CODESCROLL
CONTROLLER TESTER

User Guides

3.7 — Last update: Jun 09, 2022

Suresofttech

Table of Contents

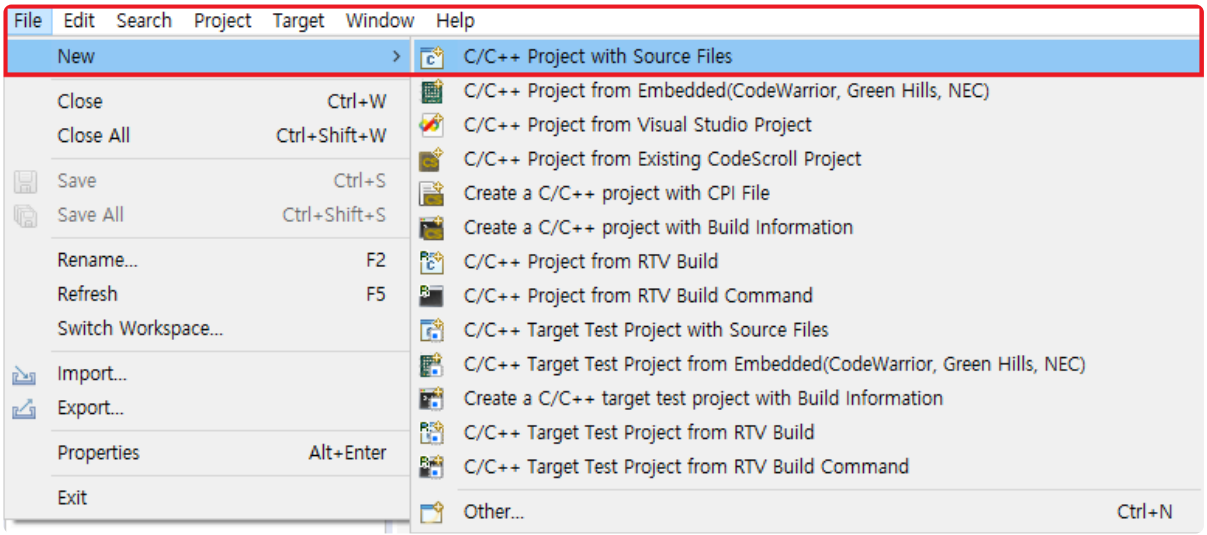
1. Target Test Guides

This user guides document describes how to execute target tests using CodeScroll Controller Tester.

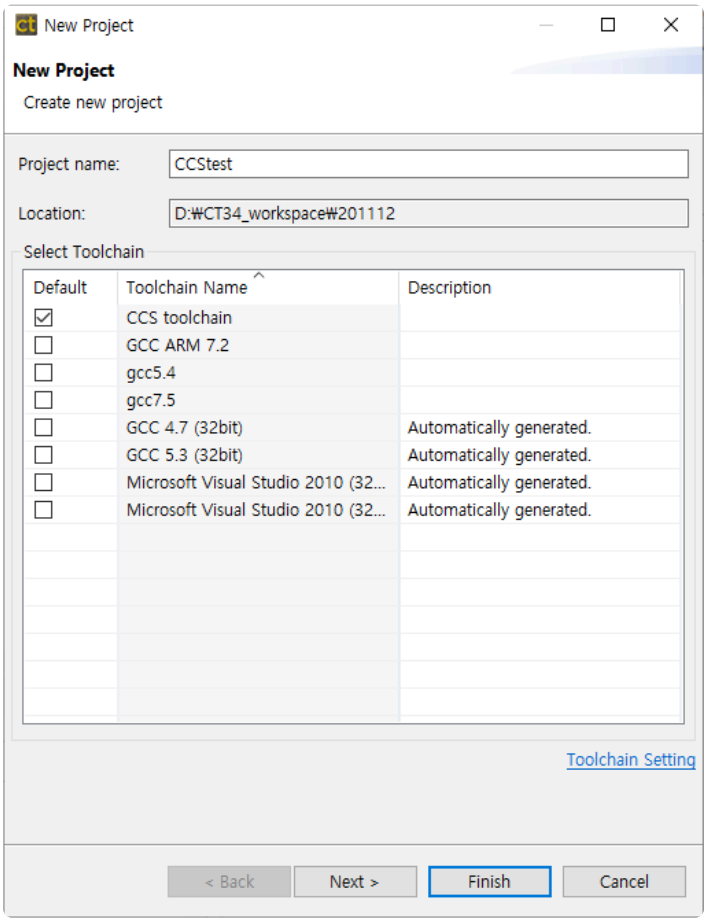
- Texas Instruments Code Composer Studio
- STM32cubeIDE

1.1. Texas Instruments Code Composer Studio

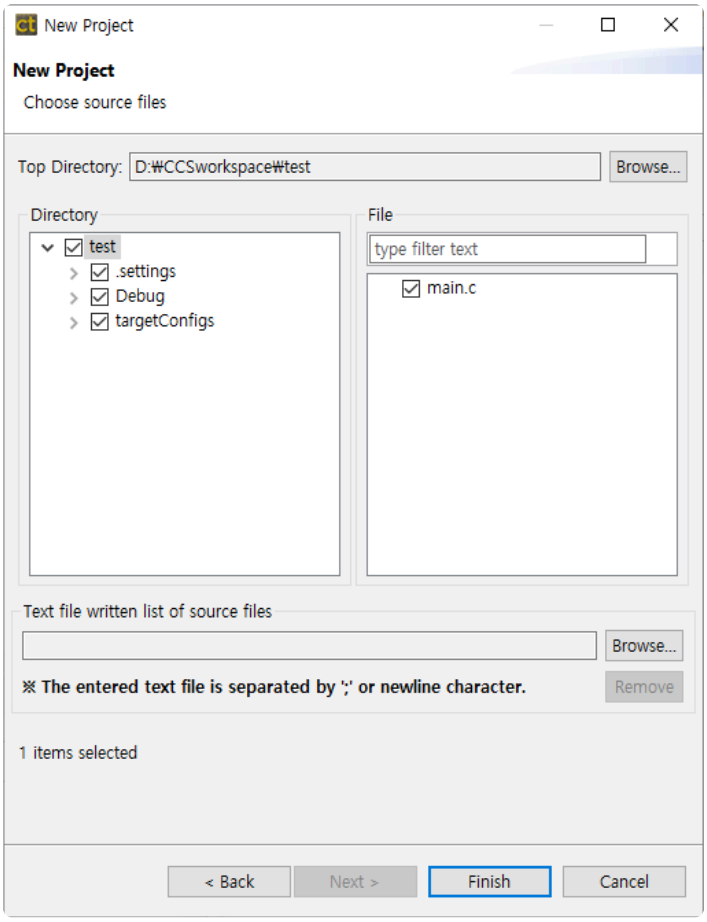
1. Creat a CodeScroll Controller Tester project.



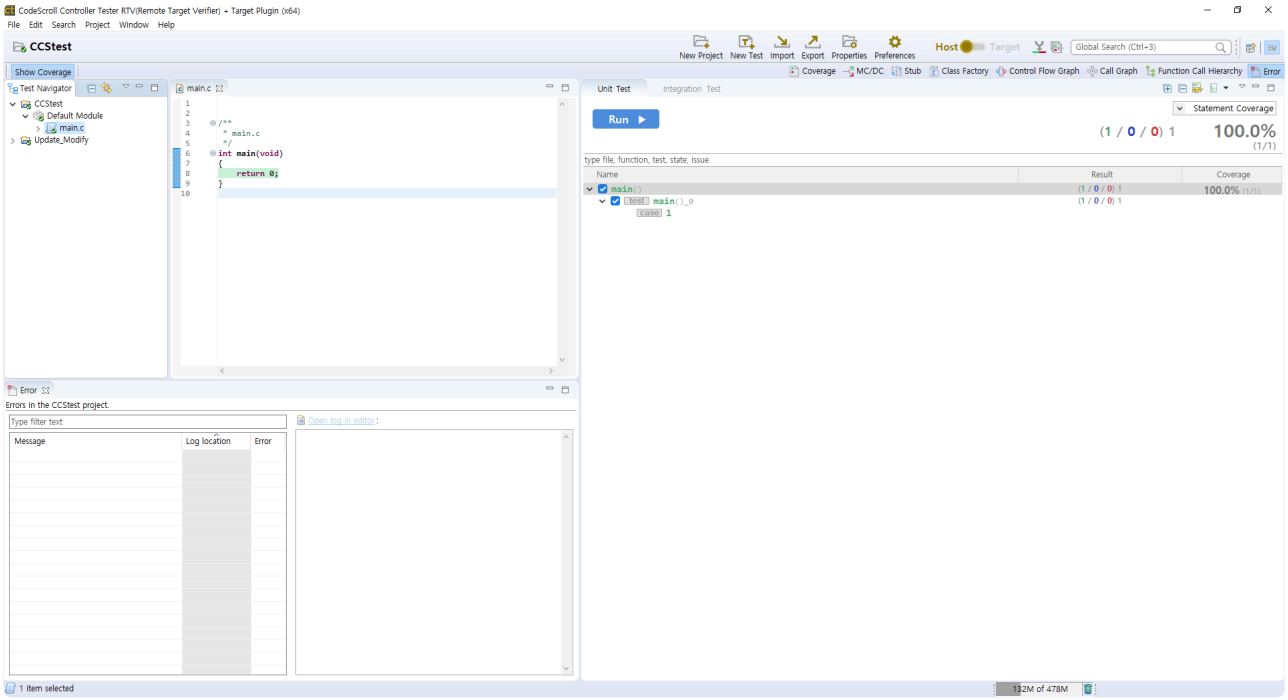
2. Select a created Code Composer Studio toolchain.



3. Select source files to test.



4. When finish the settings, click [Finish] button to create the project.



5. To use debuggers, set up in Code Composer Studio and CodeScroll Controller Tester. For more information, refer to Texas Instruments Code Composer Studio, a sub-topic of Controller Tester Debugger User Guides in this document.

1.2. STM32cubeIDE

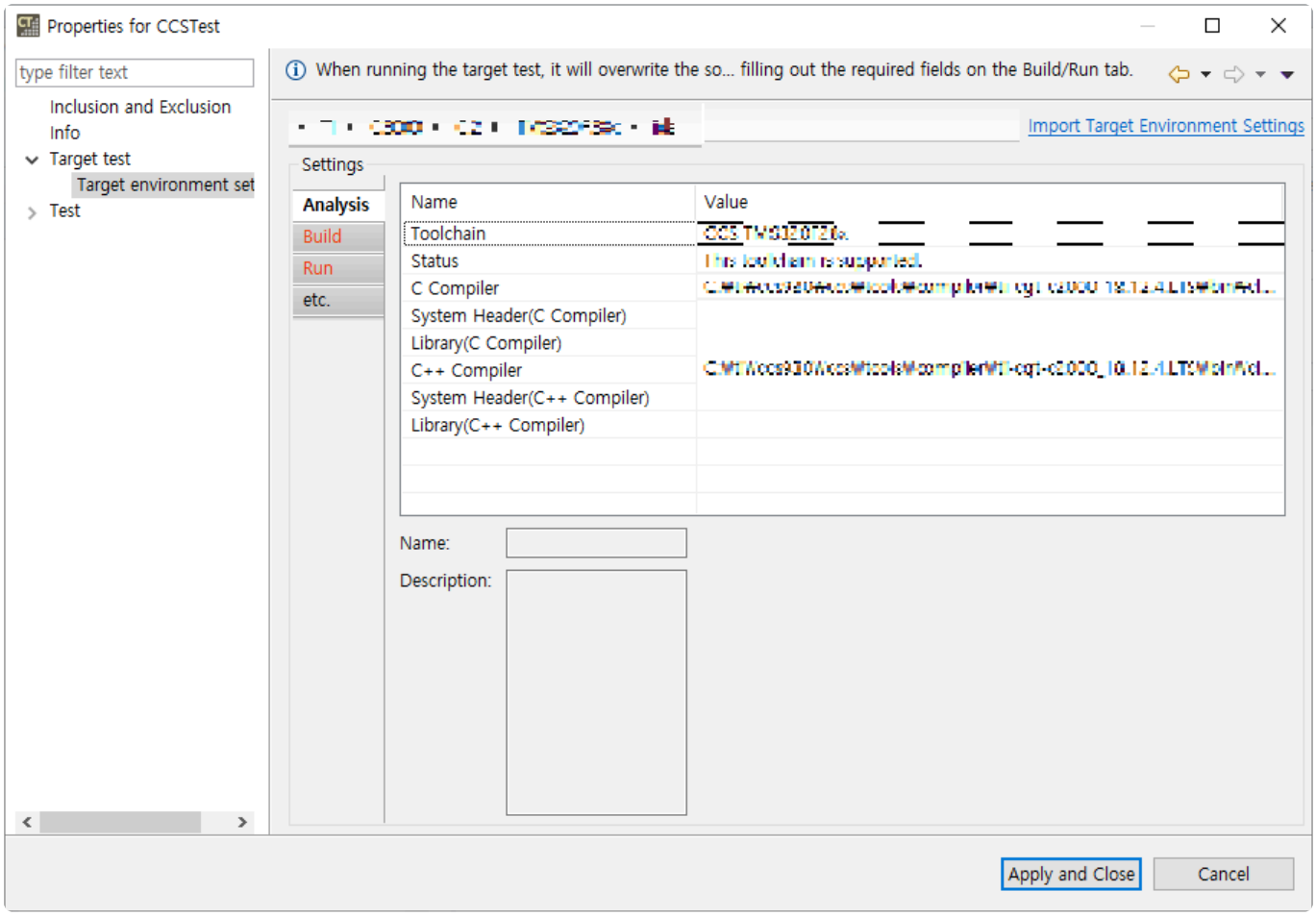
This document describes how to perform target testing using STM32cubeIDE for STM32 family targets.

The application example environment is as follows, and ST-Link debugger is used.

No.	개발 환경(OS)		빌드 환경(OS)		개발 언어	통합 개발 환경(IDE)		컴파일러		빌드 방식	타깃(실행 환경)	
	종류	버전	종류	버전		종류	버전	종류	버전	(Makefile, IDE)	아키텍처	칩셋(Chipset)
1	Windows	10	Windows	10	C	Stm32cubeide	1.6.1	Arm-none-eabi-gcc	GNU Tools for STM32 9-2020-q2-update.20201001-1621) 9.3.1 20200408	IDE	ARM Cortex-M7, 32Bit MCU	SMT32F7 Series
2	Windows	10	Windows	10	C++	Stm32cubeide	1.6.1	Arm-none-eabi-g++	GNU Tools for STM32 9-2020-q2-update.20201001-1621) 9.3.1 20200408	IDE	ARM Cortex-M7, 32Bit MCU	SMT32F7 Series

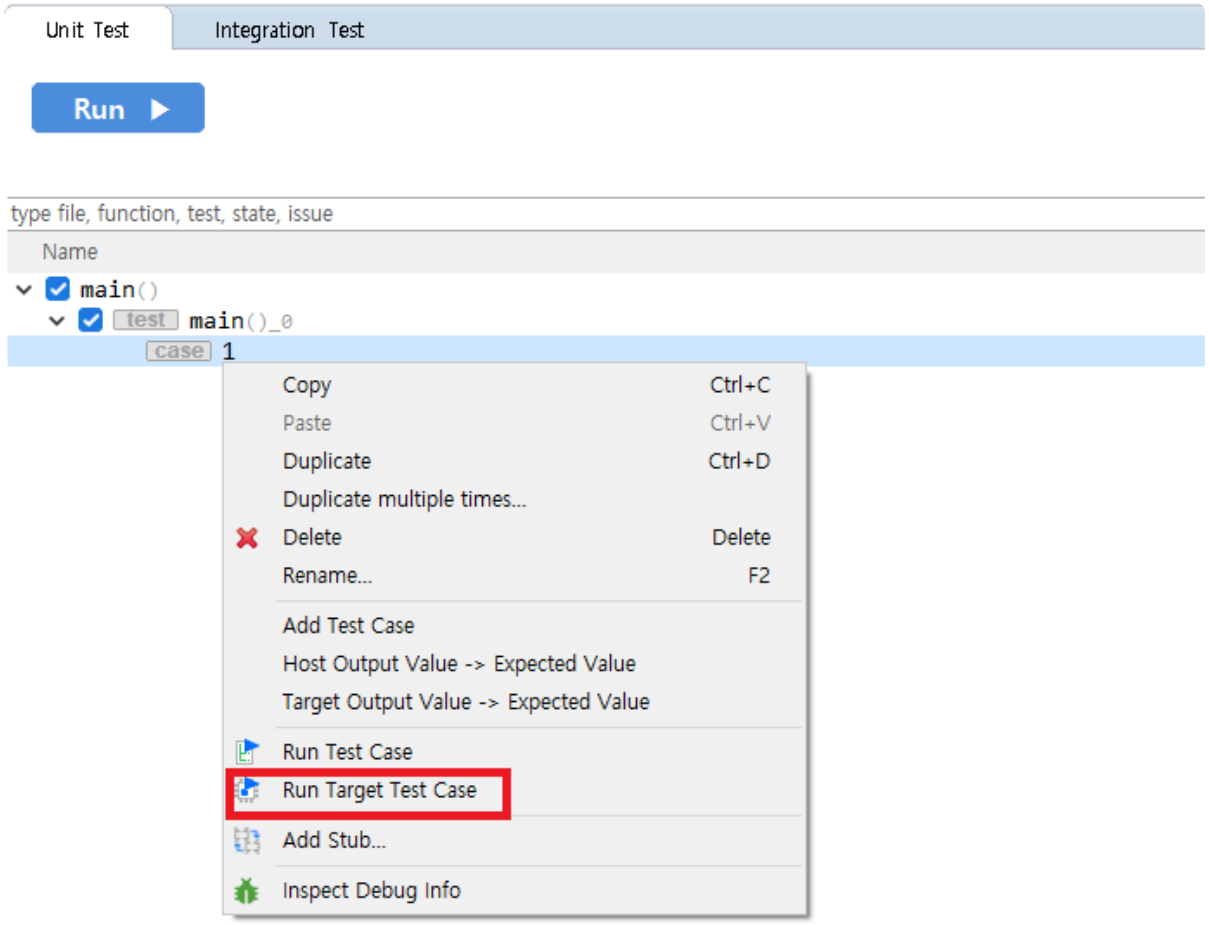
Target test application and execution order

1. Setting Target environment

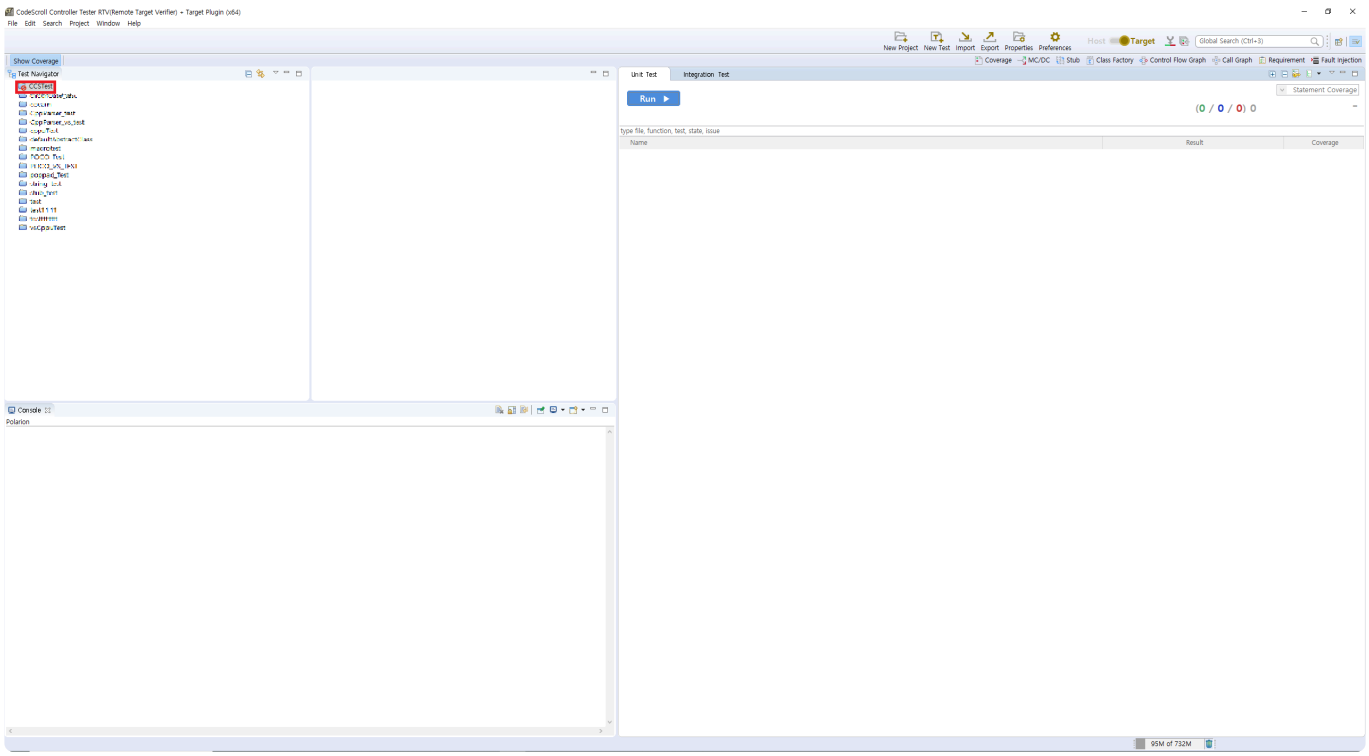


- On [right click on project] -> [properties] -> [Target test] -> [Target environment Setting], Just fill out the Property Analysis tab and close it after applying. The target test document is a manual build method, so other tabs do not affect the test.

2. Execute test case unit with [Run Target Test Case]

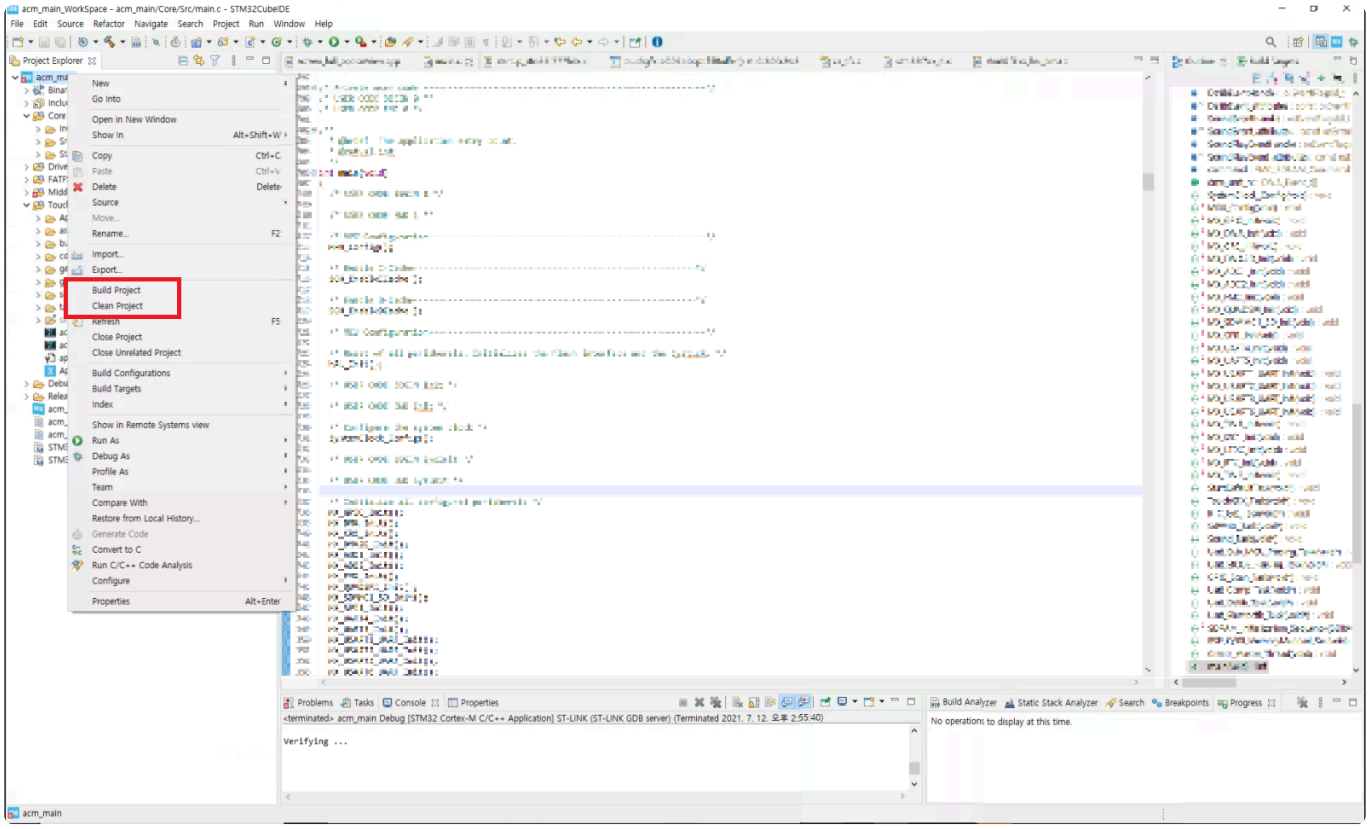


- For accurate testing, run them in test cases.



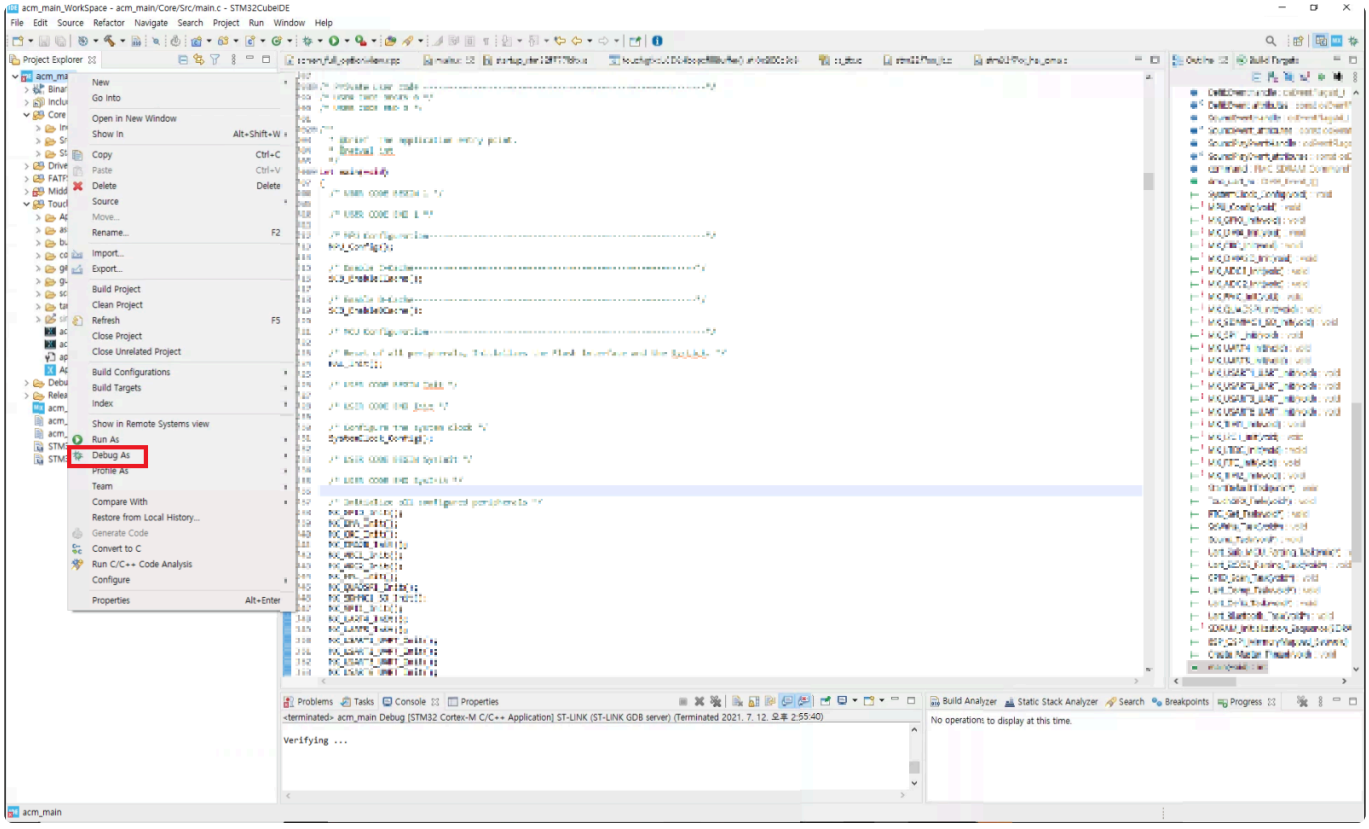
- If you go through steps 1 and 2, the project of Controller Tester will be locked as above.

3. Clean and build the project in STM32cubeIDE



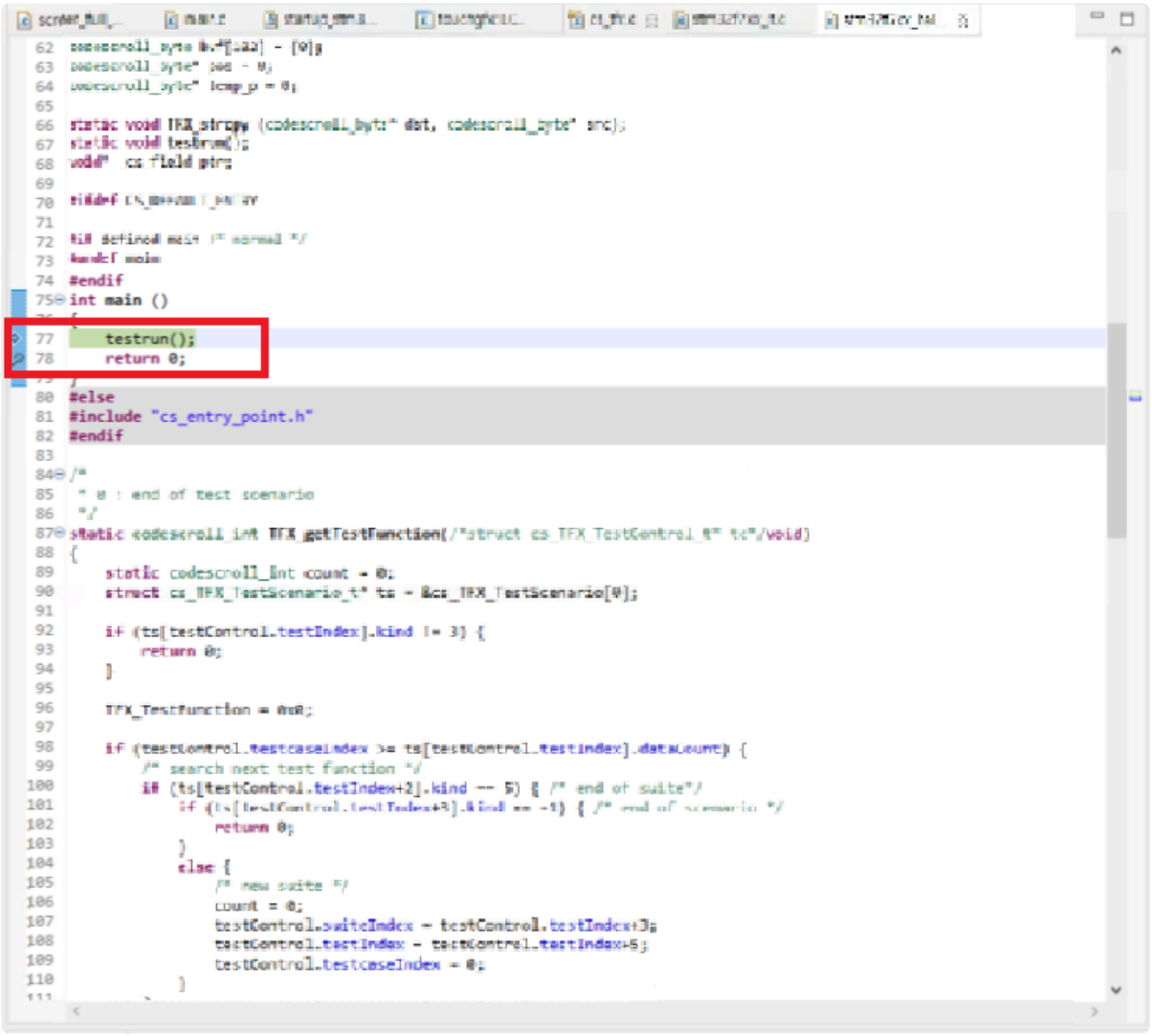
- Clean the exported source in STM32cubeIDE and build it.

4.Debug in STM32cubeIDE



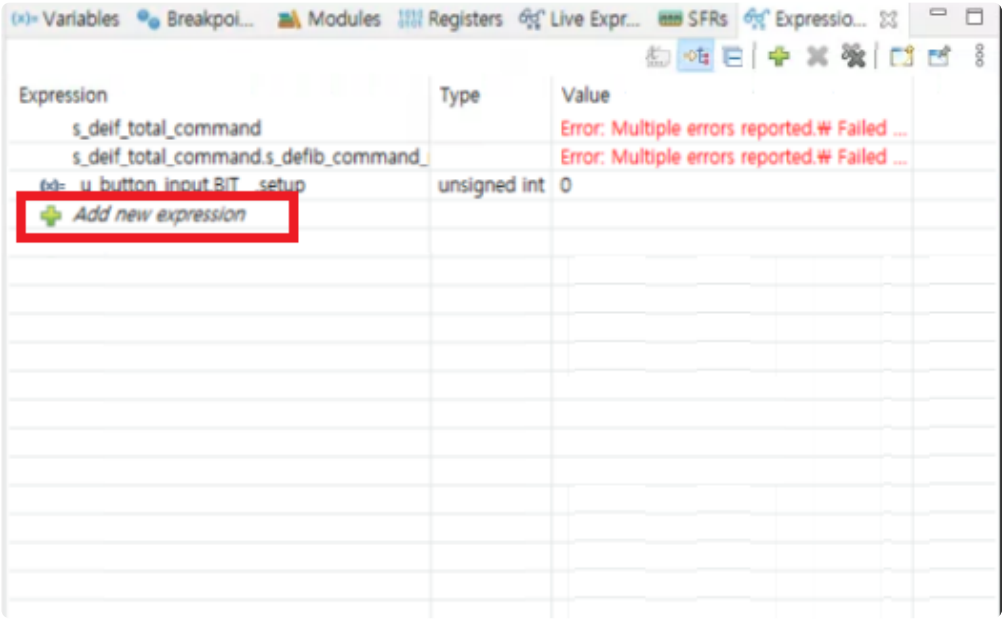
- If the build is successful, run debug.

