# SymbianUnitTest Online Help

Tasks

- Creating SymbianUnitTest suite
- Adding test cases
- Running SymbianUnitTest in emulator

References

- SymbianUnitTest user guide
  - Overview
  - SymbianUnitTest introduction
  - Write a unit test using SymbianUnitTest
  - Classes and Macros
  - ConsoleUI usages
  - SymbianUnitTest advanced features

Legal

## Tasks

This section will tell you how to work with SUT plug-in.

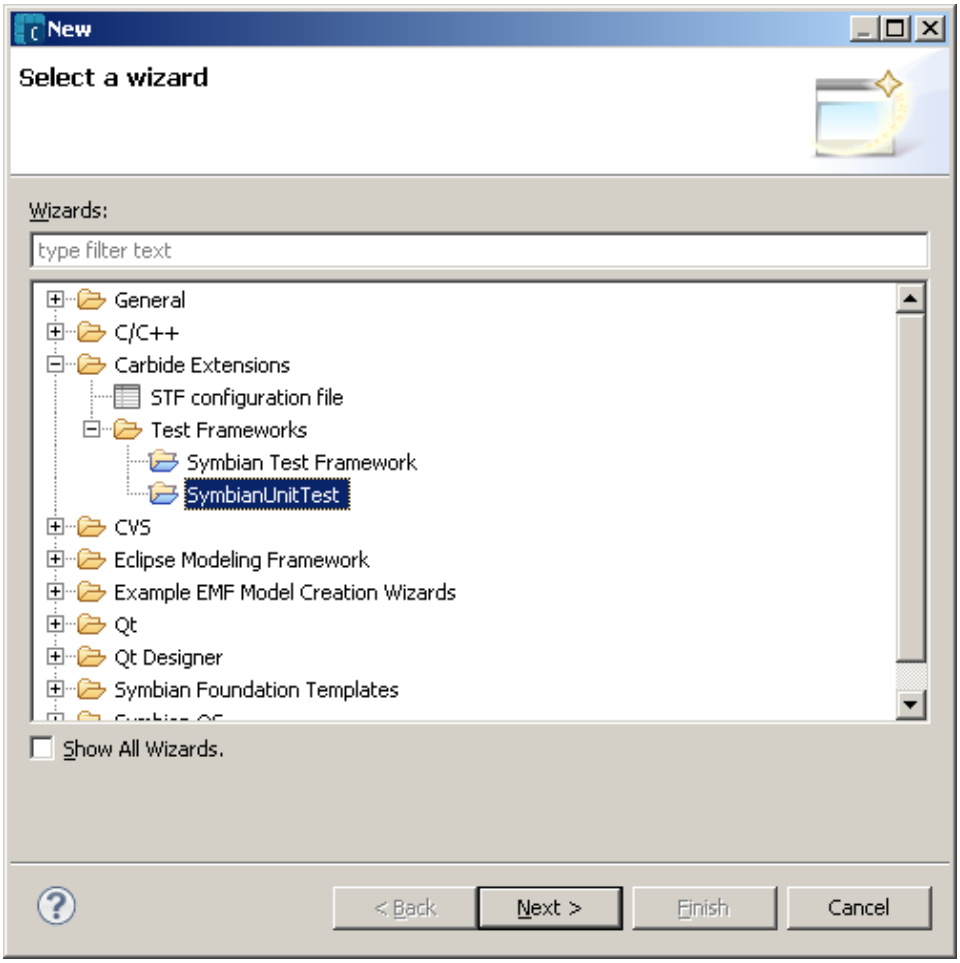
Topics in this section include:

- Creating SymbianUnitTest suite
- Adding test cases
- Running SymbianUnitTest in emulator

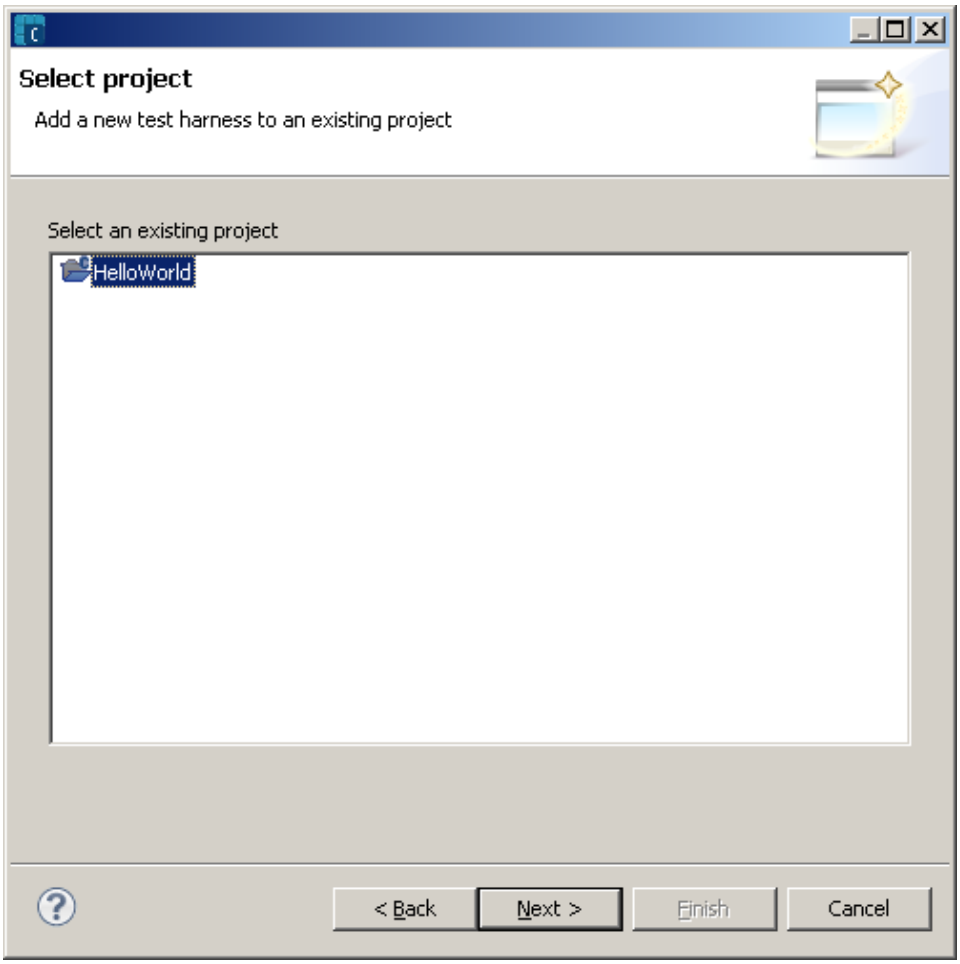
### Creating SymbianUnitTest suite

To create SymbianUnitTest suite, you must have a Symbian C++ project, which is the actual development work, before you build a test project. Once the main development project has been created you can create a Symbian Unit Test suite, the suite files are saved separately in a specified folder, for example the test folder. The steps below show how to create SymbianUnitTest suite:

