What is C-TestIt!

C-TestIt! is a product that allows users to “unit test” their C code. “Unit testing” in the case of C is understood as the ability to test a function with a number of different sets of parameters and then check the expected results, this is usually referred to as Black Box testing. C-TestIt! also allows users to specify a number of “assertions”; “assertions” are conditional statements that will be checked by C-TestIt! during execution of the function under test, this is referred to a Gray Box Testing.

C-TestIt! uses a unique approach for unit testing, in that it actually allows you to test functions in the very body of your own application, since the tests are done not on the C source but on the actual executable file that is your application. This approach guarantees that the function is working in its final environment, and it relieves the sometimes tedious steps of having to recompile and link the tested function with added code that will implement the test. In the case of C-TestIt!, no code is added to your function, no compilation or link is necessary, C-TestIt! directly uses your own application.

C-TestIt! Main features

- In Application Unit Testing.The code under test is the exact code you will later put into your target

- Can check every function of the project
  All functions compiled in debug mode can be checked, with a single file load.

- Gray Box Testing
  User can specify a number of conditional statements that will be evaluated during function execution.

- Works directly on the application code: no need to recompile or re-link

- Defines input parameters and expected results
  All parameters of the function under test can be specified, using C expressions.

- Defines values for global variables
  Global variables can de defined for every test using C expressions

- Test the function at their real location
  Since the test is done using the real application code, functions are thus tested at their real location thereby releaving the issues of function location, bank switching, ...

- Test with the same memory model and compiler options than the real project
  The real code is tested so all options and parameters of the code are the respected

- Inputs (arguments, globals) can be specified as a single value, or as a range or a set of values thereby allowing to test the same function with different inputs in the same test session.

- Create a suite of tests, or “Testoramas” to be run together.

- Supports simulator and real hardware (BDM or ICE, JTAG, Emulators)

  C-TestI! Offers several variants for executing the code under test.

- Supports all targets for which Cosmic Tools are available.

- Runs interactively or in batch mode

  Tests can be run immediately and results visualized graphically or they can be run in batch mode with logging of output results.

- Can produce reports for archive

- Can produce additional information such as Code Coverage and execution timing

Using C-TestIt!

As explained earlier C-TestIt! runs using an executable file produced by the Cosmic Tools. Tests can be created, saved, loaded and executed later, or they can be grouped in “Testoramas”.

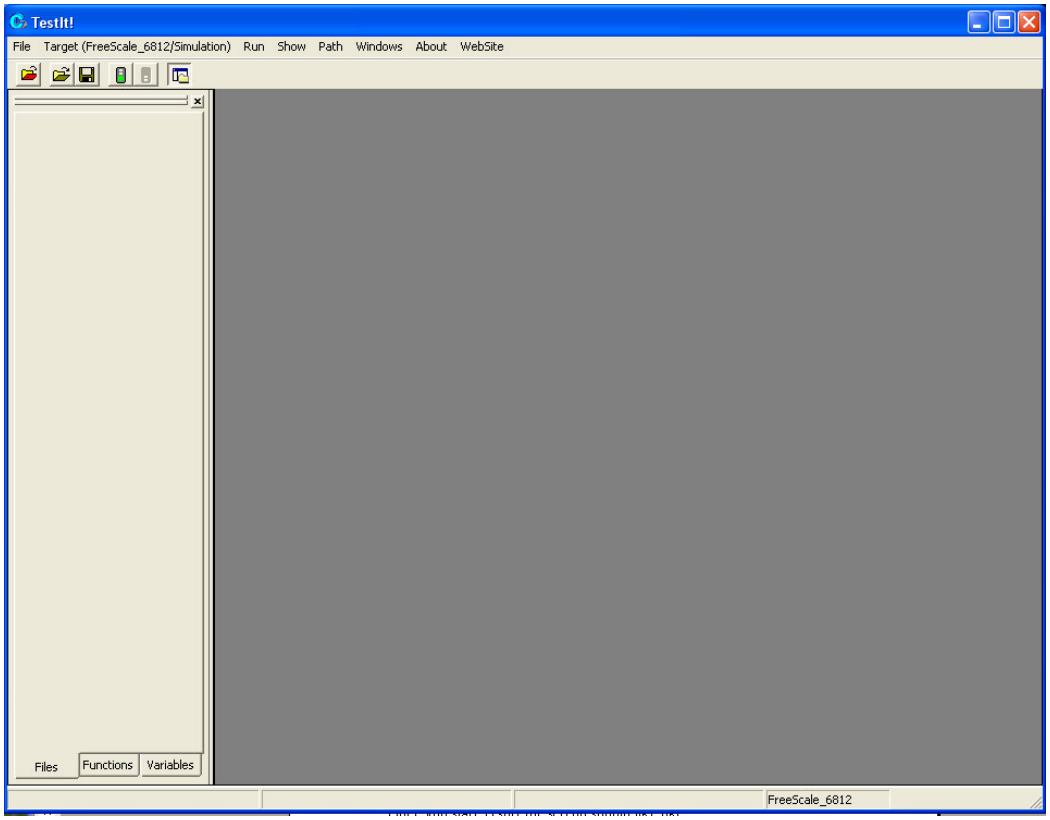
We are now going to see how:

To start C-TestIt!
Create a simple test and specify input and output values
Specifying input and output values
Save a simple Test
Run a test
Adding Assertions
Create a Testorama
Save a Testorama
Run a testorama

Starting C-TestIt!