5. Execute after setting a break point in return 0;

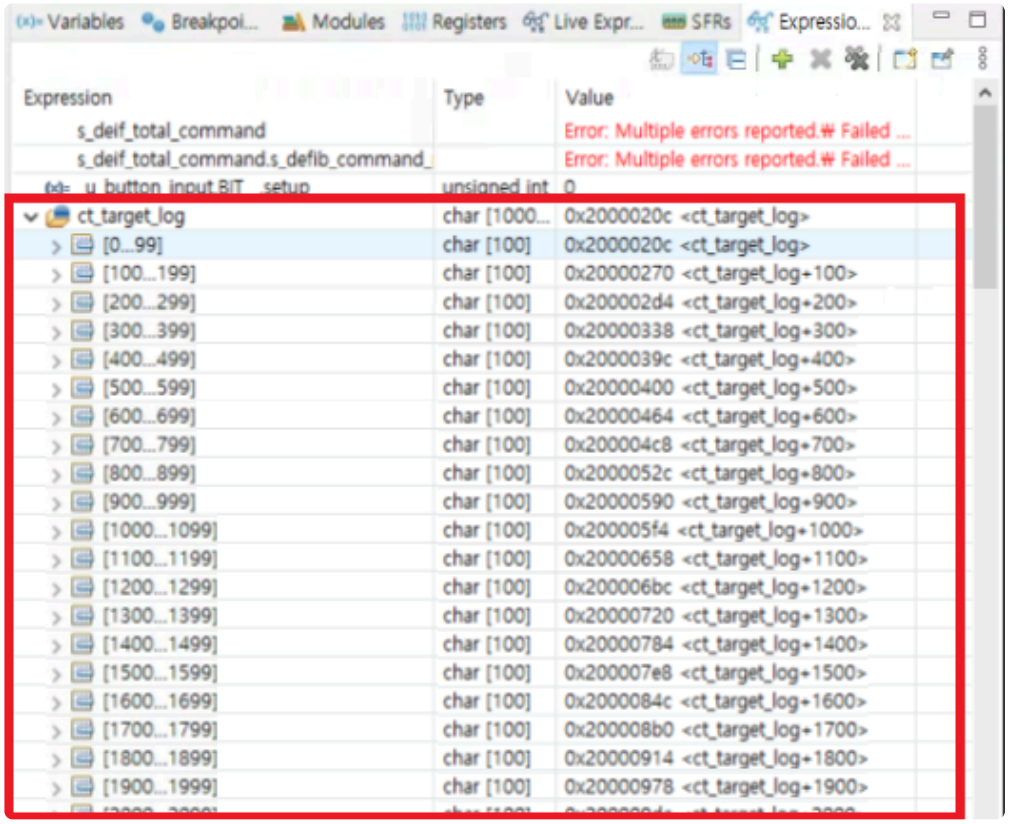


- The starting point of the code is main in cs_tfx.c. Put a break point before ‘return 0;’, which is the point at which testrun(); ends.

6. Check the log in the ct_target_log expression view

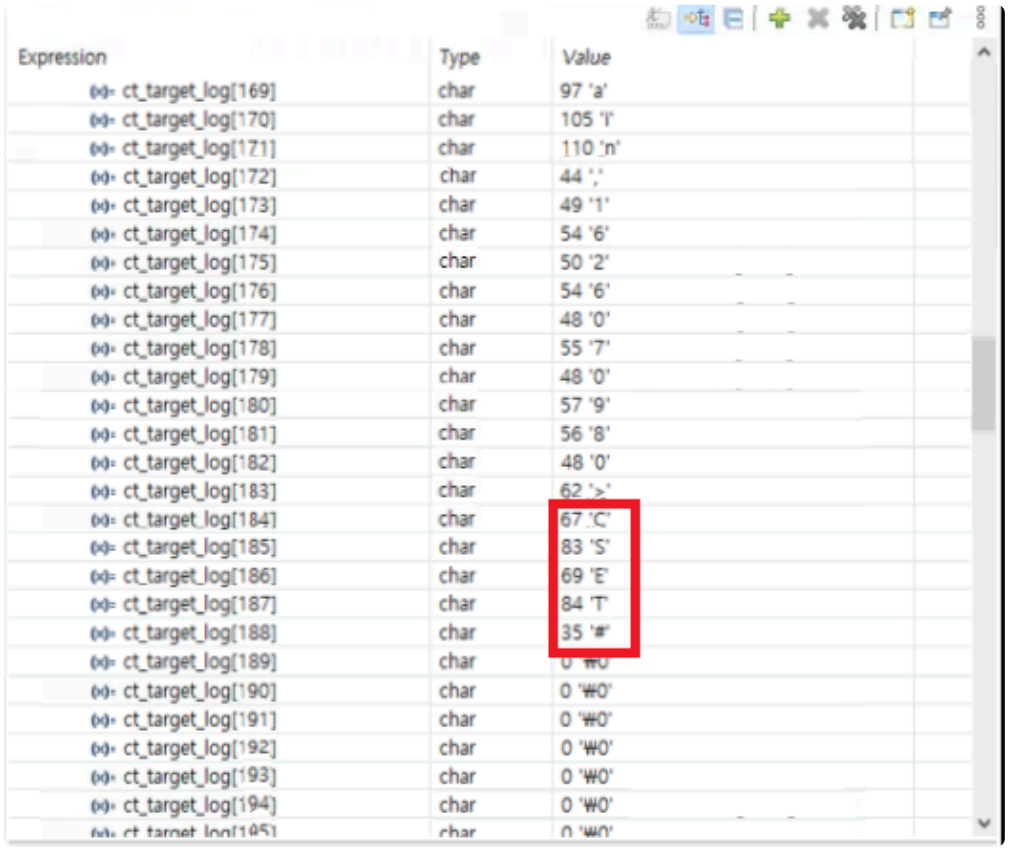


- In the expression view, click Add new expression to add an array containing the log (ct_target_log).



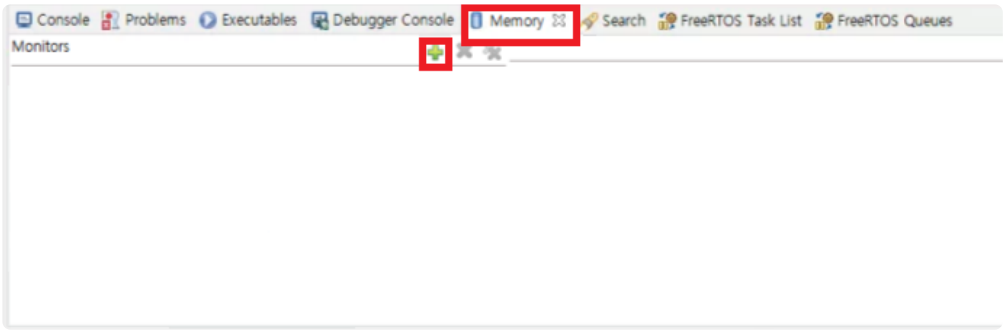
- You can check the contents of ct_target_log as above.

7. Check if it ends with CSET# (whether or not a normal test is performed)

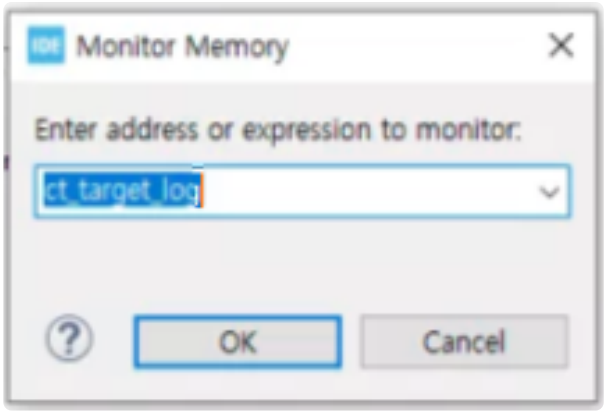


- When the last part of the log ends with CSET#, it can be judged that the test ended normally. Therefore, you can check once whether the test is running normally in the expression view.

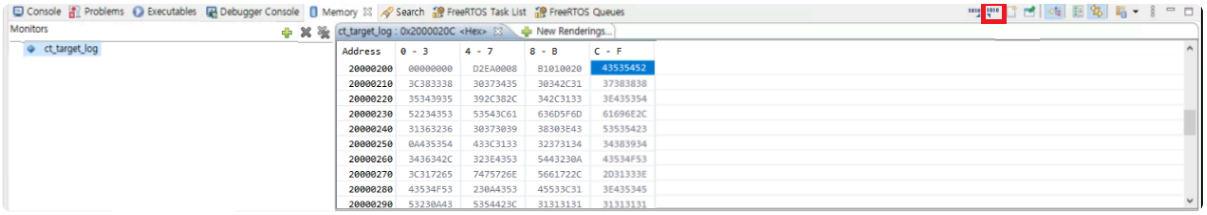
8. Add ct_target_log to monitor memory in memory view



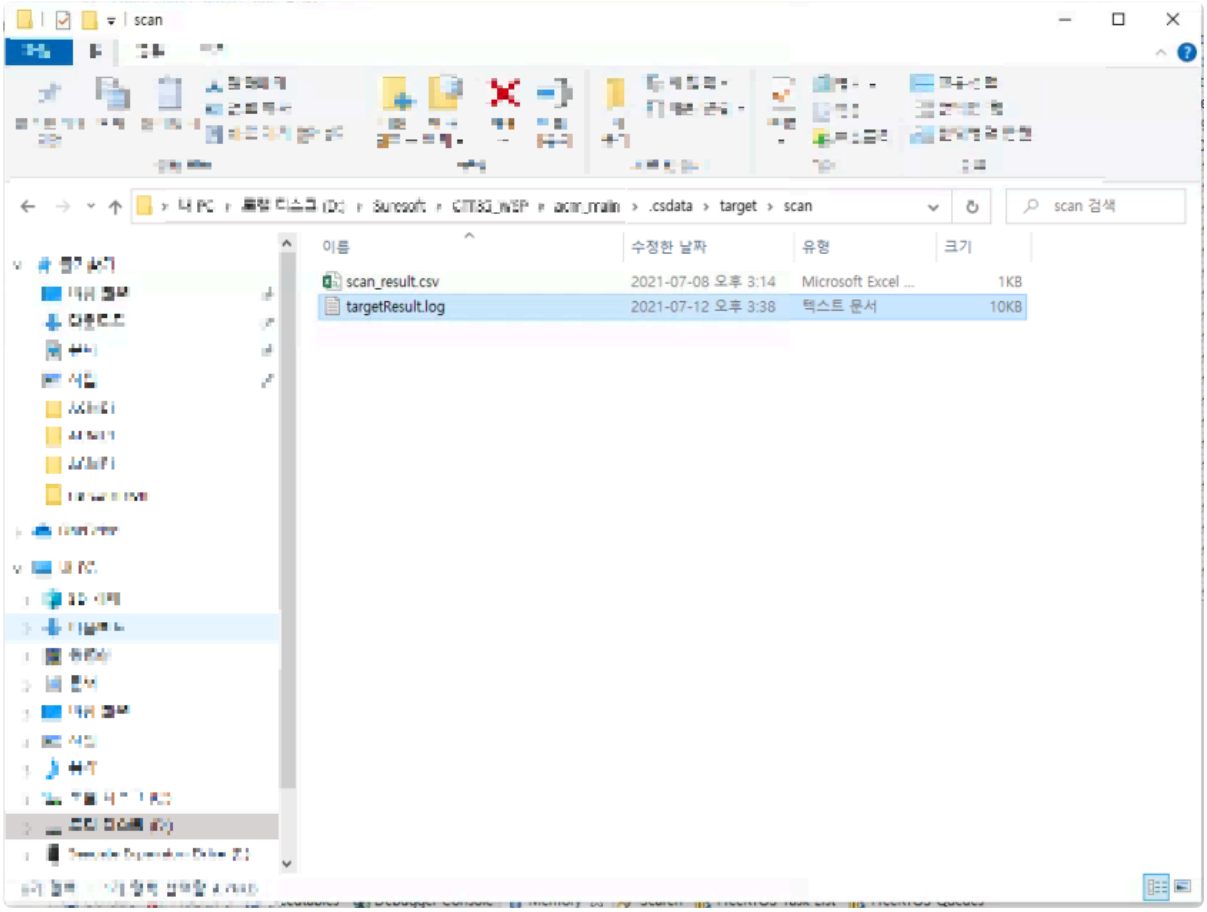
- Add ct_target_log by clicking '+' to Monitors in memory view for memory dump.



9. Export from memory view to log path of Controller Tester project / Check if the file is normally created in the path

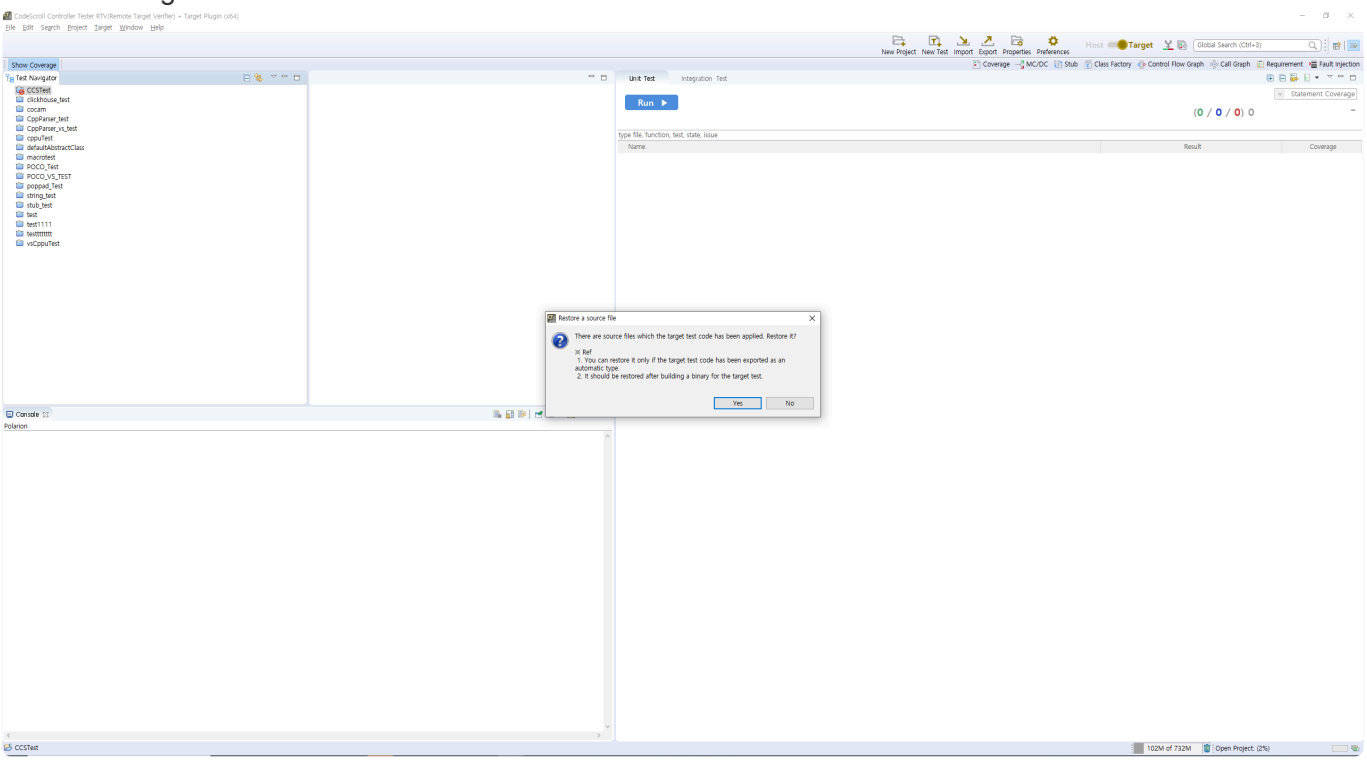


- Click the export button in the memory view to download the memory of ct_target_log to a file.
- Format is RAW Binary, Start address is the start address of ct_target_log of expression view, and Length specifies the array size of ct_target_log. (Even if the length of the log is shorter than Length, it does not affect the test.)



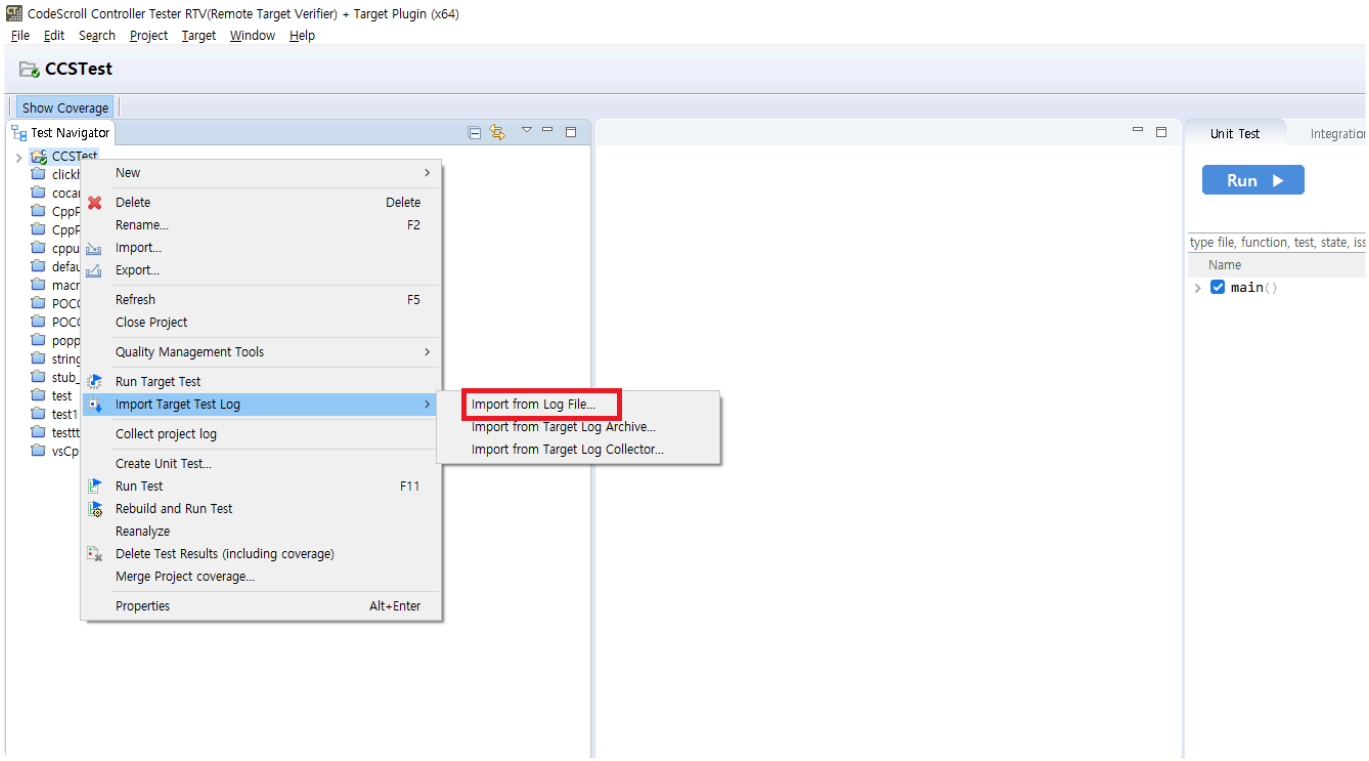
- Check if the file is normally created in the specified path.

10. Restoring source file from Controller Tester

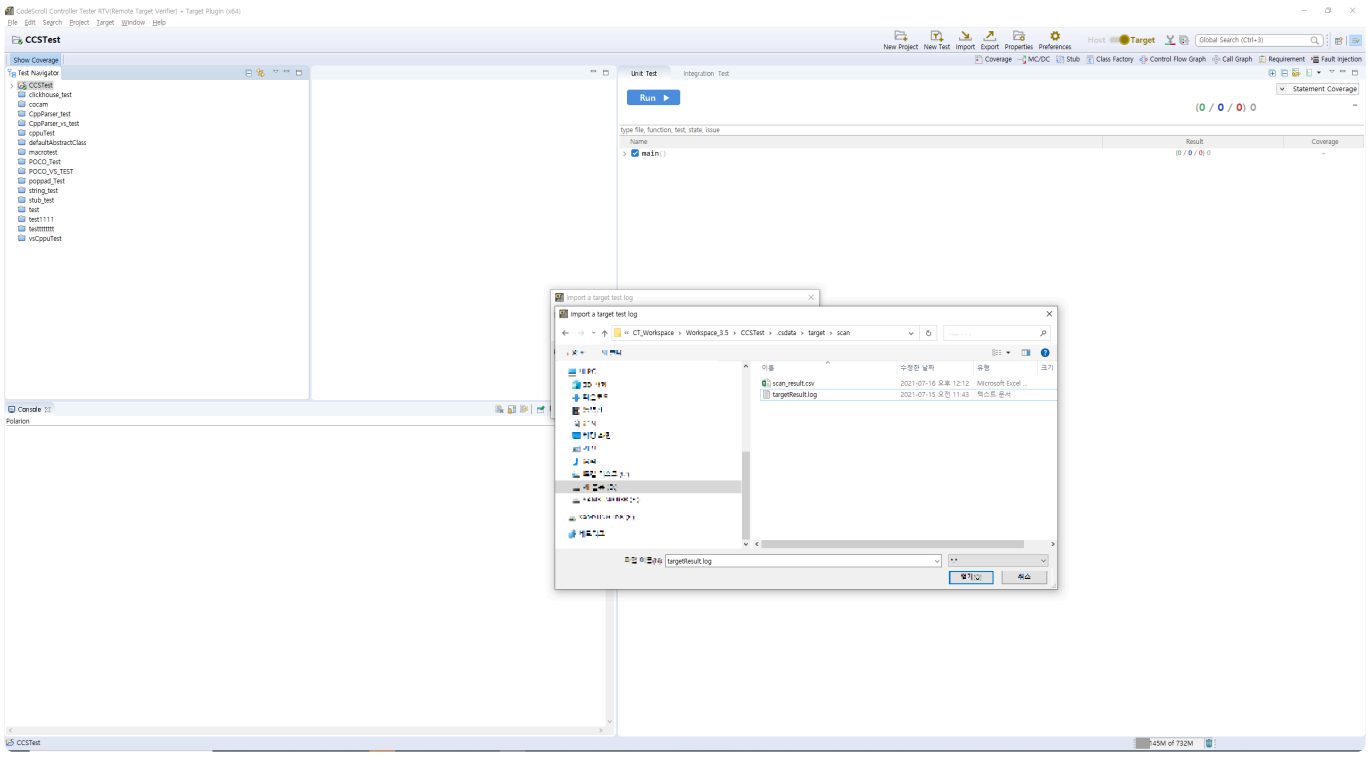


- Restore the source file from Controller Tester to get the target test log.

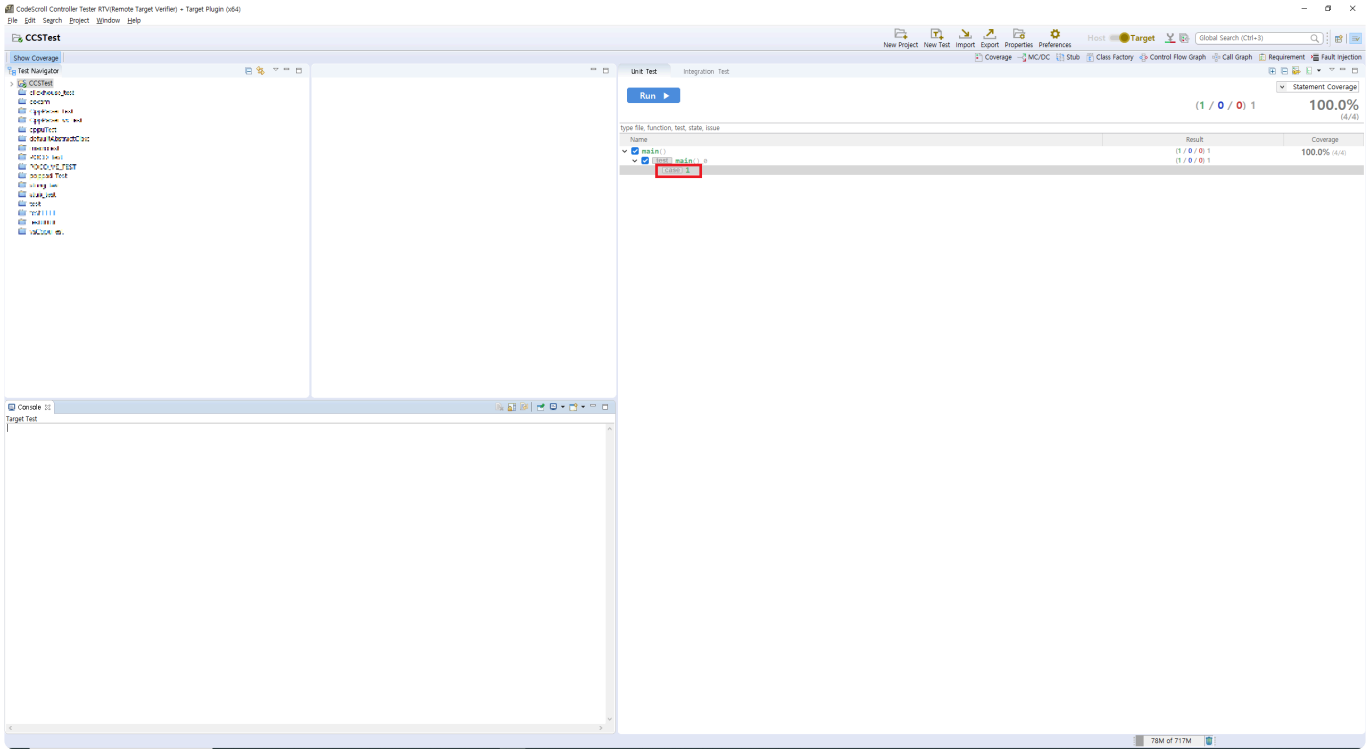
11. Import Target Test Log -> Import from Log File



- Click [Import Target Test Log] -> [Import from Log File]



- Import the file created in step 9.



- You can confirm that the test case was successfully executed and the coverage was measured.

2. Debugger User Guides

This user guides document describes how to use debugger when executing CodeScroll Controller Tester target test.

- Lauterbach TRACE32
- PLS Universal Debug Engine
- iSYSTEM winIDEA Debugger
- IAR Embedded Workbench C-SPY Debugger
- Texas Instruments Code Composer Studio
- Microchip MPLAB IDE

2.1. Lauterbach TRACE32

Controller Tester can target test using the TRACE32 debugger.
Controller Tester uses TRACE32's cmm script to run tests in the target environment and get the results.
A list of targets supported by TRACE32 can be found on the Lauterbach homepage.

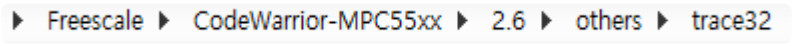
2.1.1. Supported target list that can generate cmm script automatically

Controller Tester automatically generates a cmm script file or receives it from the user.
If the cmm script can be generated automatically, you only need to enter the chip name of the target. If you cannot generate cmm scripts automatically, you must enter the cmm script file path manually.
The targets that currently support the automatic generation of cmm scripts are:

PowerPC	mpc5554, mpc5553, mpc5534, mpc556x, mpc551x, mpc560xe, spc560bxx, spc560pxx, spc560sxx, mpc560xb, mpc560xp, mpc560xs, spc563m54, mpc5632m, spc563m60, mpc5633m, spc563m64, mpc5634m, mpc564xs, mpc5668, mpc5674, mpc5644a, spc564a80, mpc5642a, spc564a70, mpc567xk, spc56hk, mpc5643l, spc56el60, spc56el70, mpc5644b, mpc5644c, spc564b64, spc56ec64, mpc5645b, spc564b70, mpc5645c, spc56ec70, mpc5646b, spc564b74, mpc5646c, spc56ec74, mpc5676r, spc56ap, mpc5746m, mpc5744k, spc574k74, mpc5777m, spc57hm90, mpc574xp, mpc574xg, mpc574xr, mpc577xk, mpc5777c, spc570s, mpc5726l, spc572l, spc574s, spc58ne, spc58eg, spc58nn, spc582b, spc58ec, spc58nh, spc584b, s32r274, s32r264, s32r372
ARM	mkw01, mkw20, mkv30, mkv40, mkv10, mkv50, mkm30, mkl0, mkl10, mkl20, mkl30, mkl40, mkl80, mk0, mk10, mk20, mk30, mk40, mk50, mk60, mk70, mk80, mac57d54h, mac71×1, mac71×2, mac71×4, mac71×5, mac71×6, mac72×1, lpc51u68, lpc54xx, lpc8xx, lpc11xx, lpc12xx, lpc13xx, lpc17xx, lpc18xx, lpc21xx, lpc22xx, lpc23xx, lpc24xx, lpc28xx, lpc29xx, lpc40xx, lpc43xx, imxrt1064, xmc1100, xmc1200, xmc1300, xmc1400, xmc4100, xmc4200, xmc4300, xmc4400, xmc4500, xmc4700, xmc4800, tle98, s3fm02g, s32k, s6e1a, s6e1c, s6j3
tricore	tc2dx, tc21x, tc22x, tc23x, tc26x, tc27x, tc29x, tc35x, tc37x, tc38x, tc39x, tc116x, tx1167, tx1197, tc1724, tc1728, tc1736, tc1762, tc1764, tc1766, tc1767, tc1782, tc1784, tc1791, tc1792, tc1793, tc1796, tc1797, tc1798

2.1.2. Step1: Setting target environment in Controller Tester

Select Debugger on the target environment setting page of the Controller Tester. Only a list of debuggers supported is displayed, depending on the toolchain selected for the project.
Set the debugger to TRACE32.



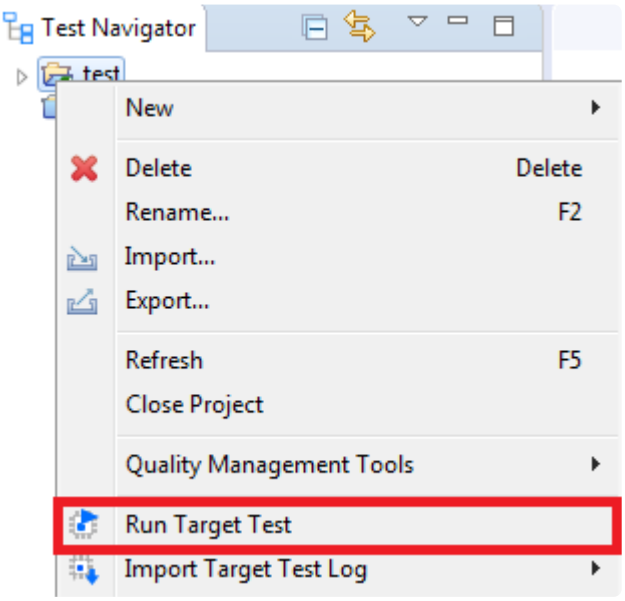
The setting items are displayed according to the selected information. The items you need to set when using the TRACE32 debugger are shown in the table below.
Some of the settings are required.

trace32_exe_file_path	TRACE32 executable file path. Each target has a different executable file, so you need to make sure that the target executable is the correct one. Required
target_binary_path	Path to the binary file for loading into the target environment. Check and enter the path where the target binary file is created in your IDE or build script. Required.
chip	Enter the chip name of the target you are using. It is used when auto-generating a cmm script, so you need to enter the correct chip name.
user_defined_cmm_script_file_path	Custom cmm script file path. For targets that do not support automatic generation of cmm scripts, you must write a script to set the debugger and target usage environment, or enter the path to the cmm script file you are using.

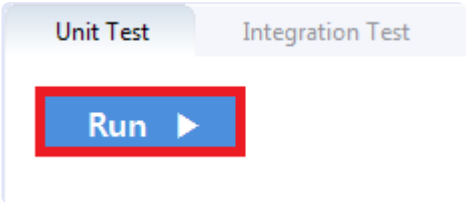
2.1.3. Step2: Run the target test

You must exit the running TRACE32 program before running the target test.
You can run a target test by selecting [Run Target Test] from the project context menu in the Test Navigator view or by clicking the [Run] button in the Test View.

- [Run Target Test]



- [Run]



When you run the target test, the TRACE32 program runs. If the test is succeeded, the TRACE32 program ends automatically.

2.1.4. Debug the target test

1. After setting it as a target, right-click the test case in the ‘Unit Test’ view and click ‘Check Debug Information’
2. Build the user project directly or execute the build script registered in the ‘target environment’ setting in the controller tester project
3. Verify that the build was successful
4. Restore the original source by opening the project in Controller Tester
5. After running Trace32, open the cmm script file (start.cmm) and execute ‘debug’ (Controller_Tester_project_path/.csdata/target/start.cmm)
6. Click the ‘step’ button to go to the first line of the target.cmm script
7. Add breakpoint to ‘Go.Hll’ in target.cmm file
8. Click ‘Var’ > ‘Show Function’
9. Double-click after searching for the function to be tested
10. Add breakpoint at the beginning of the function
11. Click the ‘step’ button and confirm that the debugging point moves to the location specified in step 10.
12. ‘Var’ > ‘Show Local…’ . Click to confirm that the value of the local variable changes
13. Run up to the debugging point

2.2. PLS Universal Debug Engine (UDE)

Controller Tester can target test using the UDE debugger.
Controller Tester uses debugging scripts supported by UDE to run tests and get results in the target environment.
A list of targets available for connection to UDE can be found on the PLS homepage.

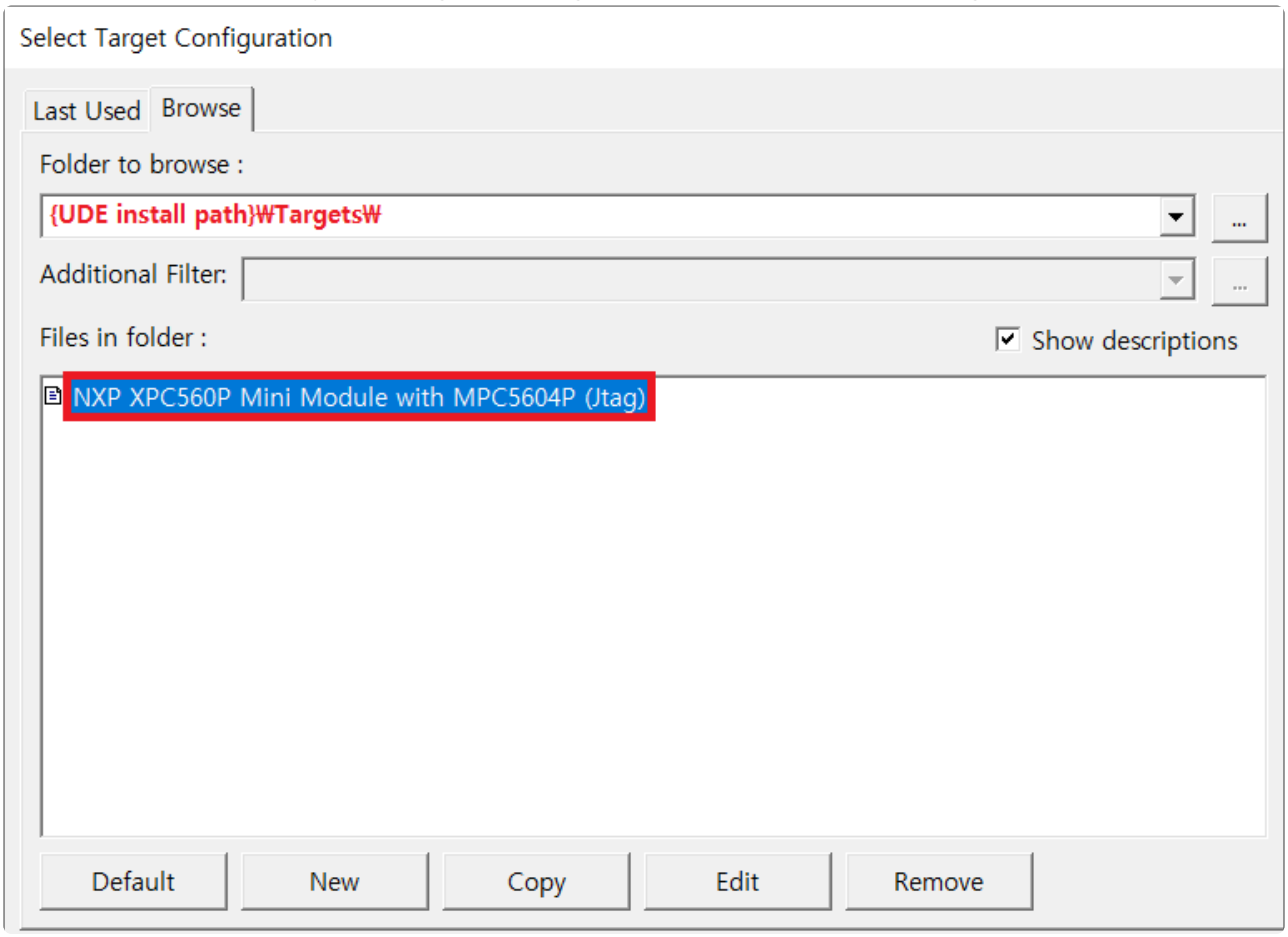
Controller Tester uses the UDE workspace information to perform target tests. For this reason, users must first create a workspace before performing a target test.

