = true;
    } else {
        data_ = 0;
        ret_code_ = false;
    }
}
```

The steps are:



Follow the steps described in the next section by running the same commands in a terminal.

5.2.1 Addition to the Makefile

The Makefile in the learning program compiles and links the test target sources and the GoogleTest library/test driver. To create an executable to test with QTE, include the Makefile from which the variables for QTE are defined from this Makefile, and add the QTE variables (configuration values) to the Makefile.

Open source/sample1_module/src/Makefile in an editor and add the deficit in the callout.

```
TARGET = sample1_test

(omitted)

%.o : %.cpp
	$(CXX) $(CXXFLAGS) -c -o $@ $<

sample1.o : sample1.cpp sample1.h

sample1Test.o : sample1Test.cpp sample1.h
	$(CXX) $(GTEST_INC_PATH) -pthread -c -o $@ $<

$(TARGET) : sample1.o sample1Test.o gtest.a gtest_main.a
	$(CXX) -o $@ $^ $(LDFLAGS) $(LDLIBS)
```

Insert "include ../../qte.make" in the first row

Add "$(QTE_SRC_FLAGS)"

Add "$(QTE_API)"

Add "$(QTE_LINK_FLAGS)"

Once the configuration is complete, the system will be in the following state.

```
include ../../qte.make

TARGET = sample1_test

(omitted)

%.o : %.cpp
	$(CXX) $(CXXFLAGS) -c -o $@ $< $(QTE_SRC_FLAGS)

sample1.o : sample1.cpp sample1.h

sample1Test.o : sample1Test.cpp sample1.h
	$(CXX) $(GTEST_INC_PATH) -pthread -c -o $@ $<

$(TARGET) : sample1.o sample1Test.o gtest.a gtest_main.a $(QTE_API)
	$(CXX) -o $@ $^ $(LDFLAGS) $(LDLIBS) $(QTE_LINK_FLAGS)
```

This concludes the Makefile modifications, next create a project.

Creating a project

Create a new project with the command below:

```
$ cd ~/QTE_EvaluationKit/QTE_Tutorial/test/workspace
$ qte-cov create-project ProjectSample1
(omitted)
```

The create-project command “creates” a project with the name given as argument (here “ProjectSample1”) by creating a directory under the workspace directory. The created directory has the Project’s name.

```
$ ls -F
ProjectSample1/  log/  report/  source/  workspace.json
```

In the ProjectSample1 directory the following directories and files are created.

```
$ cd ProjectSample1
$ ls -F
build.json      fusa_profile1/    fusa_profile2/    fusa_profile3/
gcc_profile/    project.json      testenv.json
```

Files with the '.json' extension are configuration files for QTE. The details of each file and directory are as follows.

File name / Directory name	Description
build.json (file)	Configuration information used during analysis and build of the test program
project.json (file)	Project configuration information
testenv.json (file)	Settings used during the execution of the test program information
fusa_profile1 (directory)	Profile directory (A profile is a unit that manages a set of files required for building and running the test program)
fusa_profile2 (directory)	
fusa_profile3 (directory)	
gcc_profile (directory)	

Now the project is created, we will edit its settings.

5.2.2 Project settings

We now edit the project.json file with the following settings.

Option	Value
project-name	Sample1PJ
test-target-source-path-list	@EXTERNAL_SOURCE_DIR@[1]/sample1_module/src/sample1.cpp
cov-target-header-path-list	@EXTERNAL_SOURCE_DIR@/sample1_module/src/sample1.h
gtest-source-path-list	@EXTERNAL_SOURCE_DIR@/sample1_module/src/sample1Test.cpp
build-env-executable-path-list	@BUILD_DIR@[2]/sample1_module/src/sample1_test
qte-env-gtest-include-dir	@EXTERNAL_SOURCE_DIR@/googletest/googletest/include

[1] @EXTERNAL_SOURCE_DIR@ designates the source directory.

[2] @BUILD_DIR@ designates the fusa_profile1/source directory.

Open project.json with a Text Editor and perform the changes in red below. Other fields can be left with the default value.

```
"project-name": {
   "value": "Sample1PJ"
},

"test-target-source-path-list": {
   "valueList": [
      "@EXTERNAL_SOURCE_DIR@/sample1_module/src/sample1.cpp"
   ]
},

"cov-target-header-path-list": {
   "valueList": [
      "@EXTERNAL_SOURCE_DIR@/sample1_module/src/sample1.h"
   ]
},

"gtest-source-path-list": {
   "valueList": [
      "@EXTERNAL_SOURCE_DIR@/sample1_module/src/sample1Test.cpp"
   ]
},

"build-env-executable-path-list": {
   "valueList": [
      "@BUILD_DIR@/sample1_module/src/sample1_test"
   ]
},

"qte-env-gtest-include-dir": {
   "value":"@EXTERNAL_SOURCE_DIR@/googletest/googletest/include"
},
```

Next we now edit the build.json file with the following settings.

Option	Value
compiler-path	/usr/bin/g++-11
system-include-dir-list	/usr/include/c++/11
	/usr/include/x86_64-linux-gnu/c++/11
	/usr/lib/gcc/x86_64-linux-gnu/11/include
	/usr/include/x86_64-linux-gnu
	/usr/include
user-include-dir-list	@EXTERNAL_SOURCE_DIR@/ googletest/googletest/include
	@EXTERNAL_SOURCE_DIR@/sample1_module/src
qte-api-libfile-path	/opt/qte/usrlib/Qte-native-x64-linux-gcc-11.4.0.o

Open build.json with a Text Editor and perform the changes in red below. Other fields can be left with the default value.

```
"compiler-path": {
    "value": "/usr/bin/g++-11"
},

"system-include-dir-list": {
    "valueList": [
        "/usr/include/c++/11",
        "/usr/include/x86_64-linux-gnu/c++/11",
        "/usr/lib/gcc/x86_64-linux-gnu/11/include",
        "/usr/include/x86_64-linux-gnu",
        "/usr/include"
    ]
},

"user-include-dir-list": {
    "valueList": [
        "@EXTERNAL_SOURCE_DIR@/googletest/googletest/include",
        "@EXTERNAL_SOURCE_DIR@/sample1_module/src"
    ]
},

"qte-api-libfile-path": {
    "value": "/opt/qte/usrlib/Qte-native-x64-linux-gcc-11.4.0.o"
},
```

Next we now edit the testenv.json file with the following settings.

Option	Value
test-execution-host-optional-label	Ubuntu-PC

Open testenv.json with a Text Editor and perform the changes in red below. Other fields can be left with the default value.

```
  "test-execution-host-optional-label": {
     "value": "Ubuntu-PC"
  },
```

This concludes the project settings. Next change the profile settings.

5.2.3 Profile Settings

In Exercise 1, the fusa_profile1 is used. Otherprofiles are not used.

The directories and files in the fusa_profile1 directory are as follows.

```
$ cd fusa_profile1
$ ls -F
binary/         prehook.sh*    result/          targetscript/
posthook.sh*    pretest.sh*    source/          testbinbuild.sh
posttest.sh*    profile.json   syncsource.sh*   userscript/
```

We have verified that the above directories/files exist.

We now edit the profile.json file in the fusa_profile1 with the following settings.

Option	Value
profile-name	FuSa_Coverage
execution-mode	3

Open profile.json with a Text Editor and perform the changes in red below.

```
"profile-name": {
    "value": "FuSa_Coverage"
},

"execution-mode": {
    "value": "3"
},
```

Next we now edit the testbinbuild.sh.

In QTE, you need to write the command to be executed during the build process in a script (testbinbuild.sh).

Open testbinbuild.sh in an editor and enter the red text at line 13.

```
#!/bin/bash

# This script is called by the 'qte-cov build' command.
# Usage: testbinbuild.sh <EXECUTION_MODE> <WORKSPACE_PATH> <PROJECT_PAT
H> <PROFILE_PATH> <EXTERNAL_SOURCE_DIR_PATH> <BUILD_DIR_PATH>
# Do not change the following six lines.
EXECUTION_MODE=$1
WORKSPACE=$2
PROJECT=$3
PROFILE=$4
EXTERNAL_SOURCE_DIR=$5
BUILD_DIR=$6

make -C ${BUILD_DIR}/sample1_module/src debug CC=gcc CXX=g++
```

The Profile settings are done. We can now run the test.

5.2.4 Running the Test

Run the test. Change the current directory to ProjectSample1/fusa_profile1 and run the prepare, build, and run-test commands to run the test.

```
$ cd ~/QTE_EvaluationKit/QTE_Tutorial/test/workspace/ProjectSample1/fusa_profile1
$ qte-cov prepare
**************************************************************************
**** Running Prepare (Functional safety coverage test / Extract execution values)
**************************************************************************
(omitted)
**** Running posthook.sh
(omitted)
$ qte-cov build
**************************************************************************
**** Running Build (Functional safety coverage test / Extract execution values)
**************************************************************************
(omitted)
**** Running testbinbuild.sh
(omitted)
$ qte-cov run-test
**************************************************************************
**** Running Test (Functional safety coverage test / Extract execution values)
**************************************************************************
(omitted)
**** Running Coverage Aggregate
(omitted)
```

In Functional Safety Coverage mode, the Coverage results can be checked in the Project Execution Report. Next we check the Test Results.

5.2.5 Checking the Test Results

In Functional Safety Coverage mode, Test Results can be checked in the Project Execution Report create by the create-project-execution-report command.

```
$ qte-cov create-profile-execution-report
**************************************************************
**** Create Profile Execution Report
**************************************************************
(omitted)
```

To verify the generated report using the tree command, install the tree command first.

```
$ sudo apt update && sudo apt install -y tree
(omitted)
```

The create-project-execution-report command creates a directory with the profile's name under workspace_dir/report/project_name_reports/
(the project&s name directory) and generates profile_execution_report.html in that directory.

```
$ cd ../../report
$ tree -L 4
.
`-- project_fs_reports
    `-- ProjectSample1
        `-- fusa_profile1
            |-- coverage
            |-- profile_execution_report.html
            |-- profile_execution_report_im.xml
            |-- resources
            `-- result
```

First, on your local environment, open a separate terminal (not connected via SSH), and run the following command to copy the file to your client machine. Replace the parts in parentheses with values appropriate for your environment.

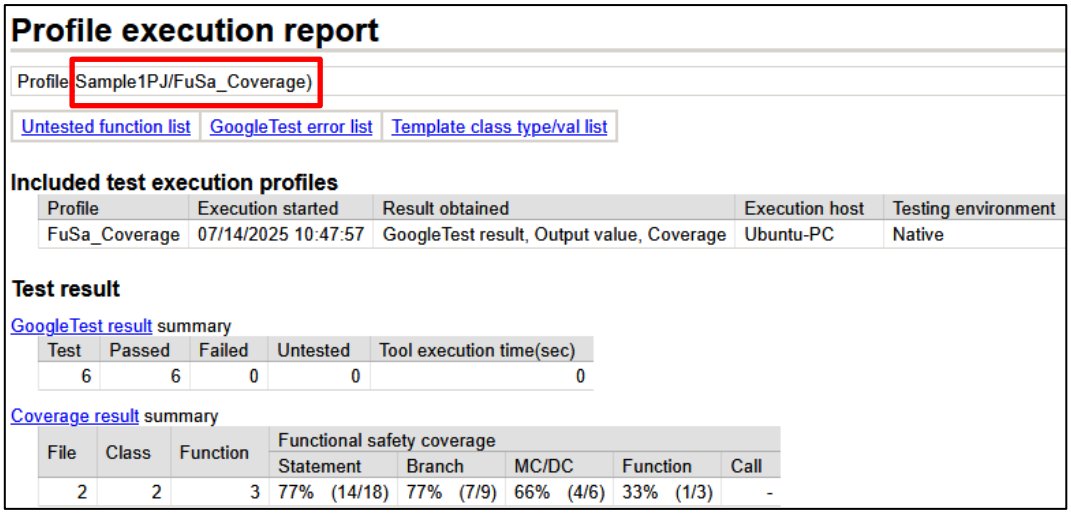
```
$ scp  -i (the path to the downloaded .pem file) -r ubuntu@(server IP address):
~/QTE_EvaluationKit/QTE_Tutorial/test/workspace/report .
```

Here is an example:

```
$ scp -i ./qte-trial-app-keypair.pem -r ubuntu@3.104.140.192:~/
QTE_EvaluationKit/QTE_Tutorial/test/workspace/report .
```

Let’ s open profile_execution_report.html on the Client machine's browser.

The red box displays the value 'Sample1PJ' that was set for the option 'project-name' of the project.json and ‘FuSa_Coverage’ that was set for the option ‘profile-name’ of the profile.json.



Profile execution report

Profile(Sample1PJ/FuSa_Coverage)

Untested function list | GoogleTest error list | Template class type/val list

Included test execution profiles

Profile	Execution started	Result obtained	Execution host	Testing environment
FuSa_Coverage	07/14/2025 10:47:57	GoogleTest result, Output value, Coverage	Ubuntu-PC	Native

Test result

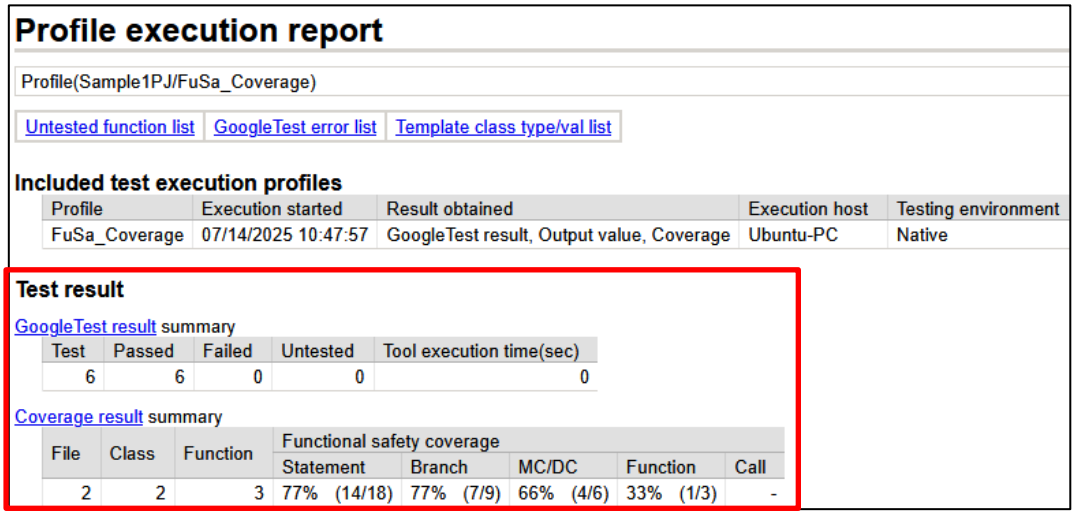
GoogleTest result summary

Test	Passed	Failed	Untested	Tool execution time(sec)
6	6	0	0	0

Coverage result summary

File	Class	Function	Functional safety coverage				
			Statement	Branch	MC/DC	Function	Call
2	2	3	77% (14/18)	77% (7/9)	66% (4/6)	33% (1/3)	-

Google Test Results will be displayed in the GoogleTest result section and Coverage Results will be displayed in the Coverage result section.



Profile execution report

Profile(Sample1PJ/FuSa_Coverage)

Untested function list | GoogleTest error list | Template class type/val list

Included test execution profiles

Profile	Execution started	Result obtained	Execution host	Testing environment
FuSa_Coverage	07/14/2025 10:47:57	GoogleTest result, Output value, Coverage	Ubuntu-PC	Native

Test result

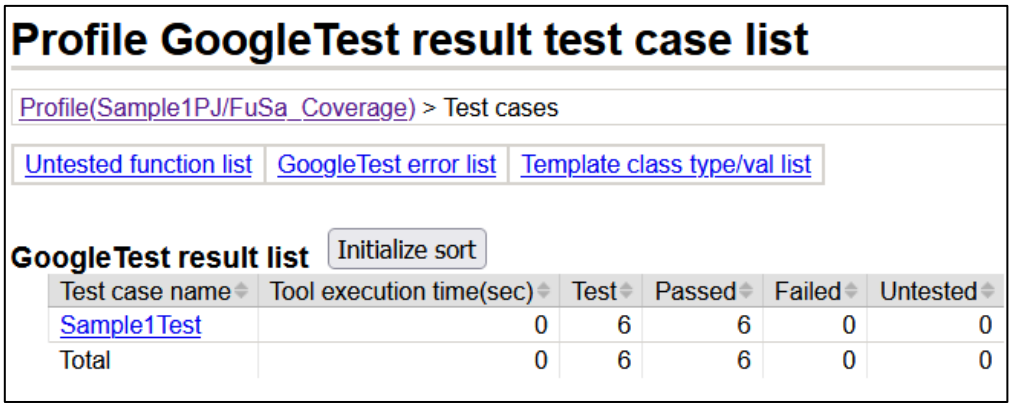
GoogleTest result summary

Test	Passed	Failed	Untested	Tool execution time(sec)
6	6	0	0	0

Coverage result summary

File	Class	Function	Functional safety coverage				
			Statement	Branch	MC/DC	Function	Call
2	2	3	77% (14/18)	77% (7/9)	66% (4/6)	33% (1/3)	-

Click the GoogleTest result link to open the detailed report with results for each Test case.



Profile GoogleTest result test case list

Profile(Sample1PJ/FuSa_Coverage) > Test cases

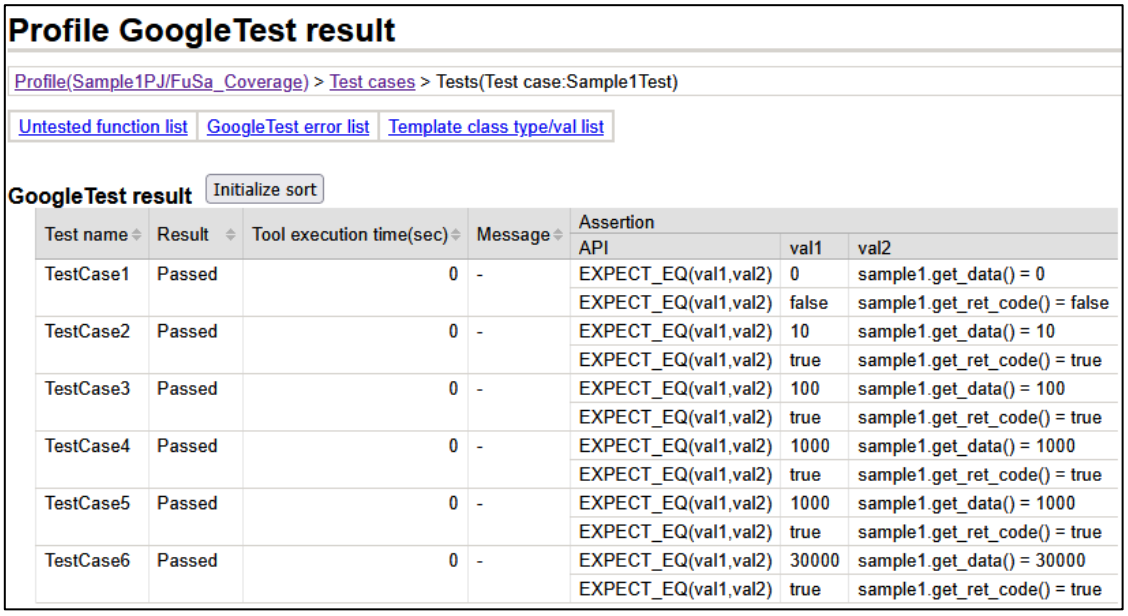
Untested function list | GoogleTest error list | Template class type/val list

GoogleTest result list Initialize sort

Test case name	Tool execution time(sec)	Test	Passed	Failed	Untested
Sample1Test	0	6	6	0	0
Total	0	6	6	0	0

The Google Test Results should look like this.

Click a Test case name to see the results of each Test Driver. For example, try with Sample1Test:



Profile GoogleTest result

Profile(Sample1PJ/FuSa_Coverage) > Test cases > Tests(Test case:Sample1Test)

Untested function list | GoogleTest error list | Template class type/val list

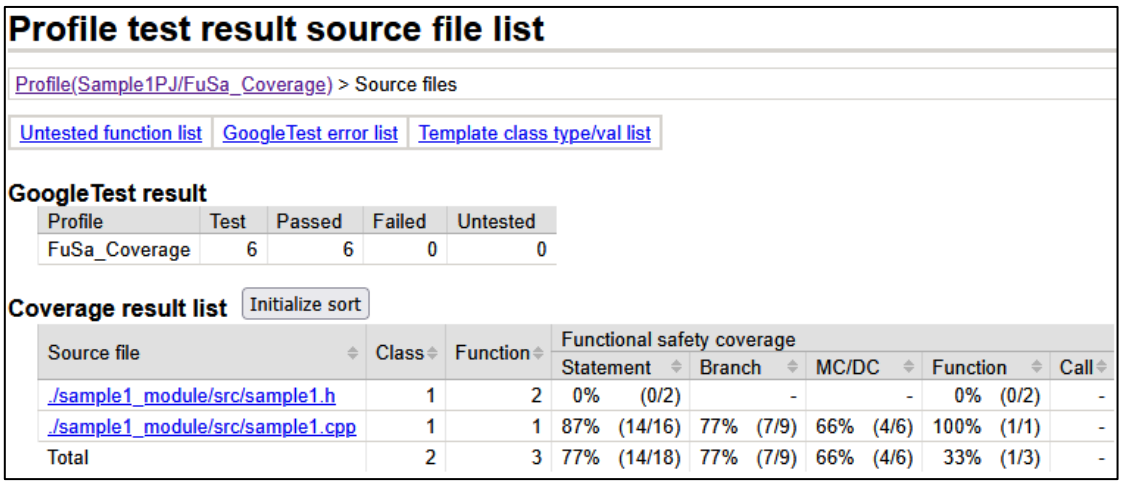
GoogleTest result Initialize sort

Test name	Result	Tool execution time(sec)	Message	Assertion		
				API	val1	val2
TestCase1	Passed	0	-	EXPECT_EQ(val1,val2)	0	sample1.get_data() = 0
				EXPECT_EQ(val1,val2)	false	sample1.get_ret_code() = false
TestCase2	Passed	0	-	EXPECT_EQ(val1,val2)	10	sample1.get_data() = 10
				EXPECT_EQ(val1,val2)	true	sample1.get_ret_code() = true
TestCase3	Passed	0	-	EXPECT_EQ(val1,val2)	100	sample1.get_data() = 100
				EXPECT_EQ(val1,val2)	true	sample1.get_ret_code() = true
TestCase4	Passed	0	-	EXPECT_EQ(val1,val2)	1000	sample1.get_data() = 1000
				EXPECT_EQ(val1,val2)	true	sample1.get_ret_code() = true
TestCase5	Passed	0	-	EXPECT_EQ(val1,val2)	1000	sample1.get_data() = 1000
				EXPECT_EQ(val1,val2)	true	sample1.get_ret_code() = true
TestCase6	Passed	0	-	EXPECT_EQ(val1,val2)	30000	sample1.get_data() = 30000
				EXPECT_EQ(val1,val2)	true	sample1.get_ret_code() = true

We can check the results of each Test Driver function/test case.

Next, check the coverage results. Click the Profile(Sample1PJ/FuSa_Coverage) link to return to the Profile Execution Report top page.

To check the Coverage at the Source File level, click Coverage result of the Profile Execution Report.



Profile test result source file list

Profile(Sample1PJ/FuSa_Coverage) > Source files

Untested function list | GoogleTest error list | Template class type/val list

GoogleTest result

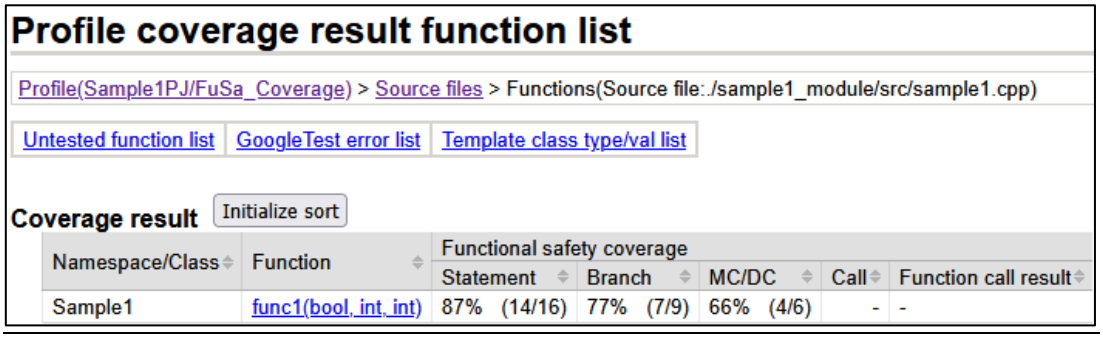
Profile	Test	Passed	Failed	Untested
FuSa_Coverage	6	6	0	0

Coverage result list Initialize sort

Source file	Class	Function	Functional safety coverage				
			Statement	Branch	MC/DC	Function	Call
./sample1_module/src/sample1.h	1	2	0% (0/2)	-	-	0% (0/2)	-
./sample1_module/src/sample1.cpp	1	1	87% (14/16)	77% (7/9)	66% (4/6)	100% (1/1)	-
Total	2	3	77% (14/18)	77% (7/9)	66% (4/6)	33% (1/3)	-

The Coverage per Source File is displayed here.

Click each Source File to see the Coverage Results for each function defined in it. For example, click ./sample1_module/src/sample1.cpp to see func1 results.



Profile coverage result function list

Profile(Sample1PJ/FuSa_Coverage) > Source files > Functions(Source file:./sample1_module/src/sample1.cpp)

Untested function list | GoogleTest error list | Template class type/val list

Coverage result Initialize sort

Namespace/Class	Function	Functional safety coverage				
		Statement	Branch	MC/DC	Call	Function call result
Sample1	func1(bool, int, int)	87% (14/16)	77% (7/9)	66% (4/6)	-	-

You can see here all the results for all functions defined in sample1.cpp.

Click a function name (here func1(bool, int, int)) to open the Coverage source viewer where Coverage is displayed on top of the code. For func1(bool, int, int) the result should look like this:

Line number	Functional safety coverage			Source code
	Statement	Branch	MC/DC	
1.				#include "sample1.h"
2.				
3.				void Sample1::func1(bool enable, int mode, int input)
4.				{
5.	6	T/F	T/F : enable	if (enable) {
6.	5	4/5		switch (mode) {
7.				case 0:
8.	1			data_ = input;
9.	1			break;
10.				case 1:
11.	1			data_ = input * 10;
12.	1			break;
13.				case 2:
14.	1			data_ = input * 100;
15.	1			break;
16.				case 3:
17.	2	/F	/F : input > 100 /F : input < 300	if ((input > 100) && (input < 300))
18.	0			data_ = 10000;
19.				else
20.	2			data_ = input * 100;
21.	2			break;
22.				default:
23.	0			data_ = -1;
24.				}
25.	5			ret_code_ = true;
26.				} else {
27.	1			data_ = 0;
28.	1			ret_code_ = false;
29.				}
30.				}

MC/DC Fully Covered
Branch Partially Covered
Line Not Executed

The Coverage for each Line can be viewed here. The SWITCH on L6 has 4 of his 5 cases covered. The red line L23 was not executed, showing the default CASE was not covered. The T/F in the Branch and MC/DC columns show which condition was TRUE or FALSE. L17 shows that the TRUE case is missing.

From the above coverage results, it is clear that test cases are insufficient.

Below is the Test driver for sample1Test.cpp.