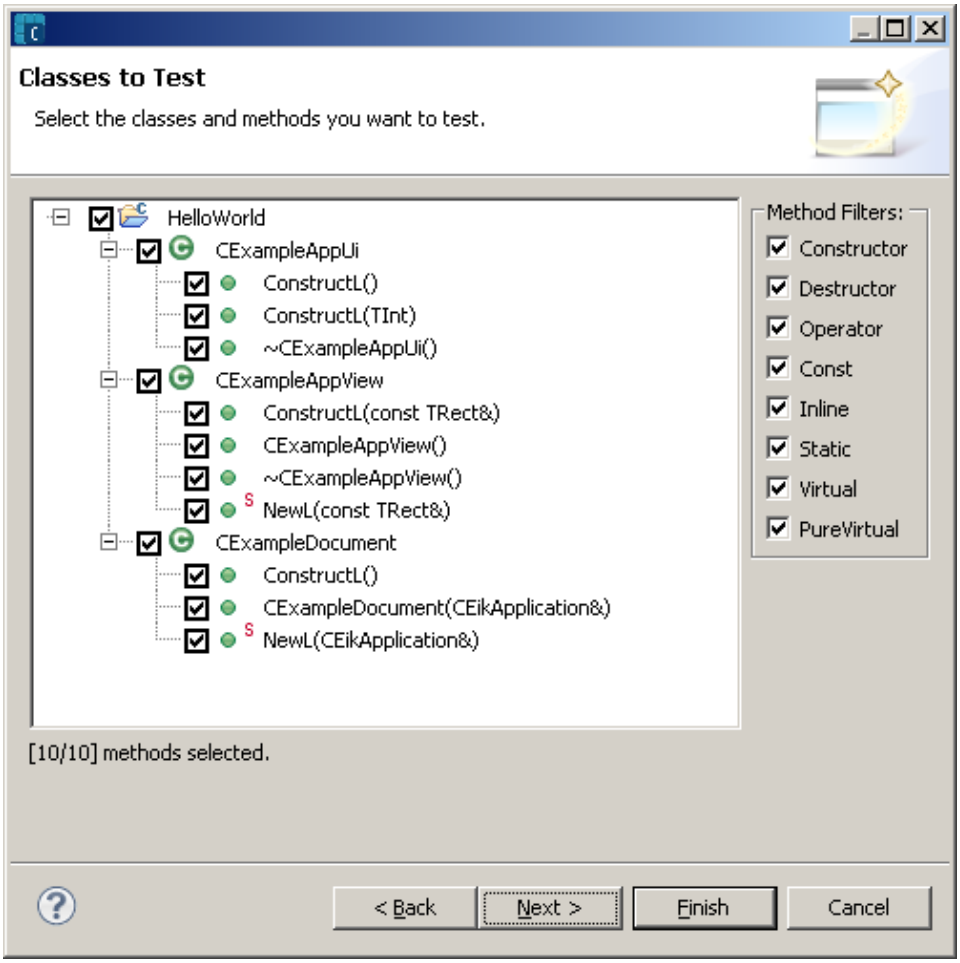
1. Select **File > New > Other**.
2. Select **Carbide Extensions > Test Frameworks > Symbian Unit Test**, and click **Next**.



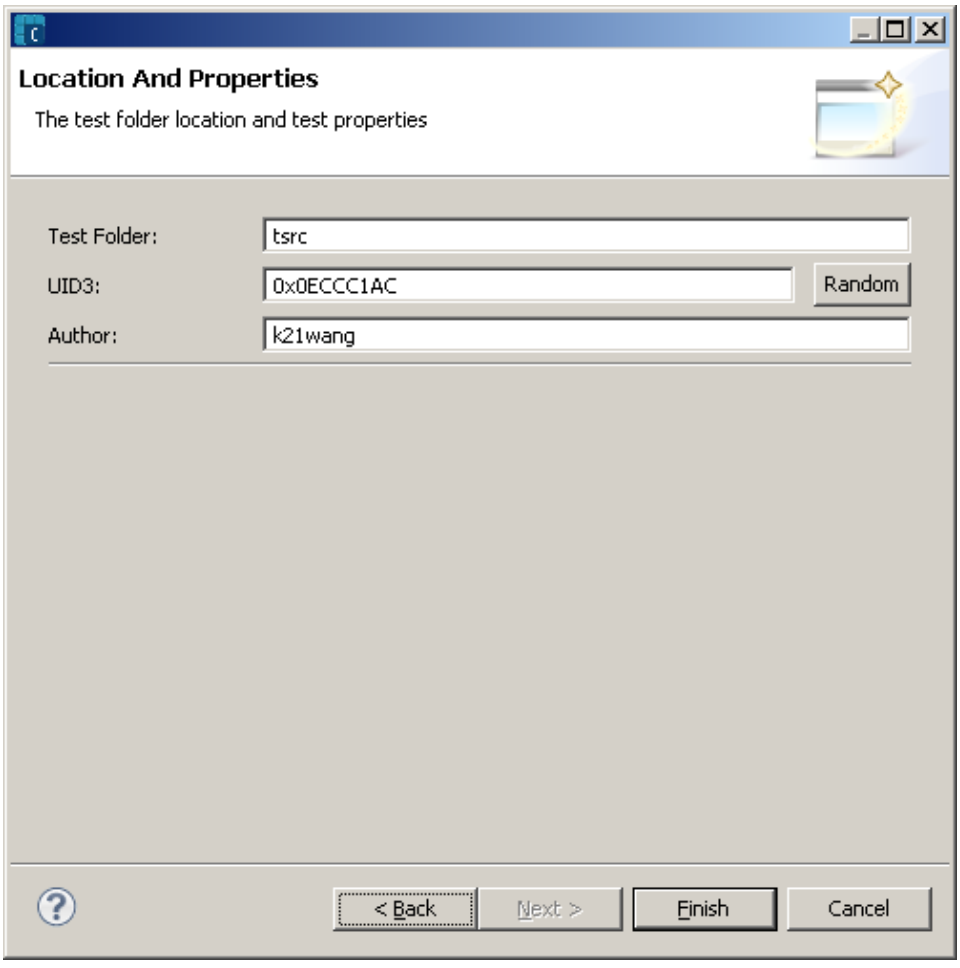

3. Select the project where the Symbian Unit Test suite will be created and click **Next**.



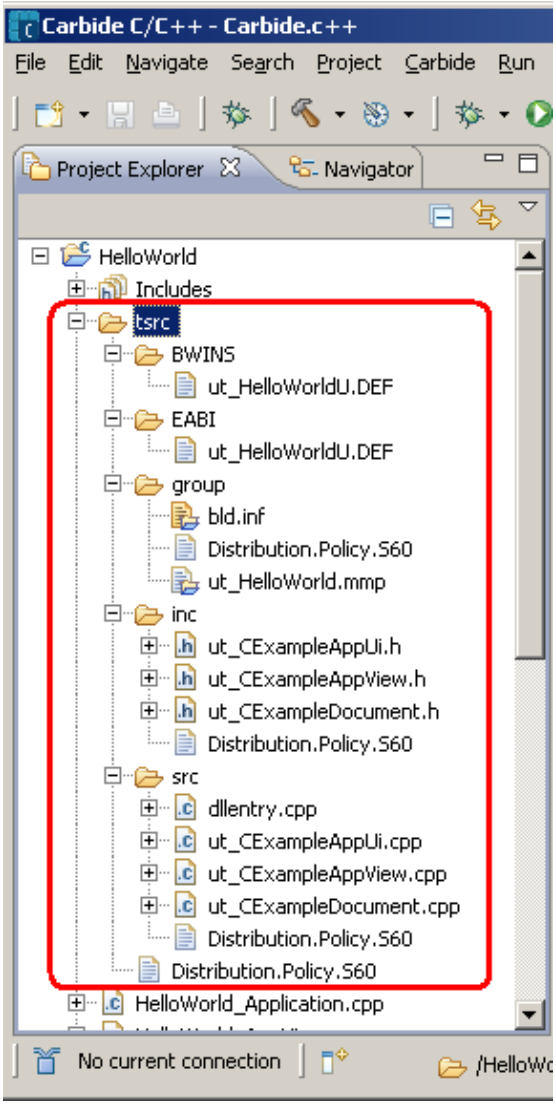

4. Select the classes or functions to test. Click **Next**.




5. If necessary, change the directory for the files generated by the wizard and click **Next**. By default, the generated files are saved under the test folder. The UID3 is generated randomly.