- Step1: Create a workspace in UDE IDE
- Step2: Setting target environment in Controller Tester
- Step3: Run the target test

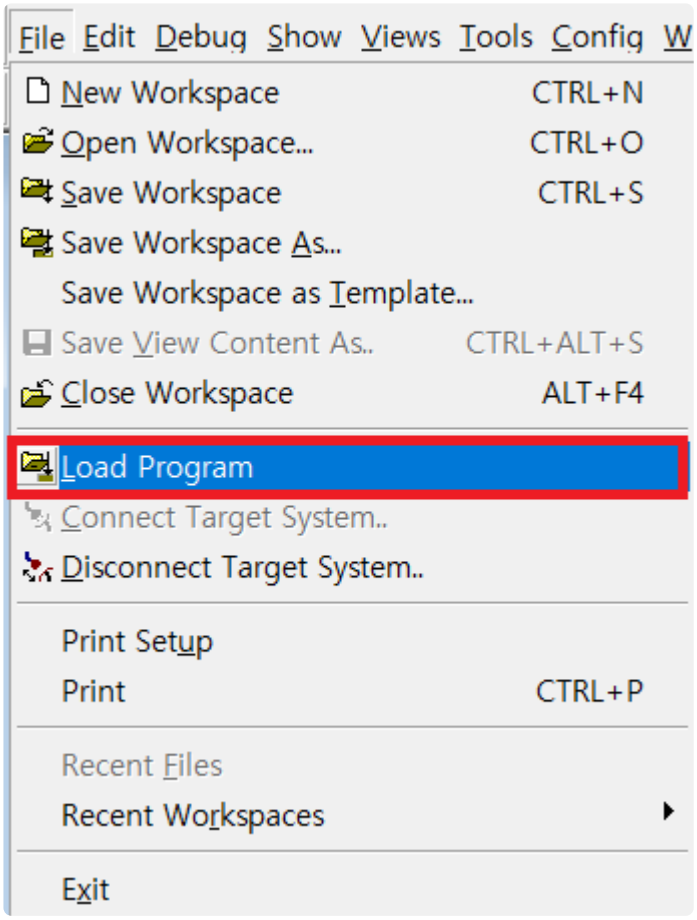
2.2.1. Step1: Create a workspace in UDE IDE

UDE can generate UDE workspaces from the UDE desktop IDE.

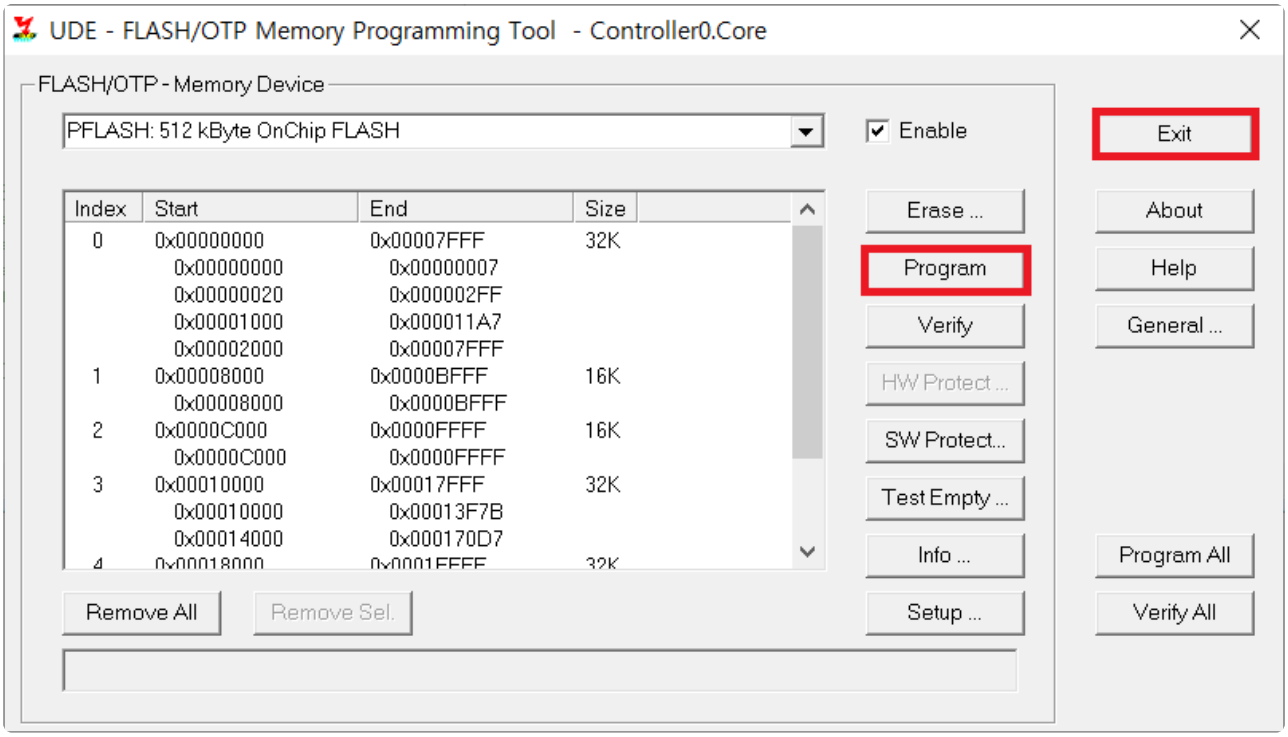
1. Create the workspace by selecting the configuration file suitable for the target used.



2. Click the [File]> [Load Program] button to load the binary file. At this point, select the binary file built from the test code.



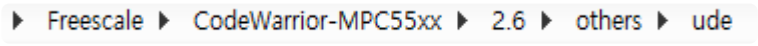
3. Follow the instructions and press the [program] button to load the binary file into the target according to the target settings. If the load completes successfully, the workspace setup is complete. Click [Exit] to exit the dialog.



See the manual provided by UDE for details.

2.2.2. Step2: Setting target environment in Controller Tester

Select Debugger on the Target Environment configuration page of the Controller Tester. Only a list of debuggers supported is displayed, depending on the toolchain selected for the project.
Set the debugger to UDE.



The setting items are displayed according to the selected information. The items you need to set when using the UDE debugger are shown in the table below.
Some of the settings are required.

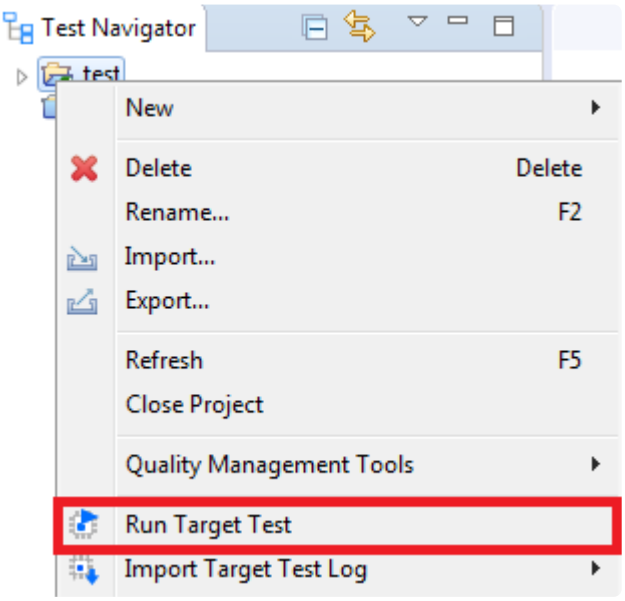
target_binary_path	Path to the binary file for loading into the target environment. Check and enter the path where the target binary file is created in your IDE or build script. Required.
ude_project_file	Path to the workspace project file (.wsx) generated by the UDE IDE. Required.

The default scripting language used by Controller Tester is visual basic script.
When the target configuration is complete, click the [OK] or [Finish] button. You are ready to execute the target test.

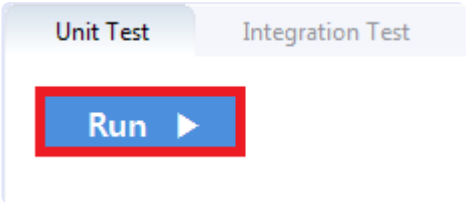
2.2.3. Step3: Run the target test

You must have exited the UDE desktop IDE to run the target test.
You can run a target test by selecting [Run Target Test] from the project context menu in the Test Navigator view or by clicking the [Run] button in the Test View.

- [Run Target Test]



- [Run]



UDE debugging scripts can be written in languages such as C ++, .NET, and Perl. See the UDE Automation Basics documentation included in the UDE manuals for other supported languages that can be scripted.

2.2.4. Debug the target test

1. After setting it as a target, right-click the test case in the ‘Unit Test’ view and click ‘Check Debug Information’
2. Build the user project directly or execute the build script registered in the ‘target environment’ setting in the controller tester project
3. Verify that the build was successful
4. Restore the original source by opening the project in Controller Tester
5. Select project after executing Pls Ude (.wsx file)
6. Select the output file built in step.2
7. Notice that the source file and function information contained in the output file are displayed on the left navigation.
8. Select a source file containing the function to be tested and add breakpoints in the function
9. Press F5 to start from the entry point

2.3. iSYSTEM winIDEA Debugger

Controller Tester provides the ability to run tests on your target environment and get results from it automatically by using winIDEA debugging scripts.
The list of targets supported by winIDEA can be found on the iSYSTEM home page.

The execution of the debugging script requires the python SDK installed together when installing winIDEA. If it is not installed, you can download it from the iSYSTEM SDK installation page. Also, you should check the version of winIDEA you use if it supports the SDK. The debugging script provided by Controller Tester is based on python 3.3.

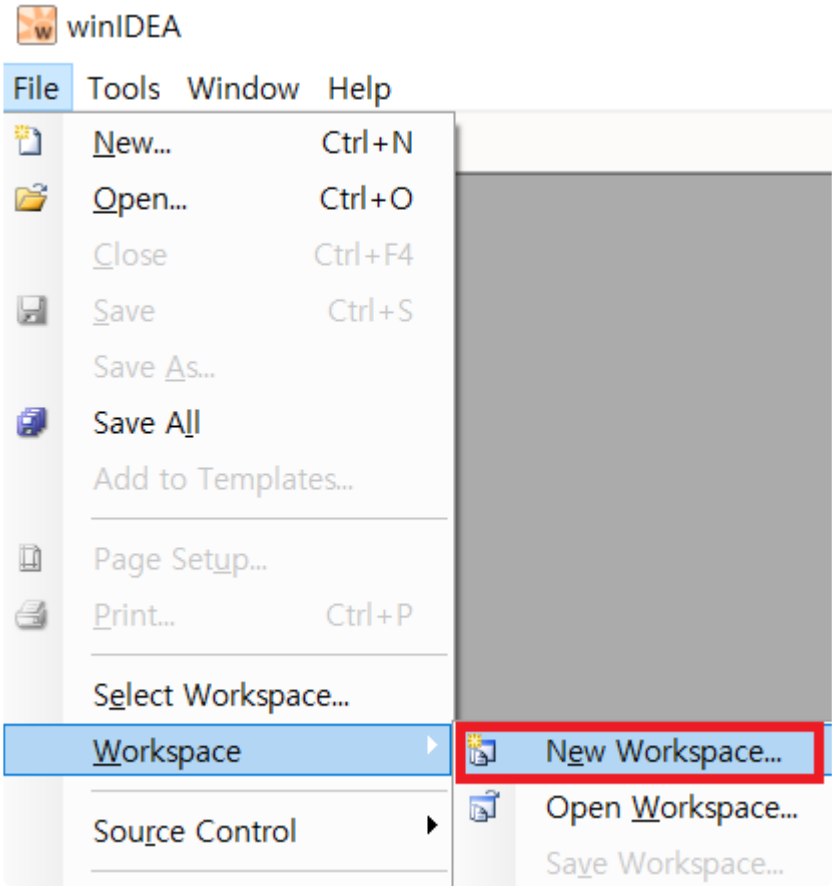
This document describes the process from creating a project in winIDEA to running a target test in Controller Tester. The iSYSTEM BlueBox iC5000 Unit debugger and NXP's MPC56xx target are used for the examples.

2.3.1. Preparation for use of iSYSTEM winIDEA

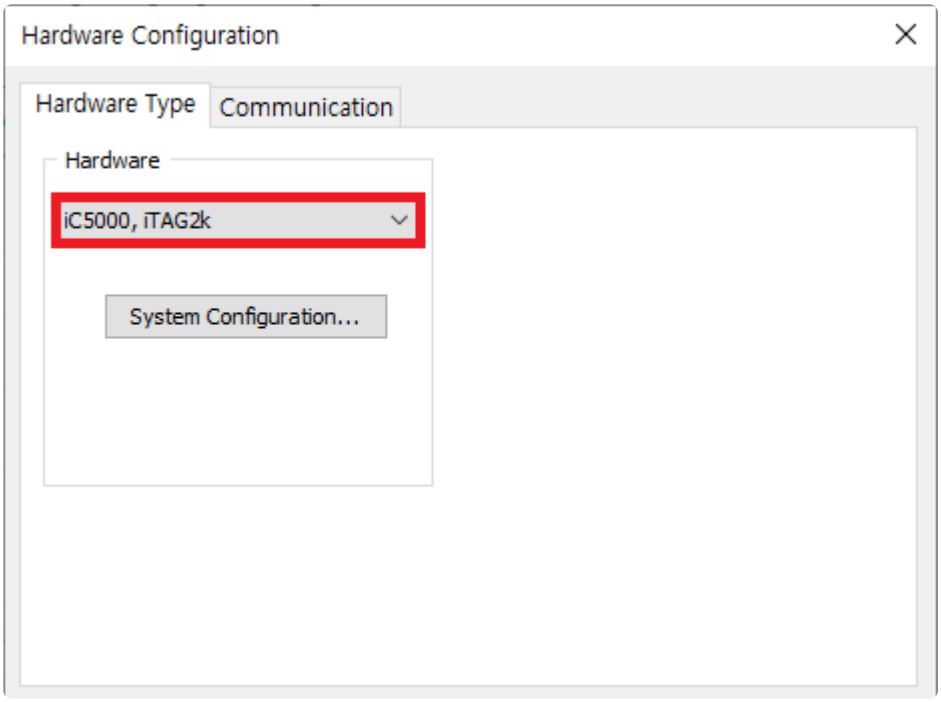
Target testing with winIDEA in Controller Tester requires a debugger that winIDEA supports.
Before running the target test, you need to create a winIDEA workspace and connect the debugger for use to the PC with Controller Tester.

2.3.2. Step1: Creating and setting up a winIDEA workspace

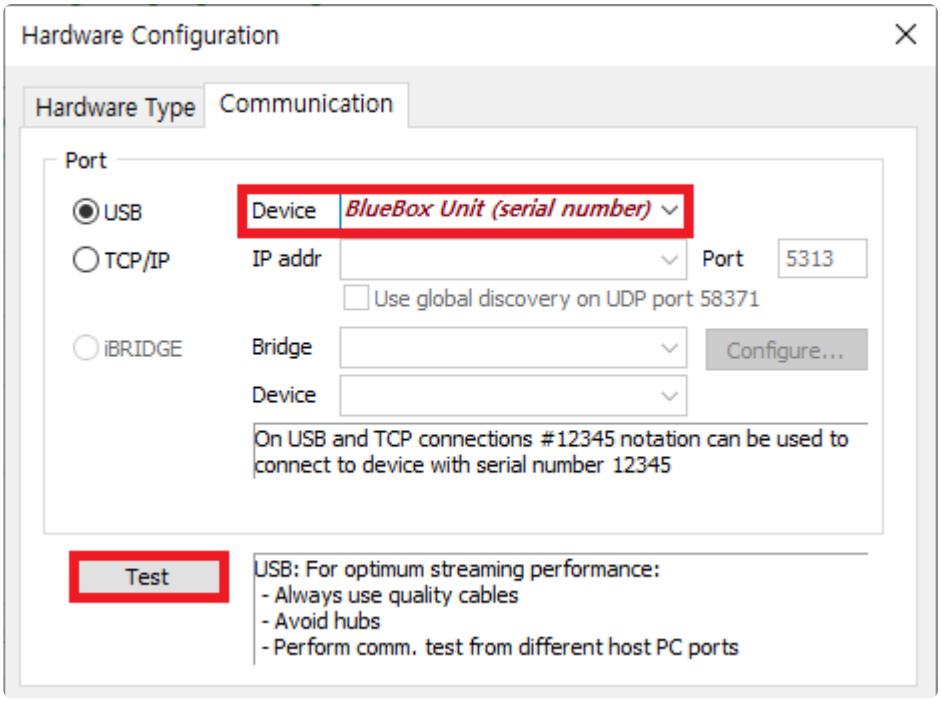
1. After running winIDEA, create a new workspace by selecting [File]> [Workspace]> [New Workspace …] from the top menu. Additional workspace settings are required to use the workspace you create for the Controller Tester target test.



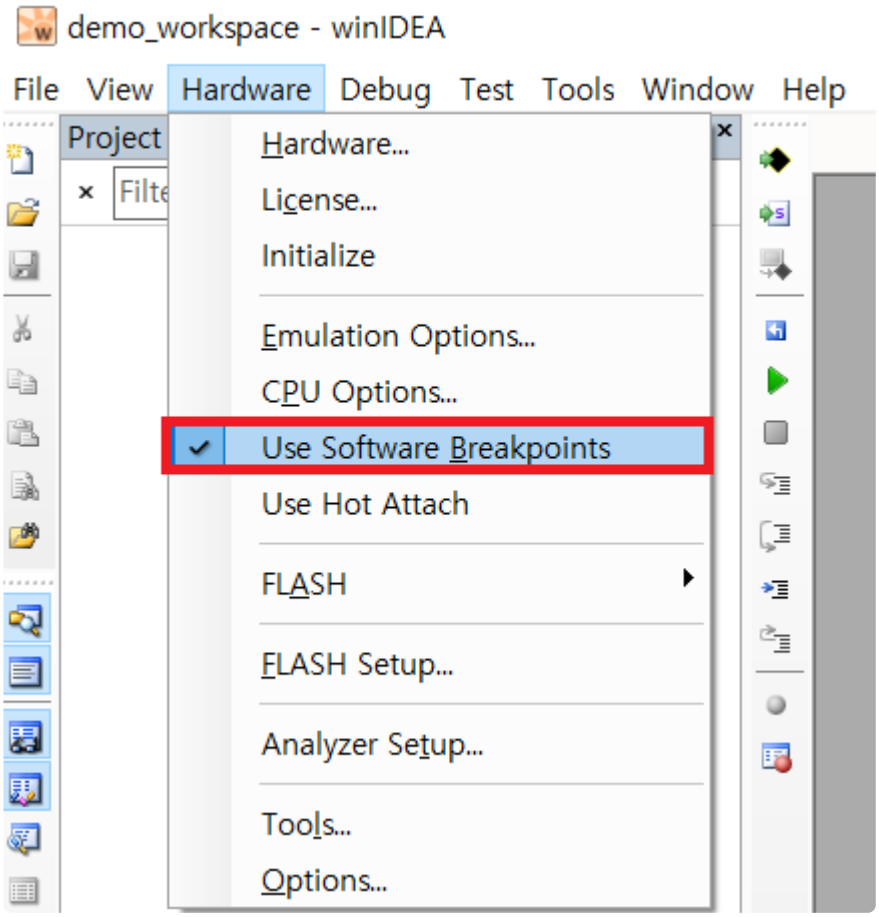
2. First, go to the top menu, select [Hardware]> [Hardware…], and then select the type of the connected BlueBox in the [Hardware Type] tab.

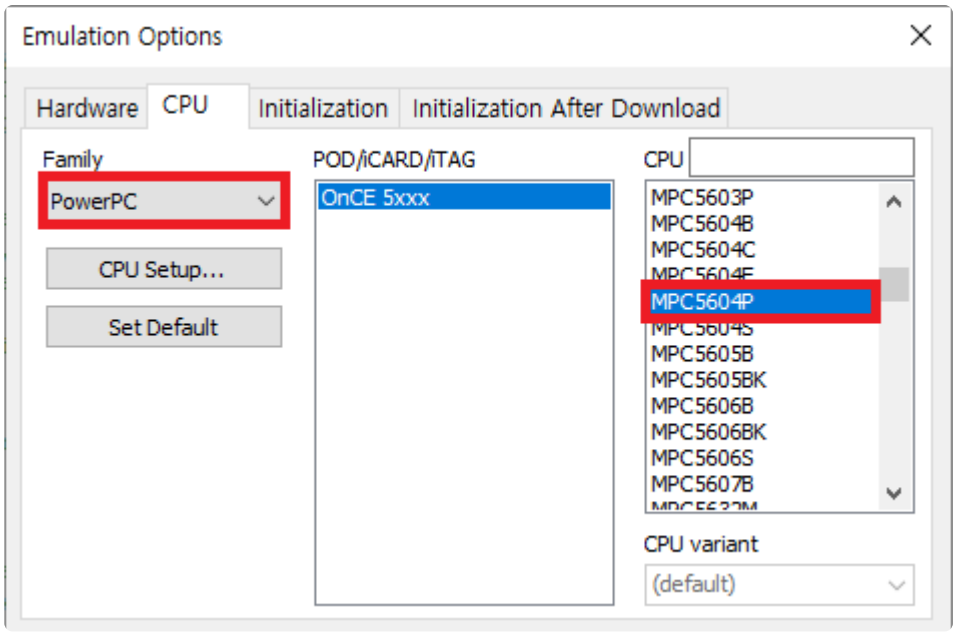


3. Next, set the communication method in the [Communication] tab, and press the [Test] button to check the connection to the debugger. Please refer to the iSYSTEM BlueBox manual for instructions on how to connect the debugger device depending on the communication method.



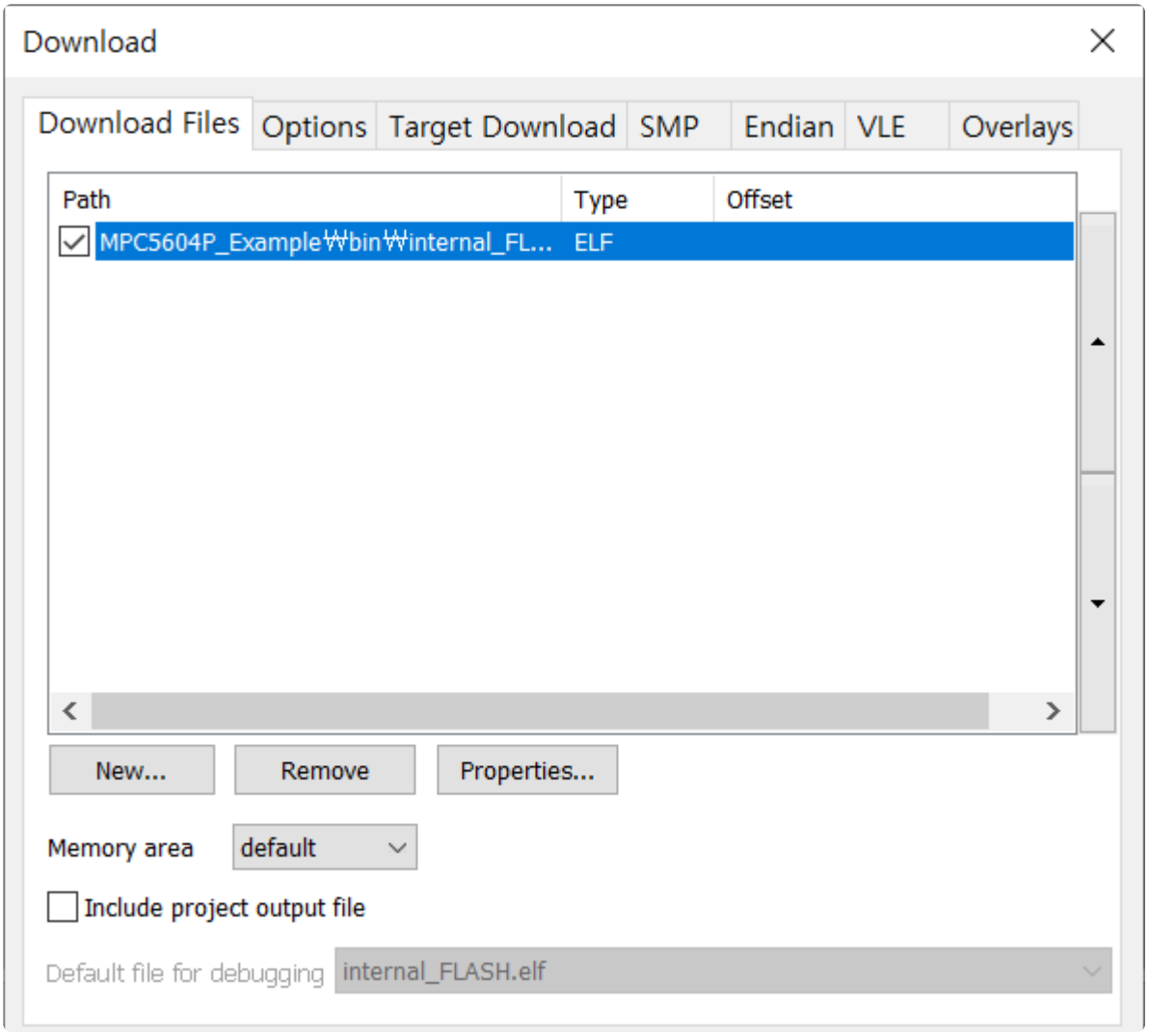
4. Click [Hardware]> [Use Software Breakpoints] on the top menu to activate it, and then select the target type to use in the [CPU] of [Hardware]> [Emulation Options…].



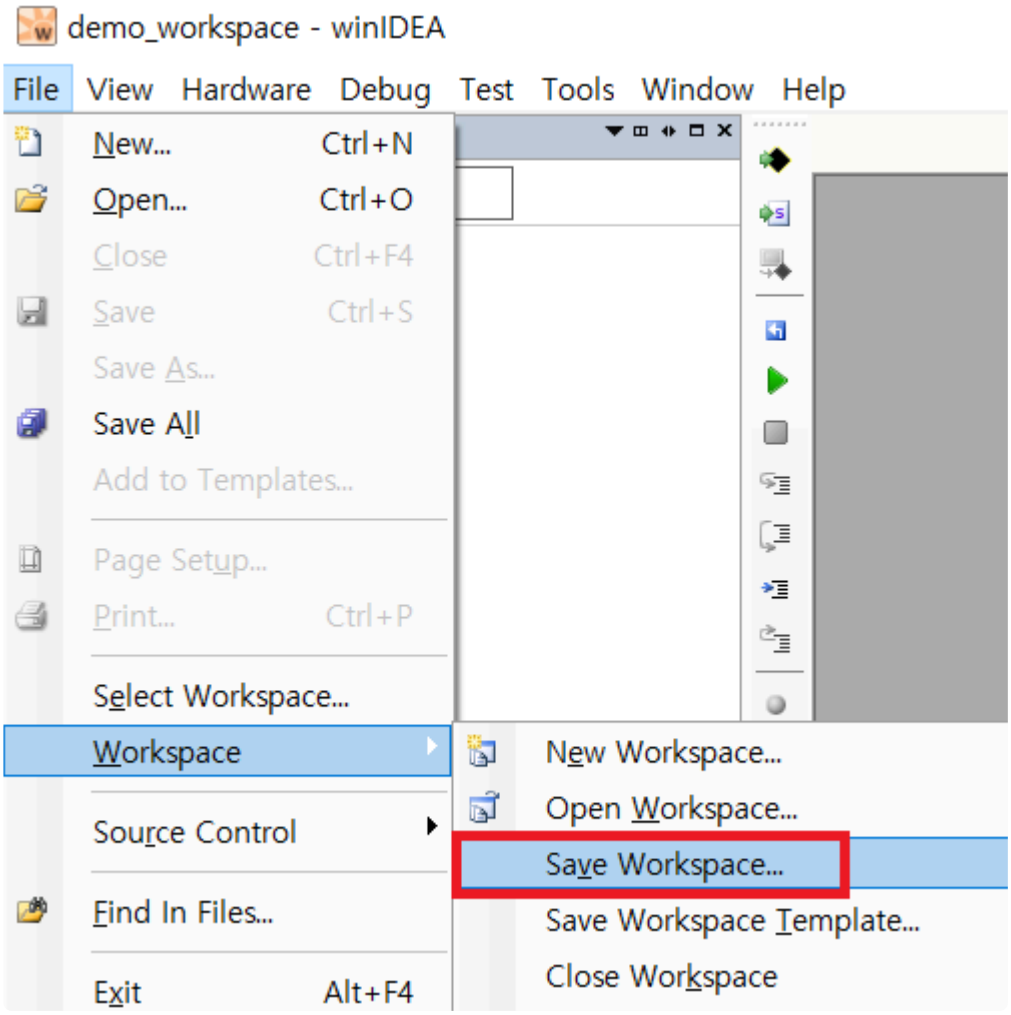


The specific options you need to set for each target may vary.

5. After the debugger setup is complete, you need to register the binary path of the software under test in the workspace. First, build the source code under test to generate the binary. Then from winIDEA's top menu [Debug]> [Files for Download …], select [New…] and add the binary generated.



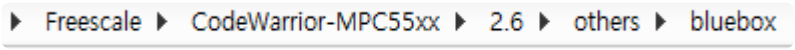
6. When everything is set up, save the workspace to create a winIDEA workspace file (.xjrf). The workspace file is used to configure the target test using winIDEA in Controller Tester.



You are now finished creating the winIDEA workspace for the target test.

2.3.3. Step2: Setting target environment in Controller Tester

Select a debugger in the [New Project] wizard of the target test project or [Target environment settings] of the project properties on Controller Tester. The list of selectable debuggers depends on the toolchain selected for the project.
Set the debugger to BlueBox.



The fields to be set are displayed according to the selection. If you are using BlueBox, the fields are shown in the table below.
Required fields are displayed in red in Controller Tester.

winidea_binary_path	The winIDEA execution file(winIDEA.exe) path. Required.
winidea_workspace_file_path	The path of the workspace file (.xjrf) created by winIDEA. Required.

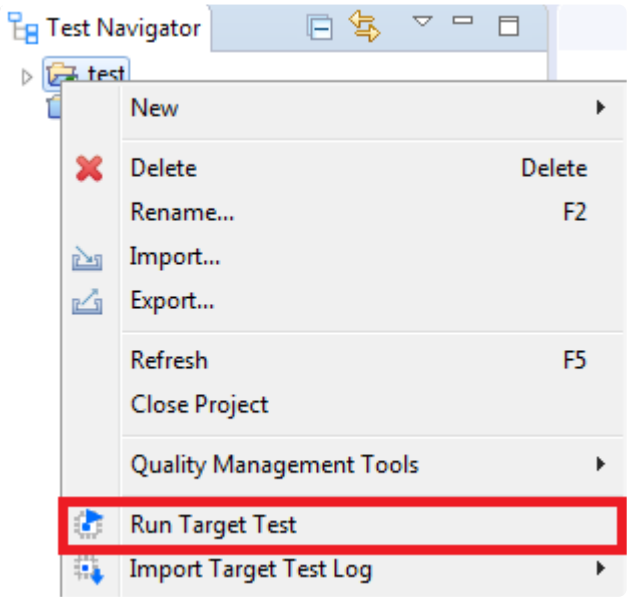
The default scripting language provided by Controller Tester is python. If you use a custom debugging script, you need to write it in python to work properly. If you write in other languages, refer to the iSYSTEM homepage to install additional SDKs.

When the target environment settings are complete, click the [OK] or [Finish] button. Now you are ready to run the target test.

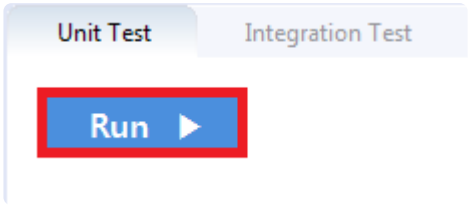
2.3.4. Step3: Run the target test

You can run a target test by selecting [Run Target Test] from the project context menu in the Test Navigator view or by clicking the [Run] button in the Test View.

- [Run Target Test]



- [Run]



Target tests cannot be run if winIDEA is running. You must exit winIDEA before running the target test in Controller Tester.

2.3.5. Debug the target test

1. After setting it as a target, right-click the test case in the ‘Unit Test’ view and click ‘Check Debug Information’
2. Build the user project directly or execute the build script registered in the ‘target environment’ setting in the controller tester project
3. Verify that the build was successful
4. Restore the original source by opening the project in Controller Tester
5. After running winIDEA, select the workspace containing the built project (.xjrf file)
6. Download to binary file target by selecting [Debug]> [Download]
7. Debugging mode by pressing the Run button at the top
8. Double-click [Project]> [Functions], move to the function location, and set the debugging point where you want
9. Press F5 to proceed debugging

2.4. IAR Embedded Workbench C-SPY Debugger

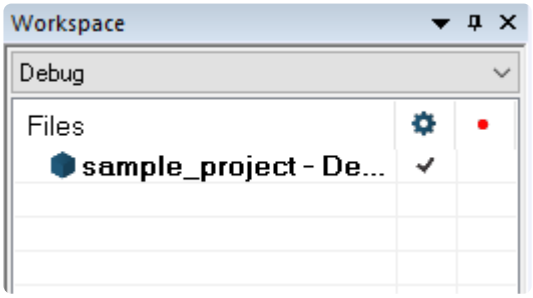
Controller Tester provides the ability to automatically run tests and get results in the target environment through the IAR Embedded Workbench C-SPY debugging function.
The list of targets supported by C-SPY can be found on the IAR website.

To test a target with the IAR Embedded Workbench C-SPY in the Controller Tester, you need a C-SPY compatible debugging probe. You need to create an IAR Embedded Workbench project and connect the debugging probe to be used with the PC where Controller Tester is installed before performing the target test.