```
TEST(Sample1Test, TestCase1)
{
   Sample1 sample1;

   sample1.func1(false, 0, 10);
   EXPECT_EQ(0, sample1.get_data());
   EXPECT_EQ(false, sample1.get_ret_code());
}
```

This Test verifies the values of data_ and ret_code_. The test cases for Sample1 are:

TestNo	Input			Output	
	enable	mode	input	data_	ret_code_
1	false	0	10	0	false
2	true	0	10	10	true
3	true	1	10	100	true
4	true	2	10	1000	true
5	true	3	10	1000	true
6	true	3	300	30000	true

From the results in profile_execution_report.html we see that some test cases are missing, so we add them:

TestNo	Input			Output	
	enable	mode	input	data_	ret_code_
7	true	3	200	10000	true
8	true	4	0	-1	true

Open source/sample1_module/src/sample1Test.cpp with a text editor and add the Tests No. 6 and 7 Test Drivers as shown below.

```
TEST(Sample1Test, TestCase7)
{
   Sample1 sample1;

   sample1.func1(true, 3, 200);
   EXPECT_EQ(10000, sample1.get_data());
   EXPECT_EQ(true, sample1.get_ret_code());
}

TEST(Sample1Test, TestCase8)
{
   Sample1 sample1;

   sample1.func1(true, 4, 0);
   EXPECT_EQ(-1, sample1.get_data());
   EXPECT_EQ(true, sample1.get_ret_code());
}
```

After saving sample1Test.cpp, re-run the prepare, build, and run-test commands.
Confirm that the MC/DC coverage has reached 100%.
This concludes Exercise 1.

5.3 Exercise 2 : Test a Template Class in Functional Safety Coverage mode

QTE measures coverage for each instance of Template Classe or Template functions. We will see this in Ex. 2 with Functional Safety Coverage mode. That will be run in the Native environment.

The Sample2 program is shown below. The Class name is Sample2.

```
template <typename T>
class Sample2 {
public:
    Sample2(T a, T b, T c, T d) : a_{ a }, b_{ b }, c_{ c } , d_{ d }
    {
    }

    bool func2(int code);
    T get_out() { return out_; };

private:
    T a_;
    T b_;
    T c_;
    T d_;
    T out_;
};
```

The Sample2 Class has func2 as member function, which is the target of our test. The last 2 lines instantiate func2 for the double and int data types.

```
template <typename T>
bool Sample2<T>::func2(int code)
{
    bool rtc = false;

    if (a_ > 10) {
        if (b_ > 20 && c_ > 30) {
            out_ = 0;
        } else {
            out_ = -1;
        }

        rtc = true;
    } else {
        switch (code) {
        case 1:
            out_ = 1;
            break;
        case 2:
            out_ = 2;
            break;
        case 3:
            out_ = 3;
            break;
        default:
            out_ = -1;
            break;
        }

        rtc = false;
    }

    a_ = out_;
    if (d_ == a_) {
        out_ = 4;
    } else {
        out_ = 5;
    }

    return rtc;
}

template class Sample2<double>;
template class Sample2<int>;
```

The steps are similar to Ex. 1 . Follow the steps described in the next section by running the same commands in a terminal.

5.3.1 Addition to the Makefile

As for Ex.1, source/sample2_module/src/Makefile must be edited so that the QTE makefile and variables are included. Open the Makefile with a ext editor and add the following (in red).

```
TARGET = sample2_test

(omitted)

%.o : %.cpp
	$(CXX) $(CXXFLAGS) -c -o $@ $<

sample2.o : sample2.cpp sample2.h

sample2Test.o : sample2Test.cpp sample2.h
	$(CXX) $(GTEST_INC_PATH) -pthread -c -o $@ $<

$(TARGET) : sample2.o sample2Test.o gtest.a gtest_main.a
	$(CXX) -o $@ $^ $(LDFLAGS) $(LDLIBS)
```

Insert "include ../../qte.make" in the first row

Add "$(QTE_SRC_FLAGS)"

Add "$(QTE_API)"

Add "$(QTE_LINK_FLAGS)"

With the Makefile ready, we can now run the test in Functional Safety Coverage mode.

5.3.2 Creating a project

Create a project in the workspace directory. Using "ProjectSample2" as the project name.

Since there is an option that uses the same value set in Exercise 1, ProjectSample2 is created by copying ProjectSample1 from Exercise 1.

Change the current directory to the workspace directory and create a project.

```
$ cd ~/QTE_EvaluationKit/QTE_Tutorial/test/workspace
$ cp -r ProjectSample1 ProjectSample2
```

The ProjectSample2 directory is created in the workspace directory. Run the following command to check that the ProjectSample 2 directory is created.

```
$ ls -F
ProjectSample1/     log/     source/
ProjectSample2/     report/  workspace.json
```

Next, we change the Project settings.

5.3.3 Project Settings

Only modify the settings related to names and paths from Ex.1.

The three settings below must be modified in the project.json file.

Option	Value
project-name	Sample2PJ
test-target-source-path-list	@EXTERNAL_SOURCE_DIR@/sample2_module/src/sample2.cpp
cov-target-header-path-list	@EXTERNAL_SOURCE_DIR@/sample2_module/src/sample2.h
gtest-source-path-list	@EXTERNAL_SOURCE_DIR@/sample2_module/src/sample2Test.cpp
build-env-executable-path-list	@BUILD_DIR@/sample2_module/src/sample2_test

Open project.json and add the values shown in red.

```
"project-name": {
    "value": "Sample2PJ"
},

"test-target-source-path-list": {
    "valueList": [
        "@EXTERNAL_SOURCE_DIR@/sample2_module/src/sample2.cpp"
    ]
},

"cov-target-header-path-list": {
    "valueList": [
        "@EXTERNAL_SOURCE_DIR@/sample2_module/src/sample2.h"
    ]
},

"gtest-source-path-list": {
    "valueList": [
        "@EXTERNAL_SOURCE_DIR@/sample2_module/src/sample2Test.cpp"
    ]
},

"build-env-executable-path-list": {
    "valueList": [
        "@BUILD_DIR@/sample2_module/src/sample2_test"
    ]
},
```

Only modify the settings related to names and paths from Ex.1.
The one settings below must be modified in the build.json file.

Option	Value
user-include-dir-list	@EXTERNAL_SOURCE_DIR@/ googletest/googletest/include
	@EXTERNAL_SOURCE_DIR@/sample2_module/src

Open build.json and add the values shown in red. Other fields can be left with the default value.

```
"user-include-dir-list": {
   "valueList": [
      "@EXTERNAL_SOURCE_DIR@/googletest/googletest/include",
      "@EXTERNAL_SOURCE_DIR@/sample2_module/src"
   ]
},
```

Next, we modify the Profile settings.

5.3.4 Profile Settings

In Exercise 2, the fusa_profile1 is used. Otherprofiles are not used.
We now edit the profile.json file with the following settings.

Option	Value
profile-name	FuSa_Coverage
execution-mode	3

Open profile.json and add the values shown in red.

```
"profile-name": {
   "value": "FuSa_Coverage"
},

"execution-mode": {
   "value": "3"
},
```

Next we now edit the testbinbuild.sh.

Open testbinbuild.sh in an editor and enter the red text at line 13.

```
#!/bin/bash

# This script is called by the 'qte-cov build' command.
# Usage: testbinbuild.sh <EXECUTION_MODE> <WORKSPACE_PATH> <PROJECT_PAT
H> <PROFILE_PATH> <EXTERNAL_SOURCE_DIR_PATH> <BUILD_DIR_PATH>
# Do not change the following six lines.
EXECUTION_MODE=$1
WORKSPACE=$2
PROJECT=$3
PROFILE=$4
EXTERNAL_SOURCE_DIR=$5
BUILD_DIR=$6

make -C ${BUILD_DIR}/sample2_module/src debug CC=gcc CXX=g++
```

Now that all settings are ready, we can run the test.

5.3.5 Running the Test

Run the test. Change the current directory to ProjectSample2/fusa_profile1 and run the prepare, build, and run-test commands to run the test.

```
$ cd ~/QTE_EvaluationKit/QTE_Tutorial/test/workspace/ProjectSample2/fusa_profile1
$ qte-cov prepare
  (omitted)
$ qte-cov build
  (omitted)
$ qte-cov run-test
  (omitted)
```

The Test should complete successfully. Let's check the results.

5.3.6 Checking the Test Results

First create the Project Execution Report with the create-project-execution-report command.

```
$ qte-cov create-profile-execution-report
**************************************************************
**** Create Profile Execution Report
**************************************************************
```

This command should create workspace/report/project_fs_reports/ProjectSample2/fusa_profile1 directory with profile_execution_report.html inside.

```
$ cd ../../report
$ ls -F project_fs_reports/ProjectSample2/fusa_profile1
coverage/                    project_execution_report_im.xml    result/
profile_execution_report.html    resources/
```

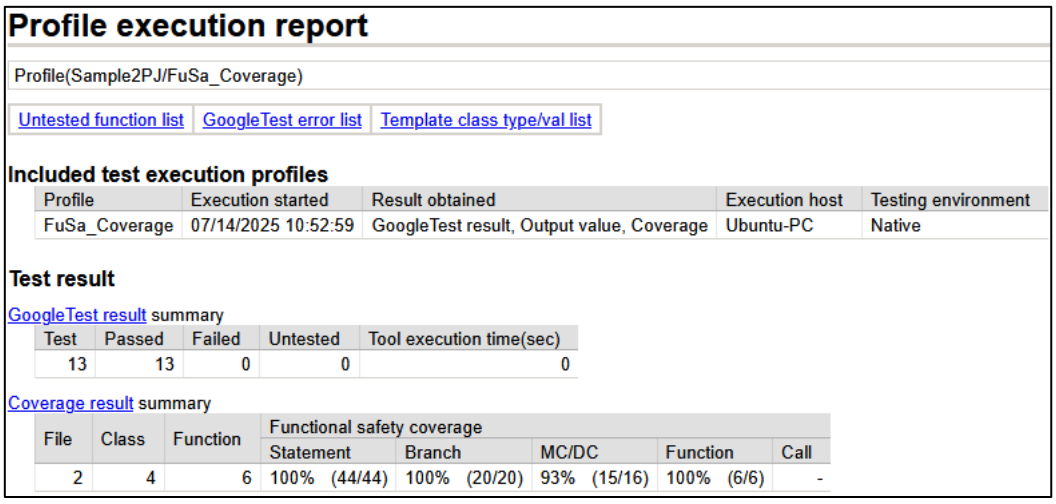
First, on your local environment, open a separate terminal (not connected via SSH), and run the following command to copy the file to your client machine. Replace the parts in parentheses with values appropriate for your environment.