6. A set of files for Symbian Unit Test suite are generated as following:

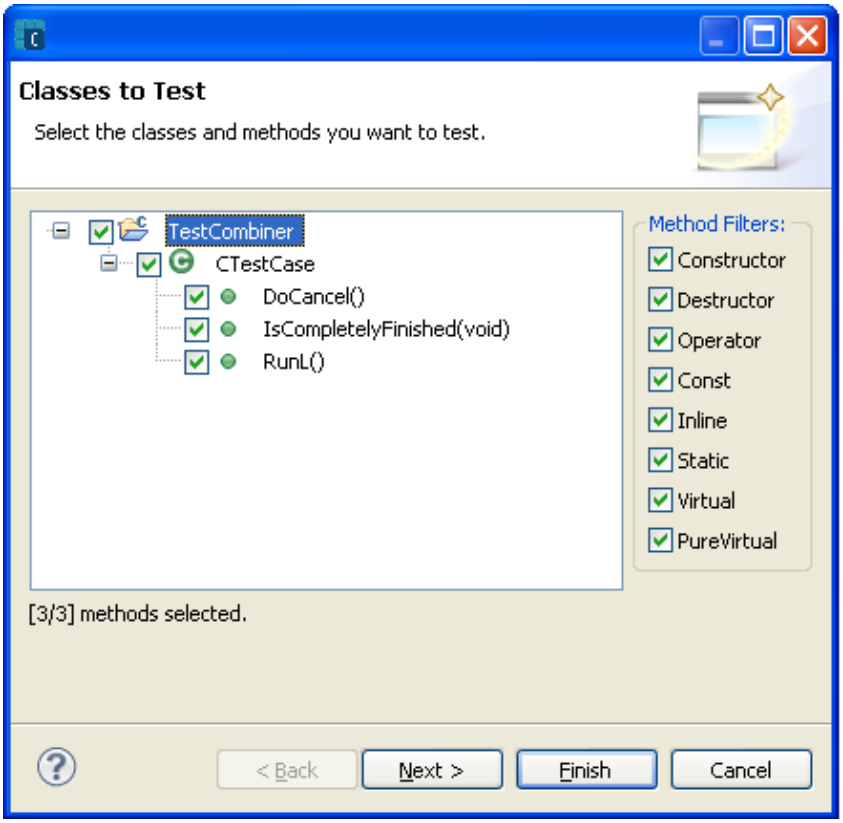



## Adding test cases

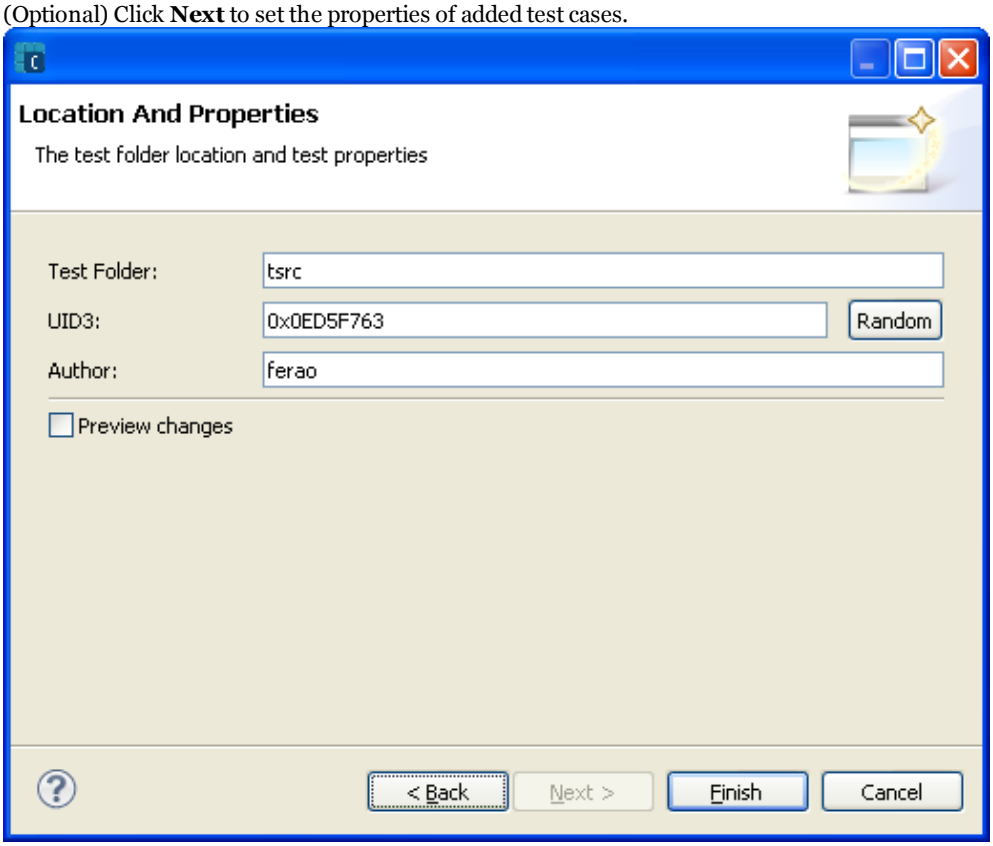
New test cases can be added when new functions are added to a class.

For example, when a new function is added to a class, do the following to add corresponding test case:

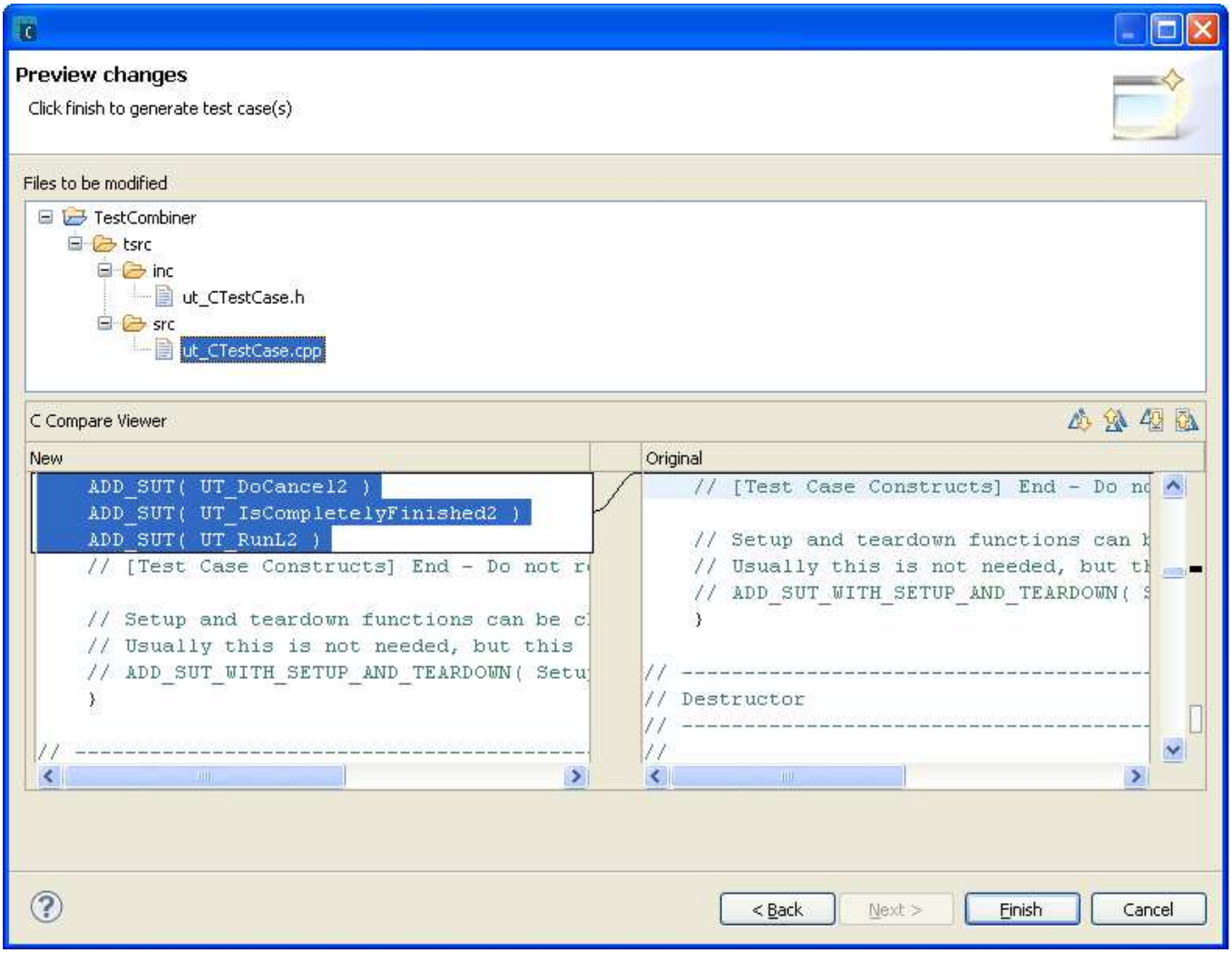
1. Open the header file of the class.
2. In outline view, right click the newly added function and select **Test Frameworks** > **Generate SymbianUnitTest Cases...**. Multiple functions can be selected by holding "CTRL" key when left clicking. A window like below will pop-up:

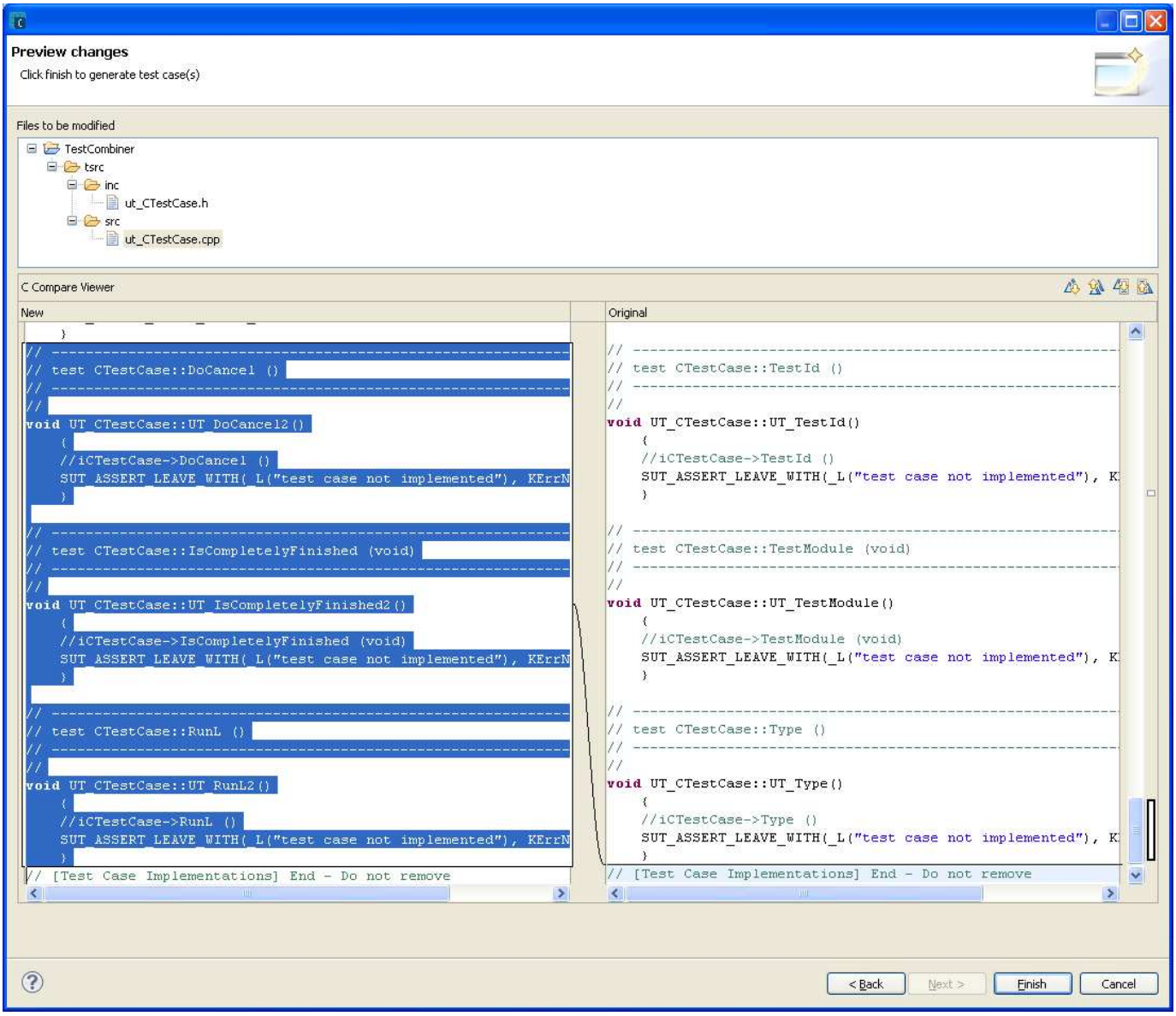



3. (Optional) Click **Next** to set the properties of added test cases.




4. (Optional) Check the "Preview changes" checkbox, the click **Next**. The SUT of ITE plugin will generate dummy test cases for you to modify.