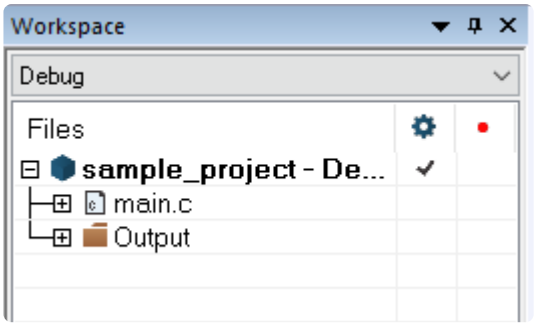
The list of debugging probes provided by IAR can be found on the homepage.

2.4.1. Step1: Creating an IAR embedded workbench project

1. Click [File]> [New Workspace] to create a new workspace and then click [Project]> [Create New Project…] to create a project file (.ewp). When a project file created, the project name is displayed in the [Workspace] view of the IAR Embedded Workbench.



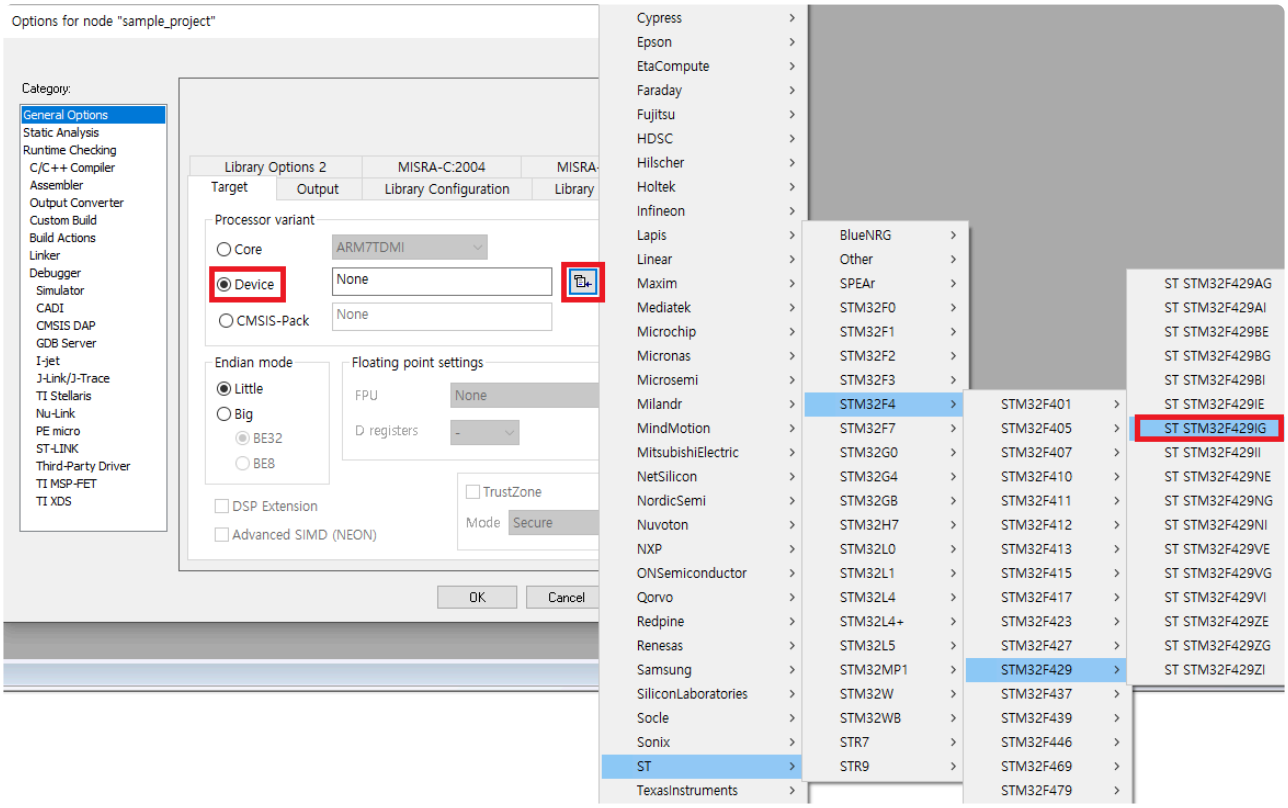
2. Next, you need to add the source files under test to the project. Right-click on the project, click [Add]> [Add Files…] and add the source files to be tested. The added source files are displayed in a hierarchical structure in [Workspace] view.



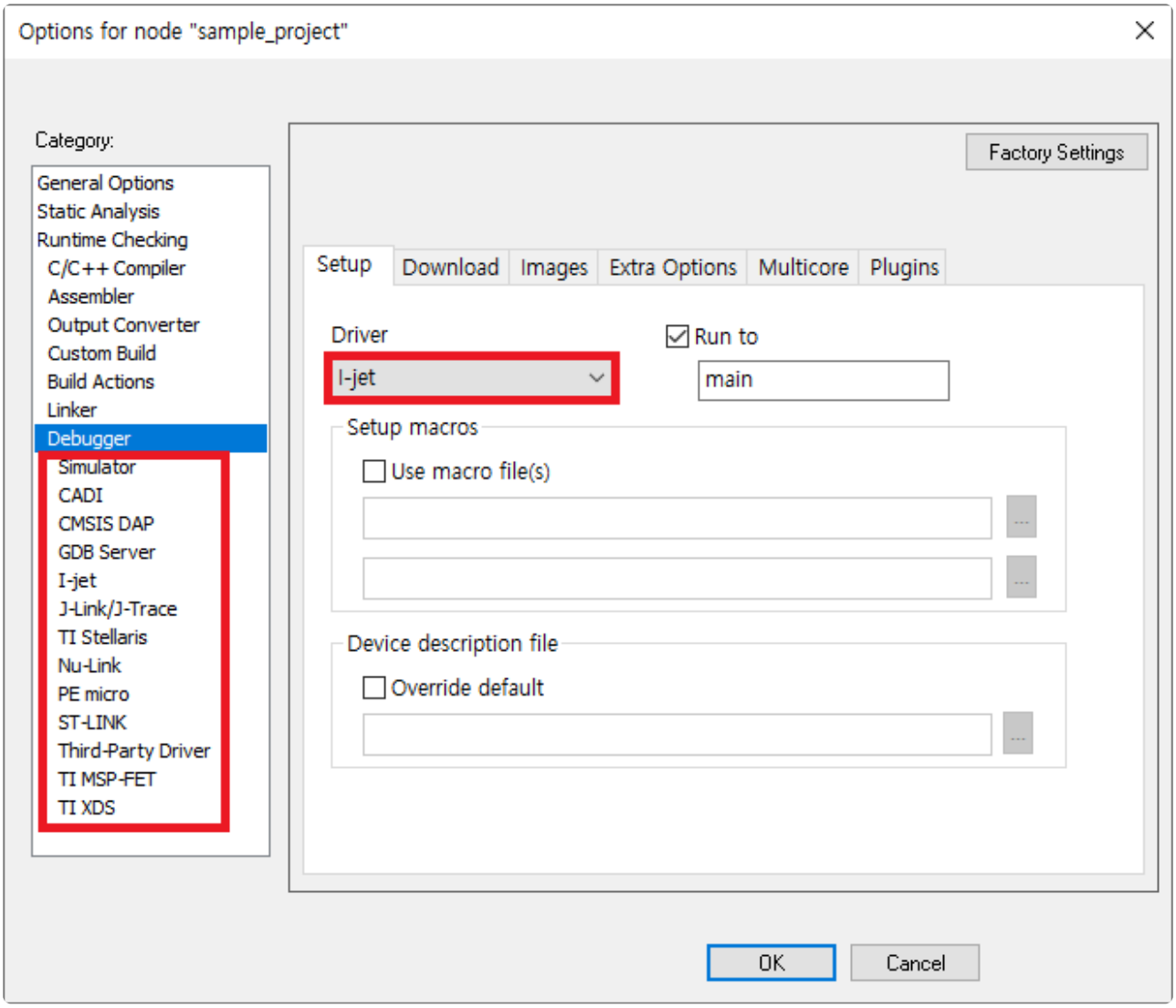
2.4.2. Step2: Setting an IAR project

If you created a project, you need to configure the project to use the C-SPY debugging feature. Right-click on the created project and select [Options …].

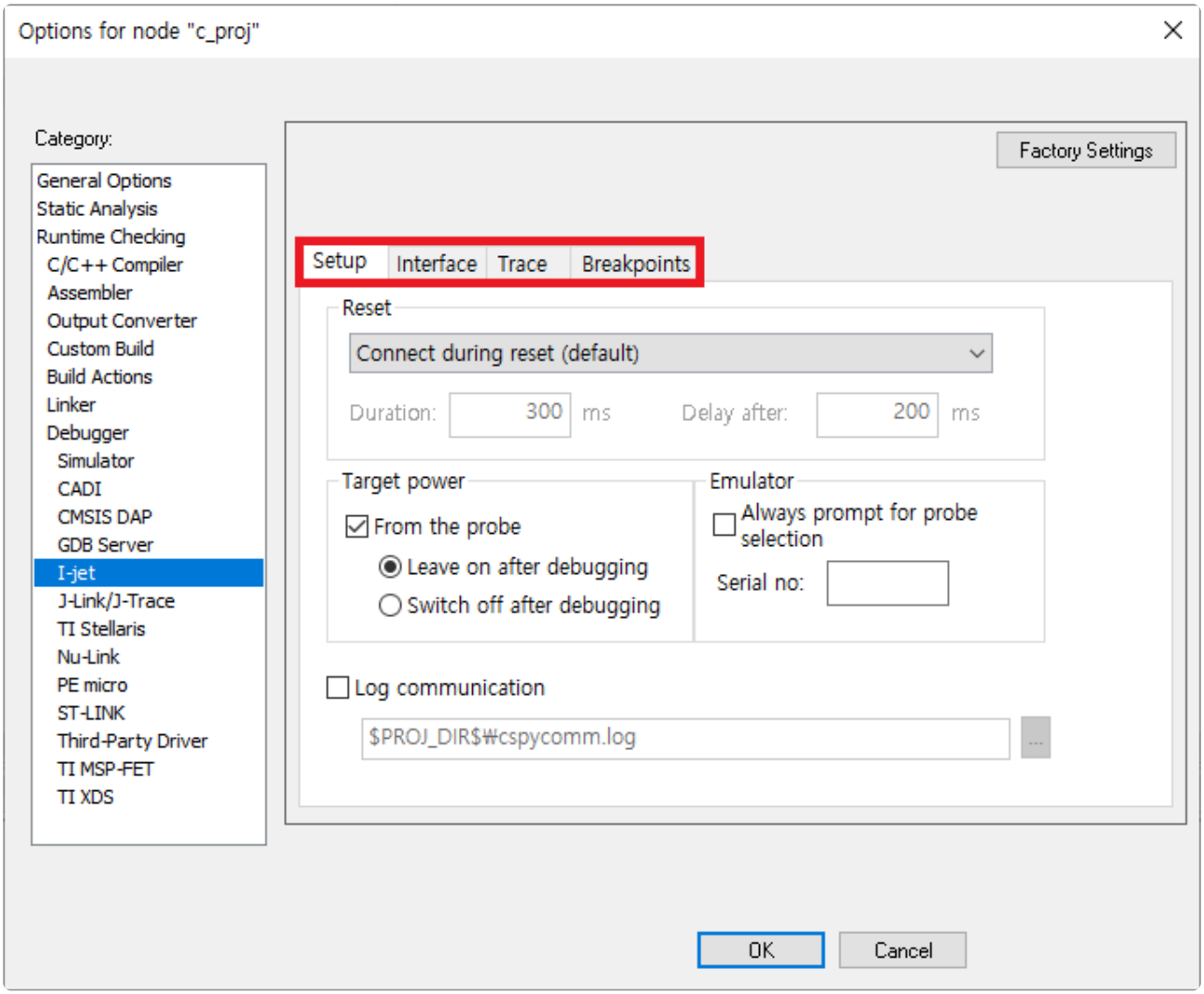
1. First, set [Processor variant] in [General Options]. For example, for ARM's STM32F429IG target, select Device and select a name that matches the target from the target list on the right.



2. Second, go to the category [Debugger] and select the debugging probe you want to use in the [Driver] field. Set the details in the Debugging Probe section at the bottom of the [Debugger] category, depending on how the selected debugging probe and PC are connected.



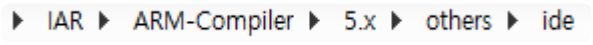
3. If I-jet is selected, select [I-jet] at the bottom of the [Debugger] category to set details. For a description of each setting tab, refer to the IAR debugger manual you want to use.



Now you are done creating and setting the IAR project for target testing.

2.4.3. Step3: Setting target environment in Controller Tester

Select a debugger in the [New Project] wizard of the target test project or [Target environment settings] of the project properties on Controller Tester. The list of selectable debuggers depends on the toolchain selected for the project.
When creating a project using the IAR toolchain, the debugger must be set to ide to use the IAR C-SPY debugging feature.



The fields to be set are displayed according to the selection. The fields for C-SPY are as shown in the table below.
Required fields are displayed in red in Controller Tester.

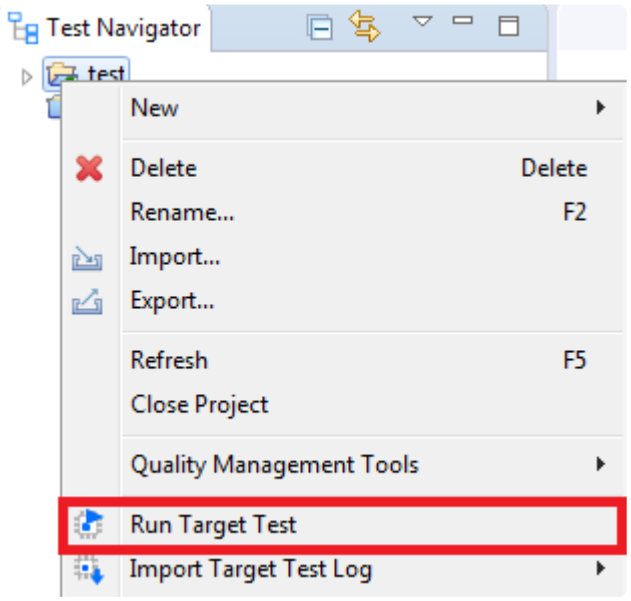
cspy_debug_general_xcl_file_path	The path to the debug.general.xcl file required when using the IAR Embedded Workbench C-SPY debugger. When creating an IAR project, the project file (.ewp) is automatically created in the [setting] folder in the saved location. Required.
cspy_debug_driver_xcl_file_path	Path to the debug.driver.xcl file required when using the IAR Embedded Workbench C-SPY debugger. When creating an IAR project, the project file (.ewp) is automatically created in the [setting] folder in the saved location. Required.

When the target environment settings are complete, click the [OK] or [Finish] button. Now you are ready to run the target test.

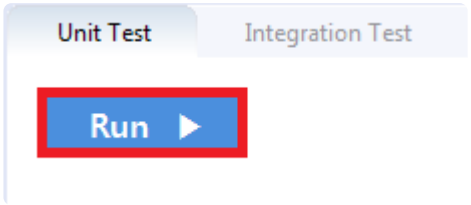
2.4.4. Step4: Run the target test

You can run a target test by selecting [Run Target Test] from the project context menu in the Test Navigator view or by clicking the [Run] button in the Test View.

- [Run Target Test]



- [Run]



2.4.5. Debug the target test

1. After setting it as a target, right-click the test case in the ‘Unit Test’ view and click ‘Check Debug Information’
2. Build the user project directly or execute the build script registered in the ‘target environment’ setting in the controller tester project
3. Verify that the build was successful
4. After IAR Workbench run, select the workspace containing the built project (.eww file)
5. Select the source file with the function to be tested in the workspace view, and click the left side of the line to add the debugging point
6. Right-click the project in the workspace view and open ‘Options …’ to check the Run to option check in the Debugger item and check that it is designated as ‘main’
7. Click the Download and Debug button at the top to start from main
8. Press F5 to proceed to the debugging point to debug

2.5. Texas Instruments Code Composer Studio (CCS v4 and later)

Controller Tester can run target tests using the Code Composer Studio debugger. Controller Tester uses debugging scripts supported by Code Composer Studio (since version 4.x) to run the tests in target environment and get results. Check the Code Composer Studio manual for a list of debugging devices you can connect to and use with Code Composer Studio.

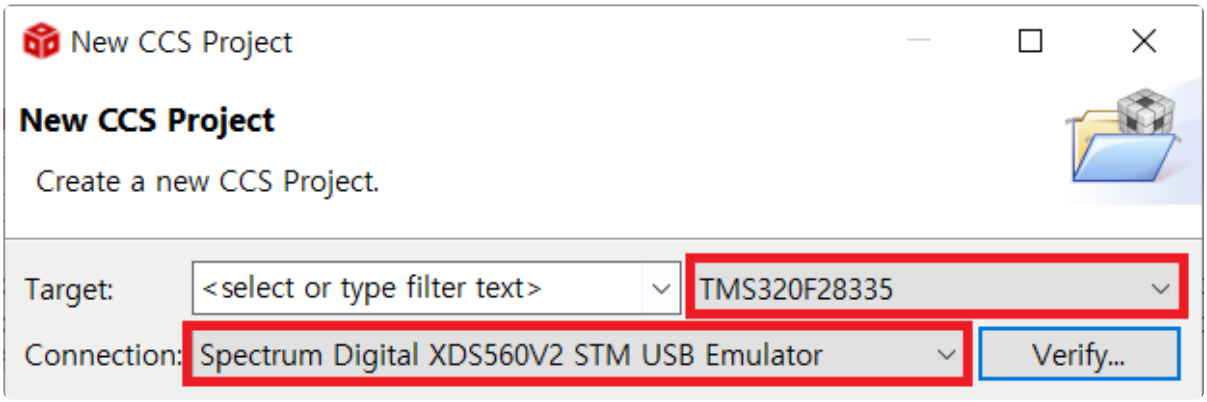
This document describes how to use Code Composer Studio debugger with following three steps.

- Step1: Create a project in Code Composer Studio
- Step2 : Setting target environment in Controller Tester
- Step3: Run the target test

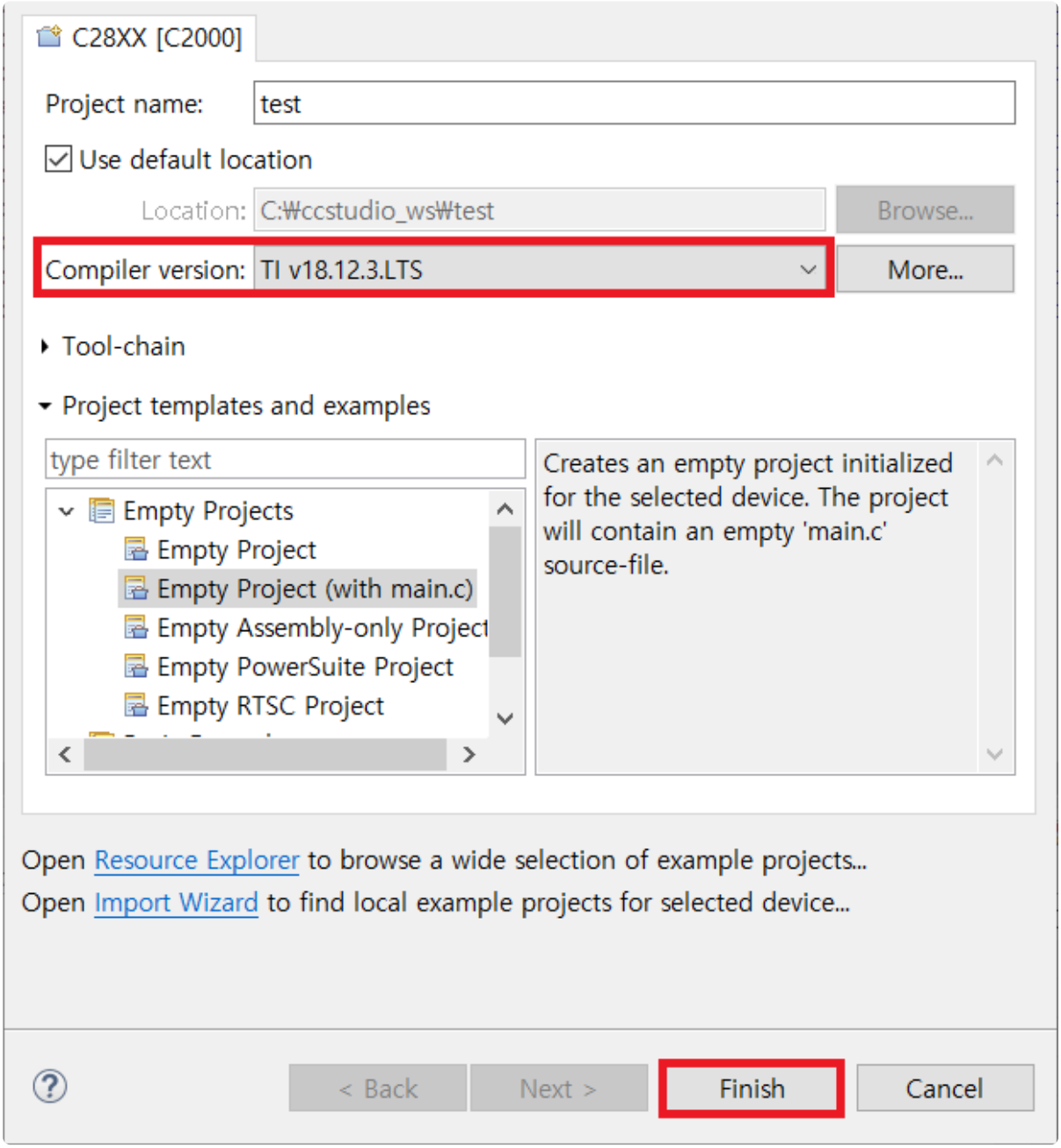
The example uses Spectrum Digital’s XDS560v2 as a debugger and Texas Instruments’ TMS320 as target device.

2.5.1. Step1: Create a project in Code Composer Studio

1. Run Code Composer Studio and create a new project. Select [File]-[New] from the top menu and select the desired project type. In this case, click [CCS Project] to create a project. After entering the target and debugger information used, click [Verify] to confirm that the connection is successful.



2. After verifying the debugger and target connections, enter the remaining settings. The example uses the C2000 Ti compiler. When you click [Finish], the CCS project is created in the workspace.



Code Composer Studio supports several more debuggers in addition to the built-in debuggers from Texas Instruments.

1. TI XDS USB (Code Composer Studio default)
2. BlackHawk JTAG emulator
3. Spectrum digital
4. MSP430 USB
5. MSP432 USB
6. Tiva/Stellaris ICDI

Controller Tester controls the debugger supported by Code Composer Studio with javascript. You can select the target and debugger details from the Project Settings screen in Code Composer Studio.

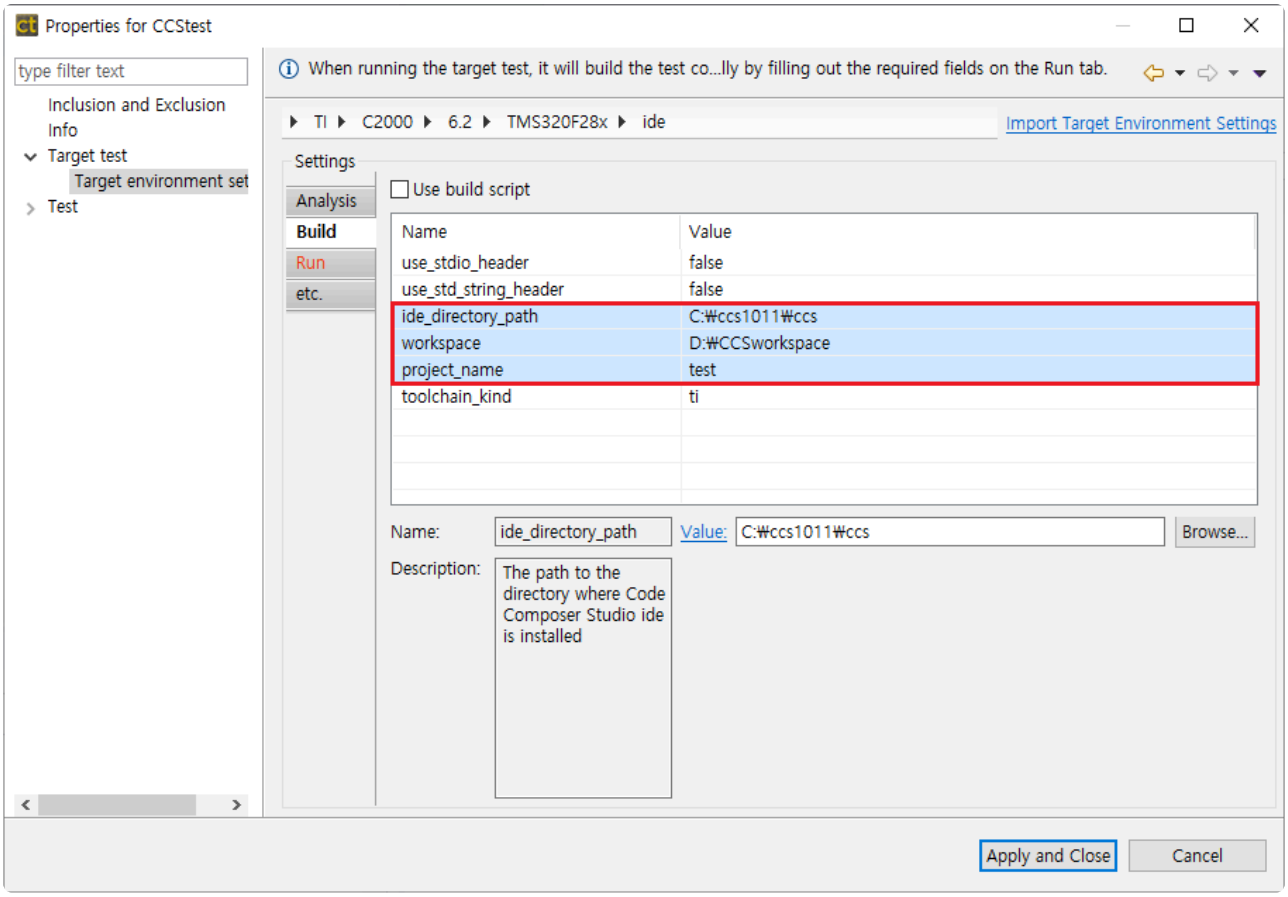
2.5.2. Step2 : Setting target environment in Controller Tester

1. Creat a CodeScroll Controller Tester project. For more information to create the project, refer to Texas Instruments Code Composer Studio in this document.

2. Right-click on the project in test navigator view and select [Properties] – [Target test] – [Target environment settings]. You can set up target environment in [Target environment settings]. Setting fields and the list of selectable debuggers depend on the toolchain selected for the project.

3. Select a debugger in [Target environment settings] of Controller Tester. This example selects IDE debugger to use Code Composer Studio debugger.
   ▶ TI ▶ C2000 ▶ 6.2 ▶ TMS320F28x ▶ ide

4. Enter needed informations on [Build] tab of [Target environment settings] for Code Composer Studio build. Following fields need to be filled and these are necessary.

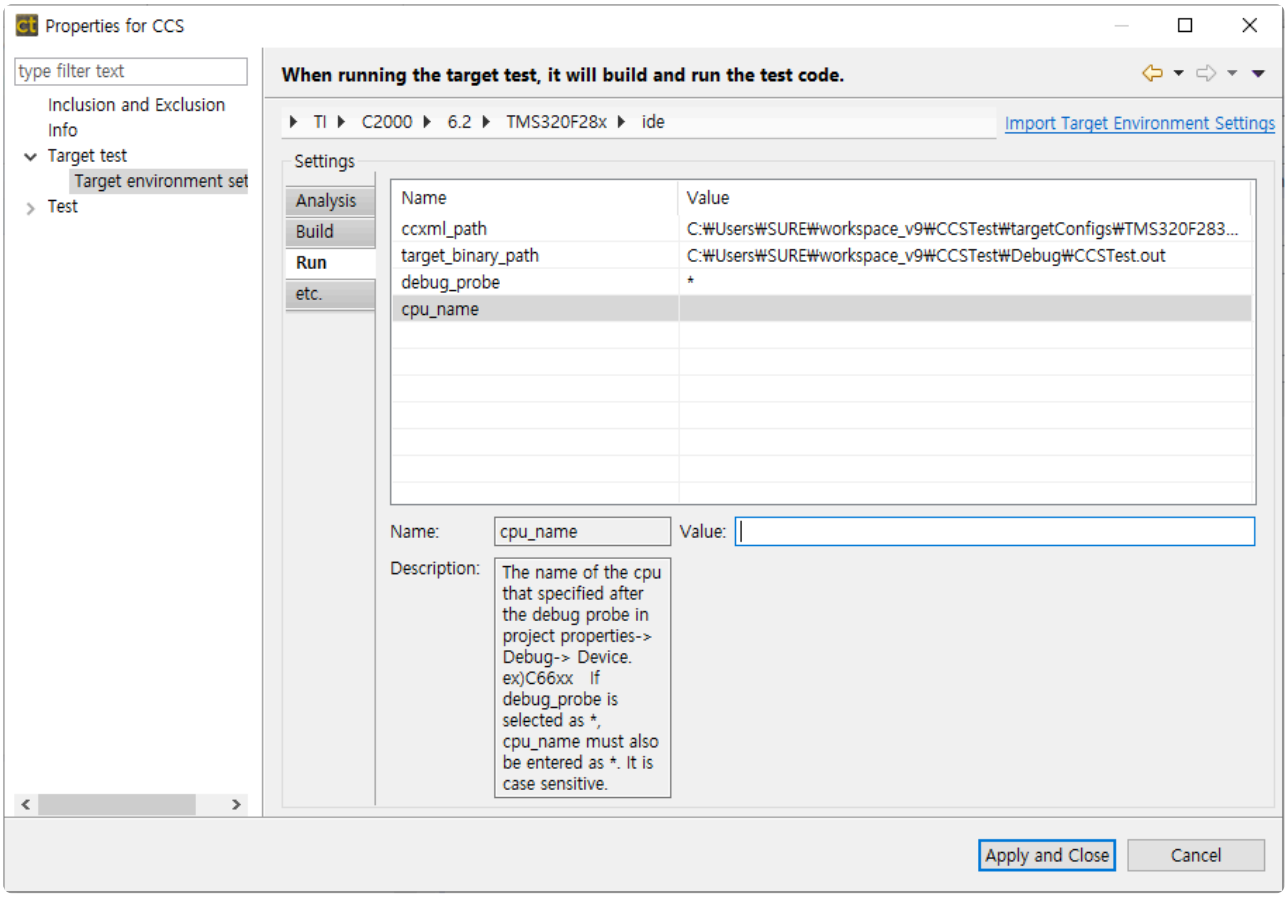


- Fileds of [Build] tab

ide_directory_path	Directory path of Code Composer Studio ex) *C:\ti\ccs930*
workspace	Directory path of Code Composer Studio workspace
project_name	Project name analyzed by Controller Tester

5. Enter needed informations on [Run] tab of [Target environment settings] for running target tests.

Following fields need to be filled and these are necessary.



- Fields of [Run] tab

ccxml_path	Enter a path of Code Composer Studio target configuration file. Check the project path and target name. File name is the target name selected in Code Composer Studio ex) *project-path*\targetConfig*target-name*.ccxml
target_binary_path	Enter a path of binary file created during build in Code Composer Studio. ex) *project-path*\Debug*project-name*.out
debug_probe	Refer to front of '/' in [Device] of Code Composer Studio properties and enter a target device name. (*Spectrum Digital XDS560V2 STM USB Emulator* in example shown below)
cpu_name	Refer to back part of '/' in [Device] of Code Composer Studio properties and enter a target device name. (*C28xx* in example shown below)

- Code Composer Studio properties
  - Right-click Code Composer Studio project and select [Properties] – [Debug] – [Device]

When only one debugger is connected to the target, debug_probe can be left as the

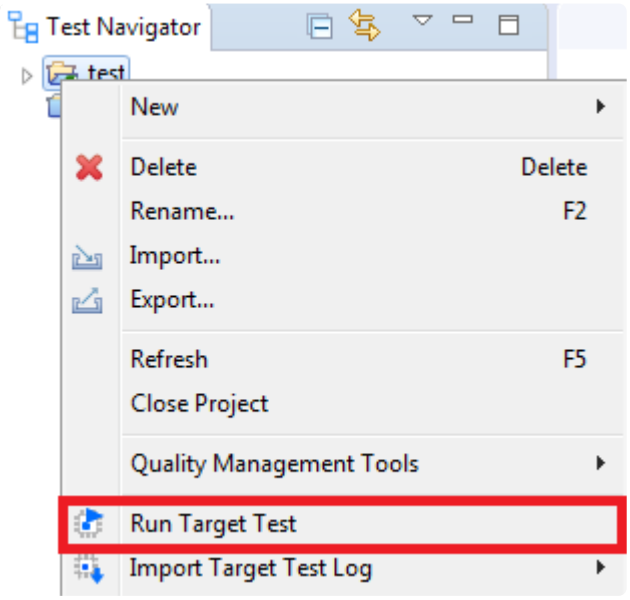
default (*). For single core cpu, you do not need to set cpu_name.

6. After finishing target environment settings, click [Finish] button. You are ready to do target tests.

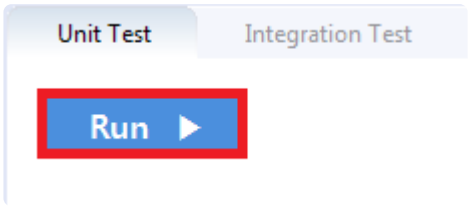
2.5.3. Step3: Run the target test

Before running the target test, you should stop using the workspace where the project you want to build is located. If you are using a workspace in the IDE, target testing does not work properly.
You can run a target test by selecting [Run Target Test] from the project context menu in the Test Navigator view or by clicking the [Run] button in the Test View.

- [Run Target Test]



- [Run]



> ! If Code Composer Studio is running during target test execution, a compilation error occurs.

> For more information on debug scripting in CCS, see the Texas Instruments home page.

2.5.4. Debug the target test

1. After setting it as a target, right-click the test case in the ‘Unit Test’ view and click ‘Check Debug Information’.
2. Run in debugging mode in Code Composer Studio.
3. Click [File] > [Open] File in Code Composer Studio.
4. Select the source file_number.c file with the function to be debugged in the Controller_Tester_workspace_path/.metadata/.plugins/com.codescroll.ut.embedded/project_name/TestFixture/cs
5. Add breakpoint where you want to debug
6. Run Debug

2.6. Microchip MPLAB IDE

This document describes how to run target tests using the Microchip MPLAB IDE in three steps.

2.6.1. Step1: Debugger script settings

In order to perform the target test in Controller Tester, the mdb.bat file included in MPLAB must be modified so that the log output from the debugger can be saved in a file format.
The mdb.bat file path is as follows.

For windows 32 bit

- `C:\Program Files\Microchip\MPLABX\vn.nn\mplab_ide\bin\mdb.bat`

For windows 64 bit

- `C:\Program Files (x86)\Microchip\MPLABX\vn.nn\mplab_ide\bin\mdb.bat`

Modify the code in the last line of the mdb.bat file as follows.

before modification

```
"%jdkhome:exe =exe%" -Dfile.encoding=UTF-8 -jar "%mdb_jar%" %1
```

after modification

```
call "%jdkhome:exe =exe%" -Dfile.encoding=UTF-8 -jar "%mdb_jar%" %1 >> %CT_TAR
GET_PATH%\mdb_log.txt
```

> ! Microchip MPLAB has a Korean encoding issue, so you should not include Korean in the Controller Tester workspace or project name.