```
$ scp  -i (the path to the downloaded .pem file) -r ubuntu@(server IP address):
~/QTE_EvaluationKit/QTE_Tutorial/test/workspace/report .
```

Here is an example:

```
$ scp -i ./qte-trial-app-keypair.pem -r ubuntu@3.104.140.192:~/
QTE_EvaluationKit/QTE_Tutorial/test/workspace/report .
```

Let’ s open profile_execution_report.html on the Client machine's browser.



Profile execution report

Profile(Sample2PJ/FuSa_Coverage)

Untested function list | GoogleTest error list | Template class type/val list

Included test execution profiles

Profile	Execution started	Result obtained	Execution host	Testing environment
FuSa_Coverage	07/14/2025 10:52:59	GoogleTest result, Output value, Coverage	Ubuntu-PC	Native

Test result

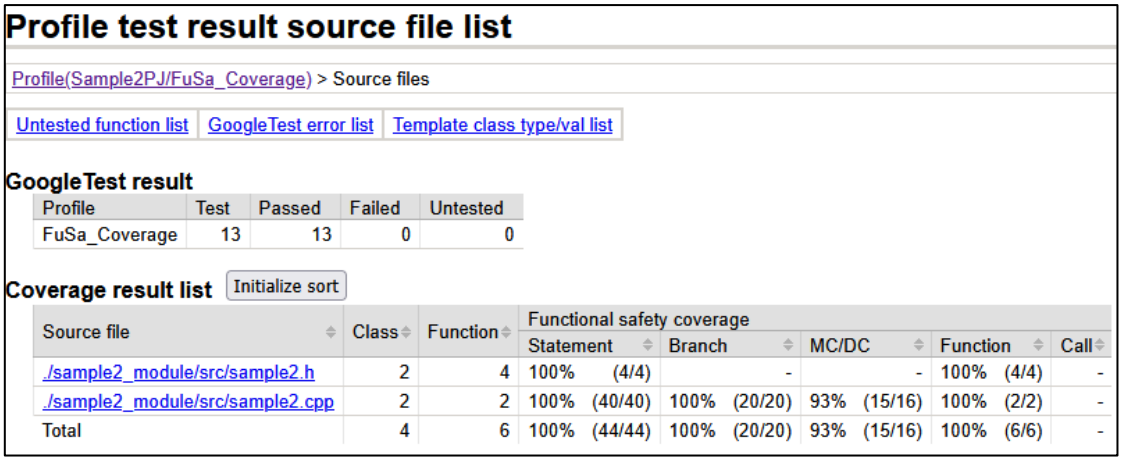
GoogleTest result summary

Test	Passed	Failed	Untested	Tool execution time(sec)
13	13	0	0	0

Coverage result summary

File	Class	Function	Functional safety coverage				
			Statement	Branch	MC/DC	Function	Call
2	4	6	100% (44/44)	100% (20/20)	93% (15/16)	100% (6/6)	-

Click the Coverage result.



Profile test result source file list

Profile(Sample2PJ/FuSa_Coverage) > Source files

Untested function list | GoogleTest error list | Template class type/val list

GoogleTest result

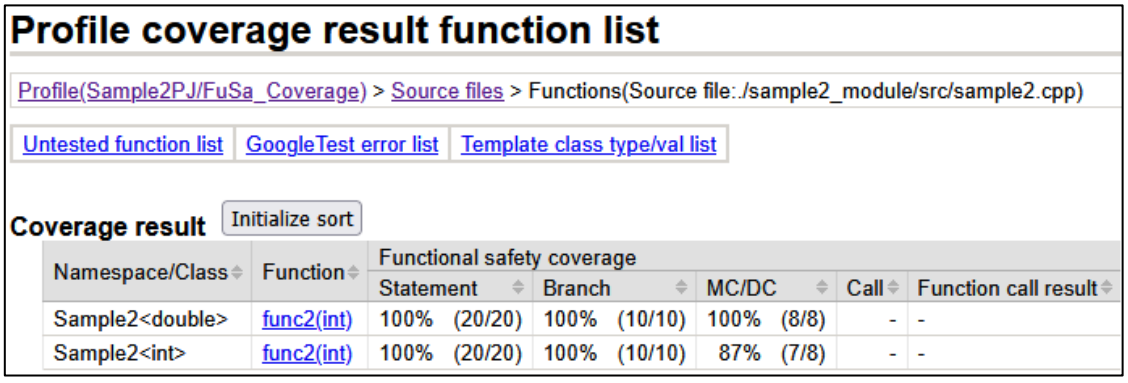
Profile	Test	Passed	Failed	Untested
FuSa_Coverage	13	13	0	0

Coverage result list Initialize sort

Source file	Class	Function	Functional safety coverage				
			Statement	Branch	MC/DC	Function	Call
./sample2_module/src/sample2.h	2	4	100% (4/4)	-	-	100% (4/4)	-
./sample2_module/src/sample2.cpp	2	2	100% (40/40)	100% (20/20)	93% (15/16)	100% (2/2)	-
Total	4	6	100% (44/44)	100% (20/20)	93% (15/16)	100% (6/6)	-

Check the Test Results summary.

To see the Coverage results of sample2.cpp, click ./sample2_module/src/sample2.cpp.



Profile coverage result function list

Profile(Sample2PJ/FuSa_Coverage) > Source files > Functions(Source file:./sample2_module/src/sample2.cpp)

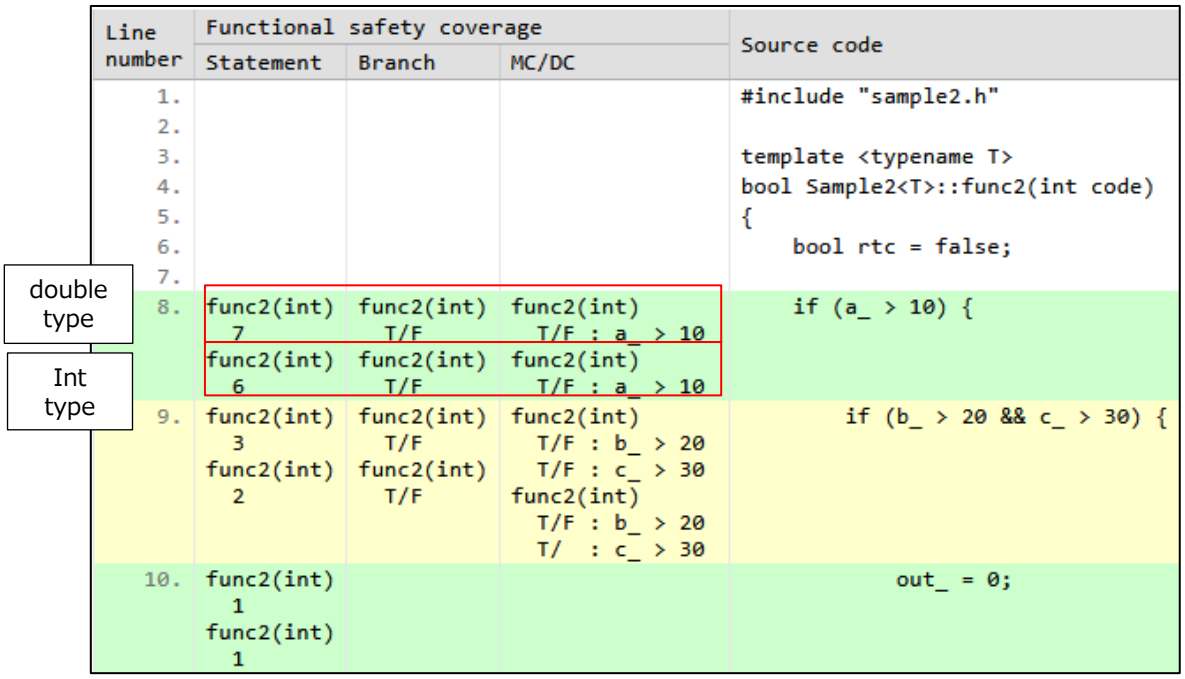
Untested function list | GoogleTest error list | Template class type/val list

Coverage result Initialize sort

Namespace/Class	Function	Functional safety coverage				
		Statement	Branch	MC/DC	Call	Function call result
Sample2<double>	func2(int)	100% (20/20)	100% (10/10)	100% (8/8)	-	-
Sample2<int>	func2(int)	100% (20/20)	100% (10/10)	87% (7/8)	-	-

All functions defined in sample2.cpp are shown here with their Coverage results.

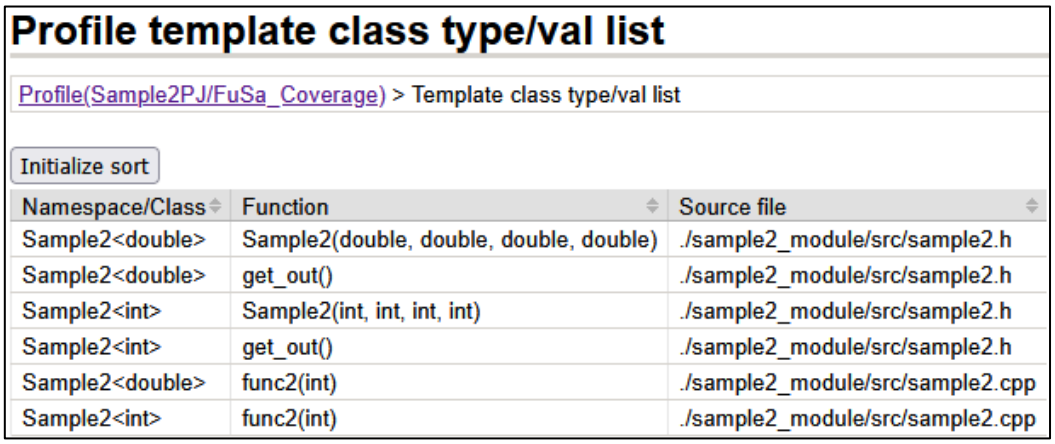
In the Coverage results the two objects for the int and double type of func2 are displayed separately. We see here that MC/DC Coverage is different between the two types. Click either of the func2(int) links to open the detailed results for that function (both links lead to the same page).



Line number	Functional safety coverage			Source code
	Statement	Branch	MC/DC	
1.				#include "sample2.h"
2.				
3.				template <typename T>
4.				bool Sample2<T>::func2(int code)
5.				{
6.				bool rtc = false;
7.				
8.	func2(int) 7 func2(int) 6	func2(int) T/F func2(int) T/F	func2(int) T/F : a_ > 10 func2(int) T/F : a_ > 10	if (a_ > 10) {
9.	func2(int) 3 func2(int) 2	func2(int) T/F func2(int) T/F	func2(int) T/F : b_ > 20 T/F : c_ > 30 func2(int) T/F : b_ > 20 T/ : c_ > 30	if (b_ > 20 && c_ > 30) {
10.	func2(int) 1 func2(int) 1			out_ = 0;

For L8-9-10 of the Functional SafetyCoverage for func2(int), two coverage results are shown, one for each data type: the upper line for int and the lower line for double. The order of the data types is the same as in the Project Coverage Results Function List.

Click the Template class type/val list link at the top to display a list of Classes and Member Functions with instantiated Template variables.



Profile template class type/val list

Profile(Sample2PJ/FuSa_Coverage) > Template class type/val list

Initialize sort

Namespace/Class	Function	Source file
Sample2<double>	Sample2(double, double, double, double)	./sample2_module/src/sample2.h
Sample2<double>	get_out()	./sample2_module/src/sample2.h
Sample2<int>	Sample2(int, int, int, int)	./sample2_module/src/sample2.h
Sample2<int>	get_out()	./sample2_module/src/sample2.h
Sample2<double>	func2(int)	./sample2_module/src/sample2.cpp
Sample2<int>	func2(int)	./sample2_module/src/sample2.cpp

From the int Template results we can see that the func2 Coverage is incomplete, so more test cases are needed (or the program is incorrect).

Open sample2_module/src/sample2Test.cpp and add the following Test Case

```
TEST(Sample2Test, TestInt2)
{
    Sample2<int> sample2(11, 21, 30, 0);

    bool rtc = sample2.func2(0);
    EXPECT_EQ(true, rtc);
    EXPECT_EQ(5, sample2.get_out());
}
```

Save sample2Test.cpp and re-run the prepare/build/run-test commands.
You should now have 100% MC/DC Coverage

This concludes Exercise 2 and this tutorial on how to use QTE and its Coverage Results.

6 Checking proceduce of Tool Validation Check

This chapter confirms that the automatically generated assertion code does not affect the original source code or objects (tool effect check). In Exercise 3, we will run the tests in Tool Validation Check Feature.

6.1 Creating the Execution environment

We will keep using the workspace used in Ex. 1 and 2, so no workspace-related steps are needed.

6.2 Exercise 3 : Tool Validation Check

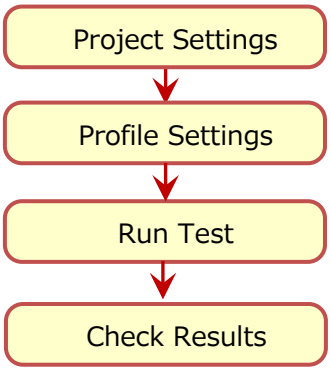
In Exercise 3, you will learn how to check the results when using the tool impact check function.

In this exercise, you will learn how to run the Sample 1 program in a native PC environment, running the object that does not contain the code for coverage measurement, respectively, and compare the results with the object containing the code for coverage measurement.

In Exercise 3, some steps from Ex.1 are not needed because the same project is used.

Since the same program Sample1 is used as in Exercise 1, you can use the same Makefile as in Exercise 1.

The steps are:



In Exercise 3, some steps from Ex.1 are not needed because the same Sample1 sourc e, workspace and projet is used.

6.2.1 Project Settings

We modify the settings in project.json with the following values;

Option	Value
gtest-result-profile-dirname	fusa_profile2

To do this open project.json with a text editor and add the following (in red).

```
"gtest-result-profile-dirname": {
   "value": "fusa_profile2"
},
```

6.2.2 Profile Settings

In Exercise 3, the fusa_profile2 is used. Otherprofiles are not used.

We modify the settings in profile.json with the following values;

Option	Value
project-name	NoCoverage
execution-mode	99

To do this open profile.json with a text editor and add the following (in red).

```
"profile-name": {
   "value": "NoCoverage"
},

"execution-mode": {
   "value": "99"
},
```

Next, configure testbinbuild.sh.

Next use testbinbuild.sh the same setting as for Ex.1. For this reason we will copy testbinbuild.sh from Ex.1.