Preview changes
Click finish to generate test case(s)
Files to be modified
TestCombiner
tsrc
inc
ut_CTestCase.h
src
ut_CTestCase.cpp
C Compare Viewer
New
Original
ADD_SUT( UT_DoCancel2 )
ADD_SUT( UT_IsCompletelyFinished2 )
ADD_SUT( UT_RunL2 )
// [Test Case Constructs] End - Do not r
// Setup and teardown functions can be c
// Usually this is not needed, but this
// ADD_SUT_WITH_SETUP_AND_TEARDOWN( Setu
}
// [Test Case Constructs] End - Do nc
// Setup and teardown functions can b
// Usually this is not needed, but th
// ADD_SUT_WITH_SETUP_AND_TEARDOWN( S
}
// Destructor
< Back
Next >
Finish
Cancel




5. Click **Finish** to apply changes.

## Running Symbian unit test

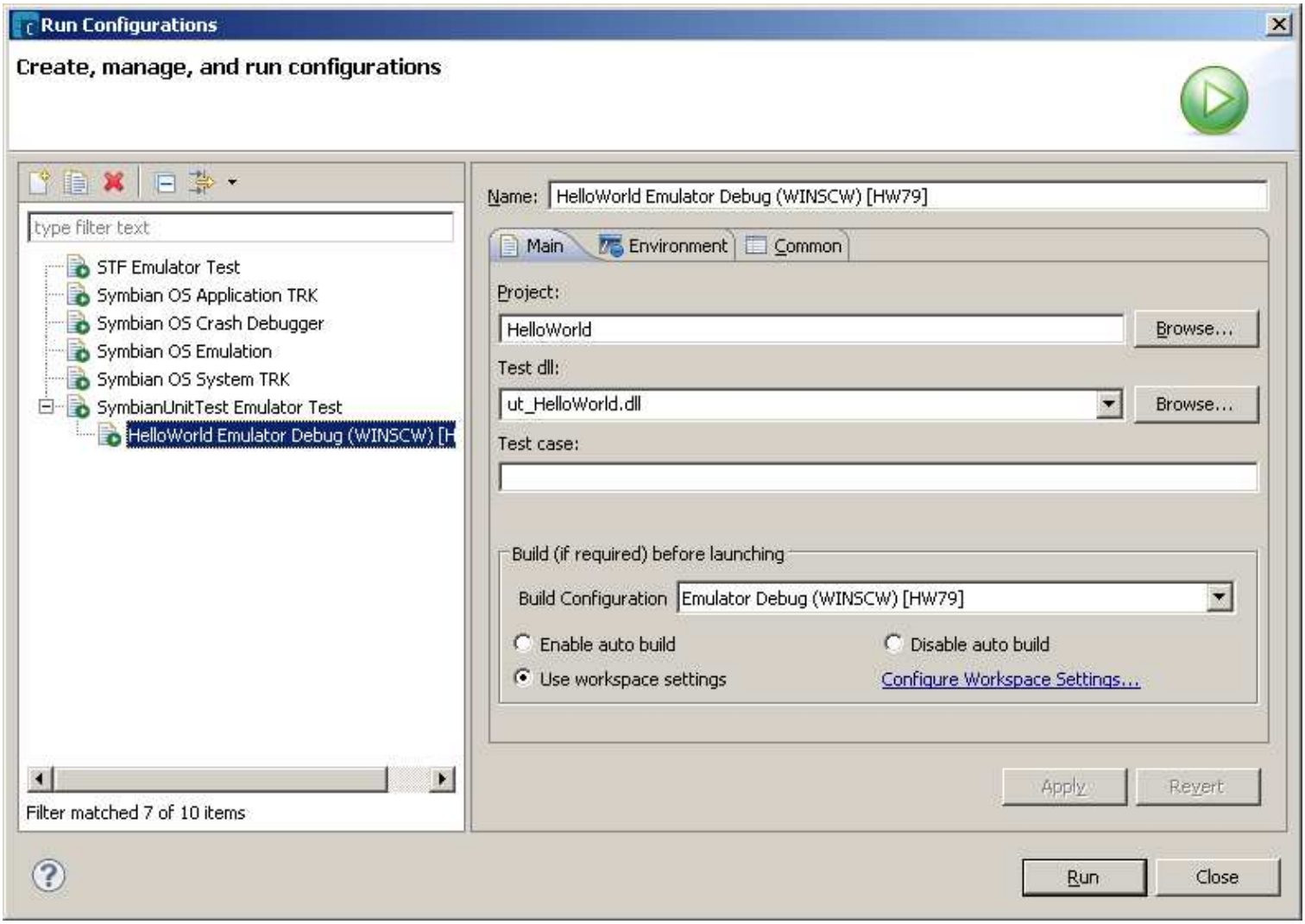
Symbian unit test are executed in emulator by defining a launch configuration. This configuration is saved, so it is an operation that needs to be done only once.

Launch configuration is created by selecting a menu item: **Run | Run Configurations...**, and then select Symbian Unit Test Emulation as your launch configuration type, then pressing the **New** button to create a new launch configuration.

This launch configuration has three tabs. The first one is specific to this launch configuration and the remaining two are common tabs for all launch configuration. It contains settings like where to save the launch configuration, and where to direct its (possible) output. More information about Environment and Common tab can be found from Carbide documentation.

If the project under Symbian Unit Test is selected in your workspace when the launch configuration is created, the wizard will fill all the fields automatically. Information can be then reviewed and launch can be preceded.

The picture below is an example:

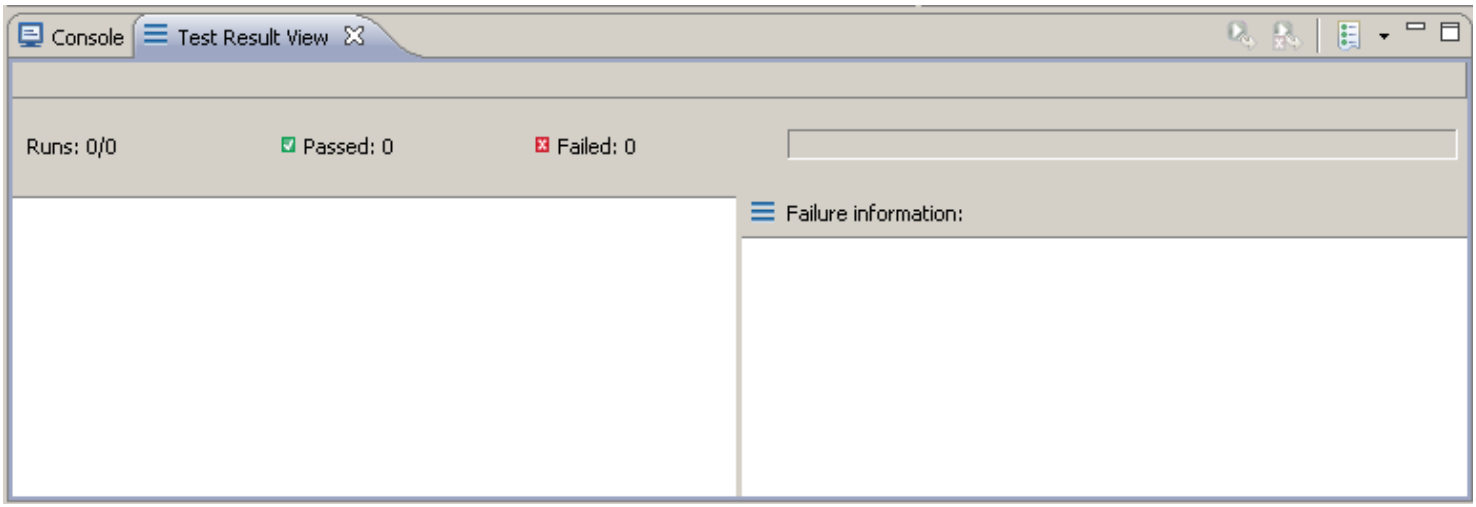



Some fields of launch configuration should be noted:

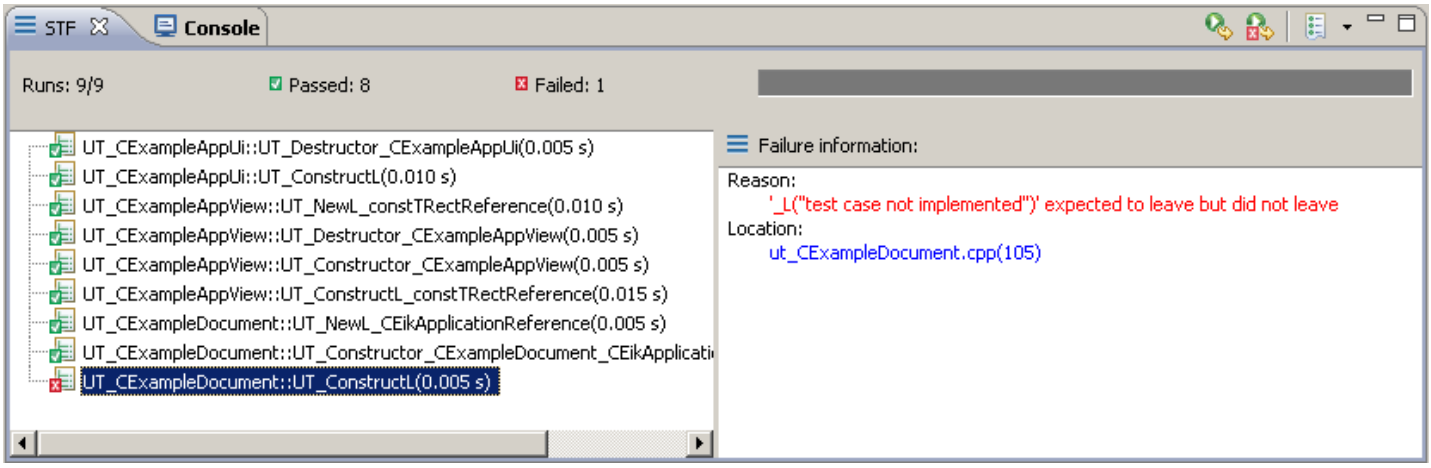
- **Project** field defines the project in which the test code is. This is used to get information about active build parameters in the project.
- **Test dll** defines the test dll which is to be executed in the emulator. Tab tries to resolve this automatically based on active build configuration in the project. For example on S60 platform, on "emulator debug" configuration, compiled dll files go to `%EPOCROOT%/Epoc32/release/winscw/udeb` folder.
- **Test case** defines the test cases which are to be executed in the emulator. To run part of test cases, use `UTClassName::TestCaseName` to specify individual case, use comma "," to seperate them and no whitespace allowed.

## Viewing Test Execution Results

When Symbian Unit Test launch configuration is launched, it will automatically show you the EUnit Results View. This view is by default located at the bottom central area of the IDE, as following:




As execution is ongoing the results will be updated on the view. After the execution the view looks like as following:




Buttons (right side of the upper tab) from left to right indicate:

- Rerun test: Executes again the tests which results are currently displayed.
- Rerun test-failed: Executes again the failed tests which results are currently displayed.
- Test Run History: All the history test result, select the drop down list to show the history result.

In the result view, double-click the link of failed case will open the source code and locate cursor at the the failed line.

## References

The following references are available for this tool:

- SymbianUnitTest user guide

## Symbian Unit Test Framework

Contents


- Overview
- SymbianUnitTest introduction
- Write a unit test using SymbianUnitTest
- Classes and Macros
- ConsoleUI usages
- SymbianUnitTest advanced features


### Overview

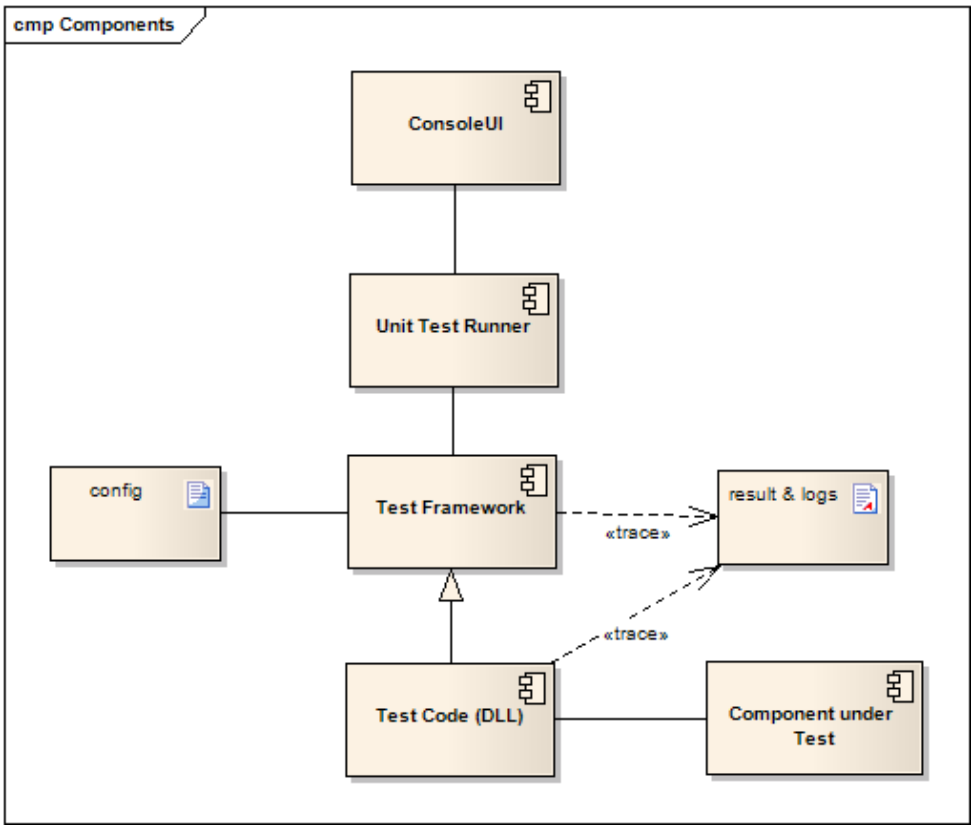
SymbianUnitTest is a unit test framework, which is a port of xUnit to Symbian C++. Its basic architecture and usage are closely linked with xUnit model.

SymbianUnitTest makes full use of Symbian C++ language features. It provides helper macros for users to easily define test cases.

Unit test frameworks are key elements of TDD(Test Driven Development). For this reason, SymbianUnitTest is not only considered as test framework to validate the software, but also the enabler for TDD and continuous integration to improve software quality and speed time to market.

### Introduction

SymbianUnitTest Framework consists of several components:




Test Framework:

Implemented as symbianunittestunitfw.dll, it is the core part of SymianUnitTest to handle all unit testing activities.

Unit Test Runner:
The test runner component is used for driving the test framework. It provides a unified interface to console and graphic UIs.