2.6.2. Step2: Setting target environment in Controller Tester

Select the debugger in the target test project creation wizard in Controller Tester or in the target environment settings in the project properties. The list of debuggers to choose from depends on the toolchain selected when creating the project.
Set the debugger to ide.

▸ Microchip ▸ XC16 ▸ others ▸ MPLAB-PIC ▸ ide

Setting items are displayed according to the selected information.
Required fields are displayed in red in Controller Tester.

ide_directory_path	The path to the directory where Mplab ide is installed. Required.
project_directory_path	The directory path of the project. Required.
make_path	The path to the make.exe file. Just enter the path to make.exe used when building in the mplab project. Required.

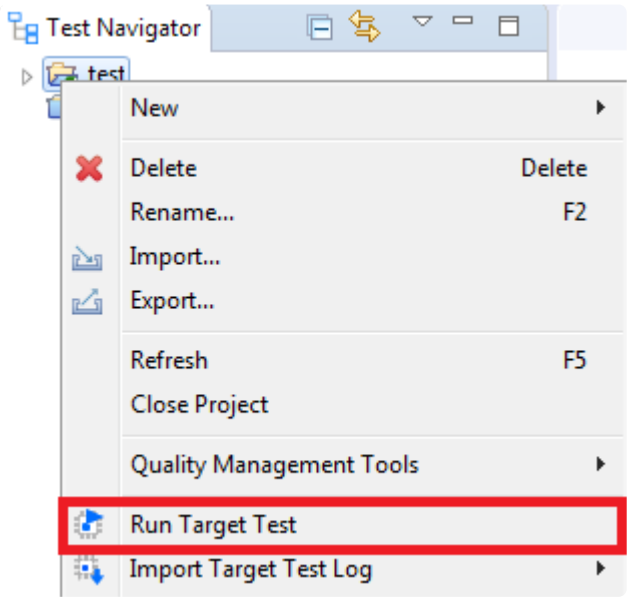
target_binary_path	Binary file to be uploaded to the target (binary location generated during build). Required.
debugger_tool	You can select the debugger tool information (select among ICD3, RealICE, PICkit3, SIM, PM3, LicensedDebugger, LicensedProgrammer, SK). Required.
chip	Product name of the chip under test (ex..dsPIC33EP512MU814). Required.

In order to perform the target tests in Controller Tester, the mdb.bat file must be modified as in Step1. When the target environment setting is finished, click the [OK] or [Finish] button. You are ready to perform target testing.

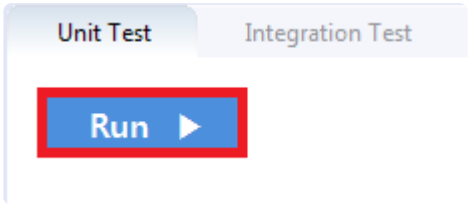
2.6.3. Step3: Run the target test

You can run a target test by selecting [Run Target Test] from the project context menu in the Test Navigator View or clicking the [Run] button in the Test View.

- [Run Target Test]



- [Run]



3. Target Build Guide

CodeScroll Controller Tester guides you through building target test code using target project information.

- IAR Embedded Workbench IDE
- Texas Instruments Code Composer Studio
- CodeWarrior IDE
- Hightec Development Platform IDE
- Tasking VX IDE
- Renesas CS+ IDE
- MPLAB X IDE
- Microsoft Visual Studio
- GNU Compiler

3.1. IAR Embedded Workbench IDE

The target preference page is automatically filled in according to the tool chain selected by the user. The type of debugger you can choose depends on your toolchain analysis settings.

To build an IAR Embedded Workbench, enter the required information in the Analysis and Build tab of the target preferences and click Done.
The contents to be filled out are as shown in the table below, which is mandatory.

- Analysis tab

cpu	CPU of target that can be selected from Core of Processor variant

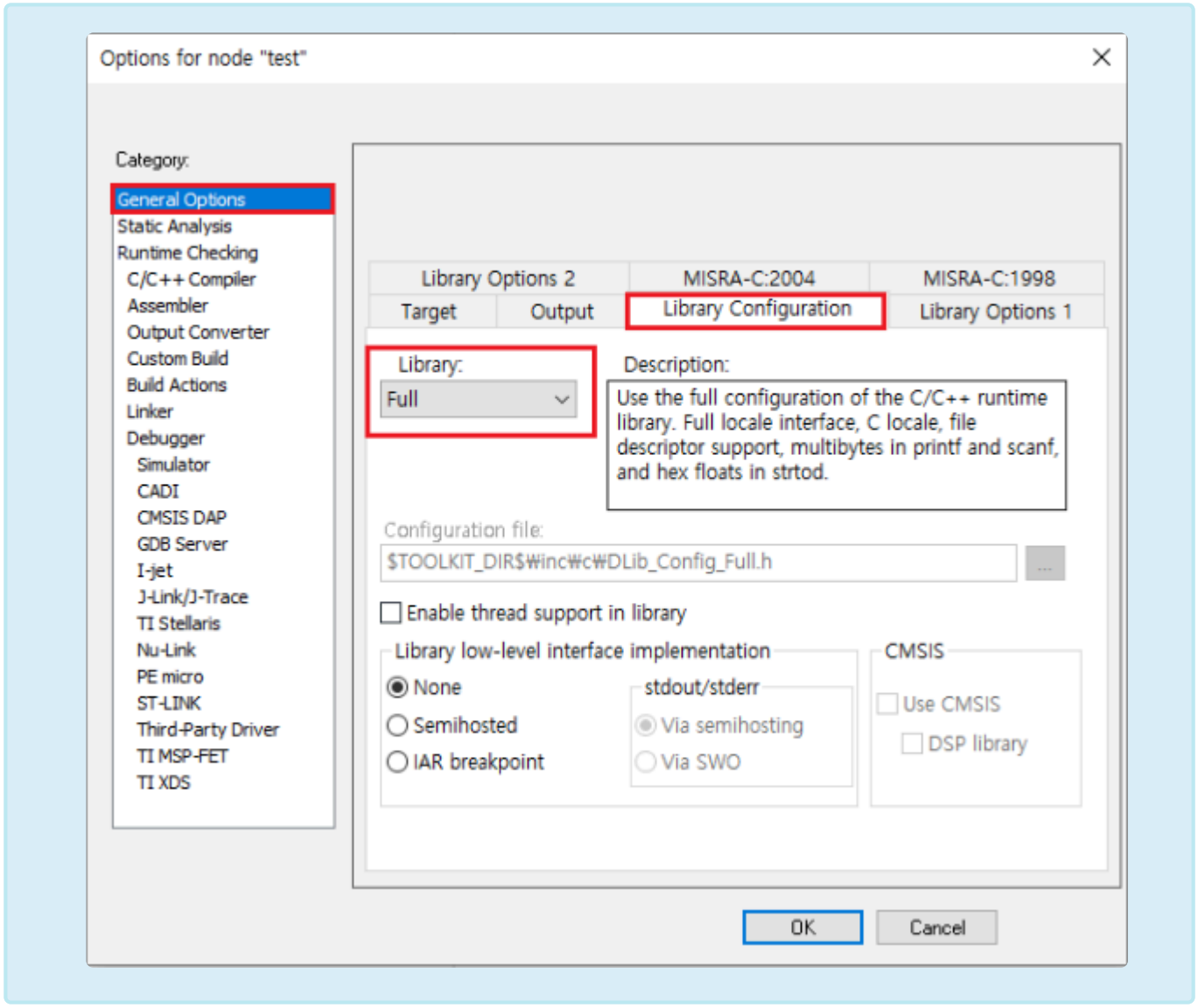
- Build tab

ide_directory_path	Installation path of the IAR Embedded Workbench IDE *ex. C:\Program Files (x86)\IAR Systems\Embedded Workbench 8.4*
project_file_path	Project file (.ewp) path of IAR Embedded Workbench

If you click the Done button on the target preference page without writing all the contents or if the path has been changed, you can set it again at ‘Right click on the project in the test navigator-> Properties-> Test target-> Target environment’.

After setting the target environment and clicking the Run button in the unit test view, the controller tester builds the target test code.

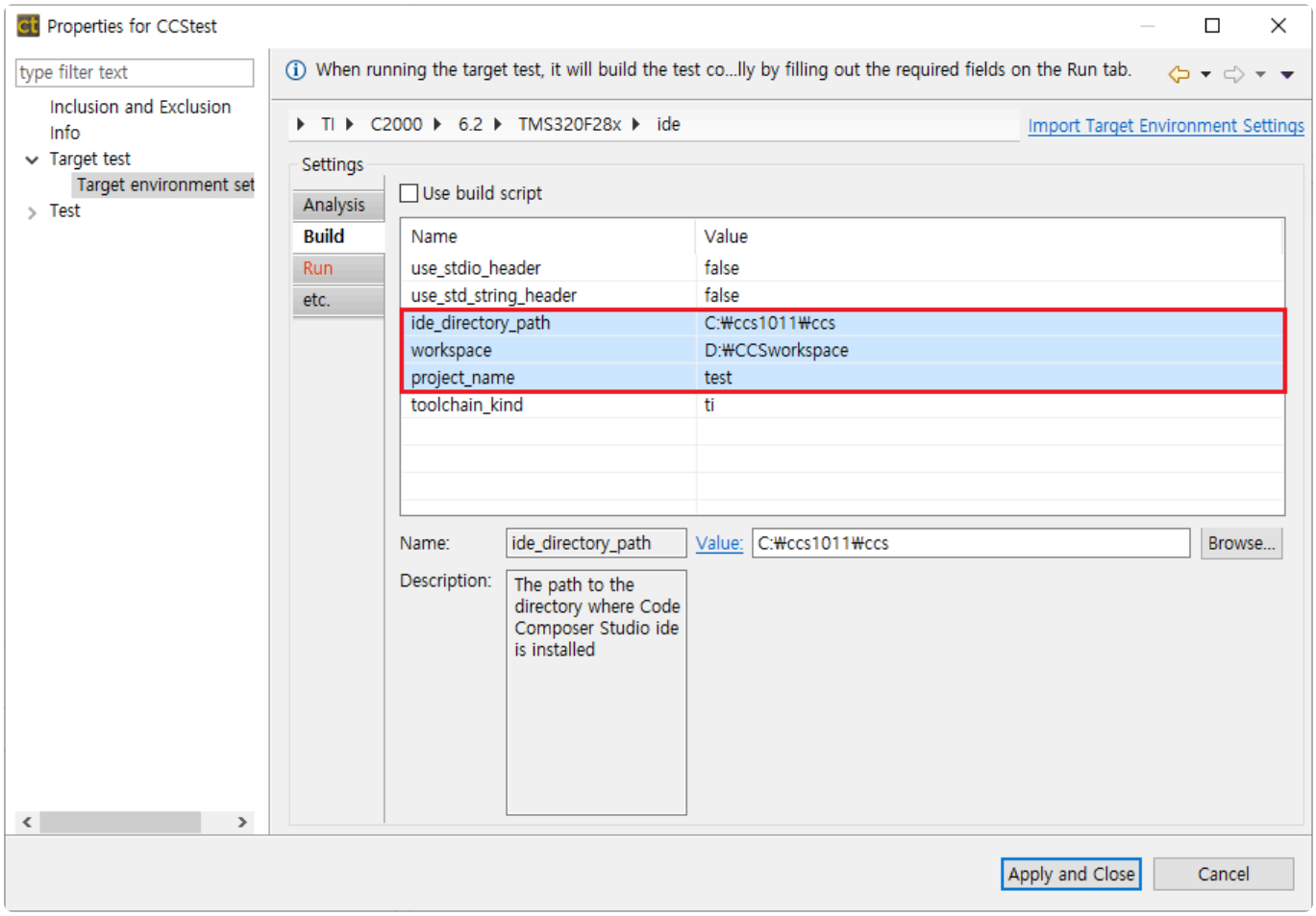
> When using the IO function of stdio.h, it is necessary to change the library settings. Right click on the project in the workspace -> Options -> General Options -> Library Configuration -> Library tab and change it to Full.


Options for node "test"
Category:
General Options
Static Analysis
Runtime Checking
C/C++ Compiler
Assembler
Output Converter
Custom Build
Build Actions
Linker
Debugger
Simulator
CADI
CMSIS DAP
GDB Server
I-jet
J-Link/J-Trace
TI Stellaris
Nu-Link
PE micro
ST-LINK
Third-Party Driver
TI MSP-FET
TI XDS
Library Options 2
MISRA-C:2004
MISRA-C:1998
Target
Output
Library Configuration
Library Options 1
Library:
Full
Description:
Use the full configuration of the C/C++ runtime library. Full locale interface, C locale, file descriptor support, multibytes in printf and scanf, and hex floats in strtod.
Configuration file:
$TOOLKIT_DIR$¥inc¥c¥DLib_Config_Full.h
Enable thread support in library
Library low-level interface implementation
None
Semihosted
IAR breakpoint
stdout/stderr
Via semihosting
Via SWO
CMSIS
Use CMSIS
DSP library
OK
Cancel

3.2. Texas Instruments Code Composer Studio

The target preference page is automatically filled in according to the tool chain selected by the user. The type of debugger you can choose depends on your toolchain analysis settings.

To build an Code Composer Studio, enter the required information in the Build tab of the target preferences and click Done.
The contents to be filled out are as shown in the table below, which is mandatory.



- Build tab

ide_directory_path	Directory path of Code Composer Studio *ex.C:\ti\ccs930*
workspace	Path to workspace directory in Code Composer Studio
project_name	The name of the Code Composer Studio project to be analyzed by Controller Tester

If you click the Done button on the target preference page without writing all the contents or if the path has been changed, you can set it again at ‘Right click on the project in the test navigator-> Properties-> Test target-> Target environment’.

After setting the target environment and clicking the Run button in the unit test view, the controller tester

builds the target test code.

> ! If Code Composer Studio is running during execution, a compile error occurs.

3.3. CodeWarrior IDE

The target preference page is automatically filled in according to the tool chain selected by the user. The type of debugger you can choose depends on your toolchain analysis settings.

To build an CodeWarrior project, enter the required information in the Build tab of the target preferences and click Done.
The contents to be filled out are as shown in the table below, which is mandatory.

- Build tab

ide_directory_path	Path to CodeWarrior IDE *ex. C:\Program Files (x86)\Freescale\CW for MPC55xx and MPC56xx 2.10, C:\Freescale\CW MCU*
ide_version	Classic or Eclipse(for MCUs)
project_file_path	In the case of Classic, the .mcp file named when creating the project, and in Eclipse, the .project file created when creating the project.

If you click the Done button on the target preference page without writing all the contents or if the path has been changed, you can set it again at ‘Right click on the project in the test navigator-> Properties-> Test target-> Target environment’.

After setting the target environment and clicking the Run button in the unit test view, the controller tester builds the target test code.

3.4. Hightec Development Platform IDE

The target preference page is automatically filled in according to the tool chain selected by the user. The type of debugger you can choose depends on your toolchain analysis settings.

To build an Hightec IDE project, enter the required information in the Build tab of the target preferences and click Done.
The contents to be filled out are as shown in the table below, which is mandatory.

- Build tab

ide_directory_path	Path to Hightec IDE *ex. C:\HIGHTEC\toolchains\arm\v4.6.5.0*
project_directory_path	Path of project directory created by HighTec IDE

If you click the Done button on the target preference page without writing all the contents or if the path has been changed, you can set it again at ‘Right click on the project in the test navigator-> Properties-> Test target-> Target environment’.

After setting the target environment and clicking the Run button in the unit test view, the controller tester builds the target test code.

3.5. Tasking VX IDE

The target preference page is automatically filled in according to the tool chain selected by the user. The type of debugger you can choose depends on your toolchain analysis settings.

To build an Tasking VX IDE project, enter the required information in the Build tab of the target preferences and click Done.
The contents to be filled out are as shown in the table below, which is mandatory.

- Build tab

ide_version	Version of Tasking VX IDE
makefile_path	Path of makefile created in Tasking VX IDE project
ide_directory_path	Path to the directory where Tasking VX IDE is installed *ex. C:\Program Files (x86)\TASKING\C166-VX v3.1r2*

If you click the Done button on the target preference page without writing all the contents or if the path has been changed, you can set it again at ‘Right click on the project in the test navigator-> Properties-> Test target-> Target environment’.

After setting the target environment and clicking the Run button in the unit test view, the controller tester builds the target test code.

3.6. Renesas CS+ IDE

The target preference page is automatically filled in according to the tool chain selected by the user. The type of debugger you can choose depends on your toolchain analysis settings.

To build an Renesas CS+ IDE project, enter the required information in the Build tab of the target preferences and click Done.
The contents to be filled out are as shown in the table below, which is mandatory.

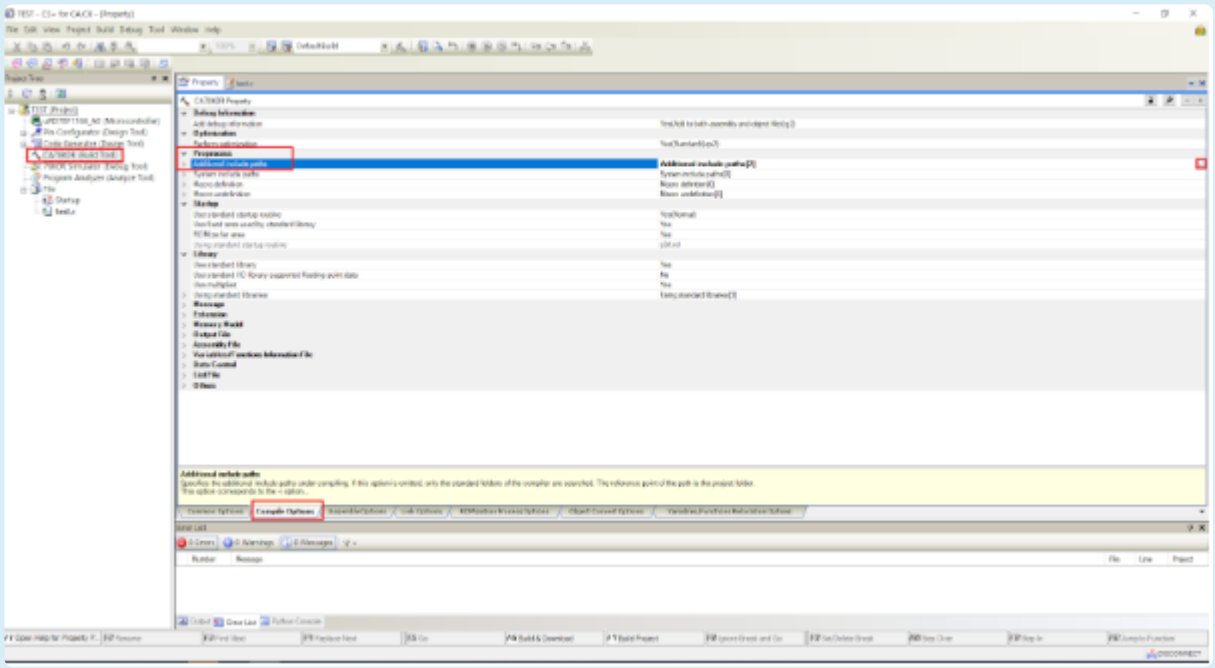
- Build tab

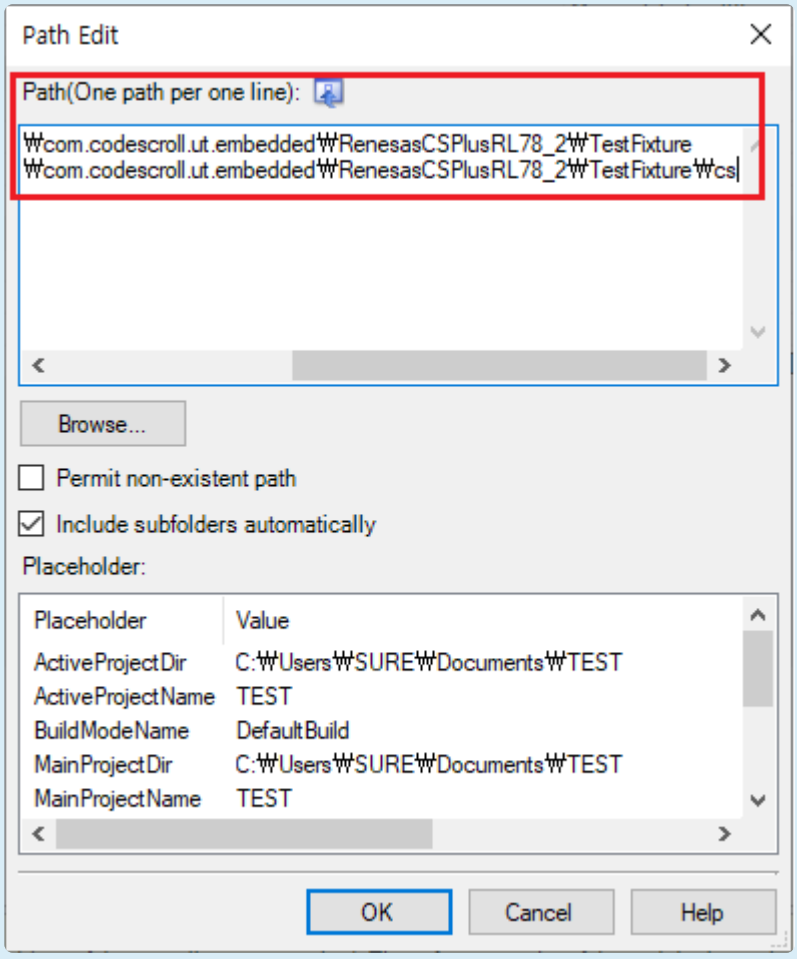
ide_directory_path	Directory path of Renesas CS + IDE *ex. C:\Program Files (x86)\Renesas Electronics*
ide_kind	IDE kind(CS+)
workspace_path	This is only necessary for the Renesas HEW IDE, so you do not need to enter it in CS +.
project_file_path	Project file path created by Renesas CS+(.mtpj)

If you click the Done button on the target preference page without writing all the contents or if the path has been changed, you can set it again at ‘Right click on the project in the test navigator-> Properties-> Test target-> Target environment’.

After setting the target environment and clicking the Run button in the unit test view, the controller tester builds the target test code.

When exporting test codes from Controller Tester, some reference relative paths.
To reference this path when building in the Renesas CS+, add an environment variable.
- Add below paths at property of Build Tool-> Compile Options -> Preprocess -> Additional include paths
(CTWORKSPACE) \.metadata\.plugins\com.codescroll.ut.embedded\ *CT project name* \TestFixture
(CTWORKSPACE) \.metadata\.plugins\com.codescroll.ut.embedded\ *CT project name* \TestFixture\cs




Path Edit
Path(One path per one line):
₩com.codescroll.ut.embedded₩RenesasCSPlusRL78_2₩TestFixture
₩com.codescroll.ut.embedded₩RenesasCSPlusRL78_2₩TestFixture₩cs
Browse...
Permit non-existent path
Include subfolders automatically
Placeholder:
Placeholder
Value
ActiveProjectDir
C:₩Users₩SURE₩Documents₩TEST
ActiveProjectName
TEST
BuildModeName
DefaultBuild
MainProjectDir
C:₩Users₩SURE₩Documents₩TEST
MainProjectName
TEST
OK
Cancel
Help

3.7. MPLAB X IDE

The target preference page is automatically filled in according to the tool chain selected by the user. The type of debugger you can choose depends on your toolchain analysis settings.

To build an MPLAB X IDE project, enter the required information in the Build tab of the target preferences and click Done.
The contents to be filled out are as shown in the table below, which is mandatory.

- Build tab

ide_directory_path	Installation path of MPLAB X IDE *ex. C:\Program Files (x86)\Microchip\MPLABX\v5.35*
project_directory_path	Project directory path created in MPLAB X IDE

If you click the Done button on the target preference page without writing all the contents or if the path has been changed, you can set it again at ‘Right click on the project in the test navigator-> Properties-> Test target-> Target environment’.

After setting the target environment and clicking the Run button in the unit test view, the controller tester builds the target test code.

3.8. Microsoft Visual Studio

The target preference page is automatically filled in according to the tool chain selected by the user. The type of debugger you can choose depends on your toolchain analysis settings.

To build an Microsoft Visual Studio project, enter the required information in the Build tab of the target preferences and click Done.
The contents to be filled out are as shown in the table below, which is mandatory.

- Build tab

ide_directory_path	Installation path of Microsoft Visual Studio *ex. C:\Program Files (x86)\Microsoft Visual Studio 10.0*	
build_configuration	Configuration and platform to test the target solution ex. Release	Win32
sin_path	File path of target solution (.sin file)	

If you click the Done button on the target preference page without writing all the contents or if the path has been changed, you can set it again at ‘Right click on the project in the test navigator-> Properties-> Test target-> Target environment’.

After setting the target environment and clicking the Run button in the unit test view, the controller tester builds the target test code.

3.9. GNU Compiler

The target preference page is automatically filled in according to the tool chain selected by the user. The type of debugger you can choose depends on your toolchain analysis settings.

To build an GNU Compiler code, enter the required information in the Build tab of the target preferences and click Done.
The contents to be filled out are as shown in the table below, which is mandatory.

- Build tab

makefile_path	Path of user-made makefile

If you click the Done button on the target preference page without writing all the contents or if the path has been changed, you can set it again at ‘Right click on the project in the test navigator-> Properties-> Test target-> Target environment’.

After setting the target environment and clicking the Run button in the unit test view, the controller tester builds the target test code.

4. Sharing Projects with Other Users

You can share the CodeScroll Controller Tester projects with others.
Controller Tester 3.3 or later uses [Export Project] and [Import Project] functions.

- Guide to Share Projects
- Guide to Share RTV Projects

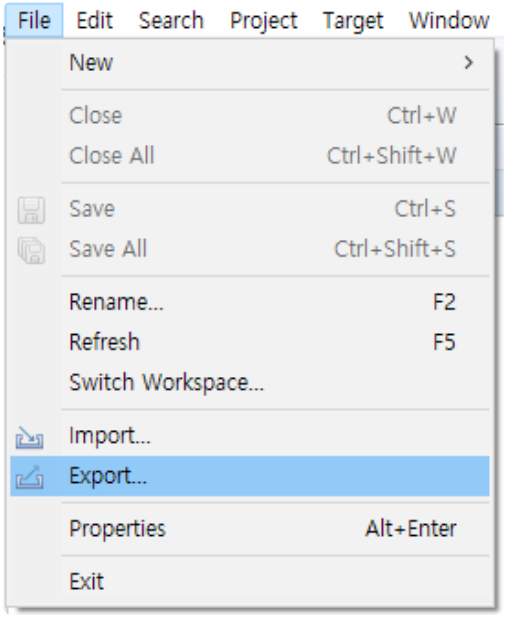
4.1. (Ver.3.3 or later) Guide to Share Projects

From Controller Tester 3.3, you can easily share a project with the [Export Project] and [Import Project] functions.

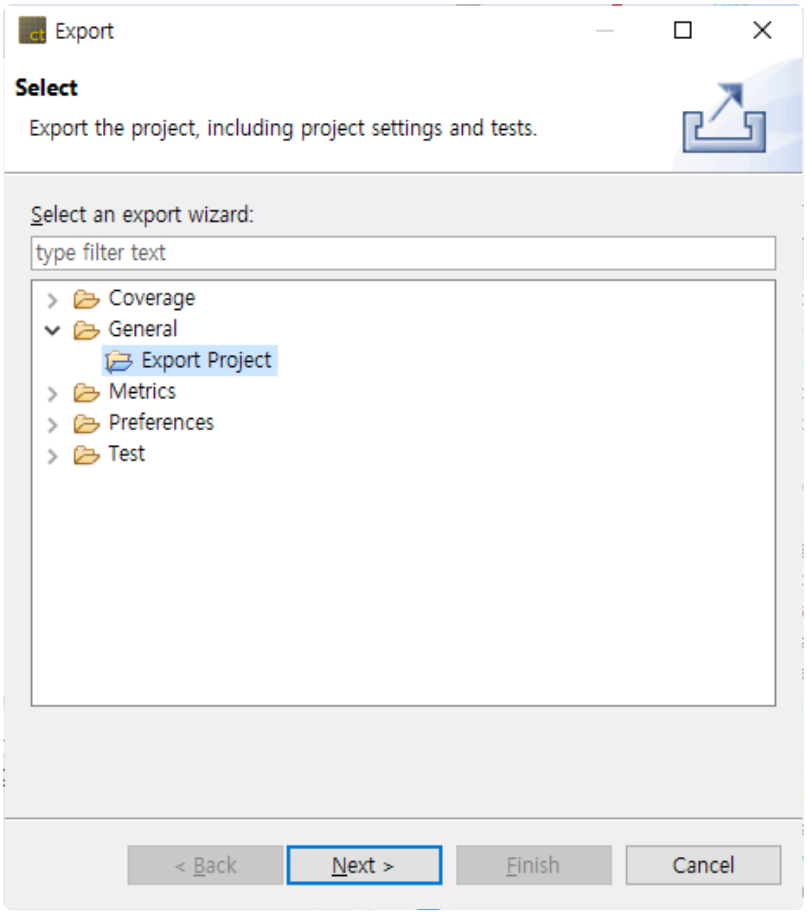
4.1.1. Export project

You can export projects, including project setup and testing.

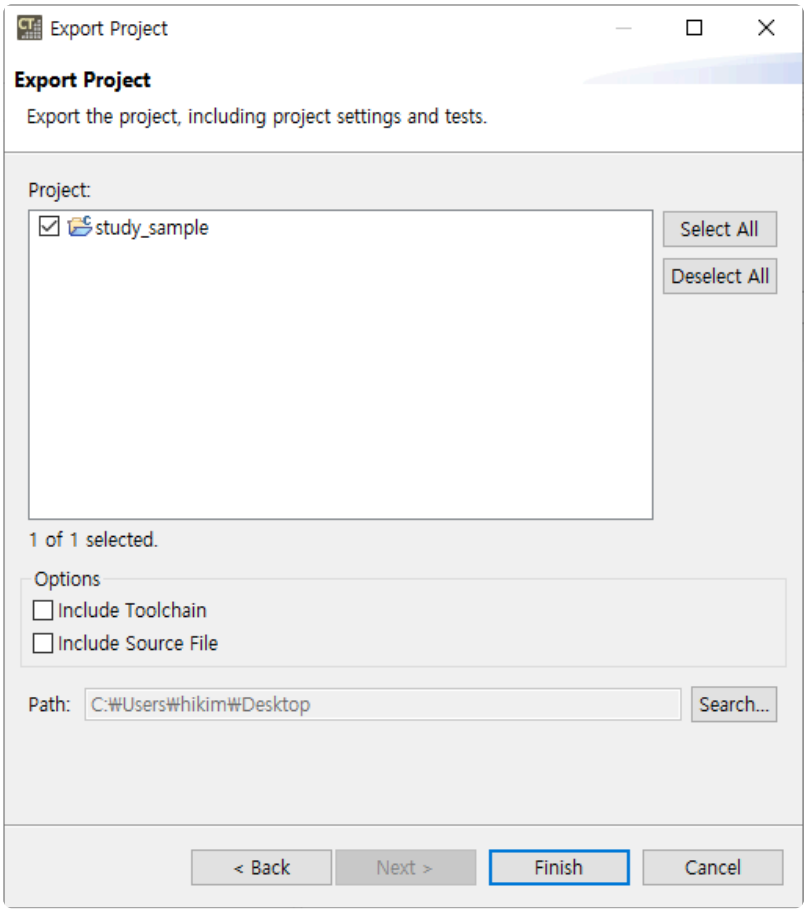
1. On the main menu, click [File] > [Export]. The Export Wizard opens.



2. Click [General] > [Export Project].



3. After selecting the project to export and the path to export, click the [Finish] button.



- (Ver.3.7 or later) When you export a project, you can export it including the toolchain and source files.

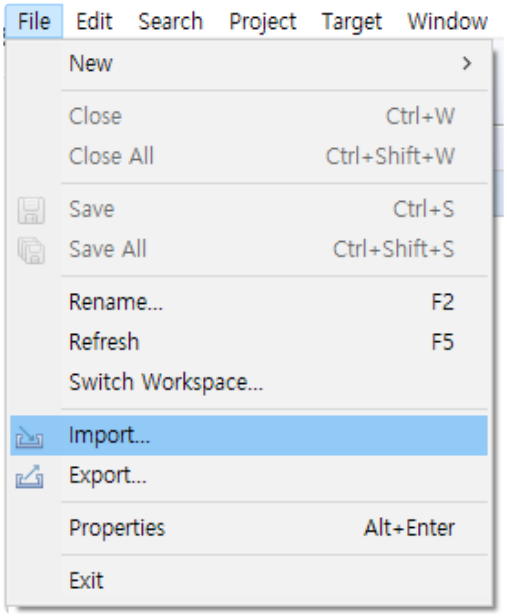
4. You can see that there is a folder containing the exported project name in the exported path. Compress the folder and move it to the computer of the user you want to share.

4.1.2. Import project

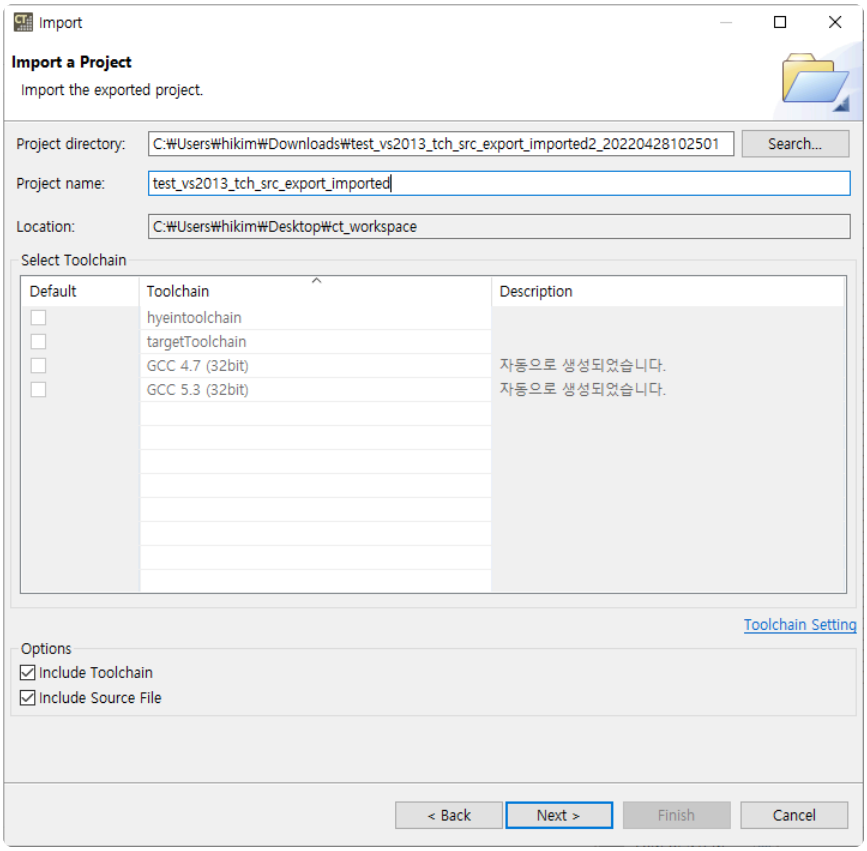
Using the Import Project function, you can import a project exported from another PC into the workspace.