```
$ cd ~/QTE_EvaluationKit/QTE_Tutorial/test/workspace/ProjectSample1
$ cp fusa_profile1/testbinbuild.sh fusa_profile2
```

After the Project settings, we can run the Test.

6.2.3 Running the Test

How to run a test is the same as in Ex.1.

Change the current directory to the fusa_profile2 directory and run the following commands, starting with running the prepare, build, and run-test commands.

```
(* The following console outputs of the commands are omitted.)
$ cd ~/QTE_EvaluationKit/QTE_Tutorial/test/workspace/ProjectSample1/fusa_profile2
$ qte-cov prepare
$ qte-cov build
$ qte-cov run-test
```

The Test should complete successfully. Let's check the results.

6.2.4 Checking the Test Results

When performing the tool impact check, compare the result of executing the object that does not contain the code for coverage measurement with the result of executing the object containing the code for coverage measurement in Exercise 1.

```
$ cd ../
$ qte-cov compare fusa_profile1/ fusa_profile2/
**************************************************************
**** Running Compare
**************************************************************
Comparing the results by full-digit exact match of variable precision.
Comparing the results by round-off (round-up/down) of variable precision.
No differences were found in the compared projects.
(omitted)
```

Check the comparison results. In the terminal output, 'No differences were found in the compared projects.' indicates that there were no differences in the results. Therefore, it can be confirmed that similar results are produced in both the object containing the code for coverage measurement and the object that does not contain the code for coverage measurement.

Next Test Results can be checked in the Project Execution Report create by the create-project-execution-report command.

```
$ qte-cov create-project-execution-report
**************************************************************
**** Create Project Execution Report
**************************************************************
(omitted)
```

The create-project-execution-report command creates a directory with the project's name under report/project_name_reports in workspace and generates project_execution_report.html in that directory.

```
$ cd ../report/project_fs_reports
$ tree -L 1 ProjectSample1/
ProjectSample1
|-- coverage
|-- diff
|-- fusa_profile1
|-- optionfile
|-- project_execution_report.html
|-- project_execution_report.xml
|-- project_execution_report_im.xml
|-- resources
`-- result
```

First, on your local environment, open a separate terminal (not connected via SSH), and run the following command to copy the file to your client machine. Replace the parts in parentheses with values appropriate for your environment.

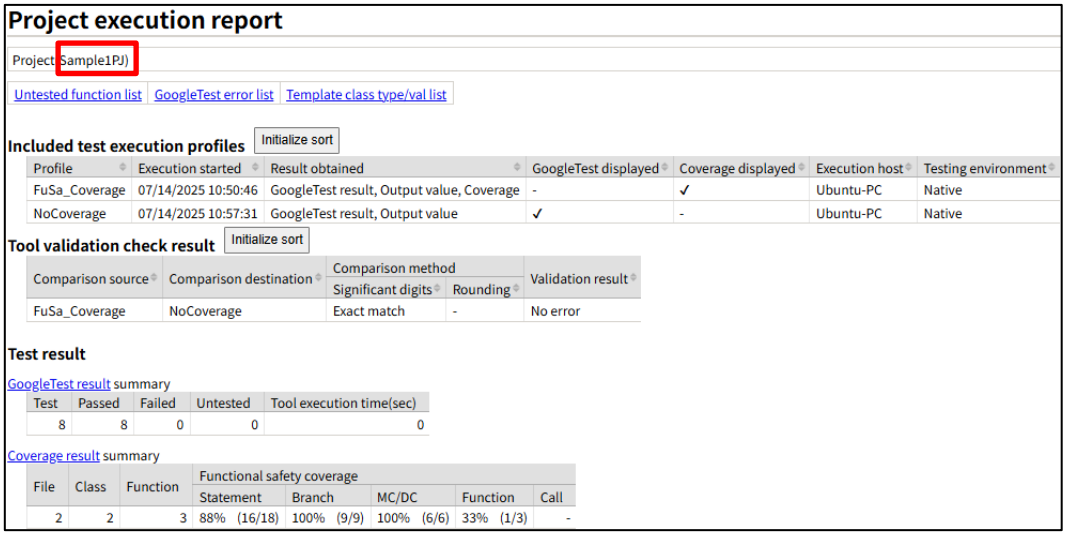
```
$ scp  -i (the path to the downloaded .pem file) -r ubuntu@(server IP address):
~/QTE_EvaluationKit/QTE_Tutorial/test/workspace/report .
```

Here is an example:

```
$ scp -i ./qte-trial-app-keypair.pem -r ubuntu@3.104.140.192:~/
QTE_EvaluationKit/QTE_Tutorial/test/workspace/report .
```

Let’ s open project_execution_report.html on the Client machine's browser.

The value "Sample1PJ" configured in the "project-name" option of project.json is displayed in the red frame.



Project execution report

Project(Sample1PJ)

Untested function list | GoogleTest error list | Template class type/val list

Included test execution profiles Initialize sort

Profile	Execution started	Result obtained	GoogleTest displayed	Coverage displayed	Execution host	Testing environment
FuSa_Coverage	07/14/2025 10:50:46	GoogleTest result, Output value, Coverage	-	✓	Ubuntu-PC	Native
NoCoverage	07/14/2025 10:57:31	GoogleTest result, Output value	✓	-	Ubuntu-PC	Native

Tool validation check result Initialize sort

Comparison source	Comparison destination	Comparison method		Validation result
		Significant digits	Rounding	
FuSa_Coverage	NoCoverage	Exact match	-	No error

Test result

GoogleTest result summary

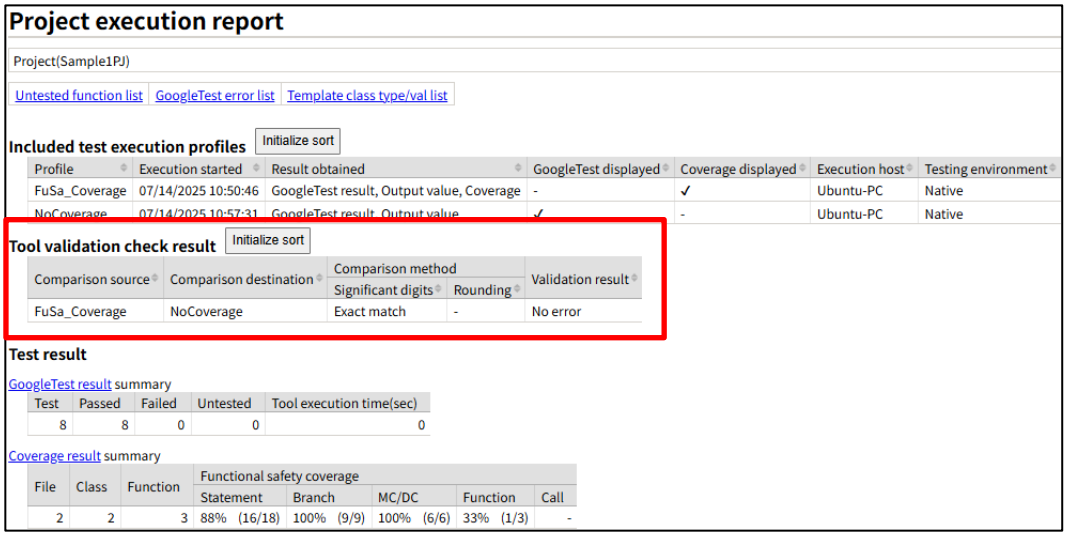
Test	Passed	Failed	Untested	Tool execution time(sec)
8	8	0	0	0

Coverage result summary

File	Class	Function	Functional safety coverage				
			Statement	Branch	MC/DC	Function	Call
2	2	3	88% (16/18)	100% (9/9)	100% (6/6)	33% (1/3)	-

The tool impact check results display a list of comparison results obtained using the compare command.

The result of comparing the test results of the object containing the code for coverage measurement and the object that does not contain the code for coverage measurement is displayed in the impact check results. Since there are no differences in the test results, 'No errors' is shown.



Project execution report

Project(Sample1PJ)

Untested function list | GoogleTest error list | Template class type/val list

Included test execution profiles Initialize sort

Profile	Execution started	Result obtained	GoogleTest displayed	Coverage displayed	Execution host	Testing environment
FuSa_Coverage	07/14/2025 10:50:46	GoogleTest result, Output value, Coverage	-	✓	Ubuntu-PC	Native
NoCoverage	07/14/2025 10:57:31	GoogleTest result, Output value	✓	-	Ubuntu-PC	Native

Tool validation check result Initialize sort

Comparison source	Comparison destination	Comparison method		Validation result
		Significant digits	Rounding	
FuSa_Coverage	NoCoverage	Exact match	-	No error

Test result

GoogleTest result summary

Test	Passed	Failed	Untested	Tool execution time(sec)
8	8	0	0	0

Coverage result summary

File	Class	Function	Functional safety coverage				
			Statement	Branch	MC/DC	Function	Call
2	2	3	88% (16/18)	100% (9/9)	100% (6/6)	33% (1/3)	-

This concludes exercise on how to use Tool Validation Check Feature.

7 Checking Integrated Report

QTE can integrate test results from multiple projects/workspaces and output HTML reports.

There are four types of integrated reports.

- Workspace integrated test report

  This is where the results of the testing of the projects in the workspace are summarized. It serves as evidence as functional safety standards.

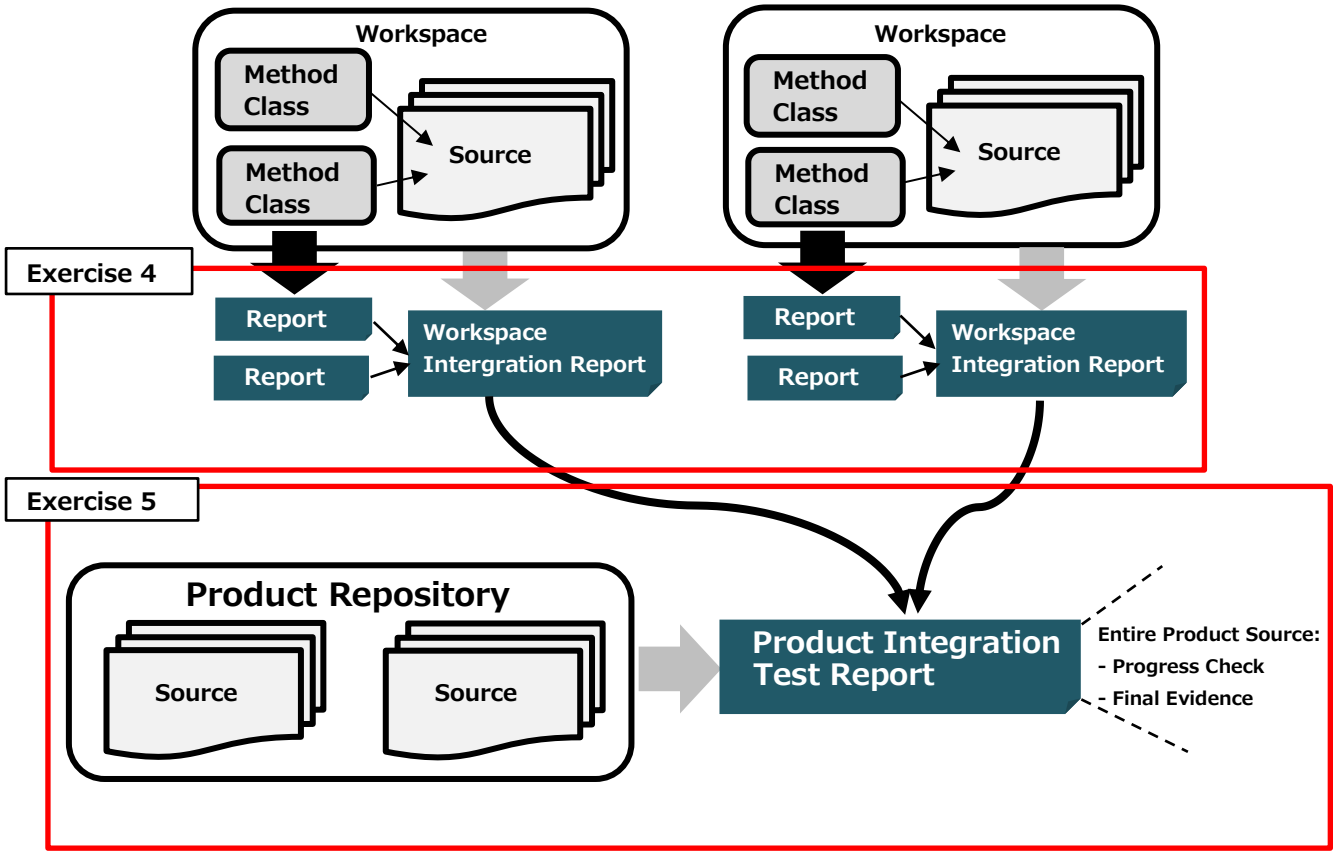
- Product integrated test report

  This is a compilation of test results from multiple workspaces (products) which will serve as evidence as functional safety standards.

In the practical training, an application consisting of four modules, sample1, sample2, sample3, and sample4, will be tested, and the overall test results will be confirmed in an integration report.

The following flow will be used to create the integration report.

In the practical training, you will output and check the Integration Report in the following flow according to the flow in actual use.



7.1 Exercise 4: Check Workspace Integrated Test Report

Create a Workspace Integration Test Report and review the test results for the entire workspace. Follow the tutorial and enter commands into the terminal.

In this exercise, the workspace created and tested in Exercises 1 and 2 will be used.

To create the Workspace Integration Test Report, the Project Execution Report from each project is required. Since the Project Execution Report for the project used in Exercise 2 has not been created, run the create-product-integrated-report command in the Exercise 2 project to create a Project Execution Report.