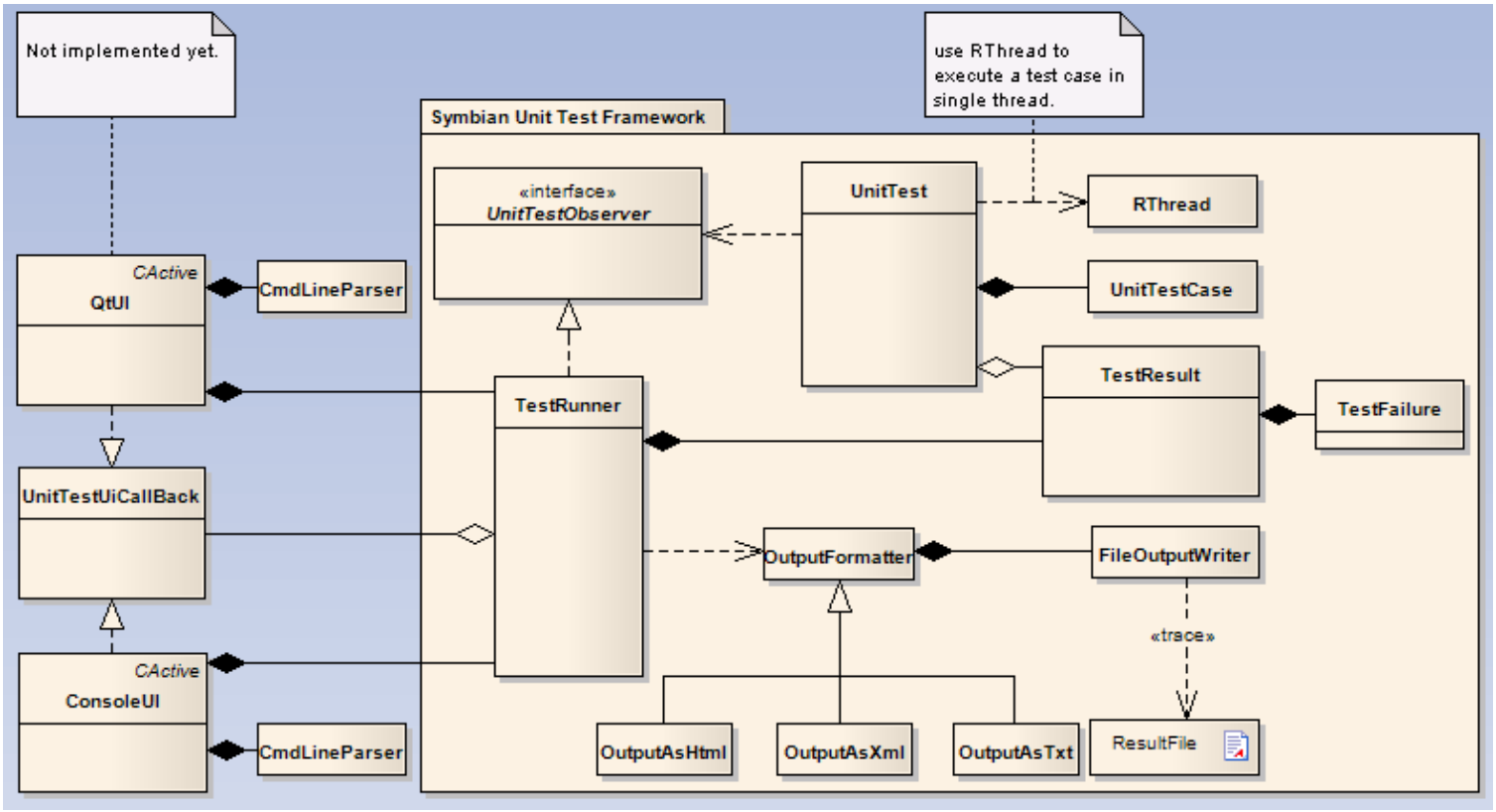
Test Code:
It is the user's test cases which are implemented as a DLL file.

Result & Logs:
The test report will be generated at c:\sut\SymbianUnitTestResults.html
If the logging is enabled while building and the c:\logs\sut directory exists, SymbianUnitTest will write debug information during test case execution.

ConsoleUI:
Implemented as symbianunittest.exe, it provides a console interface to users to run test cases via command line.

Usage (will explained in the later section):

Symbianunittest.exe –tests [-cases] [-alloc] [-help] [-output] [-timeout] [-noprompt]




ConsoleUI accepts user input from keyboard, CSymbianUnitTestCommandParser deals with command line options. Then CSymbianUnitTestRunner starts to handle the test cases by communicating with Symbian Unit Test Framework via MSymbianUnitTestObserver interface. Each test case is executed in a separate thread. At the end of the test case execution, Test Framework generates test result in user specified format.

### Write a unit test using SUT

The following code snippet is from the example Racecar, we can download it from https://developer.symbian.org/sfl/MCL/sftools/ana/testfw/testexecfw/symbianunittestfw/sutfw/sutfwexamples.

1. Select Class and methods to be tested
For instance, we want to test the class CRaceCar in the example. Two of the functions we want to test are AddFuel() and SpeedL().

```
#ifndef CRACECAR_H
#define CRACECAR_H

class CRaceCar : public CBase
    {
    // some constructors and destructor…

    public: // New functions
        TUint AddFuel( TUint aFuel );
        TUint SpeedL();

    // some other methods and attributes…
    };

#endif // CRACECAR_H
```

In order to get enough privillage to access private members in CRaceCar when creating unit test case, we need to define the test class UT_CRaceCar as friend class

```
class CRaceCar : public CBase
  {
  // ...

  private: // Test class
    #ifdef SYMBIAN_UNIT_TEST
      friend UT_CRaceCar;
    #endif // SYMBIAN_UNIT_TEST
  };
```

UT_CRaceCar will be introduced to CRaceCar as friend class if the macro SYMBIAN_UNIT_TEST has been defined in test project.

2. Define Test Class
Define a test class UT_CRaceCar, it will derive from CSymbianUnitTest,

```
#ifndef UT_RACECAR_H
#define UT_RACECAR_H

// INCLUDES
#include <symbianunittest.h>

// FORWARD DECLARATIONS
class CRaceCar;

// CLASS DECLARATION
class UT_CRaceCar : public CSymbianUnitTest
    {
    // Constructors and destructor…

    protected: // From CSymbianUnitTest
        void SetupL();
        void Teardown();

    protected: // Test functions, others are the same as this one.
        void UT_AddFuel();
        void UT_SpeedL();

    private: // Data
        // The object to be tested as a member variable:
        CRaceCar *iRaceCar;
    };

#endif // UT_RACECAR_H
```

Please notice some Symbian C++ essential functions such as NewL() and ConstructL() are ignored in this document.
Users can provide optional SetupL() and Teardown() functions. They will be called before and after test case execution. SetupL() usually initializes some of the resources, allocates memory, Teardown() usually in charge of some resource destruction so that the test object can reset its state for the next test case execution.
UT_AddFuel() and UT_SpeedL() are the unit test code to test the corresponding functions defined in the class CRaceCar.
The iRaceCar is the instance of class under test.

3. Create Test DLL entry.
The next file that we should take into consideration is dllentry.cpp which you can find in the example test project dllentry.cpp.
It only has one global function with the signature EXPORT_C MSymbianUnitTestInterface* CreateTestL(); in this function, it creates and return the test suites, so the SymbianUnitTest framework can load and run test cases.

The example code snippet is shown below:

```
EXPORT_C MSymbianUnitTestInterface* CreateTestL()
    {
    CSymbianUnitTestSuite* testSuite =
        CSymbianUnitTestSuite::NewLC( _L("ut_racecar") );

    testSuite->AddL( UT_CRaceCar::NewLC() );
    CleanupStack::Pop();

    // Add more tests to the test suite here when testing multiple classes
    CleanupStack::Pop( testSuite );
    return testSuite;
    }
```

4. Implement test cases.
One of the important functions should taken into account is the second phase constructor of the test suite. In this function, we must call the macro BASE_CONSTRUCT at the first line. Then the unit test methods can be added by using the macro ADD_SUT. Please note that SetupL() will be invoked before each test case runs and the Teardown() will be invoked after the test case execution.
We can specify the SetupL() and Teardown() functions for certain test case simply by using the macro ADD_SUTWITH_SETUP_AND_TEARDOWN.