C-TestIt! can support all targets for which a Cosmic tool chain exists. So one you have started C-TestIt! you need to specify for which target processor and which execution environment you are going to specify the tests. Please note that the target specification becomes part of the test definition, while the execution environment is not; i.e. a test for a specific target can be used with any execution environment available for that specific target.

Once you start C-Testit! the screen should look like:

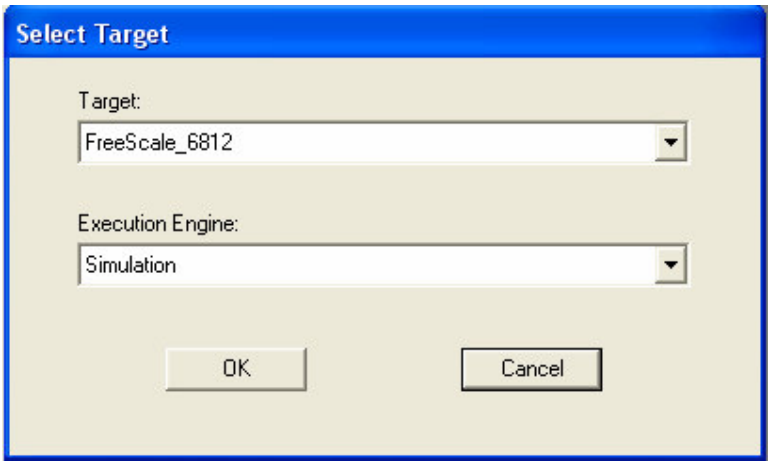


The main window is composed of:

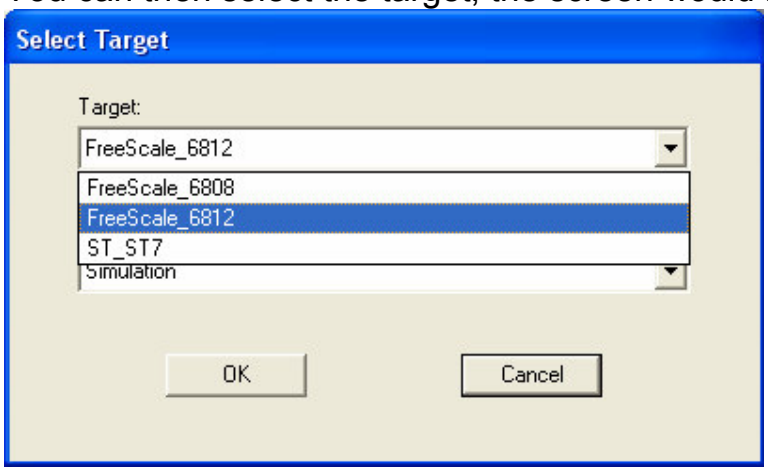
- the application pane on the left which will show all the components of the executable file for which tests will be built/run
- the test window where tests will be displayed, as well as source files if necessary.

The menu shows what is the current target and environment, in the above example the Freescale HC12 is selected and the simulation execution environment is also selected. To change these selections, use the menu entry.

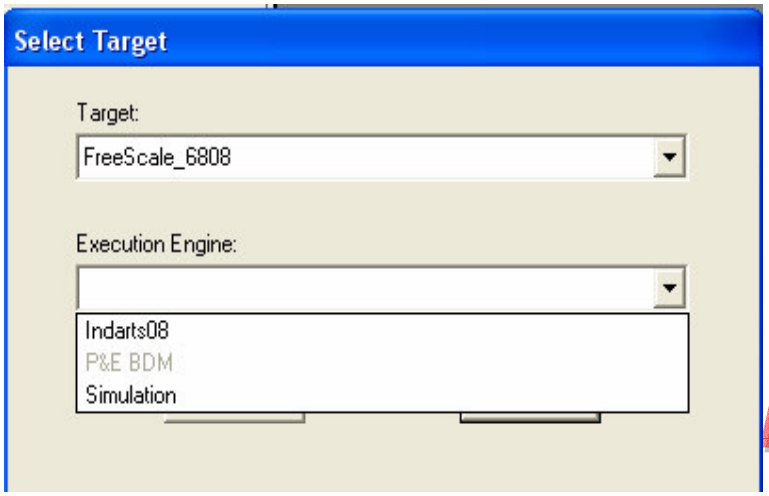
You will then get the following dialog that will allow you to make your own selection:



You can then select the target, the screen would look like:



And you can select the execution environment, the screen will look like:



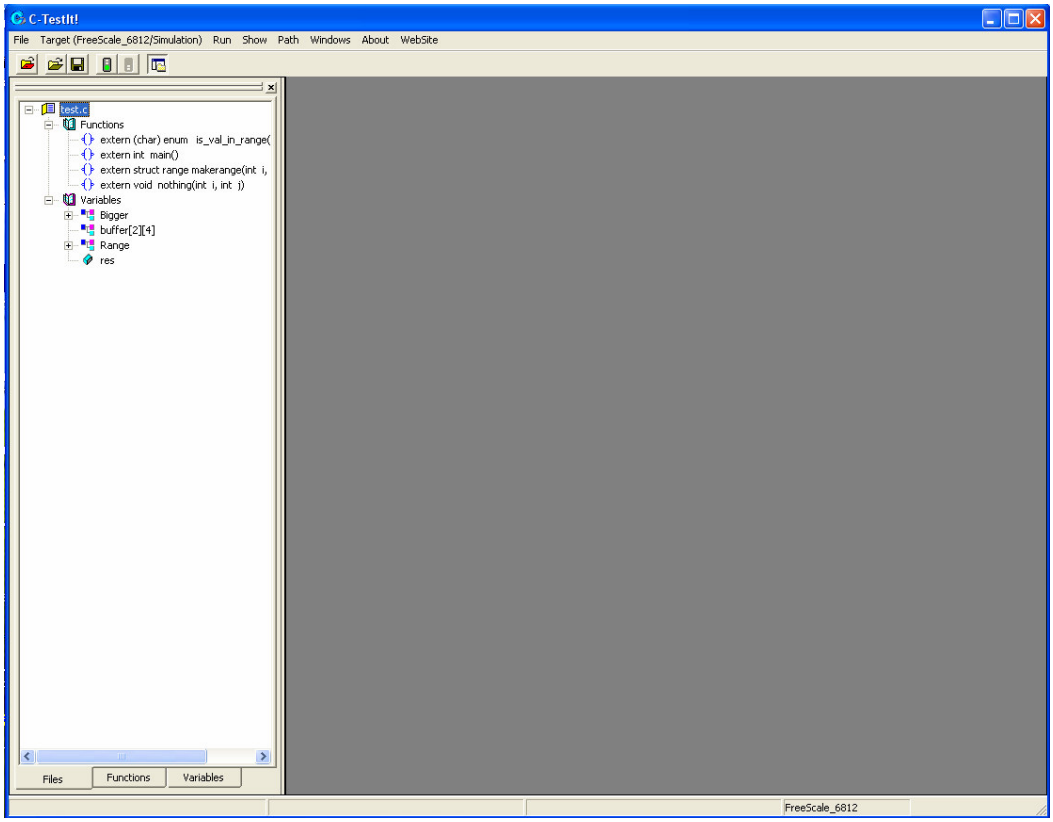
Once you have finished the selections, you can start creating new tests.

Creating a Test:

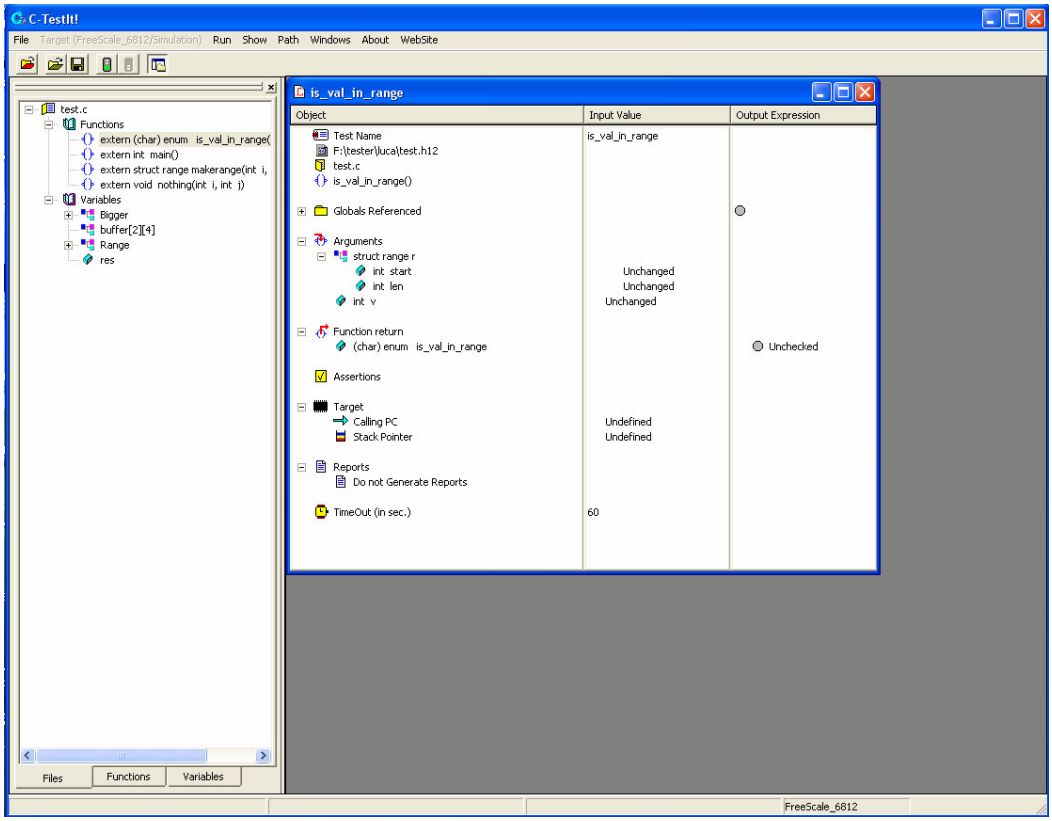
To create a test the first step is to load the executable file that contains the function to be tested. To do so you can use either the menu or the button bar.