Import general C/C++ Project

1. Click [File] > [Import] in the main menu. The Import Wizard opens.



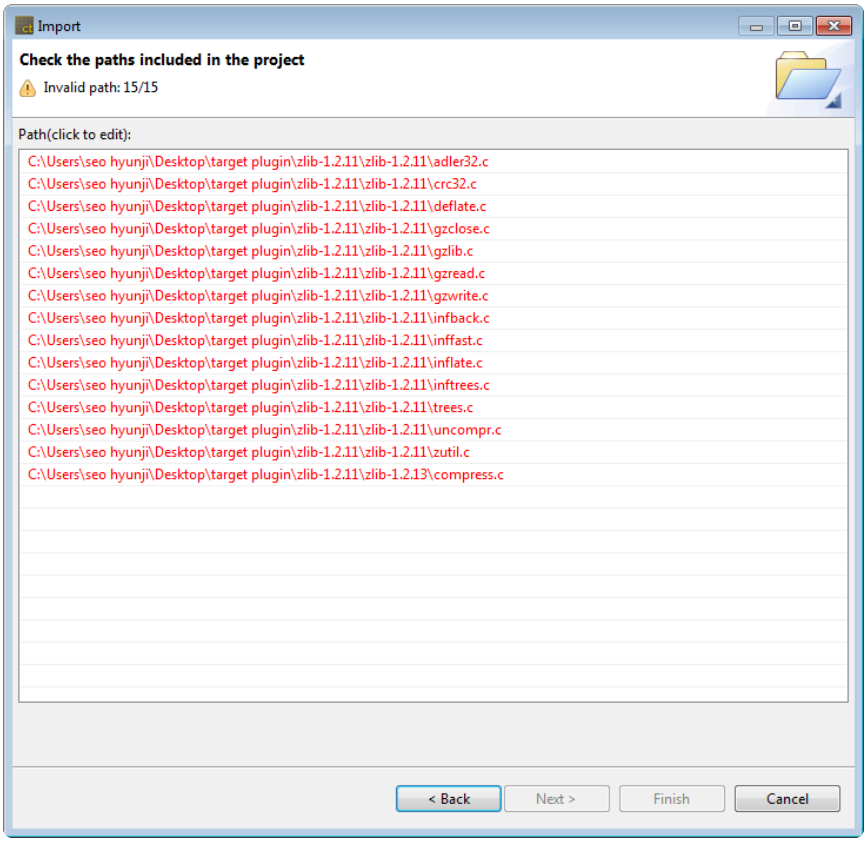
2. Click [General] > [Import Project] and then click the [Next] button.
3. Click the [Browse] button to find the directory corresponding to the exported project.
4. When you select a directory, the toolchain is automatically selected from the project information to be imported. If a project with the same name already exists in the workspace, you need to modify the project name.



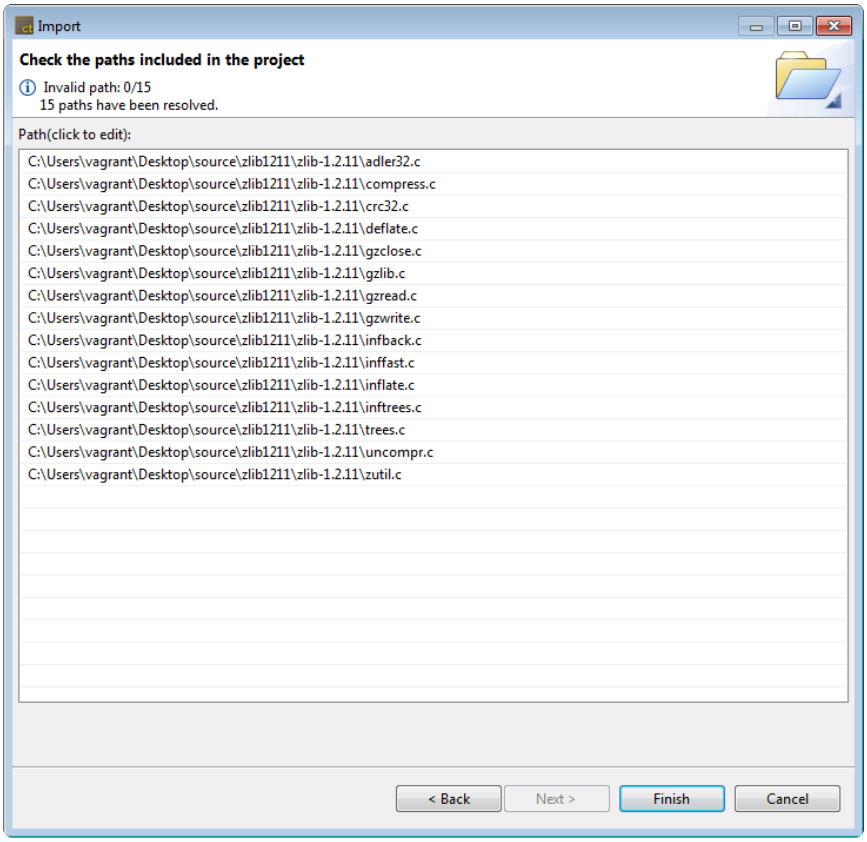
- (After Ver.3.7) If the directory contains a toolchain or source codes, its options are checked automatically.

If there is no toolchain with the same name as the toolchain of the project to be imported, you must first export and then import the toolchain of the project to be imported. For details, see [Import Toolchain] and [Export Toolchain] in the CodeScroll Controller Tester document.

5. Click the [Next] button.
6. You can check the source path included in the project to be imported. Invalid paths are marked in red and can be modified by clicking on the path window.



7. If there is an invalid path, modifying one file path automatically modifies the associated file path. At this time, you can check the number of modified routes at the top.



If is not in absolute path Windows format, the path is not checked for validity.

8. Click the [Finish] button.

Import RTV projects

RTV C/C++ projects can be imported in the same way as regular C/C++ project imports.

1. Click [File] -> [Import] in the main menu. In the Import Wizard, select [General] -> [Import Project] and click [Next].
2. Click the [Browse] button to select the directory of the project to be imported. When you select a directory, the toolchain is automatically selected from the project information to be imported. Click the [Next] button.

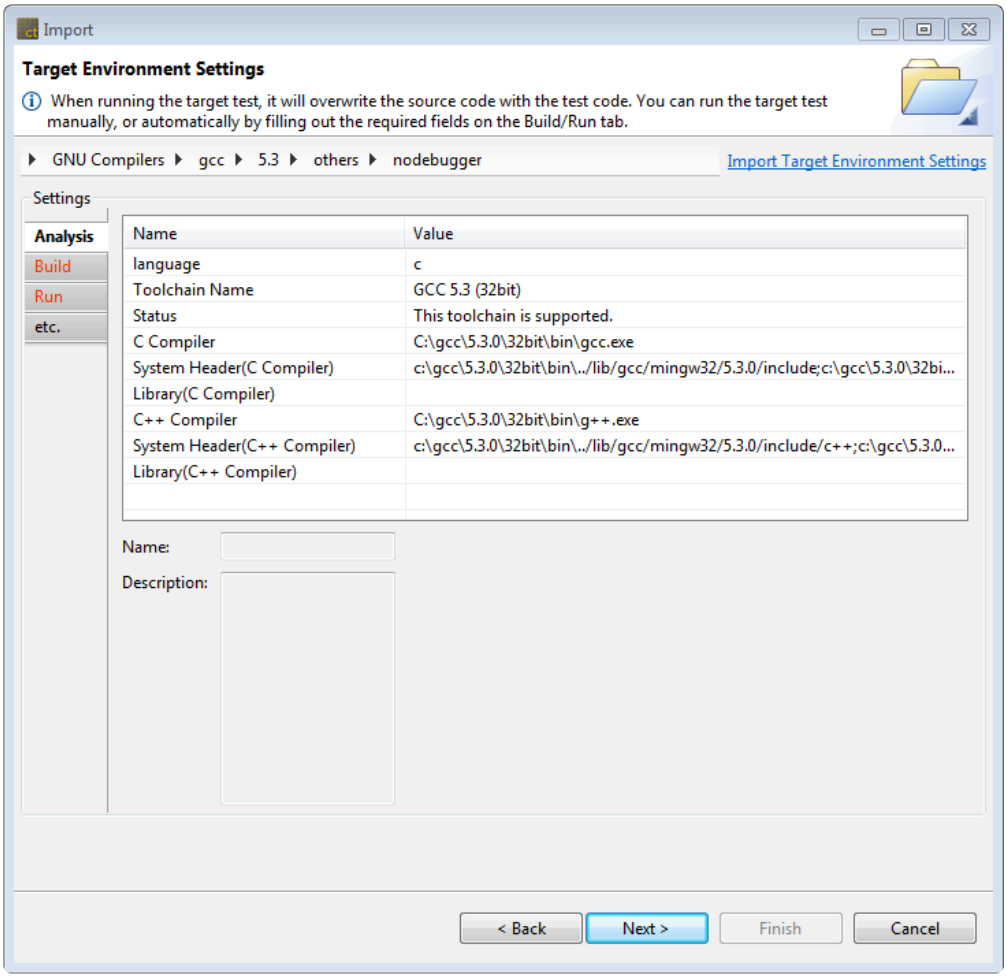
> If there is no RTV server and toolchain information identical to the project to be imported, RTV server and toolchain information is automatically generated from the project to be imported.

3. You can check the source path included in the project to be imported. Invalid paths are marked in red and can be modified by clicking on the path list.
4. Click the [Finish] button.

Import target project

When importing a target C/C++ project, additional target preferences must be created.

1. Click [File] -> [Import] in the main menu. In the Import Wizard, select [General] -> [Import Project] and click [Next].
2. Click the [Browse] button to select the directory of the project to be imported. When you select a directory, the toolchain is automatically selected from the project information to be imported. Click the [Next] button.
3. In the case of a target project, the [Target Environment setting] window appears. The target environment setting is loaded from the project information to be imported. Items with invalid paths are displayed in red.



✻ Even if it is not a target C/C++ project, if it is a project that includes target environment settings, the target environment setting window appears when [Import Project] is executed.

! Even if it contains an invalid path, you can complete the target environment setup and proceed to the next one, but the one-click target test may not be executed.

4. Complete the target environment settings and click the [Next] button.
5. You can check the source path included in the project to be imported. Invalid paths are marked in red and can be modified by clicking on the path list.
6. Click the [Finish] button.

4.2. (Ver.3.2 or earlier) Guide to Share RTV Projects

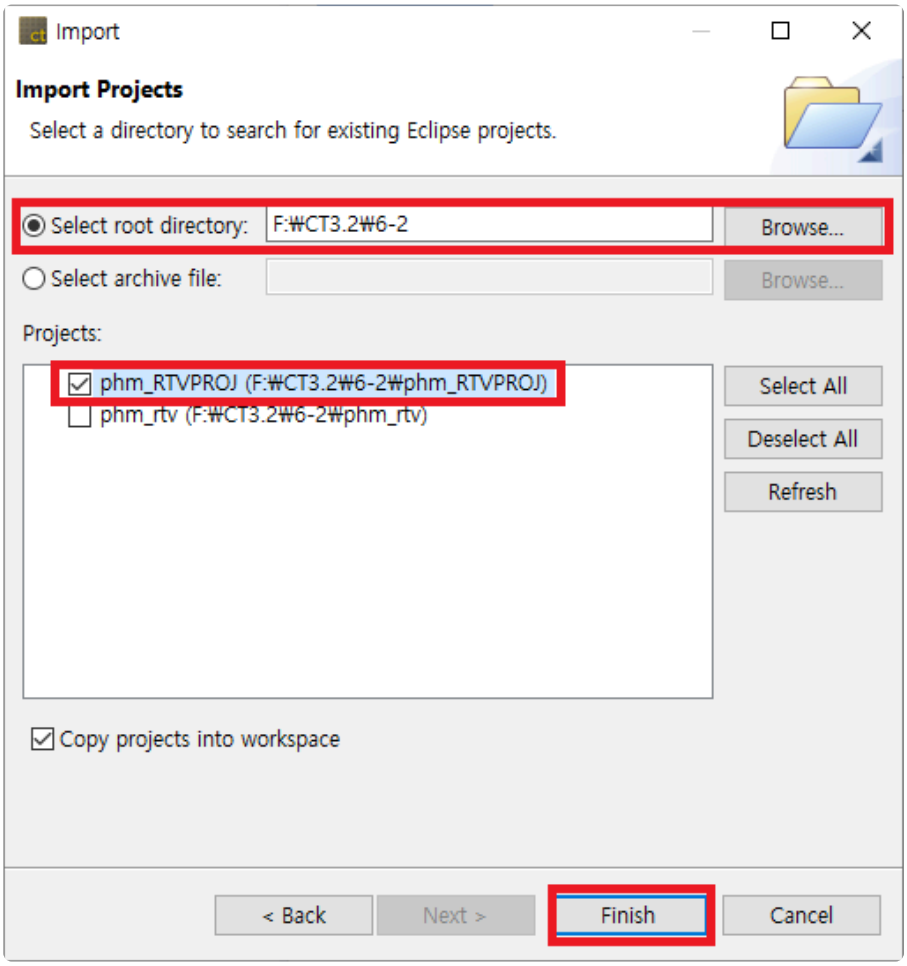
RTV projects can be easily shared because the toolchain and source file information can be fetched from the RTV server.
The step-by-step scenario according to the usage environment is as follows, and the RTV project can be shared when the scenario is followed.

- Project sharing scenario
- RTV server user guide

4.2.1. Project sharing scenario

When using the [Existing Projects into Workspace] function

1. When you create a RTV project, a RTV project directory (hereinafter referred to as RTV_A project) is created under the Controller Tester workspace.
2. The user who wants to share the project receives the RTV_A project directory created in the above step, and copies and pastes the RTV_A project directory into the Controller Tester workspace path that he uses.
3. Select top-level path to the project directory to import the projects using [Import] > [General] > [Existing Projects into Workspace] function.



4. Information required for the project is received from the RTV server, and [the toolchain or resource setting of the project is incorrect. If you want to reset automatically?], Click ‘Yes’ to complete the RTV setup (RTV server and toolchain registration used when creating the project).
5. You can see that an RTV project (hereinafter RTV_A’ project) with the same name as RTV_A has been created in the Controller Tester test navigator view.
6. Right-click the RTV_A’ project in [Test Navigator View] and perform [Reanalysis].
7. This should be done when connected to the same RTV server.

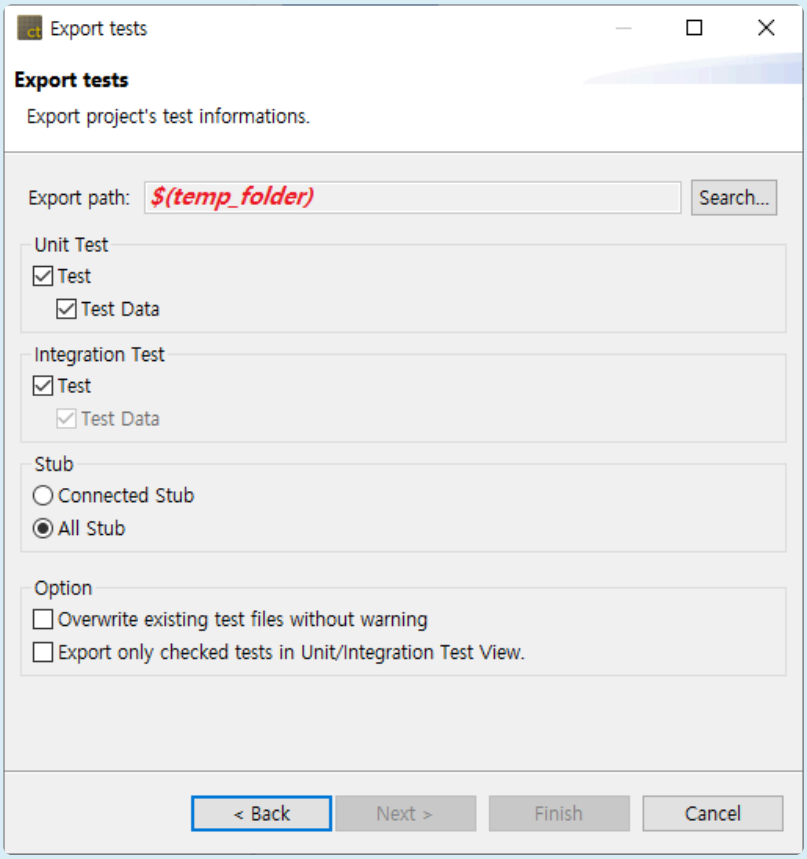
When using the [C/C++ Project from RTV Build] function

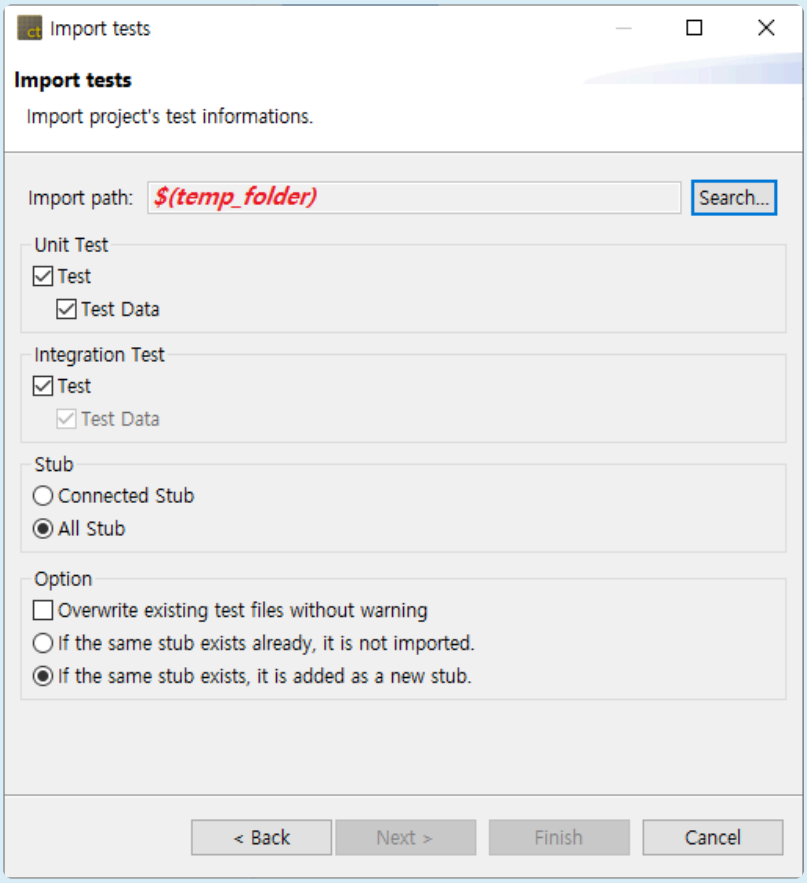
1. When you create an RTV project, an RTV project directory (hereinafter RTV_A project) is created under the Controller Tester workspace.
2. The user who wants to share the project connects to the same RTV server where the above project was created from Controller Tester that he uses, and registers the same RTV toolchain.
3. In the project creation wizard, select [C/C++ Project from RTV Build] to create an RTV project (hereinafter RTV_A' project).
4. Import the *$(project folder)/.csdata/link.mk* file from the RTV_A project folder in the Controller Tester workspace and overwrite the link.mk file in the RTV_A' project folder.
5. If you want to share the same test data, check the below.

If the path where the source files are located is long, the entire source file may not be imported properly. If the path where the source files are located is too long, make sure to specify the CT's global path just below the drive. (ex. *C:\temp*)
To modify the CT global path, open the CodeScroll.ini file in the location where the CT package is installed and replace the default under the -g option with the new global path to set.

```
-startup
plugins/org.eclipse.equinox.launcher_1.4.0.v20161219-1356.jar
--launcher.library
plugins/org.eclipse.equinox.launcher.win32.win32.x86_64_1.1.500.v20170531-1133
-data
@noDefault
-g
C:\temp
-vmargs
```

Even if you share the same project, coverage results may differ if you create each unit test. When you share a test, you must export the test using the [Export] > [Export Test] feature, and then import the test you exported using the [Import] > [Import Test] feature.


Export tests
Export tests
Export project's test informations.
Export path: $(temp_folder)
Search...
Unit Test
Test
Test Data
Integration Test
Test
Test Data
Stub
Connected Stub
All Stub
Option
Overwrite existing test files without warning
Export only checked tests in Unit/Integration Test View.
< Back
Next >
Finish
Cancel


Import tests
Import tests
Import project's test informations.
Import path: $(temp_folder)
Search...
Unit Test
Test
Test Data
Integration Test
Test
Test Data
Stub
Connected Stub
All Stub
Option
Overwrite existing test files without warning
If the same stub exists already, it is not imported.
If the same stub exists, it is added as a new stub.
< Back
Next >
Finish
Cancel

4.2.2. RTV server user guide

When using one RTV server

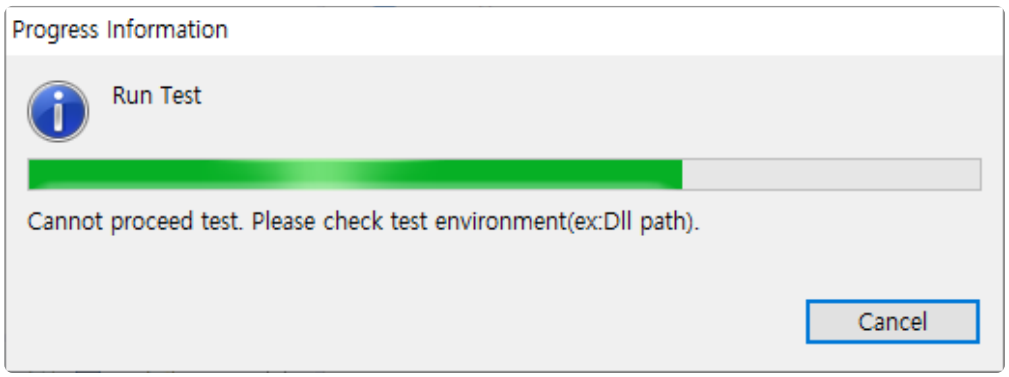
1. When RTV server has a project built using the csbuild capture function
    a. Projects can be imported according to the project sharing scenario above, without the need for additional settings.
2. When the RTV server is connected, but the server (IP/Port) information is different
    a. Since server (IP/Port) information at the time of project creation is imported, existing server information is imported and toolchain information is not imported.
    b. After modifying the server information to access the existing server, import the toolchain with the same name by importing the toolchain. At this time, the path of the tool chain used in the project should be the same.

> ! Sharing of RTV projects can be difficult if you are using more than one RTV server (same source file, tool chain, or if you want to receive and use a virtual machine file with RTV server installed).

5. Identifying the Cause of a Test Error

Occasional errors occur when performing tests on CodeScroll Controller Tester. At this time, the user can find out the cause of the test error by checking the debug information of Controller Tester.

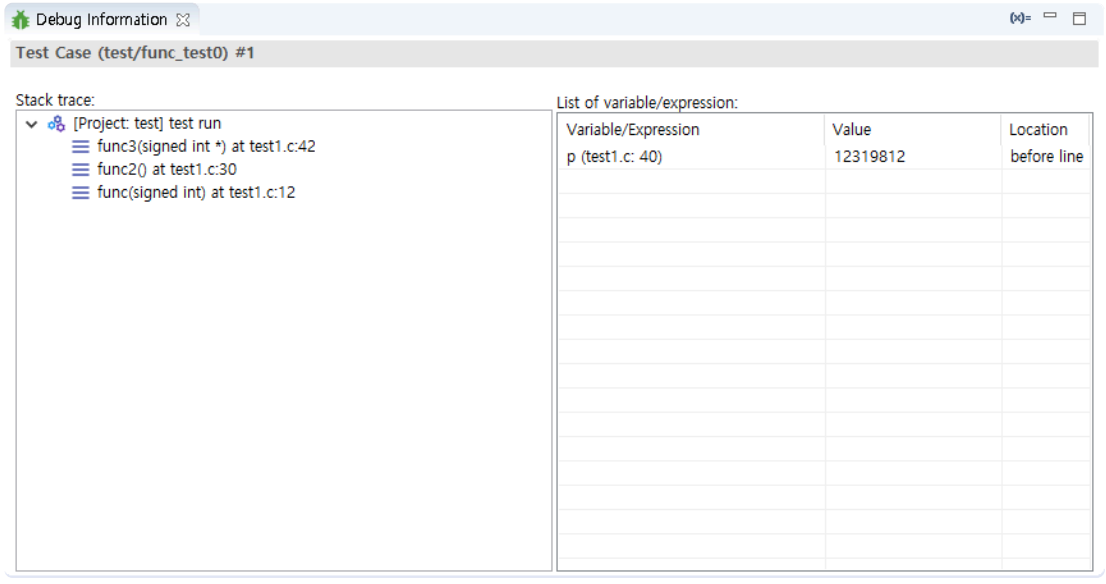
If test execution fails



Debug information verification can be performed even if the test fails. When you execute the [Inspect Debug Info] of the generated test case, the stack trace is displayed in the [Inspect Debug Info], and you can know where the test failed.

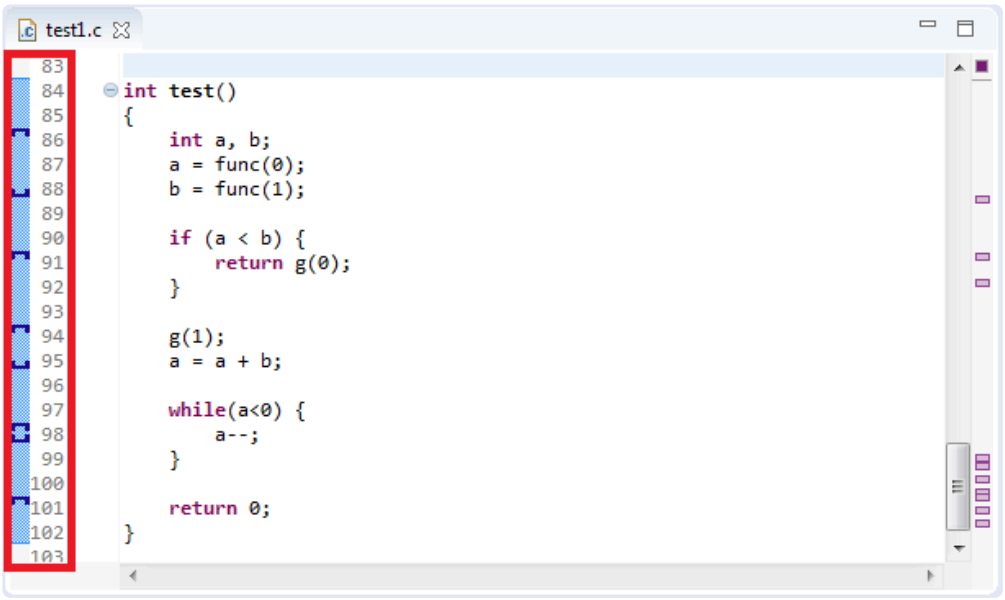
If the result contains errors after running the test

In some cases, after performing a test in Controller Tester, error results such as Signaled and Abnormal Exit are displayed. When you execute the [Inspect Debug Info] of the failed test case, the function call stack trace is displayed in the [Debug Information View]. If you added a variable/expression to debug, a list of executed variables/expressions is also displayed.

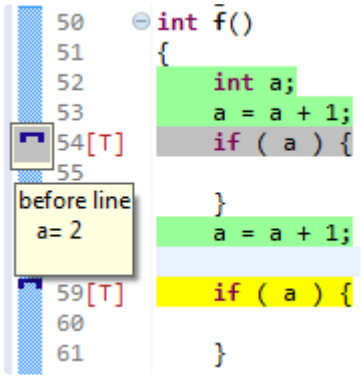


The Stack trace indicates the order of the function calls. The location where the function was called is recorded, and the last execution location is recorded at the top of the Stack trace.
The List of variable/expression represents the variable/expression values executed with the test case.

The list of variables/expressions added to the entire source code can be checked in [List of Variable/ Expressiont] in the toolbar menu of [Debug Information View].



You can also check the variable/expression information to debug in the marker in the source code editor. When you add a variable/expression to debug, the additional position is expressed as a marker in the source code editor, and when you mouse over each marker, you can see the list of variables/ expressions added at that position.
If you select a test case that contains debug information, each marker displays the result of the variable/ expression executed by the test case.



The stack trace and the executed variable/expression value can be used to identify the cause of the error in the test case that executed [Inspect Debug Info].

> For more information on adding variables/expressions to debug, see [Add Variables/ Expressions to Debug] in the CodeScroll Controller Tester document.

6. Source Code Modification and Test Reconfiguration

After designing tests source code can be modified. CodeScroll Controller Tester offers [Test reconfiguration] feature to detect source code modifications and help reconfiguring tests affected by the modification.

> Reflect modified source codes using [Refresh RTV Source File] feature before using [Test reconfiguration] in case of RTV projects and RTV target projects.

Controller Tester divide source code modifications into four cases.

- Modifying names of test or stub functions.
- Modifying names or type of global variables used in tests.
- Modifying names or the numbers of return type or parameter of test functions.
- Modifying the code of the target function to be injected with the fault.

In cases of detectable modification by Controller Tester, refer to In Cases of Detected Modification by Controller Tester and in cases of undetectable modification by Controller Tester, refer to In Cases of Undetected Modification by Controller Tester.

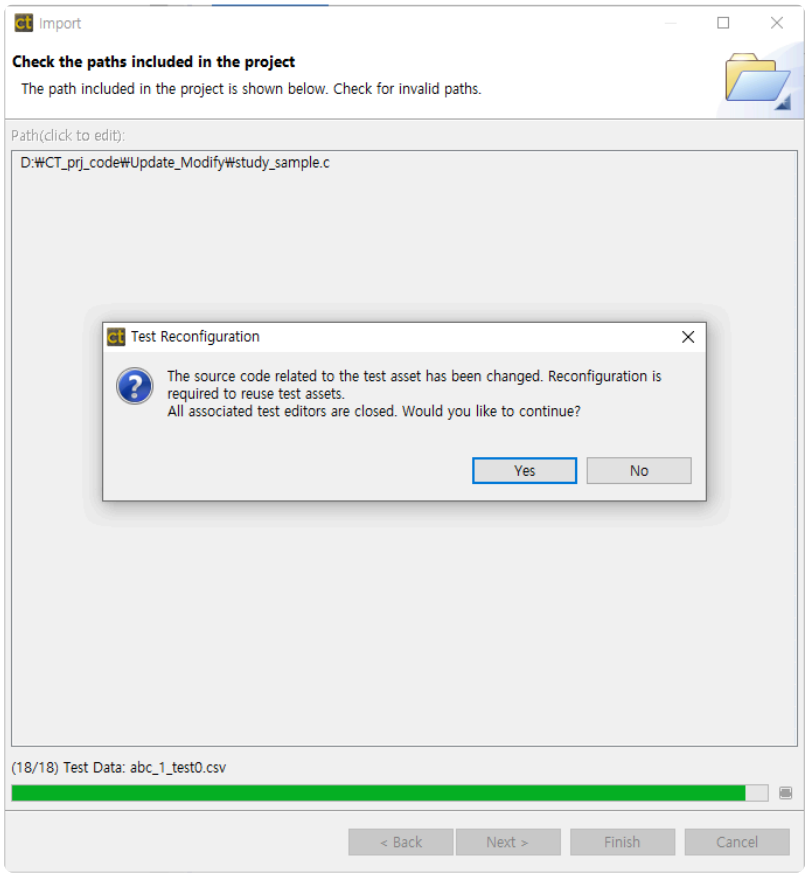
6.1. Run [Test Reconfiguration]

How to automatically use [Test Reconfiguration] feature

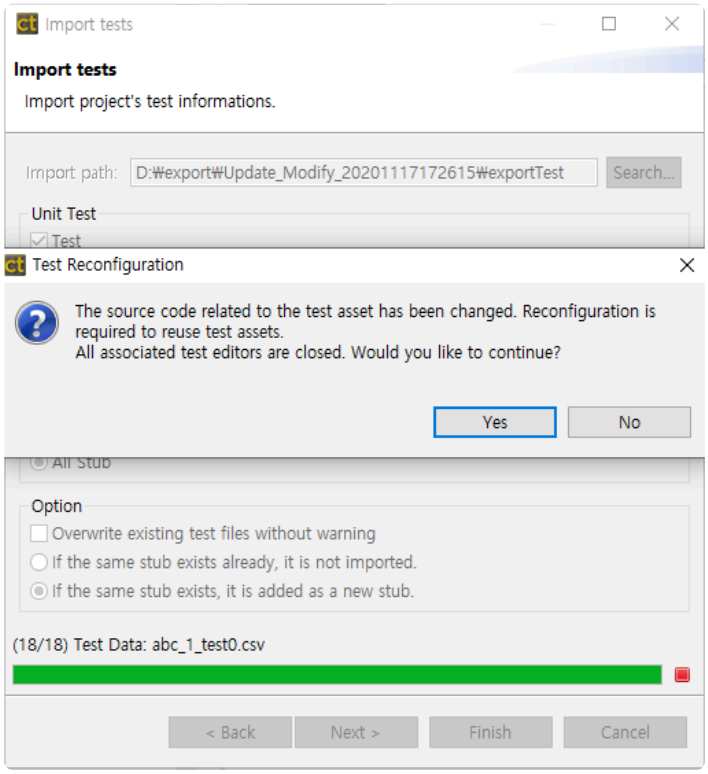
Controller Tester automatically detects following modifications.

- When differ present project information from imported project information using [Import Project] feature.
- When differ present project information from imported test information using [Import test] feature.
- When detect source code modification after analyzing project.
- When analyze the project after writing the fault injection code in a location where fault injection is not possible.

When differ present project information from imported project information using [Import Project] feature



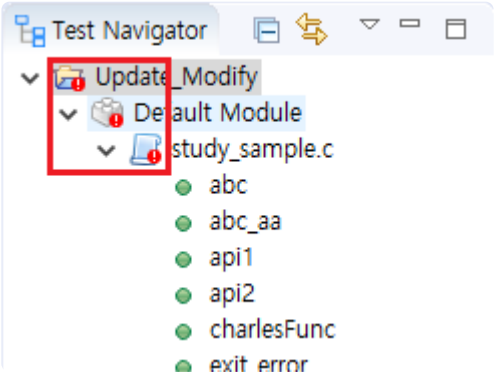
When differ present project information from imported test information using [Import test] feature



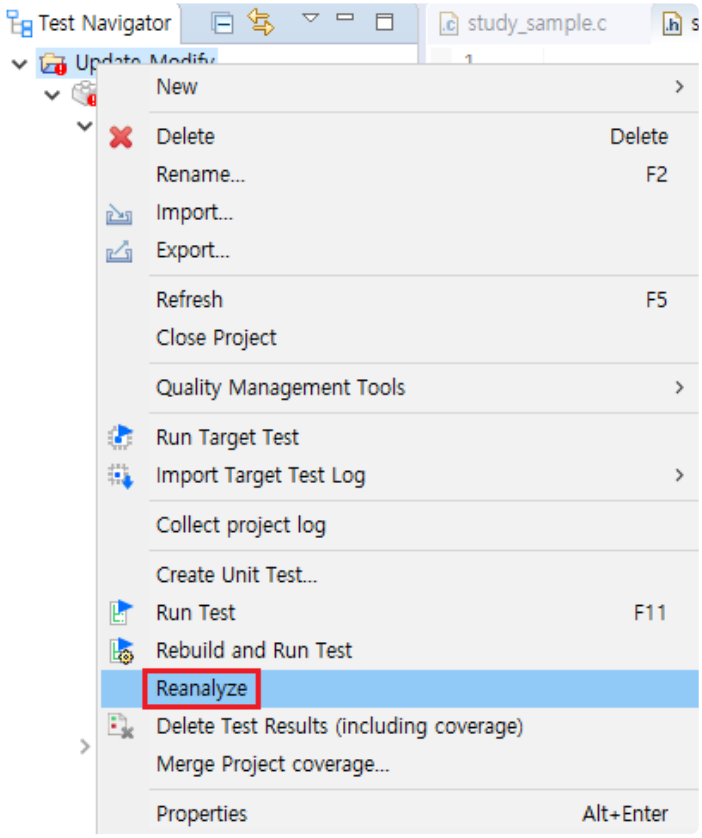
When detect source code modification after analyzing project

You can use [Test Reconfiguration] feature when Controller Tester detects source code modification after project analysis or reanalysis.