```
$ cd ~/QTE_EvaluationKit/QTE_Tutorial/test/workspace/ProjectSample2
$ qte-cov create-project-execution-report
************************************************************
**** Create Project Execution Report
************************************************************
(omitted)
```

Output the overall test results of the Workspace to a Workspace Integration Test Report. To create a Workspace Integration Test Report, run the create-workspace-integrated-report command.

```
$ cd ~/QTE_EvaluationKit/QTE_Tutorial/test/workspace
$ qte-cov create-workspace-integrated-report
************************************************************
**** Create Workspace Integrated Report
************************************************************
Target project for reporting: ProjectSample1, ProjectSample2
(omitted)
```

The create-integrated-workspace-report command creates a directory under report/project_name_reports in workspace and generates workspace_integrated_report.html in that directory.

```
$ tree -L 1 report/project_fs_reports/
report/project_fs_reports/
|-- ProjectSample1
|-- ProjectSample2
|-- integration
|-- resources
|-- workspace_integrated_report.html
|-- workspace_integrated_report.xml
`-- workspace_integrated_report_im.xml
```

First, on your local environment, open a separate terminal (not connected via SSH), and run the following command to copy the file to your client machine. Replace the parts in parentheses with values appropriate for your environment.

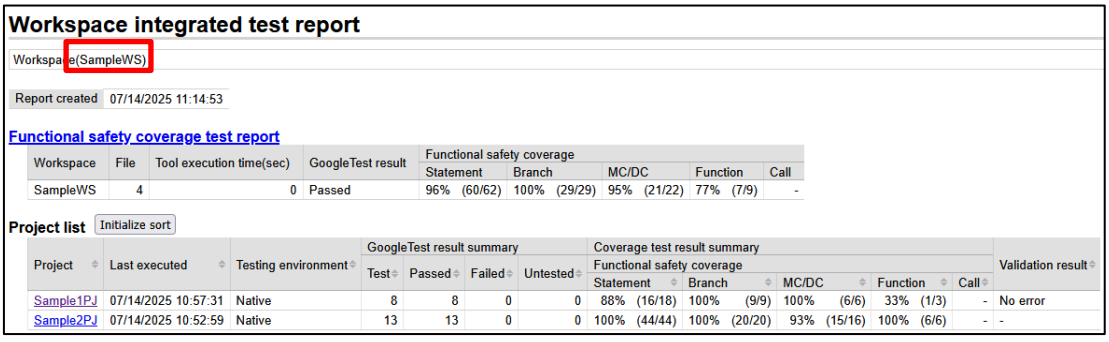
```
$ scp  -i (the path to the downloaded .pem file) -r ubuntu@(server IP address):~/QTE_EvaluationKit/QTE_Tutorial/test/workspace/report .
```

Here is an example:

```
$ scp -i ./qte-trial-app-keypair.pem -r ubuntu@3.104.140.192:~/QTE_EvaluationKit/QTE_Tutorial/test/workspace/report .
```

Let's open workspace_integrated_report.html on the Client machine's browser.

The red box displays the value 'SampleWS' that was set for the option 'workspace-name' of the workspace.json.



Workspace integrated test report

Workspace(SampleWS)

Report created 07/14/2025 11:14:53

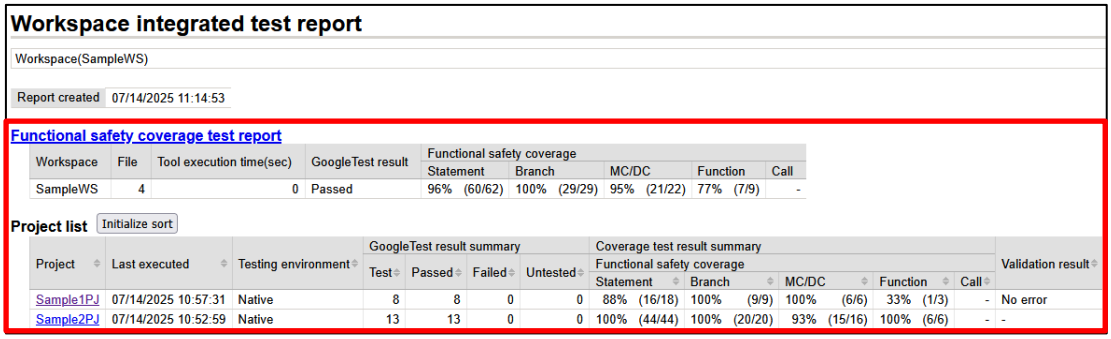
Functional safety coverage test report

Workspace	File	Tool execution time(sec)	GoogleTest result	Functional safety coverage				
				Statement	Branch	MC/DC	Function	Call
SampleWS	4	0	Passed	96% (60/62)	100% (29/29)	95% (21/22)	77% (7/9)	-

Project list Initialize sort

Project	Last executed	Testing environment	GoogleTest result summary				Coverage test result summary					Validation result
			Test	Passed	Failed	Untested	Functional safety coverage					
							Statement	Branch	MC/DC	Function	Call	
Sample1PJ	07/14/2025 10:57:31	Native	8	8	0	0	88% (16/18)	100% (9/9)	100% (6/6)	33% (1/3)	-	No error
Sample2PJ	07/14/2025 10:52:59	Native	13	13	0	0	100% (44/44)	100% (20/20)	93% (15/16)	100% (6/6)	-	-

Summary information of the test results of the integrated projects is displayed in the **Functional safety coverage test report**, and the test results of each project are displayed in the **Project list**.



Workspace integrated test report

Workspace(SampleWS)

Report created 07/14/2025 11:14:53

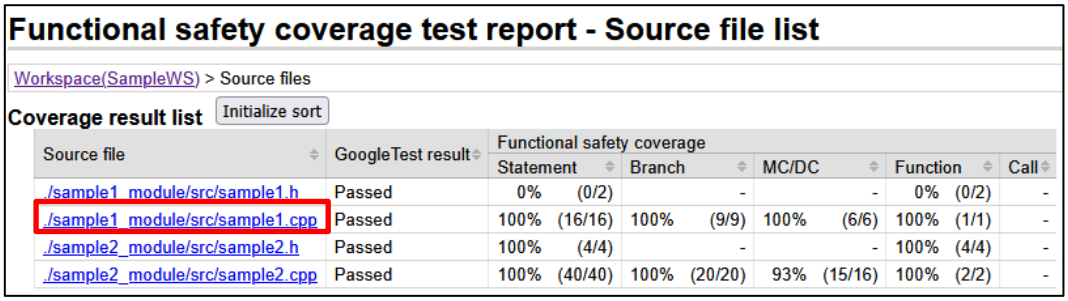
Functional safety coverage test report

Workspace	File	Tool execution time(sec)	GoogleTest result	Functional safety coverage							
				Statement		Branch		MC/DC		Function	Call
SampleWS	4	0	Passed	96%	(60/62)	100%	(29/29)	95%	(21/22)	77% (7/9)	-

Project list Initialize sort

Project	Last executed	Testing environment	GoogleTest result summary				Coverage test result summary								Validation result
			Test	Passed	Failed	Untested	Functional safety coverage								
							Statement		Branch		MC/DC		Function	Call	
Sample1PJ	07/14/2025 10:57:31	Native	8	8	0	0	88%	(16/18)	100%	(9/9)	100%	(6/6)	33% (1/3)	-	No error
Sample2PJ	07/14/2025 10:52:59	Native	13	13	0	0	100%	(44/44)	100%	(20/20)	93%	(15/16)	100% (6/6)	-	-

To check the test results at the Source File level, click the Functional safety coverage test report. Pass/fail and coverage results of test execution are displayed for each source file.



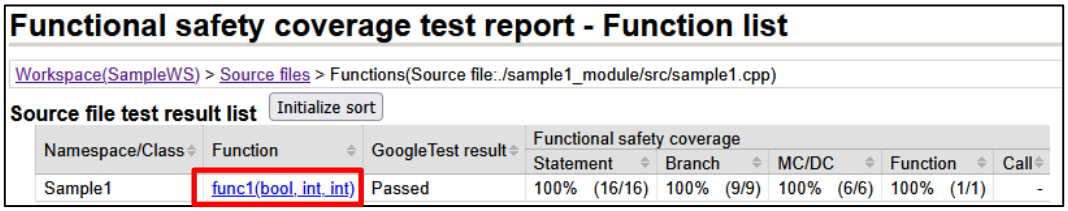
Functional safety coverage test report - Source file list

Workspace(SampleWS) > Source files

Coverage result list Initialize sort

Source file	GoogleTest result	Functional safety coverage								
		Statement		Branch		MC/DC		Function		Call
./sample1_module/src/sample1.h	Passed	0%	(0/2)	-		-		0%	(0/2)	-
./sample1_module/src/sample1.cpp	Passed	100%	(16/16)	100%	(9/9)	100%	(6/6)	100%	(1/1)	-
./sample2_module/src/sample2.h	Passed	100%	(4/4)	-		-		100%	(4/4)	-
./sample2_module/src/sample2.cpp	Passed	100%	(40/40)	100%	(20/20)	93%	(15/16)	100%	(2/2)	-

To see the test results of the functions defined in the source files, click on the file nam e. To view the coverage results for sample1.cpp, click on. Click on ./sample1_module/src/sample1.cpp.



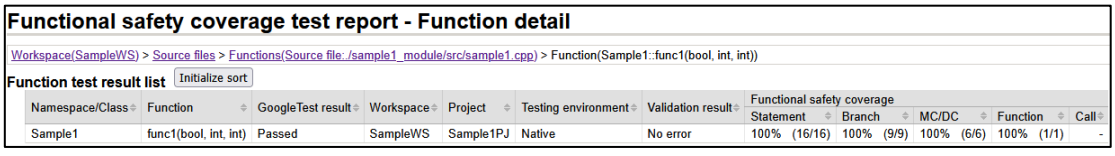
Functional safety coverage test report - Function list

Workspace(SampleWS) > Source files > Functions(Source file:./sample1_module/src/sample1.cpp)

Source file test result list Initialize sort

Namespace/Class	Function	GoogleTest result	Functional safety coverage								
			Statement		Branch		MC/DC		Function		Call
Sample1	func1(bool, int, int)	Passed	100%	(16/16)	100%	(9/9)	100%	(6/6)	100%	(1/1)	-

Click on the function func1(bool, int, int) to see the test results for each project of the function.



Functional safety coverage test report - Function detail

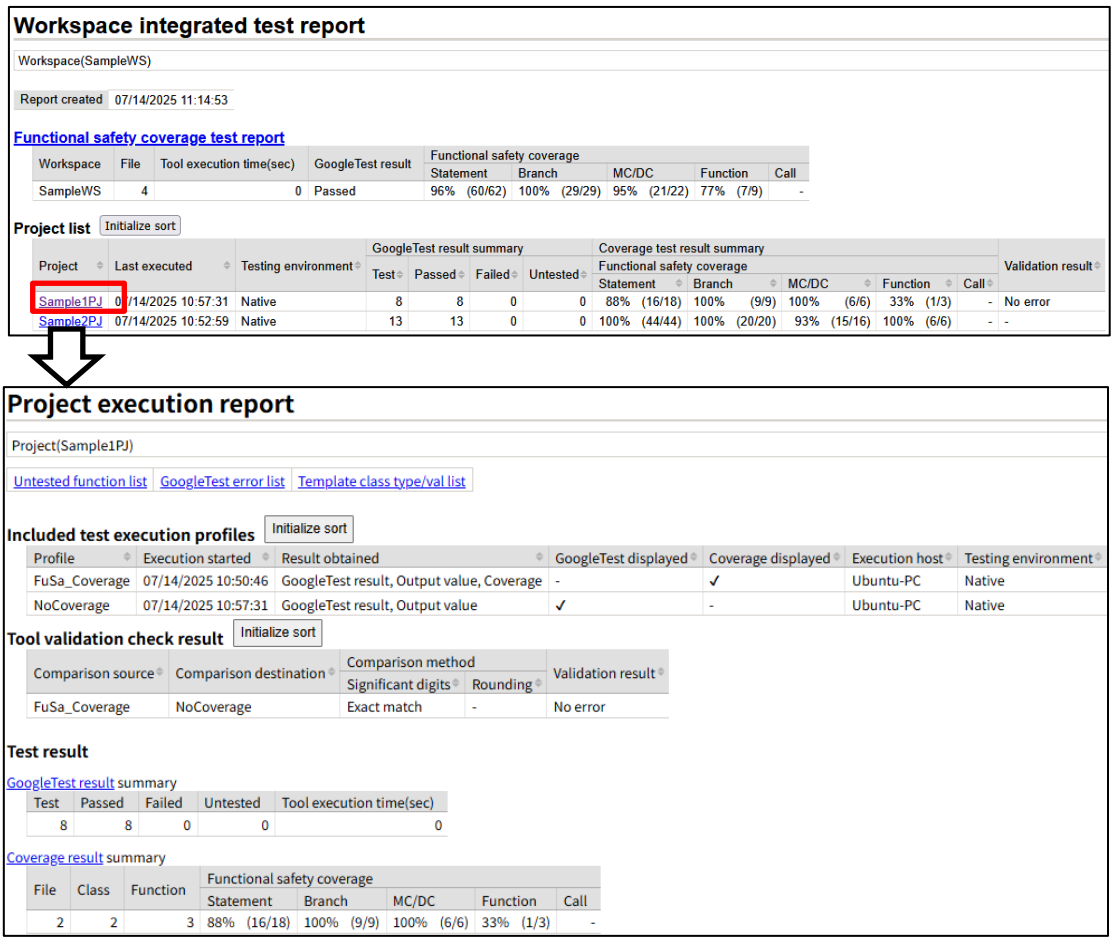
Workspace(SampleWS) > Source files > Functions(Source file: ./sample1_module/src/sample1.cpp) > Function(Sample1::func1(bool, int, int))

Function test result list Initialize sort

Namespace/Class	Function	GoogleTest result	Workspace	Project	Testing environment	Validation result	Functional safety coverage									
							Statement		Branch		MC/DC		Function		Call	
Sample1	func1(bool, int, int)	Passed	SampleWS	Sample1PJ	Native	No error	100%	(16/16)	100%	(9/9)	100%	(6/6)	100%	(1/1)	-	

Next, check the test results of each project. Click the Workspace(SampleWS) link to return to the Workspace Integration Test Report's top page.

To check the test results, click the project name in the project list of the Workspace Integration Test Report.



Workspace integrated test report

Workspace(SampleWS)

Report created 07/14/2025 11:14:53

Functional safety coverage test report

Workspace	File	Tool execution time(sec)	GoogleTest result	Functional safety coverage				
				Statement	Branch	MC/DC	Function	Call
SampleWS	4	0	Passed	96% (60/62)	100% (29/29)	95% (21/22)	77% (7/9)	-

Project list Initialize sort

Project	Last executed	Testing environment	GoogleTest result summary				Coverage test result summary					Validation result
			Test	Passed	Failed	Untested	Functional safety coverage					
							Statement	Branch	MC/DC	Function	Call	
Sample1PJ	07/14/2025 10:57:31	Native	8	8	0	0	88% (16/18)	100% (9/9)	100% (6/6)	33% (1/3)	-	No error
Sample2PJ	07/14/2025 10:52:59	Native	13	13	0	0	100% (44/44)	100% (20/20)	93% (15/16)	100% (6/6)	-	-

Project execution report

Project(Sample1PJ)

Untested function list | GoogleTest error list | Template class type/val list

Included test execution profiles Initialize sort

Profile	Execution started	Result obtained	GoogleTest displayed	Coverage displayed	Execution host	Testing environment
FuSa_Coverage	07/14/2025 10:50:46	GoogleTest result, Output value, Coverage	-	✓	Ubuntu-PC	Native
NoCoverage	07/14/2025 10:57:31	GoogleTest result, Output value	✓	-	Ubuntu-PC	Native

Tool validation check result Initialize sort

Comparison source	Comparison destination	Comparison method		Validation result
		Significant digits	Rounding	
FuSa_Coverage	NoCoverage	Exact match	-	No error

Test result

GoogleTest result summary

Test	Passed	Failed	Untested	Tool execution time(sec)
8	8	0	0	0

Coverage result summary

File	Class	Function	Functional safety coverage				
			Statement	Branch	MC/DC	Function	Call
2	2	3	88% (16/18)	100% (9/9)	100% (6/6)	33% (1/3)	-

Clicking on the link will display the Project Execution Report for ProjectSample1. As you can see, the Project List of the Workspace Integration Test Report is a link to the Project Execution Report of each project confirmed in Practice 3.

This completes Exercise 4.

7.2 Exercise 5 : Check Product Integrated Test Report

Create a Product Integration Test Report. Follow the tutorial and enter the commands in the terminal.

In order to create a product integration test report, it is necessary to create a workspace integration test report for the workspace to be integrated. In this exercise, in addition to the workspace workspace used in Exercise 4, two workspaces (workspace2 and workspace3) will be used. workspace2 and workspace3 are workspaces for which the workspace integration test report has already been created

Therefore, you can use these workspaces as they are, without the need to run tests.

To create a Product Integration Test Report, it is necessary to compile the Workspace Integration Test Report to be integrated in one directory.

Create a directory to consolidate workspace integration test reports.