Once you have loaded the executable file the application pane will display information about the application. You will then be able to list the functions names and the variable names included in your application.

Your screen will look like:



Now to create a test for a specific function just right click on the function name in the application pane; this will open a test window with all the components of the test displayed with their default values.



The test window is composed of three columns. The leftmost one lists all the objects that are manipulated by the test, the middle one shows input values, when appropriate, and the right most one shows output values where appropriate. When a test is created all appropriate values are set to their defaults. “Unchanged” is used to indicate an input value that is not specified and “Unspecified” is used to indicate an output value that needs not be checked for this test.

The leftmost window lists all the objects of the test in the following order:

- Test Name: this is a name that by default is the same as the function name under test. The user can edit this by right clicking on it.
- The Executable file name used for the test.
- The name of the source file that includes the function under test.
- The name of the function under test

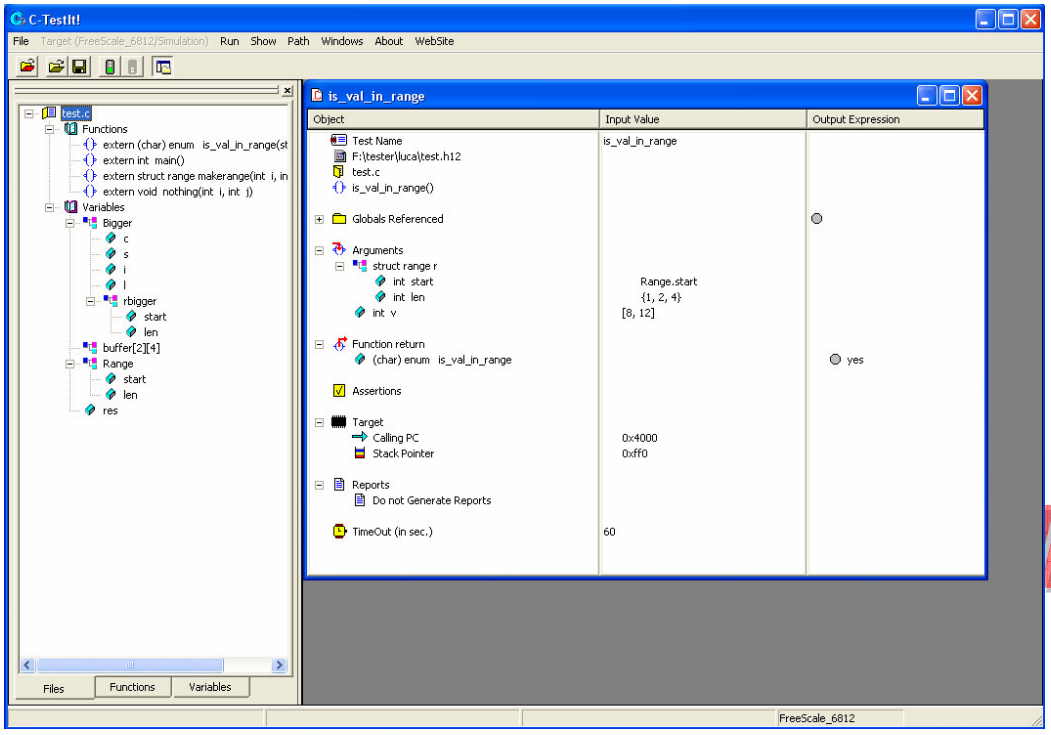
- Then we find the Globals entry. This entry exists if and only if the function under test uses globals variables of the program. This entry can be expanded to view all the use variables as well as their components for aggregate variables. Each of these variables can receive an input value for the test by right clicking the correspondent entry in the middle window, and receive an output value by right clicking the correspondent entry in the rightmost window.
- This is then followed by the list of arguments to the function if appropriate. Each argument can receive an input value by right clicking on the correspondent entry in the middle window.
- The function return value. In case of a function returning an aggregate, this entry can be expanded to show all components. The return value can have a specified output value by right clicking the correspondent entry in the rightmost window.
- Then one finds the “assertions”. Assertions are conditional expressions that will be tested during the execution of the function. To add an “assertion” simply right click on the assertion Icon or text in the leftmost window; this will bring up a dialog that is used to specify assertions.
- After the Target entry is displayed. This entry allows to specify the Stack value used for the test, and the address from which the function call should be executed.
- The Reports entry that allow to specify whether reports should be created and where they should be saved To modify the report status simply right click on, the Reports entry in the leftmost window.
- Finally there is the Time-Out entry. This entry allows to specify a time out for the execution of a function. This is to cope with situations where the code being tested does not “end” execution. The Time out value is used to stop a test in such cases. To modify the Time out value simply right click on the correpondent middle window entry.