The example code snippet is shown below:

```
void UT_CRaceCar::ConstructL()
    {
    BASE_CONSTRUCT
    ADD_SUT( UT_AddFuel )
    ADD_SUT_WITH_SETUP_AND_TEARDOWN( SetupL, UT_ SpeedL, Teardown )
    }
```

Here is an example a a unit test function.

```
void UT_CRaceCar::UT_FuelL()
    {
    SUT_ASSERT_EQUALS( 0, iRaceCar->FuelLeft() )
    SUT_ASSERT_EQUALS( 0, iRaceCar->AddFuel( KInitialFuel ) )
    SUT_ASSERT_EQUALS( KInitialFuel, iRaceCar->FuelLeft() )
    SUT_ASSERT_EQUALS( 2, iRaceCar->AddFuel( 1 ) )
    SUT_ASSERT_EQUALS( KInitialFuel, iRaceCar->FuelLeft() )
    }
```

In this function, we use the macro SUT_ASSERT_EQUALS to assert if the test function's return value is what we expect, more macros will be introduced in later section. It will be marked as failed if the actual return value doesn't match the expected value when using the macro SUT_ASSERT_EQUALS. The rest of the functions should be tested follow the same way as we mentioned in this section.

5. Create project mmp file.
In the next step, we will create a project file. Showing below is a snippet of the mmp file:

```
1) TARGET       ut_racecar.dll
2) TARGETTYPE   dll
3) UID          0x20022E76 0x0E6A3C34
4) MACRO        SYMBIAN_UNIT_TEST
5) SOURCE       racecar.cpp
6) SOURCE       dllEntry.cpp
7) SOURCE       ut_racecar.cpp
8) SYSTEMINCLUDE /epoc32/include/symbianunittest
9) LIBRARY      symbianunittestfw.lib
```

The type of the test project is a dll project, so we should indicate the target type of the project, like line 2, we specify dll as the TARGETTYPE.
At line 1, we set a target file name for the file.
At line 3, we must specify the UID2 as it is identified for the SymbianUnitTest's DLL.
At line 4, we must add the SYMBIAN_UNIT_TEST macro to enable the macros the test framework provided.
We also need to list the source code under test (in this case, the source file of the testing class is racecar.cpp), so we add it at line 5.
We added dllEntry.cpp to the mmp file as stated in section "3 -- Write dll entry" at line 6.
ut_racecar.cpp is test case implementation, we add it at line 7.
Line 8 and line 9 contain the directory of the header files and the library file of the SymbianUnitTest we need in our test.

6. Create bld.inf for the test project.
We also need to add the newly created project mmp file to bld.inf, just like the following shown below:

```
PRJ_TESTMMPFILES
ut_racecar.mmp
```

7. Build project and run it.
In the test project group directory, execute the following command to build it:

```
bldmake bldfiles
abld test clean winscw
abld test reallyclean winscw
abld test build winscw udeb
```

Run "symbianunittest.exe -t=ut_racecar.dll" to execute the test, we should get the test result at "c:\sut".

### Classes and macros

1. There are several macros used in Symbian Unit Test Framework, we should define the macro SYMBIAN_UNIT_TEST in our test project's mmp file to enable them.

BASE_CONSTRUCT:
Calls the base class constructor that sets the name of unit test, usually used in ConstructL in wrapper class.

ADD_SUT( aTestPtr ):
Adds a new unit test case to this unit test. The default setup and teardown functions will be used.

ADD_SUT_WITH_SETUP_AND_TEARDOWN( aSetupPtr, aTestPtr, aTeardownPtr ):
Adds a new unit test case to this unit test. The user can specify the customized setup and teardown functions.

SUT_ASSERT( aCondition ):
Asserts a condition in a unit test case. Leaves with a Symbian unit test framework specific error code if the condition evaluates to EFalse.

SUT_ASSERT_EQUALS( aExpected, aActual ):
Asserts that two values are equal. Leaves with a Symbian unit test framework specific error code if the values are not equal.

SUT_ASSERT_LEAVE_WITH( aStatement, aError ):
Asserts that a statement leaves an expected value. Leaves with a Symbian unit test framework specific error code if the leave code is not the expected one.

SUT_ASSERT_LEAVE( aStatement ):
Asserts that a statement leaves. The macro itself leaves with a Symbian unit test framework specific error code if the statement leaves.

The detailed explaination can be found in symbianunittestmacros.h.

2. Classes in Symbian Unit Test Framework.

In this section, we will review some key classes in SymbianUnitTest Framework. While writing a test project, below classes should be aware of from the client's perspective:

CSymbianUnitTest
CSymbianUnitTest is a basic test unit; it corresponds to the class which we will test. Client programmer should derive from it and implement test code in CSymbianUnitTest's derived class.
Client user needs to override two methods SetupL and Teardown if extra initialization needed to run test case.

CSymbianUnitTestSuite

CSymbianUnitTestSuite is the collection/container of CSymbianUnitTest. On the other words, it simply collects a number of CSymbianUnitTest's derived classes and run them during a particular test run.

### Console UI

There is a console based application provided by Symbian Unit Test Framework which is used to execute test cases.

filename:
  symbianunittest.exe

allowed arguments:
  -tests|t=<dllfilename1,dllfilename2,...>
  -cases|c=<case1,case2,...>
  -alloc|a
  -help|h
  -output|o=<html|xml|txt>
  -timeout|to
  -noprompt

each option has a short term form such as -h is the short term form of -help.

-help/-h : print help message (the same effect as run symbianunittest.exe without any parameter)
  Example : symbianunittest -help

-tests/-t : specify the test dll files which include the symbian unit test cases, seperated by ‘,’.
  Example : symbianunittest -tests=testdllfilename1,testdllfilename2,testdllfilename3

-cases/-c : specify which test cases should be ran during the test procedure. the names of the test cases are seperated by ‘,’. (optional)
  Example : symbianunittest -tests=testdll -cases=testcasename1,testcasename2,testcasename3

-timeout/-to : set the time out value for test execution, 30 secondes by default, 0 for never timeout. (optional)
  Example : symbianunittest -tests=testdll1,testdll2,testdll3 -to=3
  Set time out to 3 seconds.

-output/-o : set the output format, default is html. Test result file is stored in c:\sut.
  Example : symbianunittest -tests=testdll1,testdll2,testdll3 -to=3 -o=xml

-alloc/-a : Memory allocation failure simulation. (optional)
  Example : symbianunittest -tests=testdll1,testdll2,testdll3 -to=3 -o=xml -alloc

-noprompt : perform a quiet test for unit test execution. (optional)
  Example : symbianunittest -tests=testdll1,testdll2,testdll3 -to=3 -o=xml -alloc –noprompt

### Further Study in SymbianUnitTest

1. SymbianUnitTest Framework can also detect memory leaks. It will leave with a specified leave code if memory leak occurs.

2. Interact with ATS.
Test cases created for SymbianUnitTest can run with ATS, please refer to ATS User Guide for more information.

3. Convert EUNIT test case to SymbianUnitTest.
There is a Perl script called eunit_to_symbianunit.pl, which is used to convert the existing EUNIT test cases to SymbianUnitTest. Run it under the eunit test cases folder then the test cases are converted to SymbianUnitTest test cases.

## License Information

#### COPYRIGHTS



Initial Contributors:
Nokia Corporation - initial contribution

---