```
$ cd ~/QTE_EvaluationKit/QTE_Tutorial/test
$ mkdir product_integrated_report
```

Next, each Workspace Integration Test Report is copied to the directory where it was created. To avoid duplicate directory names, change the directory name of the Workspace Integration Test Report to the workspace name when copying.

```
$ cp -r workspace/report/project_fs_reports product_integrated_report/workspace
$ cp -r workspace2/report/project_fs_reports product_integrated_report/workspace2
$ cp -r workspace3/report/project_fs_reports product_integrated_report/workspace3
```

Now that you are ready to create a Product Integration Test Report, run the create-product-integrated-report command to create a Product Integration Test Report.

```
$ qte-cov create-product-integrated-report product_integrated_report/ --product-name SampleProduct
**********************************************************************
**** Create Product Integrated Report
**********************************************************************
Input directories for Product Integrated Report: workspace, workspace2, workspace3
(omitted)
```

The create-product-integrated-report command will output product_integrated_report.html in the product_integrated_report directory.

```
$ tree -L 1 product_integrated_report/
product_integrated_report/
|-- integration
|-- product_integrated_report.html
|-- product_integrated_report.xml
|-- product_integrated_report_im.xml
|-- resources
|-- workspace
|-- workspace2
`-- workspace3
```

First, on your local environment, open a separate terminal (not connected via SSH), and run the following command to copy the file to your client machine. Replace the parts in parentheses with values appropriate for your environment.

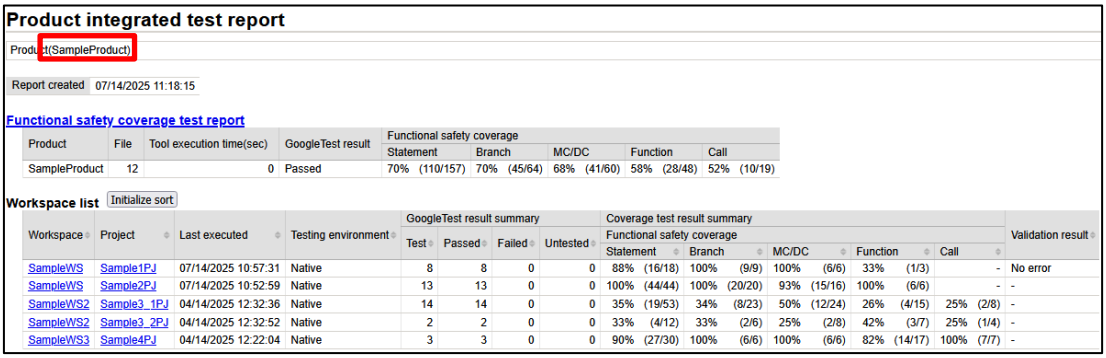
```
$ scp  -i (the path to the downloaded .pem file) -r ubuntu@(server IP address):
~/QTE_EvaluationKit/QTE_Tutorial/test/ .
```

Here is an example:

```
$ scp -i ./qte-trial-app-keypair.pem -r ubuntu@3.104.140.192:~/
QTE_EvaluationKit/QTE_Tutorial/test/ .
```

Let's open product_integrated_report.html on the Client machine's browser.

The red box displays the value 'SampleProduct' that was set for the command-line argument '--product-name' of the create-product-integrated-report command.



Product integrated test report

Product(SampleProduct)

Report created 07/14/2025 11:18:15

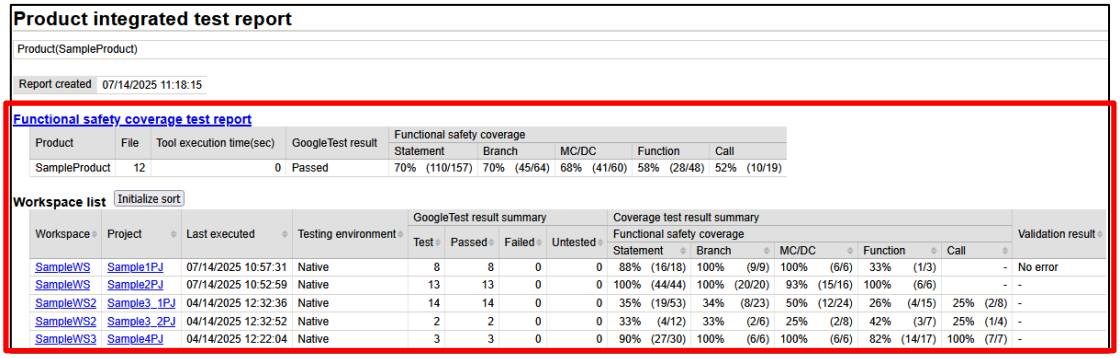
Functional safety coverage test report

Product	File	Tool execution time(sec)	GoogleTest result	Functional safety coverage									
				Statement		Branch		MC/DC		Function		Call	
SampleProduct	12	0	Passed	70%	(110/157)	70%	(45/64)	68%	(41/60)	58%	(28/48)	52%	(10/19)

Workspace list Initialize sort

Workspace	Project	Last executed	Testing environment	GoogleTest result summary				Coverage test result summary								Validation result
				Test	Passed	Failed	Untested	Functional safety coverage								
								Statement		Branch		MC/DC		Function		Call	
SampleWS	Sample1PJ	07/14/2025 10:57:31	Native	8	8	0	0	88%	(16/18)	100%	(9/9)	100%	(6/6)	33%	(1/3)	-	No error
SampleWS	Sample2PJ	07/14/2025 10:52:59	Native	13	13	0	0	100%	(44/44)	100%	(20/20)	93%	(15/16)	100%	(6/6)	-	-
SampleWS2	Sample3_1PJ	04/14/2025 12:32:36	Native	14	14	0	0	35%	(19/53)	34%	(8/23)	50%	(12/24)	26%	(4/15)	25%	(2/8) -
SampleWS2	Sample3_2PJ	04/14/2025 12:32:52	Native	2	2	0	0	33%	(4/12)	33%	(2/6)	25%	(2/8)	42%	(3/7)	25%	(1/4) -
SampleWS3	Sample4PJ	04/14/2025 12:22:04	Native	3	3	0	0	90%	(27/30)	100%	(6/6)	100%	(6/6)	82%	(14/17)	100%	(7/7) -

Summary information of the test results of the integrated workspace is displayed in the **Functional safety coverage test report**, and the test results of each project included in the integrated workspace are displayed in the **Workspace list**.



Product integrated test report

Product(SampleProduct)

Report created 07/14/2025 11:18:15

Functional safety coverage test report

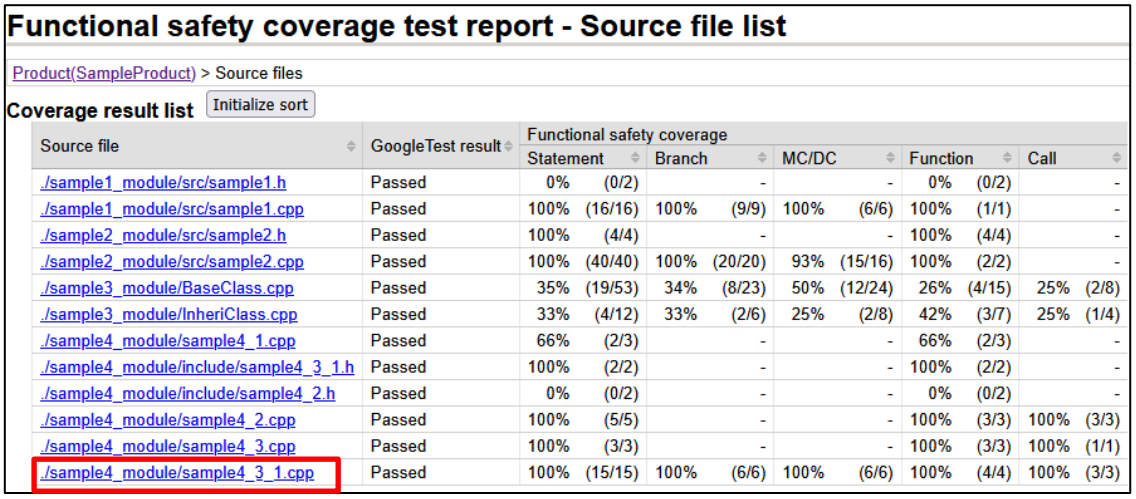
Product	File	Tool execution time(sec)	GoogleTest result	Functional safety coverage				
				Statement	Branch	MC/DC	Function	Call
SampleProduct	12	0	Passed	70% (110/157)	70% (45/64)	68% (41/60)	58% (28/48)	52% (10/19)

Workspace list Initialize sort

Workspace	Project	Last executed	Testing environment	GoogleTest result summary				Coverage test result summary								Validation result
				Test	Passed	Failed	Untested	Functional safety coverage								
								Statement		Branch		MC/DC		Function		Call		
SampleWS	Sample1PJ	07/14/2025 10:57:31	Native	8	8	0	0	88%	(16/18)	100%	(9/9)	100%	(6/6)	33%	(1/3)	-		No error
SampleWS	Sample2PJ	07/14/2025 10:52:59	Native	13	13	0	0	100%	(44/44)	100%	(20/20)	93%	(15/16)	100%	(6/6)	-		-
SampleWS2	Sample3_1PJ	04/14/2025 12:32:36	Native	14	14	0	0	35%	(19/53)	34%	(8/23)	50%	(12/24)	26%	(4/15)	25%	(2/8)	-
SampleWS2	Sample3_2PJ	04/14/2025 12:32:52	Native	2	2	0	0	33%	(4/12)	33%	(2/6)	25%	(2/8)	42%	(3/7)	25%	(1/4)	-
SampleWS3	Sample4PJ	04/14/2025 12:22:04	Native	3	3	0	0	90%	(27/30)	100%	(6/6)	100%	(6/6)	82%	(14/17)	100%	(7/7)	-

To check the test results at the Source File level, click the Functional safety coverage test report.

Pass/fail and coverage results of the test execution will be displayed.



Functional safety coverage test report - Source file list

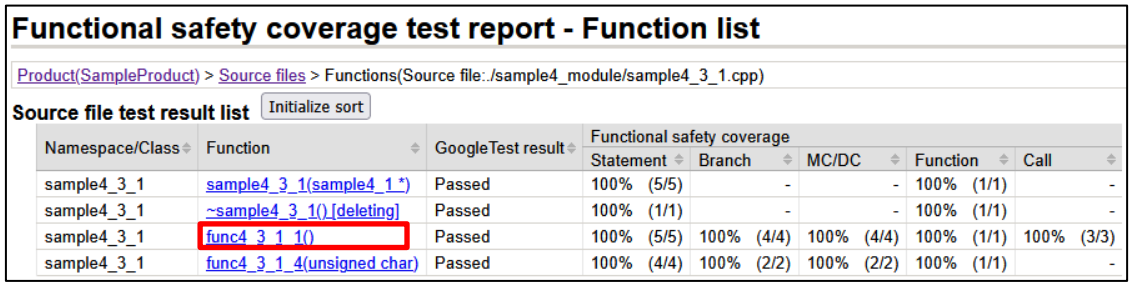
Product(SampleProduct) > Source files

Coverage result list Initialize sort

Source file	GoogleTest result	Functional safety coverage									
		Statement		Branch		MC/DC		Function		Call	
./sample1_module/src/sample1.h	Passed	0%	(0/2)	-		-		0%	(0/2)	-	
./sample1_module/src/sample1.cpp	Passed	100%	(16/16)	100%	(9/9)	100%	(6/6)	100%	(1/1)	-	
./sample2_module/src/sample2.h	Passed	100%	(4/4)	-		-		100%	(4/4)	-	
./sample2_module/src/sample2.cpp	Passed	100%	(40/40)	100%	(20/20)	93%	(15/16)	100%	(2/2)	-	
./sample3_module/BaseClass.cpp	Passed	35%	(19/53)	34%	(8/23)	50%	(12/24)	26%	(4/15)	25%	(2/8)
./sample3_module/InheriClass.cpp	Passed	33%	(4/12)	33%	(2/6)	25%	(2/8)	42%	(3/7)	25%	(1/4)
./sample4_module/sample4_1.cpp	Passed	66%	(2/3)	-		-		66%	(2/3)	-	
./sample4_module/include/sample4_3_1.h	Passed	100%	(2/2)	-		-		100%	(2/2)	-	
./sample4_module/include/sample4_2.h	Passed	0%	(0/2)	-		-		0%	(0/2)	-	
./sample4_module/sample4_2.cpp	Passed	100%	(5/5)	-		-		100%	(3/3)	100%	(3/3)
./sample4_module/sample4_3.cpp	Passed	100%	(3/3)	-		-		100%	(3/3)	100%	(1/1)
./sample4_module/sample4_3_1.cpp	Passed	100%	(15/15)	100%	(6/6)	100%	(6/6)	100%	(4/4)	100%	(3/3)

To see the test results of the functions defined in the source files, click on the file name.

To view the coverage results for sample4_3_1.cpp, click on. Click on ./sample4_module/sample4_3_1.cpp.



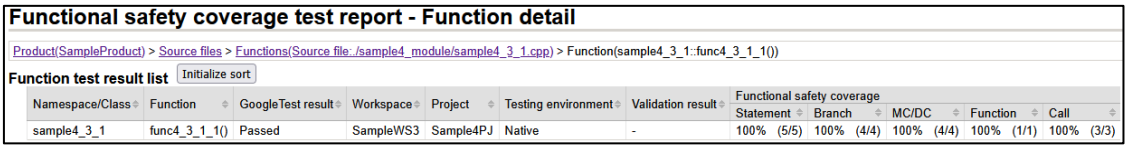
Functional safety coverage test report - Function list

Product(SampleProduct) > Source files > Functions(Source file:./sample4_module/sample4_3_1.cpp)

Source file test result list Initialize sort

Namespace/Class	Function	GoogleTest result	Functional safety coverage									
			Statement		Branch		MC/DC		Function		Call	
sample4_3_1	sample4_3_1(sample4_1 *)	Passed	100%	(5/5)	-		-		100%	(1/1)	-	
sample4_3_1	~sample4_3_1() [deleting]	Passed	100%	(1/1)	-		-		100%	(1/1)	-	
sample4_3_1	func4_3_1_1()	Passed	100%	(5/5)	100%	(4/4)	100%	(4/4)	100%	(1/1)	100%	(3/3)
sample4_3_1	func4_3_1_4(unsigned char)	Passed	100%	(4/4)	100%	(2/2)	100%	(2/2)	100%	(1/1)	-	

Click on the function func4_3_1_1() to see the test results for each project of the function.



Functional safety coverage test report - Function detail

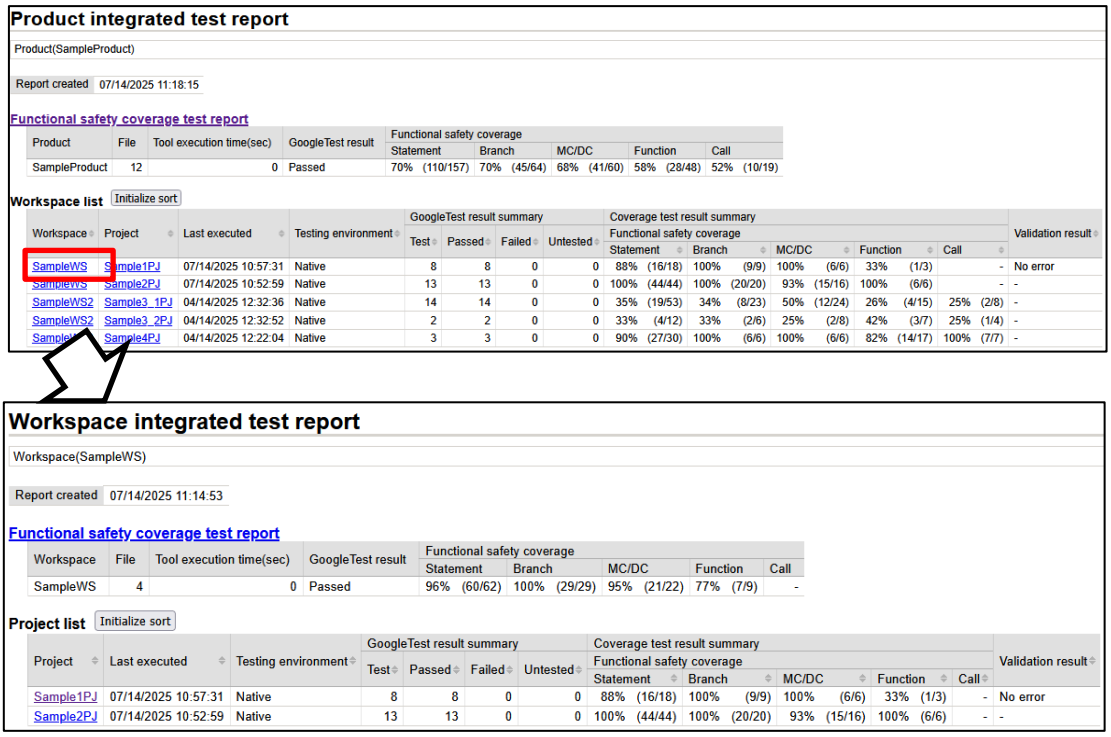
Product(SampleProduct) > Source files > Functions(Source file: /sample4_module/sample4_3_1.cpp) > Function(sample4_3_1::func4_3_1_1())

Function test result list Initialize sort

Namespace/Class	Function	GoogleTest result	Workspace	Project	Testing environment	Validation result	Functional safety coverage									
							Statement		Branch		MC/DC		Function		Call	
sample4_3_1	func4_3_1_1()	Passed	SampleWS3	Sample4PJ	Native	-	100%	(5/5)	100%	(4/4)	100%	(4/4)	100%	(1/1)	100%	(3/3)

Next, check the test results for each workspace. Click the Product(SampleProduct) link to return to the Integration Test Report's top page.

To check the test result of each workspace, click the workspace name in the workspace list in the Product Integration Test Report.



Product integrated test report

Product(SampleProduct)

Report created 07/14/2025 11:18:15

Functional safety coverage test report

Product	File	Tool execution time(sec)	GoogleTest result	Functional safety coverage									
				Statement		Branch		MC/DC		Function		Call	
SampleProduct	12	0	Passed	70%	(110/157)	70%	(45/64)	68%	(41/60)	58%	(28/48)	52%	(10/19)

Workspace list Initialize sort

Workspace	Project	Last executed	Testing environment	GoogleTest result summary				Coverage test result summary								Validation result
				Test	Passed	Failed	Untested	Functional safety coverage								
								Statement		Branch		MC/DC		Function		Call	
SampleWS	Sample1PJ	07/14/2025 10:57:31	Native	8	8	0	0	88%	(16/18)	100%	(9/9)	100%	(6/6)	33%	(1/3)	-	No error
SampleWS	Sample2PJ	07/14/2025 10:52:59	Native	13	13	0	0	100%	(44/44)	100%	(20/20)	93%	(15/16)	100%	(6/6)	-	-
SampleWS2	Sample3_1PJ	04/14/2025 12:32:36	Native	14	14	0	0	35%	(19/53)	34%	(8/23)	50%	(12/24)	26%	(4/15)	25%	(2/8) -
SampleWS2	Sample3_2PJ	04/14/2025 12:32:52	Native	2	2	0	0	33%	(4/12)	33%	(2/6)	25%	(2/8)	42%	(3/7)	25%	(1/4) -
[illegible]	Sample4PJ	04/14/2025 12:22:04	Native	3	3	0	0	90%	(27/30)	100%	(6/6)	100%	(6/6)	82%	(14/17)	100%	(7/7) -

Workspace integrated test report

Workspace(SampleWS)

Report created 07/14/2025 11:14:53

Functional safety coverage test report

Workspace	File	Tool execution time(sec)	GoogleTest result	Functional safety coverage									
				Statement		Branch		MC/DC		Function		Call	
SampleWS	4	0	Passed	96%	(60/62)	100%	(29/29)	95%	(21/22)	77%	(7/9)	-	

Project list Initialize sort

Project	Last executed	Testing environment	GoogleTest result summary				Coverage test result summary								Validation result
			Test	Passed	Failed	Untested	Functional safety coverage								
							Statement		Branch		MC/DC		Function		Call	
Sample1PJ	07/14/2025 10:57:31	Native	8	8	0	0	88%	(16/18)	100%	(9/9)	100%	(6/6)	33%	(1/3)	-	No error
Sample2PJ	07/14/2025 10:52:59	Native	13	13	0	0	100%	(44/44)	100%	(20/20)	93%	(15/16)	100%	(6/6)	-	-

When clicked, the Workspace Integration Test Report of SampleWS is displayed. As you can see, the workspace list in the Product Integration Test Report is a link to the Workspace Integration Test Report confirmed in Practice 4.

This completes Exercise 5.

8 Appendix

8.1 Open Source Software (OSS)

Refer to https://docs.gaioaws.com/qte/oss.html

Quality Town for Embedded Grade Tutorial

The content of this document is subject to change without prior notice.
GAIO is not responsible for any problem or loss caused by the errors in contents of this document.
Unauthorized use or reprint of the document is not allowed.

QTE is a product of the GAIO Technology Co., Ltd.
Other product names mentioned in this document are trademarks or registered trademarks of their respective companies.

GAIO TECHNOLOGY CO.,LTD.

User Support Information
https://www.en.gaio.co.jp/eng/support/support.html

Inquiry regarding product usage
For any questions, please contact GAIO user support
(https://www.en.gaio.co.jp/eng/support/support_entry.html)

For any inquiries to the user support team, a User ID is required.
Please mention your User ID when you contact us for any inquiries.
No support of any kind will be provided without a maintenance contract.