Specifying Input and Output values

To specify an input or an output value right click on the entry where appropriate. Input output values for globals, arguments, as well as function return, can be specified either as a simple value, or as a valid C expression. Additionnaly Inputs can be specified as a range of constant values or as a set of constant values. To specify a range the following notation is used: [<low_val>,<high_val>], the test will be run for every value in that range; to specify a set the following notation is used: {<val1>, <val2>, ..., <valn>}, the test will be run for every value in the set.

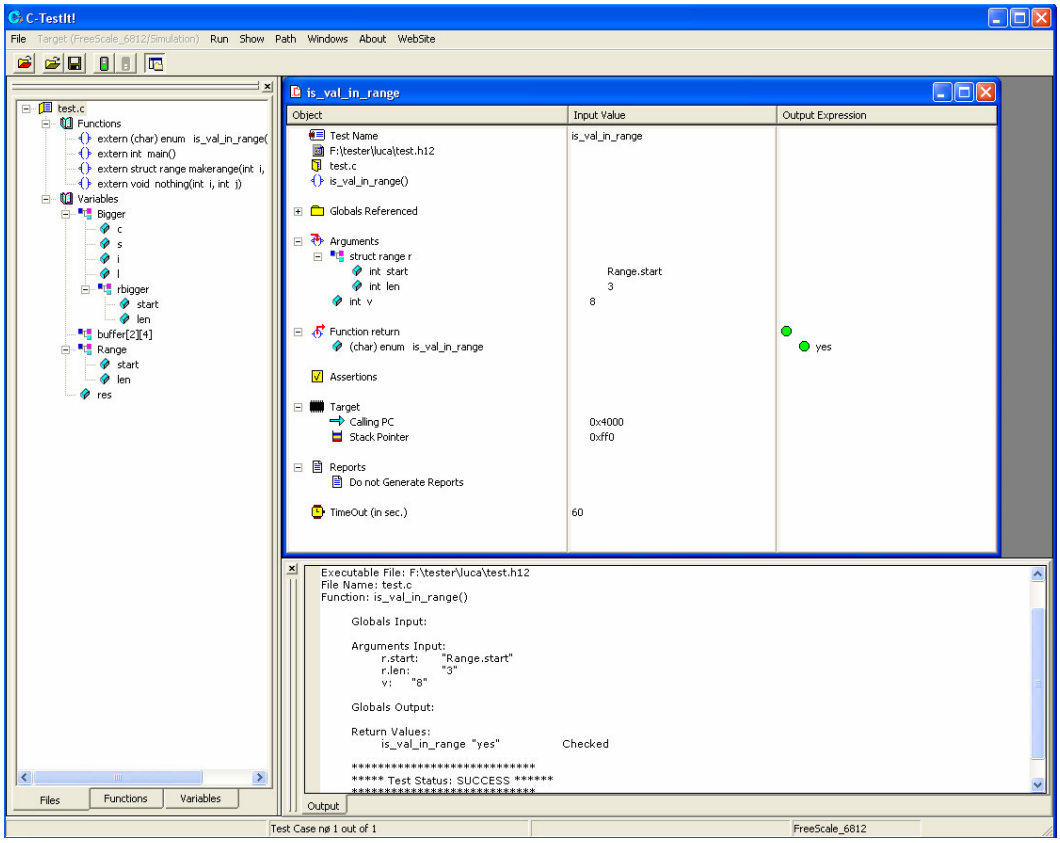
If an output value is specified as a constant or a C expression without any comparators (i.e. <, > ...), the it is taken to specify the exact value of the corresponding object; otherwise it is taken to be an expression to be evaluated and the value thus obtained is tested for true or false.