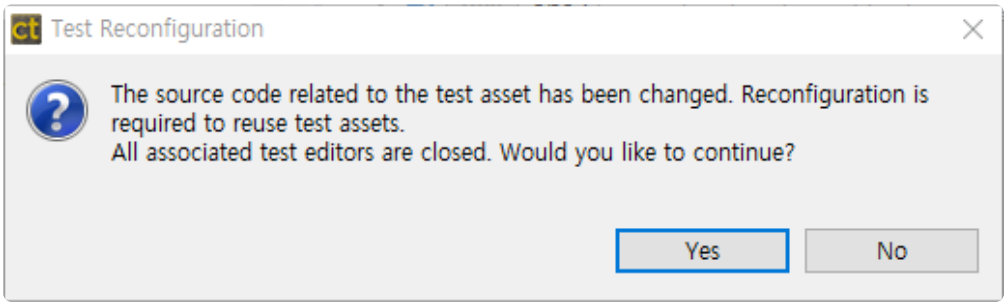
1. When the source codes modify, the Test Navigator View indicates whether the change was made.



2. Select [Reanalyze] in project context menu or run tests to analyze the source codes.



3. Click [Yes] button in [Test Reconfiguration] dialog, then a dialog for recofiguration appears.

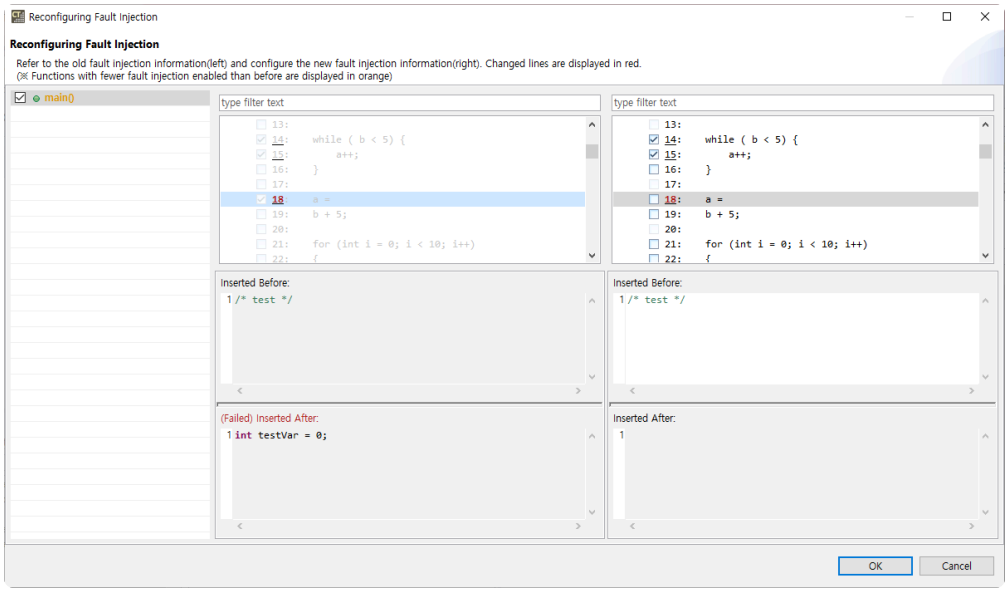


When analyze the project after writing the fault injection code in a location where fault injection is not possible

When reanalyzing the project, if there is fault injection information that satisfies the condition below, the Reconfiguring Fault Injection dialog appears.

- When activate a line in a location where the fault cannot be injected and write fault injection code.

In the Reconfiguring Fault Injection dialog, you can see where faults cannot be injected and fault injection information.

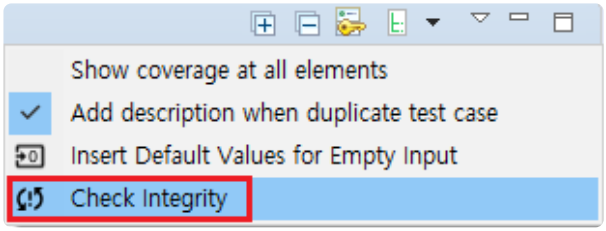


The Reconfiguring Fault Injection dialog allows you to reuse fault injection information previously written.

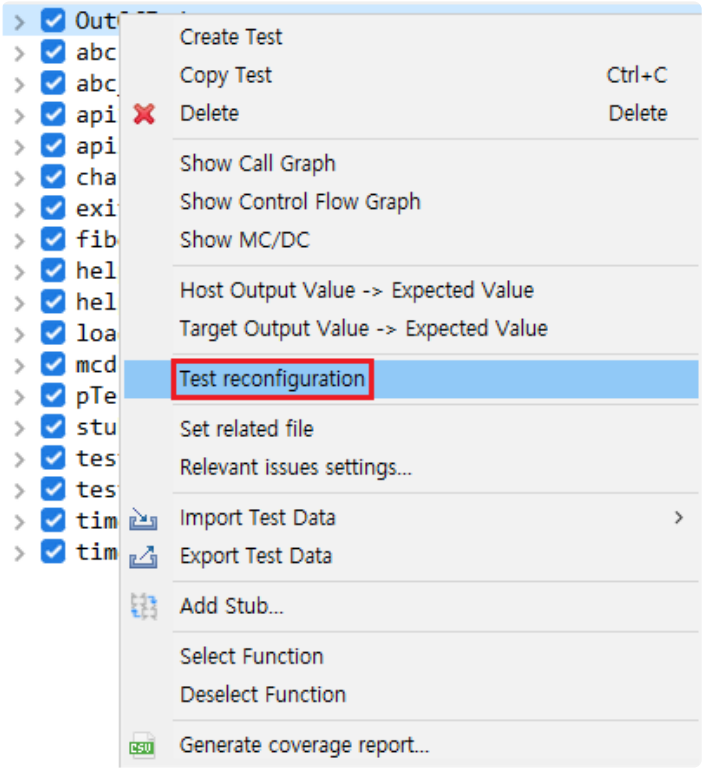
How to manually use [Test Reconfiguration] feature

If you click [No] button in [Test Reconfiguration] dialog or [Cancel] button while reconfiguration, following three method allow to use [Test Reconfiguration] feature.

- Select [Integrity Check] in pull-down menu of the Unit Test View.

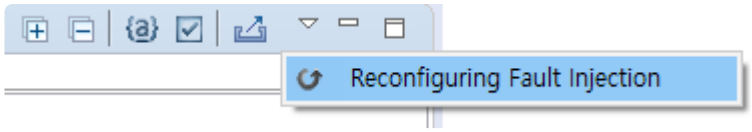


- Select [Test reconfiguration] to use [Test Reconfiguration] feature in function context menu or test context menu of the Unit Test View.

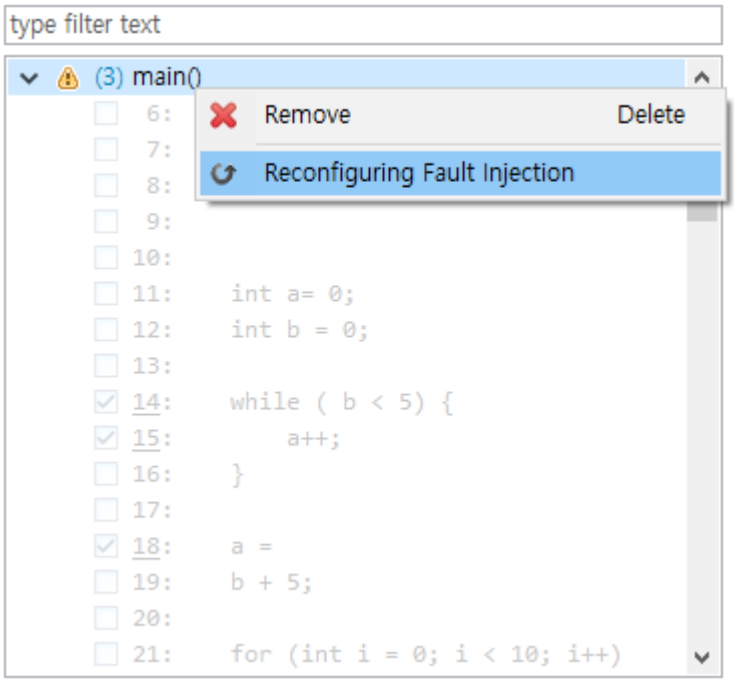


You can design a new test based on original test using [Test reconfiguration] feature.

- You can run Reconfiguring Fault Injection from the context menu or from the pull-down menu in the Fault Injection View.
  - Use the Reconfiguring Fault Injection feature in the menu at the top right of the Fault Injection View.



  - The Fault Injection View marks fault injection functions that need reconfiguring with a reconfiguration-required status marker. Reconfiguring Fault Injection can be executed by double-clicking or right-clicking on the fault injection function that needs to be reconfigured.



> ! The fault injection information cannot be modified where the fault injection function with reconfiguration-required status marker.

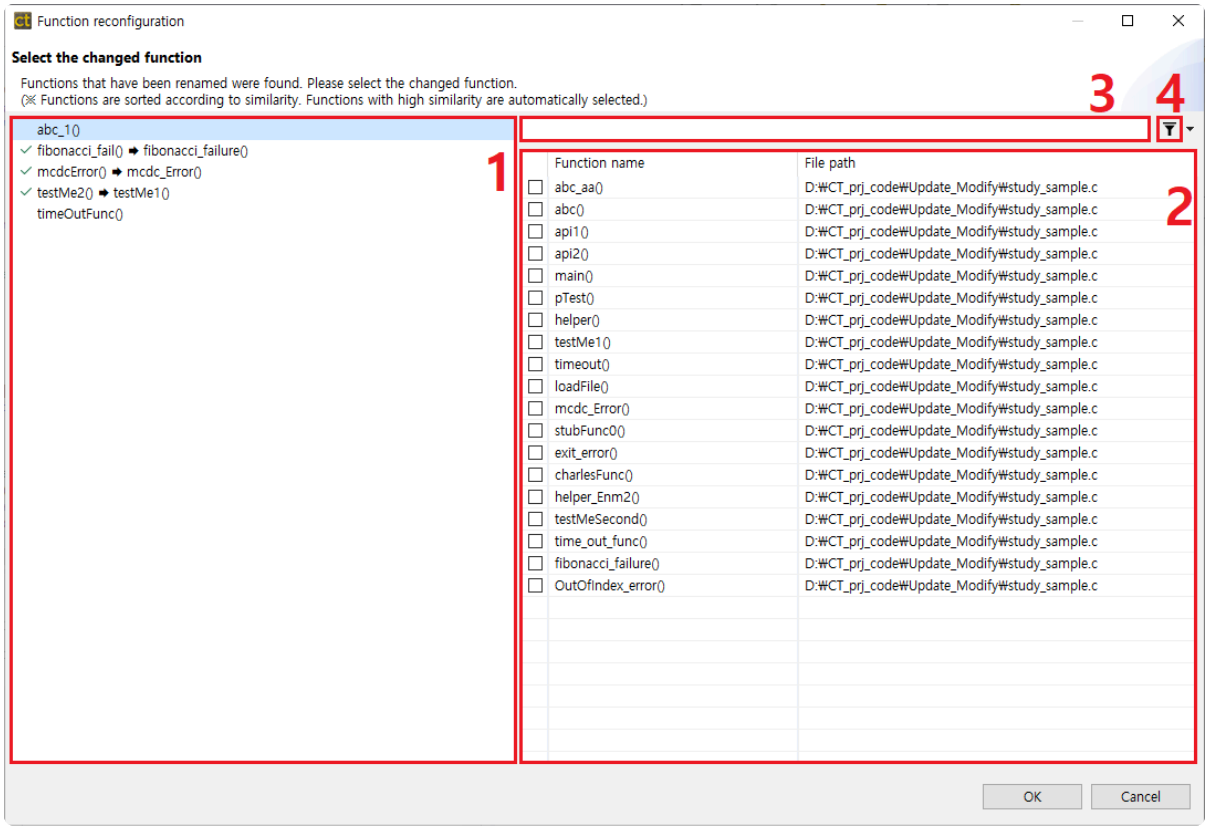
6.2. In Cases of Detected Modification Automatically

When re-analyze or run the tests after modifying source codes, Controller Tester detects modifications with integrity checker. Controller Tester divide source code modification with four cases.

- Modifying names of test or stub functions.
- Modifying names or type of global variables used in tests.
- Modifying name or number of return type or parameter of test functions.
- Modifying the code of the target function to be injected with the fault.

Modifying names of test or stub functions

When modifying names of test or stub functions, the Function Reconfiguration dialog shows up.

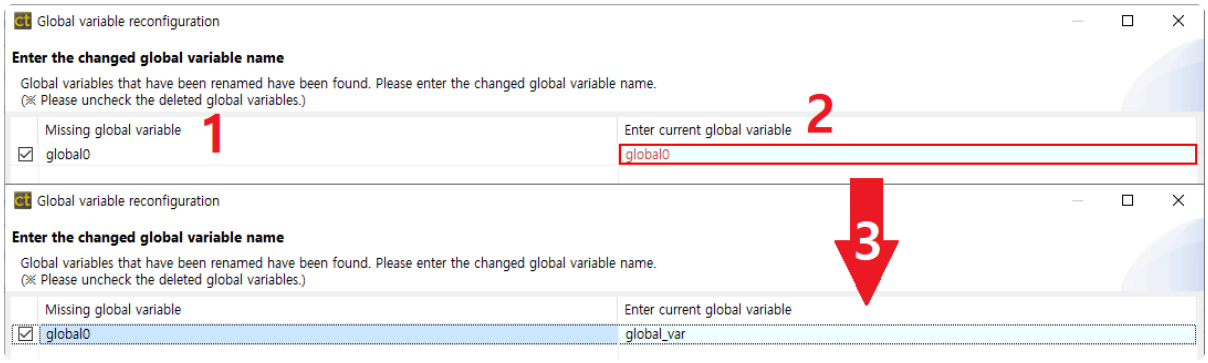


1. Left area is a list of function that modification detected. Functions that finish reconfiguration are marked with ✓.
2. Left area is a list of function contained in present source code.
    - It's sorted by similarity of function name.
    - Function with high similarity is connected automatically.
3. It allow to search a function name. (*: any string, ?: any letter)
4. It show or hide functions with tests.

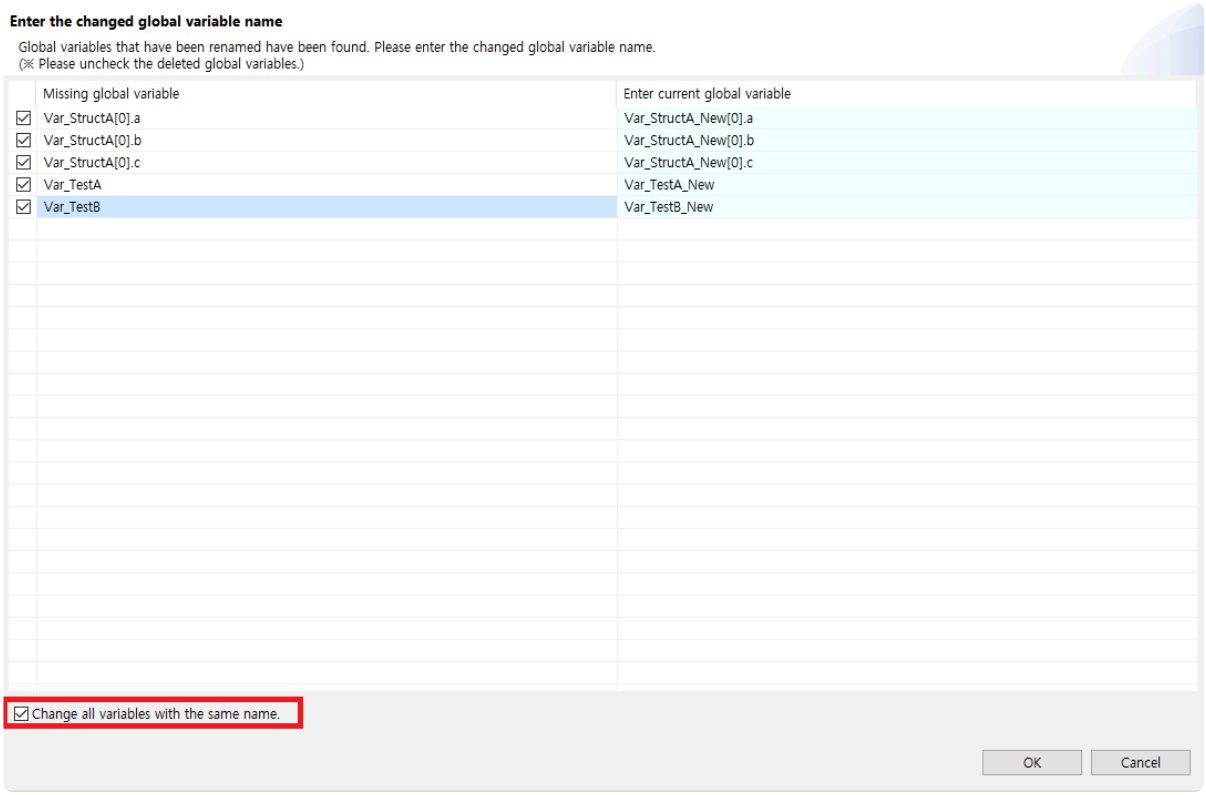
Modifying names or type of global variables used in tests

When modifying names or type of global variables used in tests, the Global Variable Reconfiguration

dialog shows up.



1. Left area is a list of global variables that cannot find.
    - Uncheck check boxes when variables are deleted.
2. Right area contain text boxes for entering present global variable.
    - When user modify a global variable name, it shows global variable list in order of similarity.
3. When user enter a valid variable, red mark in the text box disappear.

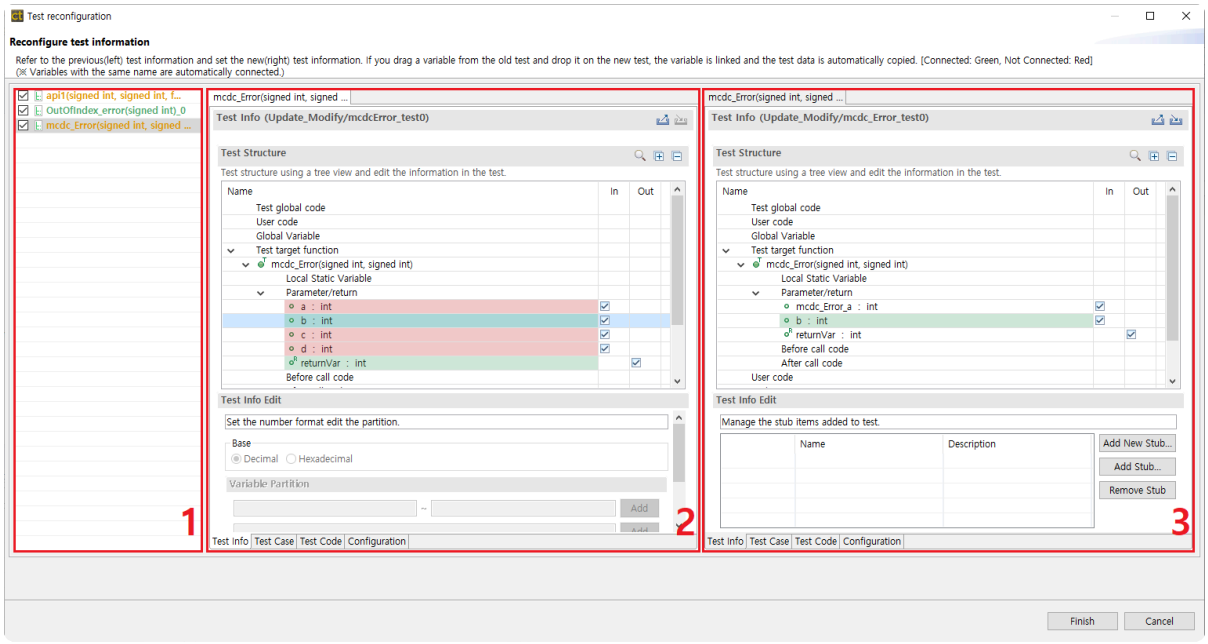


- If many global variables with similar names, such as member variables of a structure, have been changed, you can use the [Change all variables with the same name] feature to modify the global variable names at the same time. If the [Change all variables with the same name] checkbox is checked and the name of a global variable is modified, the name of a global variable with a similar name is modified at the same time. If you uncheck the checkbox, you can edit the names of global variables individually.

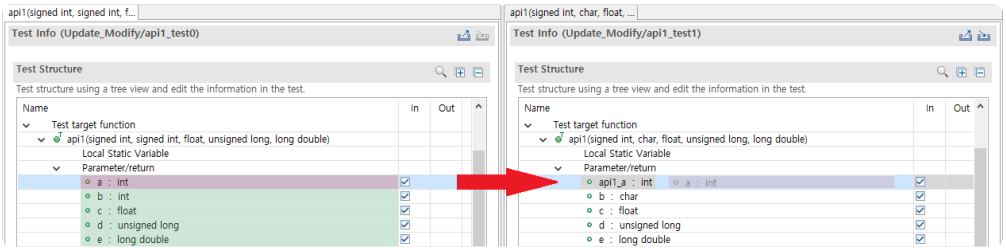
Modifying name or number of return type or parameter of test functions

When modifying name or number of return type or parameter of test functions, the Test Reconfiguration

dialog shows up.

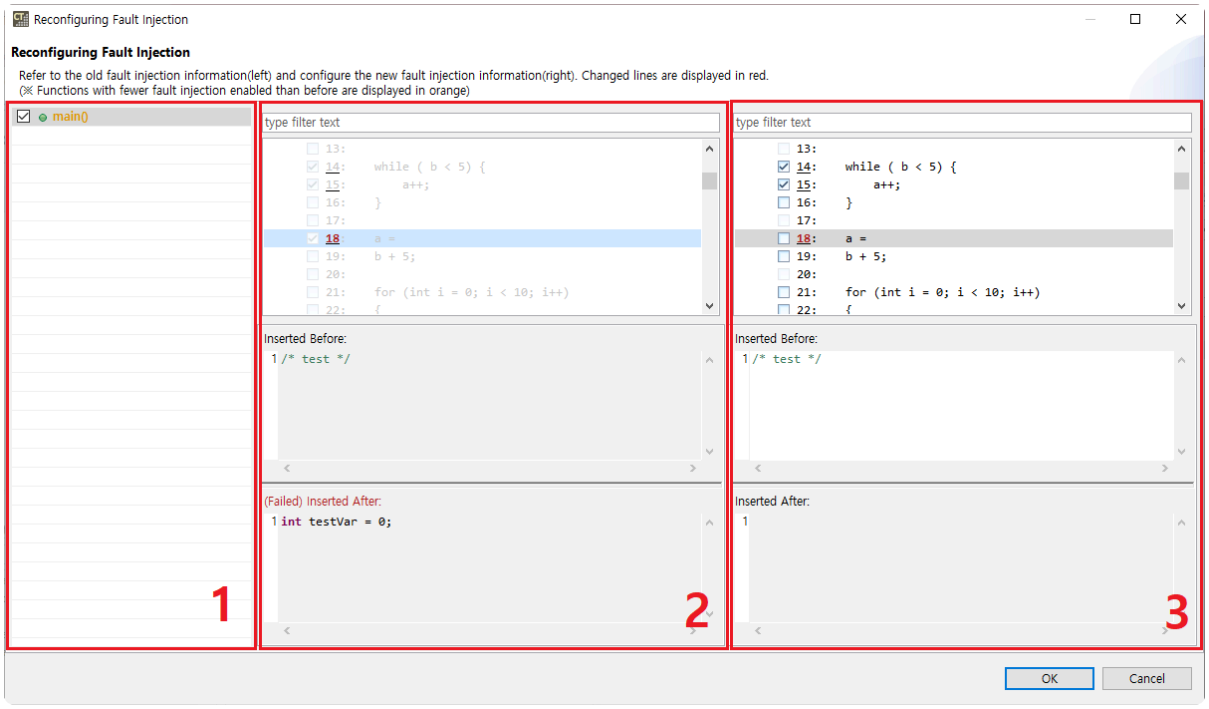


1. A list of modified functions.
2. Test information about function before modifying.
    - If a variable connect to test information after modifying, it's displayed in green and if not, it's displayed in red.
3. Test information about function after modifying.
    - When select a variable of function before modifying, it shows connected variable with selected variable.
    - When drag a variable of function before modifying and drop to a variable of function after modifying, test data are copied.



Modifying the code of the target function to be injected with the fault

If the code of the fault injection function has changed, the Reconfiguring Fault Injection dialog appears.



The list of fault injection functions is displayed in area 1, pre-change fault injection information is displayed in area 2, and after-change fault injection information is displayed in area 3.

- The list of fault injection functions
  - If the checkbox is unchecked, the previous fault injection information is retained without saving changes.
- The Fault Injection Information window
  - Pre-change fault injection information can only be copied. You can copy by shortcut(`Ctrl` + `C`) or right-click.
  - After-change, the fault injection information can be modified. You can copy/paste by shortcut (`Ctrl` + `C` / `V`) or right-click.
  - Changed lines are marked with a line number in red.
  - Double-clicking on a line selects the same line as the one selected in the other Fault Injection Information window.
  - The code written on the selected line can be shown in the Fault Injection Code window at the bottom.
- The Fault Injection code window
  - The code written before and after the selected line is displayed.
  - Locations where fault injection is not allowed are disabled so that you cannot write code.

✻ If there are no tests generated in the project, or if the fault injection line is enabled but no code is written, the Reconfiguring Fault Injection dialog does not appear.

6.3. In Cases of Undetected Modification Automatically

Controller Tester cannot detect following types of modification with integrity check.

- Modify value type of global variable that the type is not defined with `typedef`.
- Test build error (when implicit type conversion is unable)
- Test run error (runtime error including memory overflow, etc)
- Modify symbols excluding global variables.
  - Modify lower type of parameters, symbol added with macro by user, static variable, etc.
- Side effect by modifying function position
  - error that test cannot access to global variable
- Modify build stubs.

When modify value type of global variable, symbols excluding global variables, and function position, user reconfigures test using [Test reconfiguration] feature. When modify build stubs, user delete build stubs because build stubs are not target of integrity check.

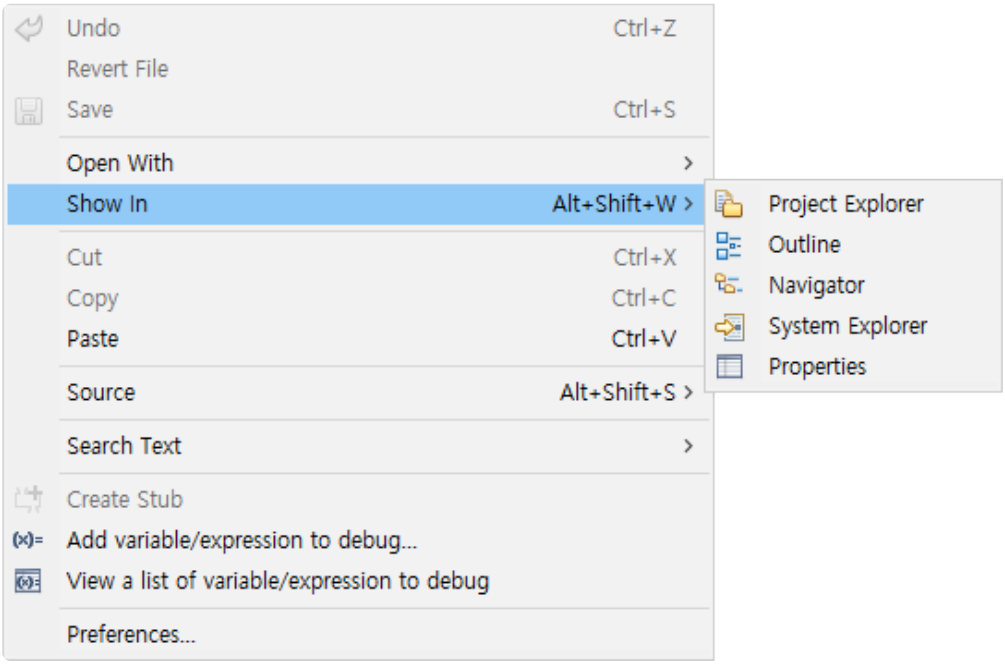
7. Navigate Source Codes

Controller Tester provides shortcuts and context menu in Source Code Editor for user convenience.

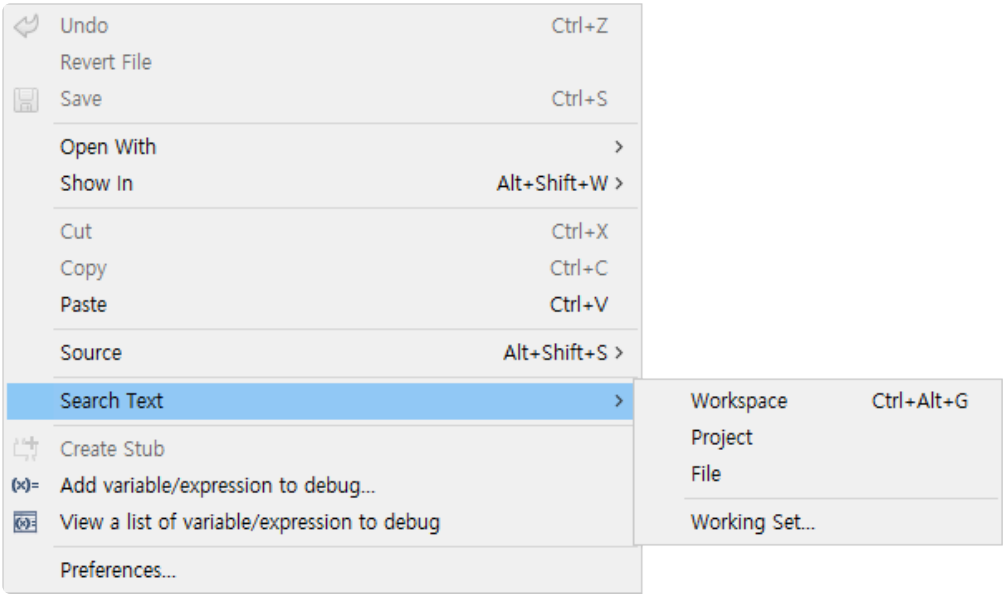
Shortcuts

Item	Shortcut	Description
Open Include Browser	`Ctrl` + `Alt` + `I`	Display the include relationship of the selected file in the [Include Browser View].
Show outline	`Ctrl` + `O`	Show outline of selected file in outline popup.
Toggle Source/ Header	`Ctrl` + `Tab`	Toggle source file and header file.
Open type in Hierarchy	`Ctrl` + `Alt` + `H`	Display hierarchy of the selected item in [Call Hierarchy View]. (Funtion/Global Variable)
Toggle Mark Occurrences	Alt + `Shift` + `O`	Turns the mark occurrence on/off for the item that is positioned by cursor or is specified by block.
Open Declaration	`F3`, `Ctrl` + `Click`	Move to the declaration of the selected item or open the file if it is an include file.
Open Resource	`Ctrl` + `Shift` + `R`	Open a file by searching by name.
References	`Ctrl` + `Shift` + `G`	Display reference to selected item in Search View.
Forward/ Backward history	`Alt` + `Right` / `Left`	Move editor history forward/backward.
Find Next/ Previous	`Ctrl` + `K` / `Ctrl` + `Shift` + `K`	Search the selected text forward/backward in the current file.
Toggle Folding	`Ctrl` + `Numpad_Divide`	Show/Hide folding icon.
Zoom Out/In	`Ctrl` + `-` / `Ctrl` + `Shift` + `=`	Zoom out/in source code editer.
Expand/ Collaspe	`Ctrl` + `Numpad_Add` / `Numpad_Subtract`	Expand/collapse the item on the cursor.
Move Line Down/UP	`Alt` + ↓ / ↑	Move line down/up.
Copy/Duplicate Lines	`Ctrl` + `Alt` + ↓ / ↑	Copy lines down/up.

Context menu



Item	Description
Outline	[Display the outline of the current file in [Outline View].
System Explorer	Open the current file location in Windows Explorer.



Item	Description
Search Text	Search the selected character string in the target (workspace/project/file) and display it in [Search View].

8. Guides for C++ Test Using the Class Factory View

Purpose of using class factories

When testing C++ source code, it is difficult to test because abstract classes cannot create objects. Class factories can facilitate testing of abstract classes and reduce the iterations that occur when designing class objects.

The main features of class factories

- Automatically create concrete classes that inherits from an abstract class
- Minimize repetitive tasks by applying them to tests all together

Utilizing class factories

This document explains the basic concepts for testing C++ before using class factories. After that, it explains how to utilize class factories.

8.1. Basic Concept for C++ Test

It outlines the basic concepts needed before testing C++ using the Class Factory View.

Pure virtual functions and abstract classes

Pure virtual functions

- Virtual function with declaration but no definition .
- Displayed as `= 0`.
- Virtual function implemented in derived class .

Abstract classes

- Classes that have pure virtual functions as members.
- Abstract classes cannot create objects.
  - Declare a variable as a pointer or reference type.
    - ex. `AbstractClass * class1;`.
- Support for polymorphism in object-oriented programming.
- Classes that inherit from an abstract class must override pure virtual functions.
  - If a derived class that inherits from an abstract class does not override a pure virtual function, the derived class is also an abstract class.

```
class Abstract {
        virtual void f() = 0; // pure virtual
}; // "Abstract" is abstract

class Concrete : Abstract {
        void f() override {} // non-pure virtual
        virtual void g();     // non-pure virtual
}; // "Concrete" is non-abstract

class Abstract2 : Concrete {
        void g() override = 0; // pure virtual overrider
}; // "Abstract2" is abstract

int main()
{
        // Abstract a; // Error: abstract class
        Concrete b; // OK
        Abstract& a = b; // OK to reference abstract base
        a.f(); // virtual dispatch to Concrete::f()
        // Abstract2 a2; // Error: abstract class (final overrider of g() is p
ure)
}
```

8.2. Using the Object Creation Code of Abstract Class for Testing

When analyzing the source code, the object creation code of the concrete class that inherits the abstract class is automatically generated in the class factory so that the object of the abstract class can be created. In the object creation code of the abstract class, a framework for the concrete class is provided so that the user can easily create the concrete class.

When creating a test, if a concrete class that inherits that abstract class exists in the source code, that class is linked with the test, and if the concrete class does not exist, the object creation code in the class factory is linked.

You can apply different types of abstract classes to your tests by adding object creation code.

8.3. Design C++ Tests Using Class Factory

After Controller Tester 3.5, you can use class factories for most classes, not just abstract classes.

Advantages of Controller Tester 3.5 Class Factory

Class factories can be used to reduce simple repetitive tasks.

- Class objects that get external data
  - Database, external input/output, server, and so on.
- In the case of class objects that need to be designed in a complex way in the Test Editor, but the same should be used for multiple tests.

How to create and apply an object using a class factory

1. Right-click the class in the Class Factory View and use [Create] to create the class object creation code.
2. Modify the class object creation code according to the test design.
3. Apply the class object creation code to the tests.
   - Apply all together
   - Apply individually

8.4. Using Mock Objects in C++ Test

Purpose of using mock objects

When testing C++ source code, it is sometimes difficult to test because it costs much to create the actual object or the test depends on the object a lot. In such cases, using a mock object that mimics the real object can effectively reduce dependencies on the object. Additionally, you can generate specifications, such as the expected number of calls of the mock to verify that the object is being used as intended.

Available toolchians

- GCC 6.0 or later
- Visual Studio 2015 and later

The main features of a mock object

- Setting return parameters and return values of a mock object
- Setting call count for mock object
- Checking whether the calls occured in a specific order
- Adding constraints to parameters
- etc

Mock object usage

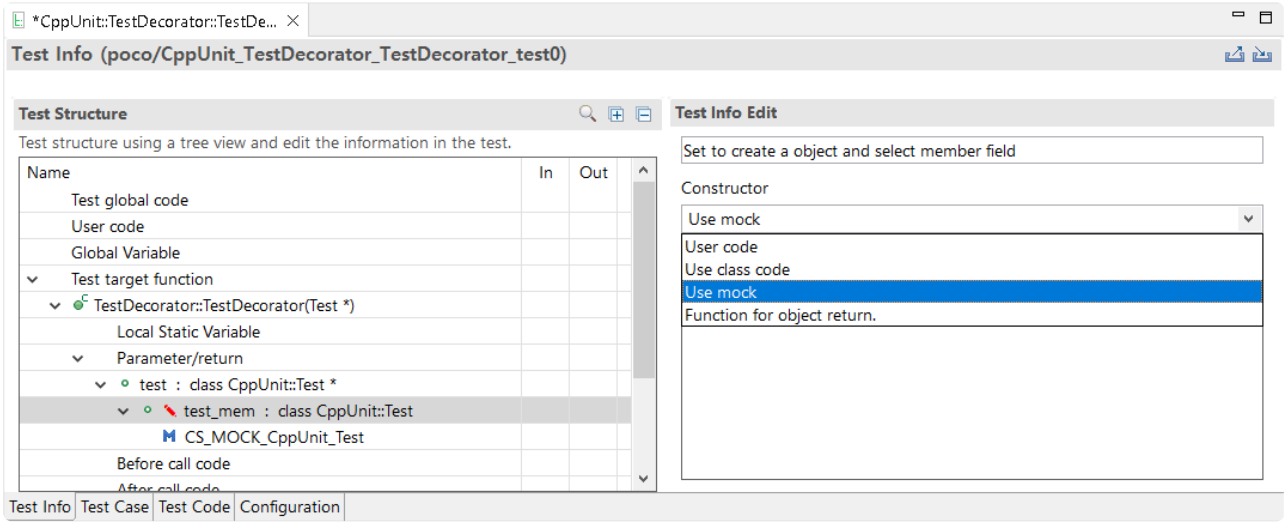
This article explains how to use mock objects in C++ tests.

- Creating mock objects
- Generate specifications about mock objects

8.4.1. Creating mock objects

Creating mock objects

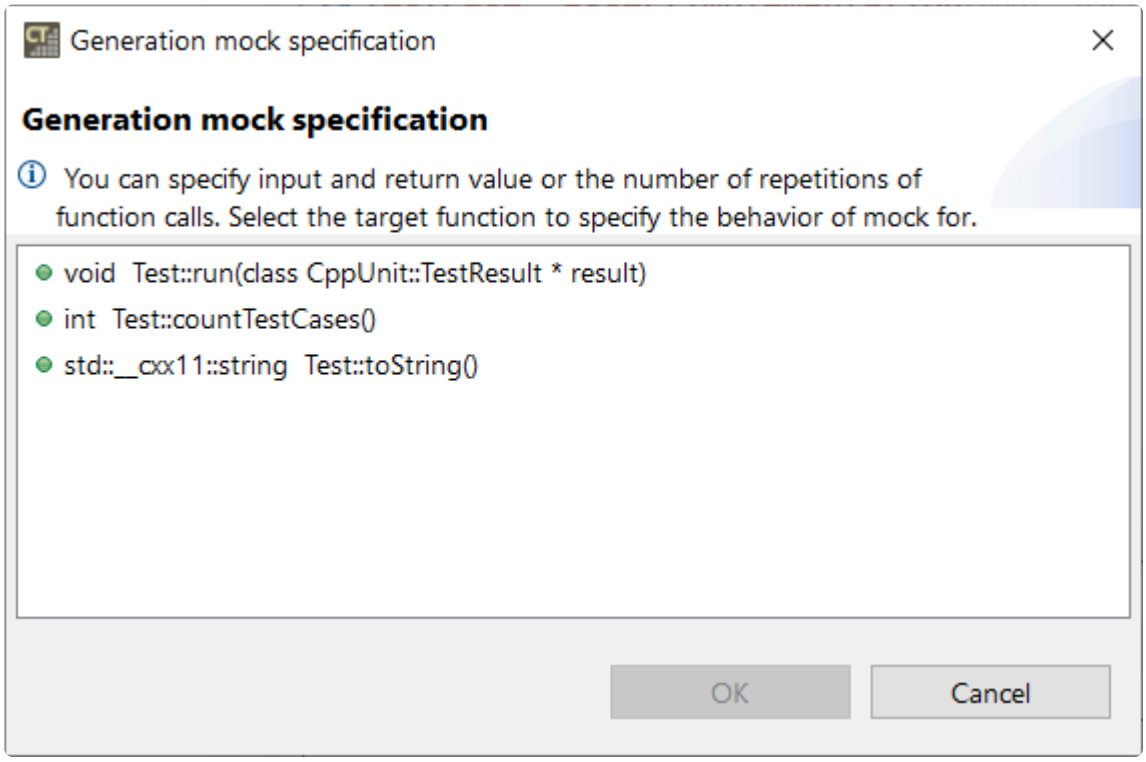
1. Open [Test Editor], by double-clicking the test for which to create a mock object.
2. In the [Test Info tab], expand the test structure tree and select the object to create a mock.
3. Select [Use mock] at the constructor in the test information edit area on the right.



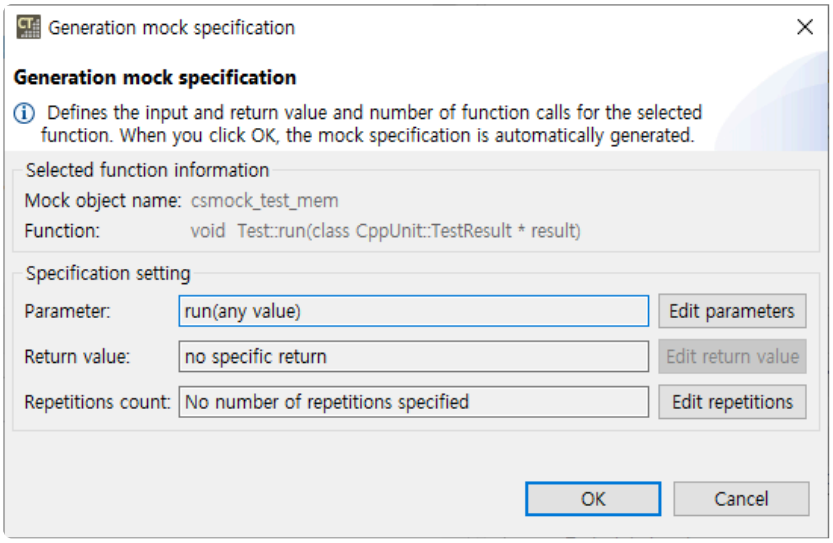
8.4.2. Generate specifications about mock objects

Generate specifications about mock objects automatically.

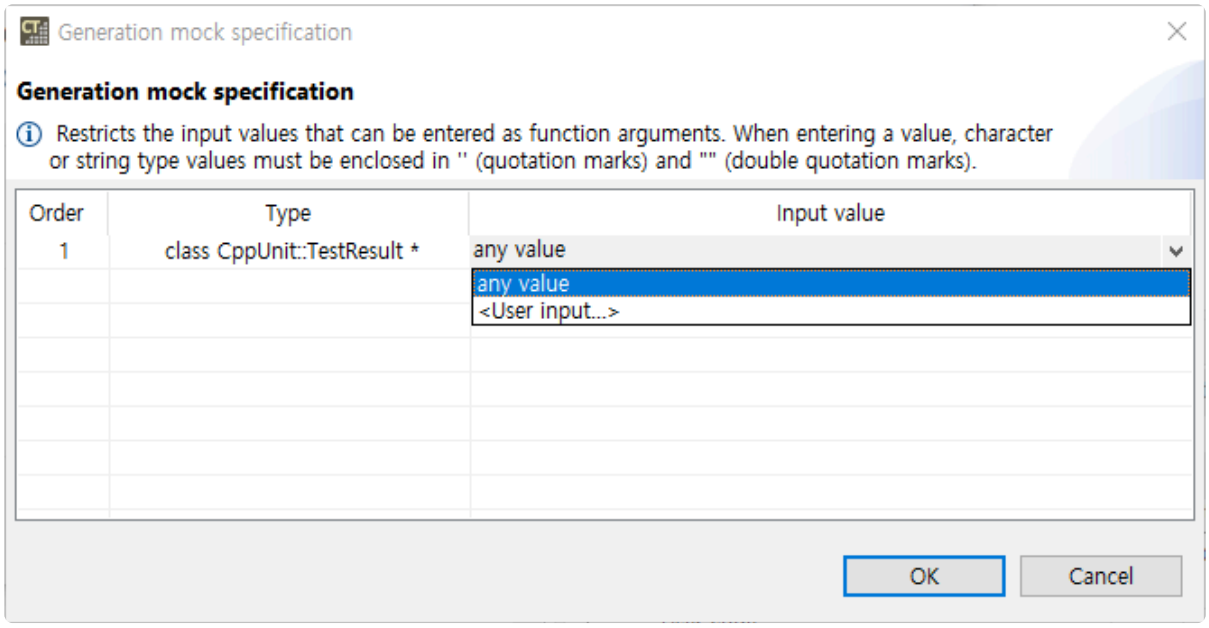
1. In the Test Information tab, click the mock object that you created.
2. In the Test Info Edit area on the right, Click [Generate Sepcification Wizard…] button.
    - If specification about the mock object is empty, [Generation mock specification] wizard automatically appears when you click the mock object.



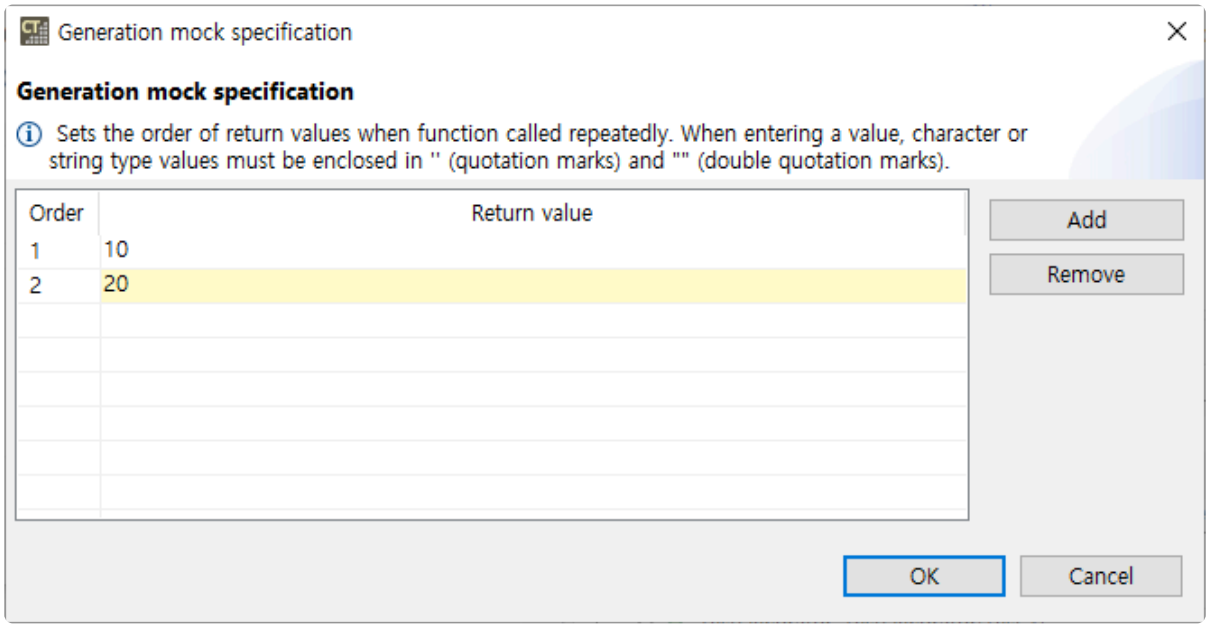
3. In [Generation mock specification] wizard, select the target function to specify and click [OK] button.
4. Edit parameters, return values, and repetitions.



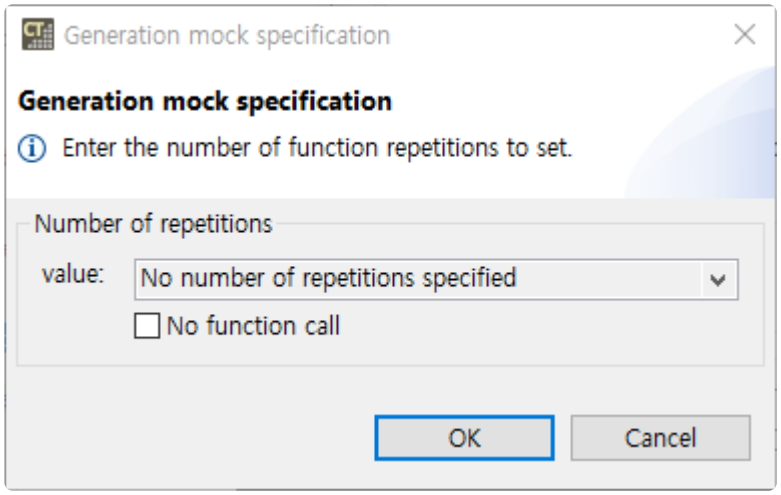
    - Click [Edit parameters] button to create a specification of the parameters used by the function.



  - Selecting [any value] does not restrict the value of that parameter.
  - You can restrict parameter values through [< User input… >]. For example, when you type `1` in the input value and run tests, the test fails if the parameter is not `1`.
- Click [Edit return value] button to determine the return value of the function.



  - Select [Add] button to add the value to return when the function is called.
  - Select [Remove] button to remove the last added return value.
  - If you specify one return value, it will be returned repeatedly.
  - When multiple return values are specify, the function returns them in order when called. In this case, the test fails if the function is not called by the corresponding number of return values.
- Click [Edit repetitions] button to create a specification of the number of calls to that function.



- If you select [No number of repetitions specified], you do not restrict the number of calls.
- Use [< User input… >] to limit the number of function calls. For example, if you set the number of function calls to 3 and run a test, the test fails if the function is not called 3 times.
- [No function call] is the same as specifying a zero number of calls. In this case, the test fails when the function is called.

> ! To set the return value and the repetitions at the same time, you must write it directly in the Test Editor, referring to the specification you created in [Generation mock specification] wizard.

5. Click [OK] button to generate a specification.

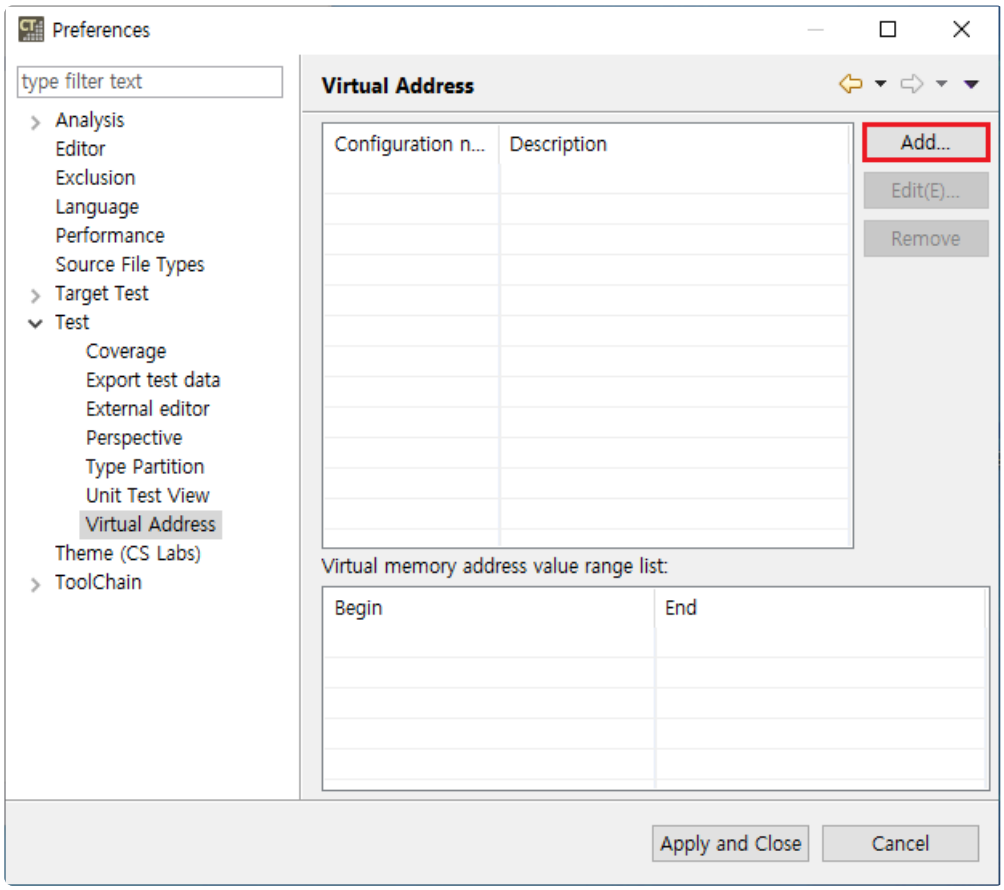
Generate specifications about the mock object yourself

You can modify the specifications created by [Generate Sepcification Wizard…] on Controller Tester or create various specifications yourself. See this document for more information.

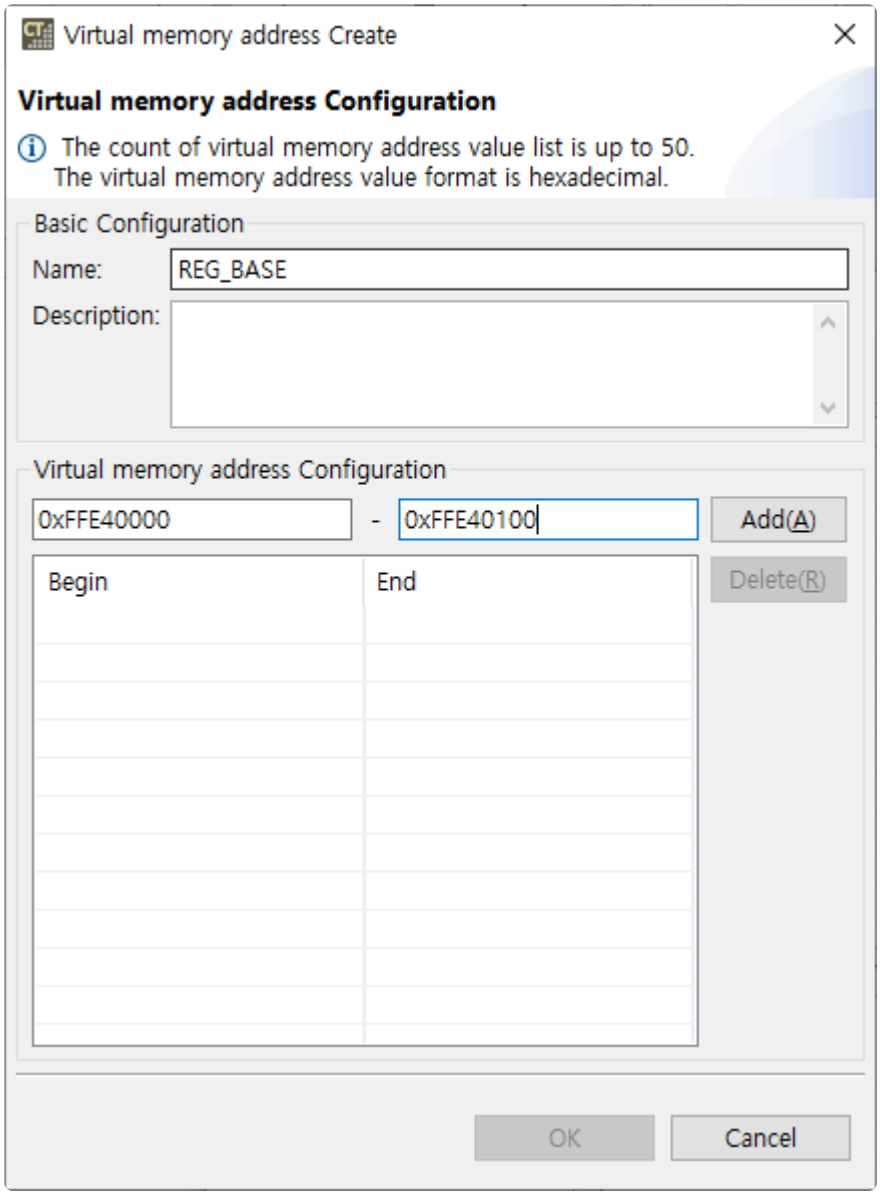
9. Virtual Address Usage Guide

You can set the memory for testing the embedded environment by setting the virtual memory address.

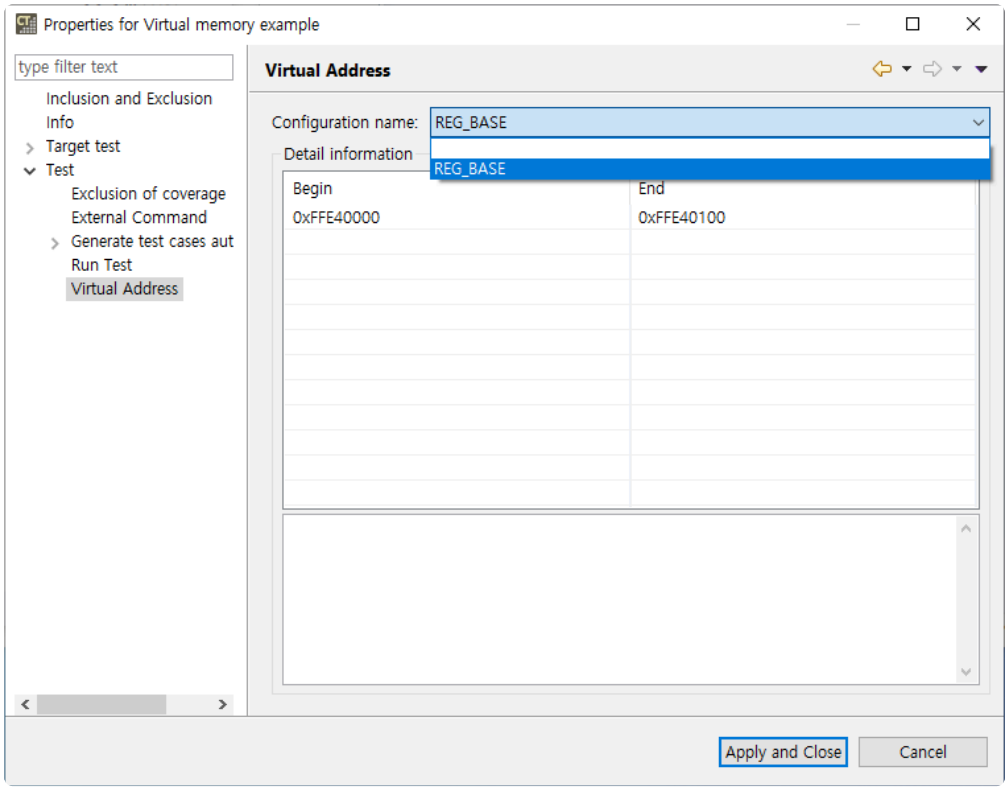
1. Top menu [Window] > [Preferences] > [Unit Test] > [Virtual Address] > [Add…] Selection



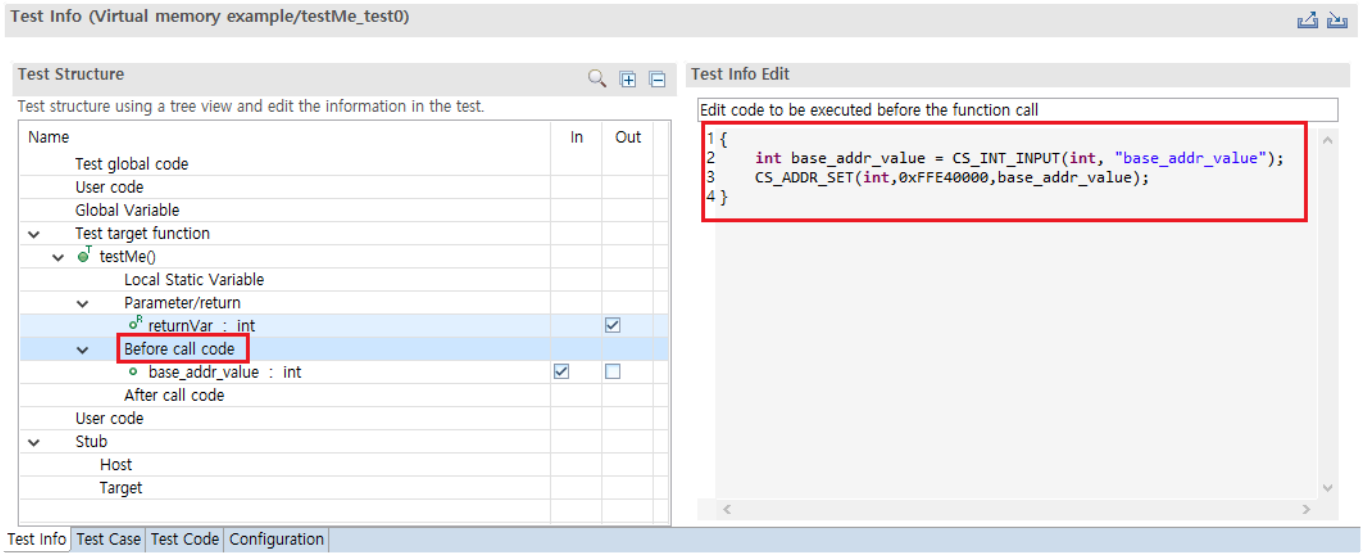
2. After entering the name and range of the virtual address, click the [Add(A)] button



3. Right-click the project [Properties] > [Unit Test] > [Virtual Address] and select the registered virtual address range in the [Configuration Name] combo box

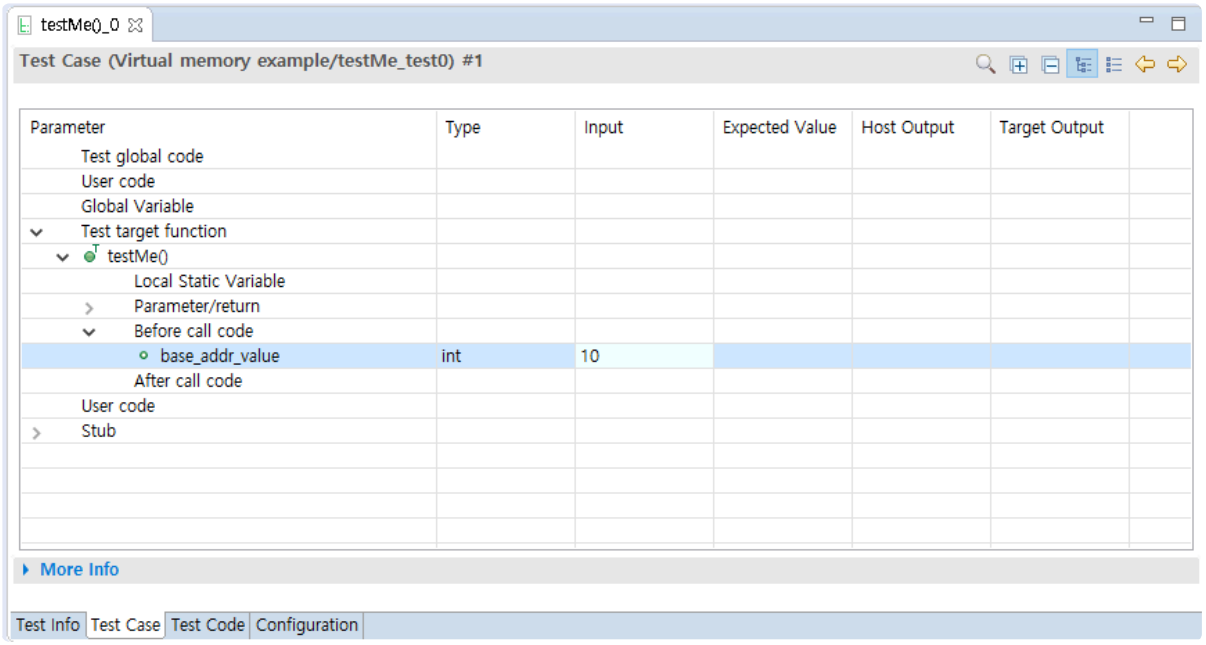


4. Using a macro to set a value to a virtual address in [Before call code] of the test structure editor



For details about macro, please refer to the Test Maco page in User Manual

5. Edit test case values


testMe()_0
Test Case (Virtual memory example/testMe_test0) #1
Parameter
Type
Input
Expected Value
Host Output
Target Output
Test global code
User code
Global Variable
Test target function
testMe()
Local Static Variable
Parameter/return
Before call code
base_addr_value
int
10
After call code
User code
Stub
More Info
Test Info
Test Case
Test Code
Configuration

10. Guides to Import Coverages

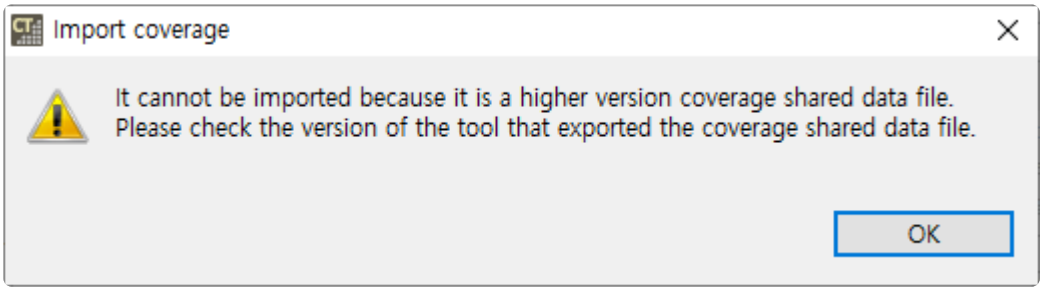
When importing coverages from Controller Tester in another environment or COVER, these are imported by the following three criteria. If the criteria are not met, coverage imports may fail.

- version of coverage shared file
- ternary operation option
- coverage type

10.1. Import Coverages by Version

When Controller Tester import coverages, it checks the version of the coverage shared file.

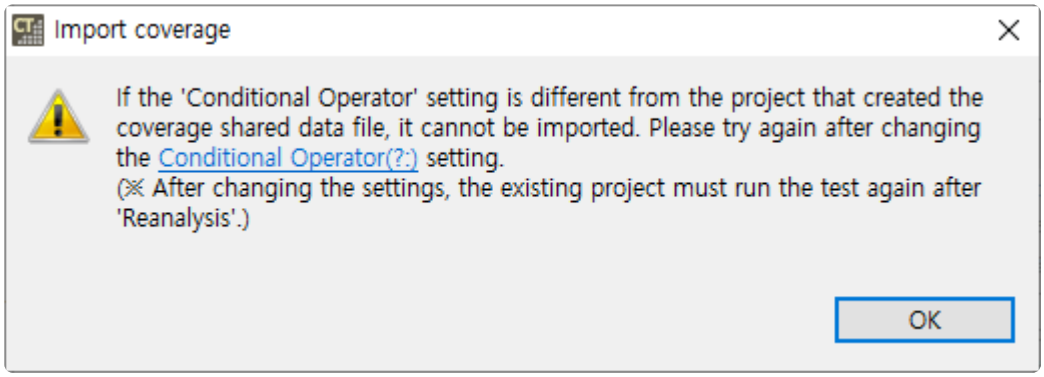
- When importing the higher-versioned coverage shared file, the coverage cannot be imported.



- When importing the lower-versioned coverage shared file, importing the coverages for some functions may fail depending on the option of the tool that exported coverages.

10.2. Import Coverages by Conditional Operation Option

When the conditional operation option of Controller Tester differs from the tool that exported the coverage shared files, the coverages cannot be imported.



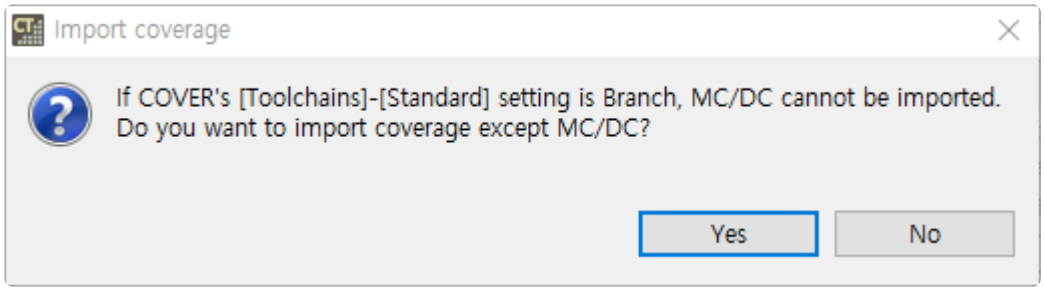
Click the link in the warning window or [Preferences] > [Test] > [Coverage] > Branch coverage, MC/DC measurement operator]. Then, change the [Conditional Operator(?:)] option to match the file you want to import.

- When [Toolchains] > [Standard] in COVER is [COVER] > [Branch], turn off the [Conditional Operator(?:)] option.
- When [Toolchains] > [Standard] in COVER is [COVER] > [MC/DC], turn on the [Conditional Operator(?:)] option.

> ! When changing the option, run the tests again after [Reanalyze].

10.3. Import Coverages by Coverage Type

After Controller Tester 3.6, users can import coverages when coverage types are different.



When selecting [Yes], statement/branch coverages are imported except MC/DC. When selecting [No], coverages are not imported.