Once input, output have been specified the screen will look like:



Run a test

One a test is completely specified, you can launch the test, and the result will be something like:



You can see the GREEN light icon next to the function output, which highlights the fact that the return value of the function does match the output value specified, if that was not the case the Icon would be a red light. You can also see the output window which contains a textual report as to the test execution.

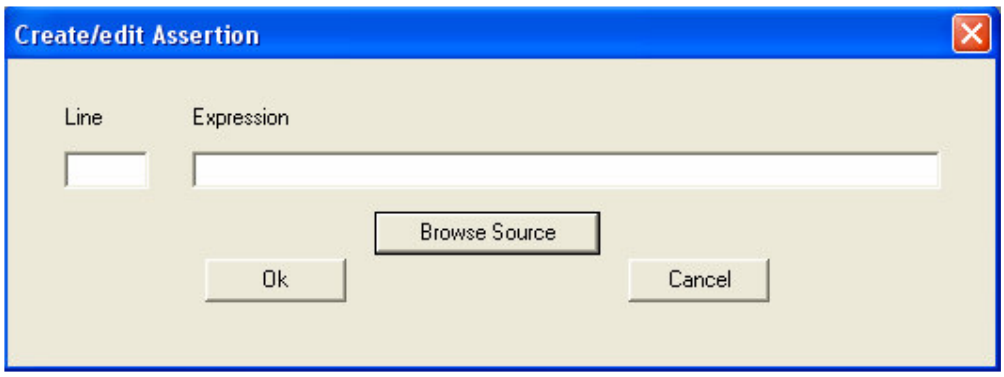
Adding Assertions

Assertions are expressions that will be evaluated while the program is running. Assertions are attached to a particular source line in the function under test. They allow to test a condition every time this line of code will be executed. Please note that the assertions are evaluated BEFORE that line of code they are attached to, is executed.

Assertions can for example allow you to test that a particular variable does meet some specified condition when a line of code is reached.

To add Assertions to a test, simply right Click on the assertion Icon in the leftmost part of the test window.

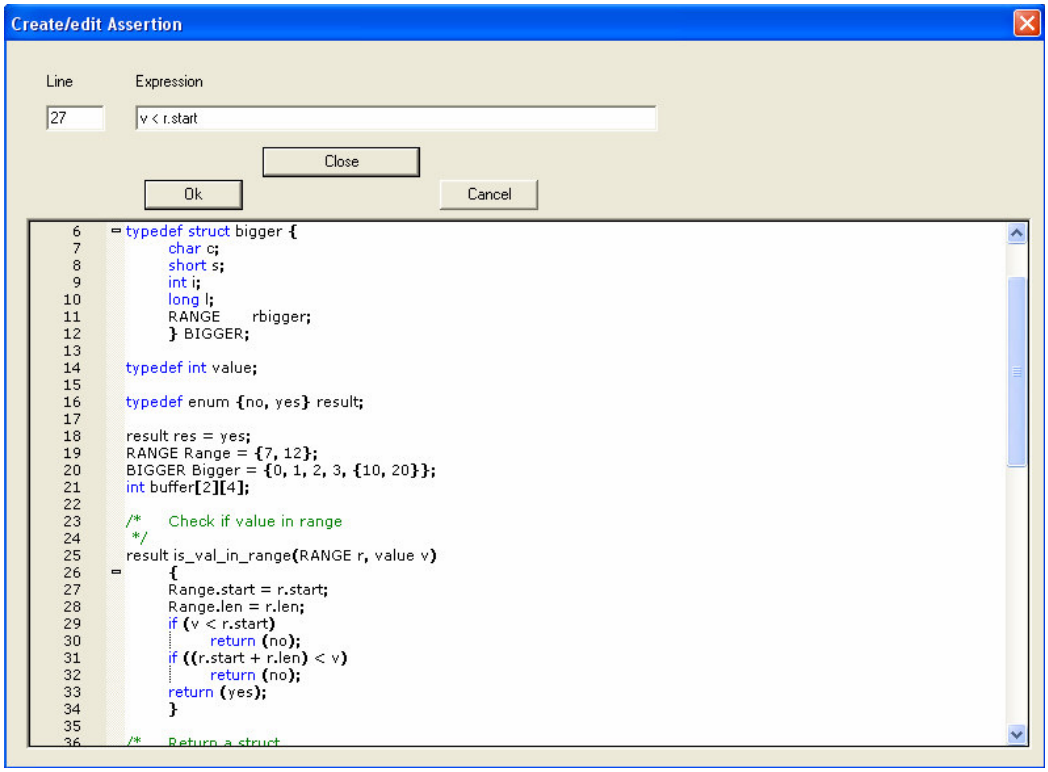
The following dialog is then displayed:



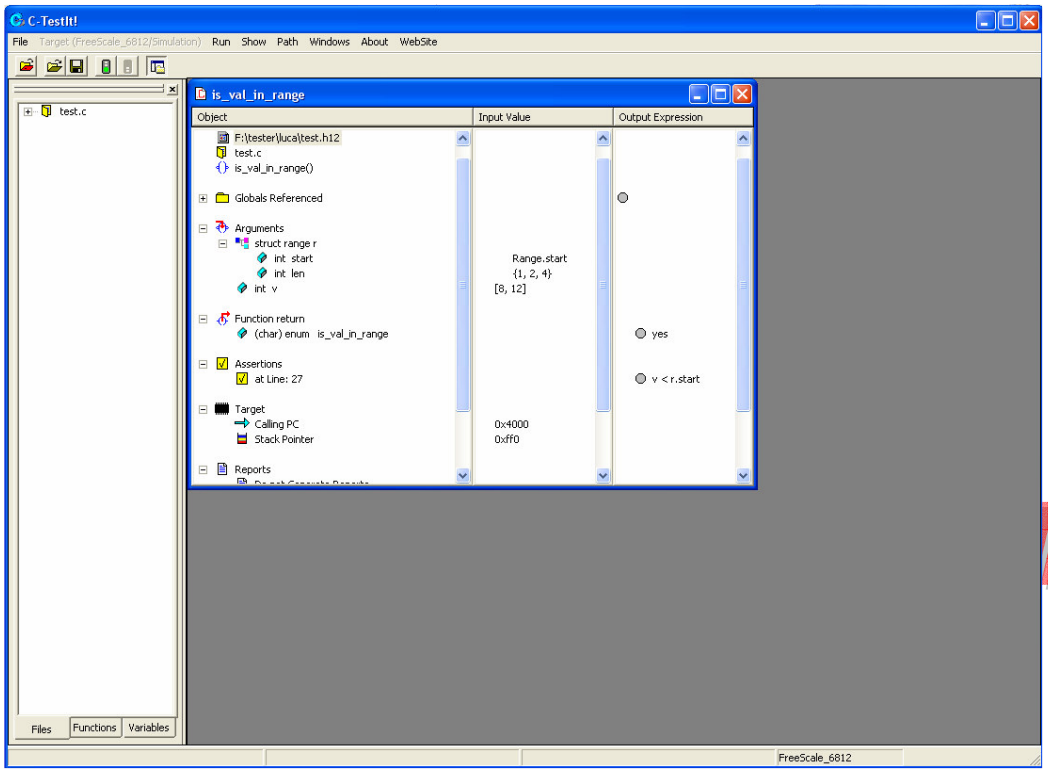
The Browse button can be used to display the code of the function under test. You must then specify a line number, and a valid C expression, that will be evaluated when that particular line of code is reached.

When a test is executed, assertions will be displayed with a GREEN light icon is their expression is TRUE, and will be displayed with a RED light Icon if their expression is false.

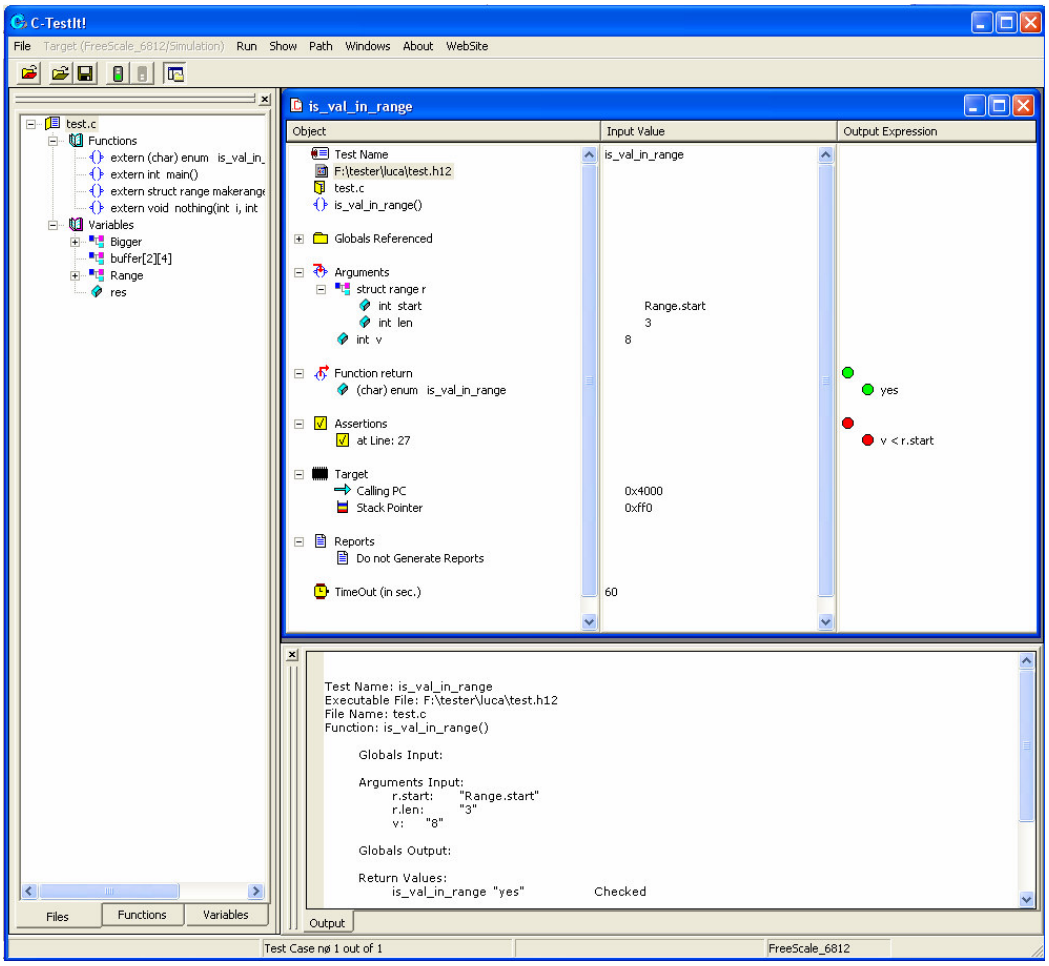
Here is an example of assertion:



Once you validate the assertion (by clicking OK), the screen will look like:



If we run a test with assertions here is what the screen may look like:



In this case you can see that the assertion is displayed with a RED LIGHT icon because it did not evaluate to TRUE.

Assertions are the means to create “GRAY BOX Tests”, i.e. tests where the user has visibility of what happens inside the function under test.

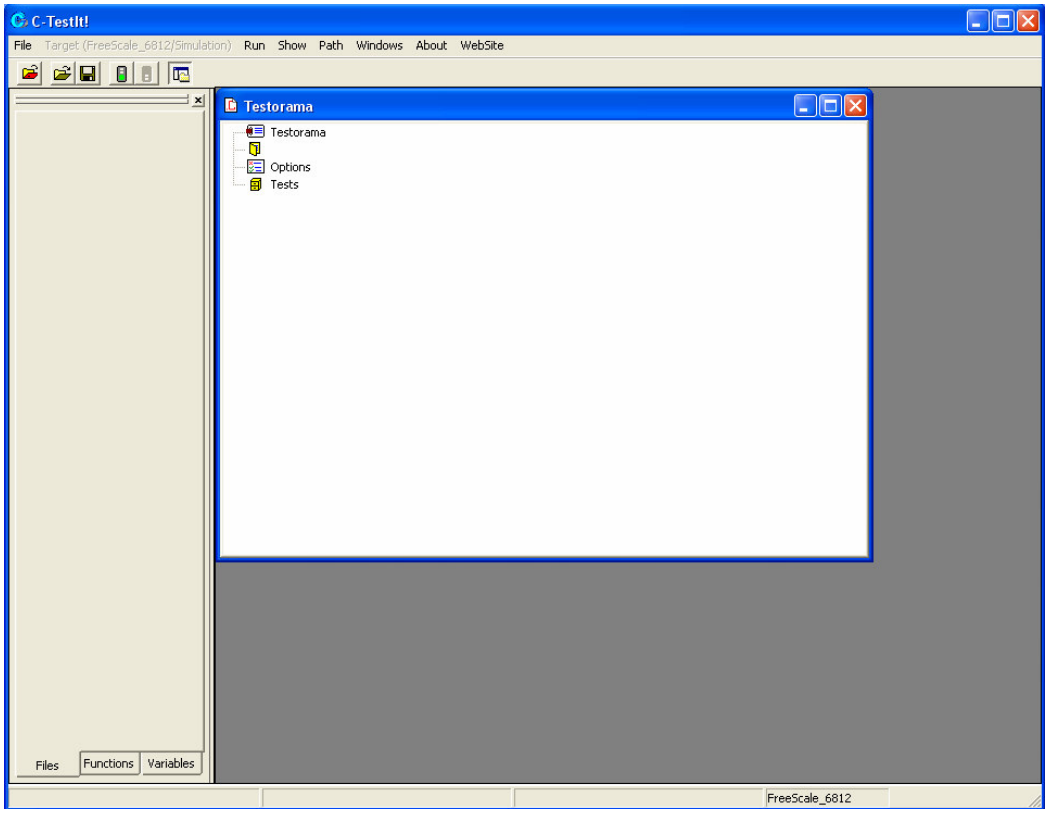
Save a Test

To save a test, select the appropriate menu option under the File Menu. You will have to enter a name for the saved test. Tests are saved in .CTH files, and can then be later reloaded for execution or editing/updating.

Creating a Testorama

A Testorama is a suite of tests all using the same executable file. When a testorama is executed reports for all of the tests can be collated in a unique report thus allowing the user to gather test information for a set of related functions.

To create a Testorama, use the menu Option File->New->Testorama, this creates aTestorama window and the screen should look like:

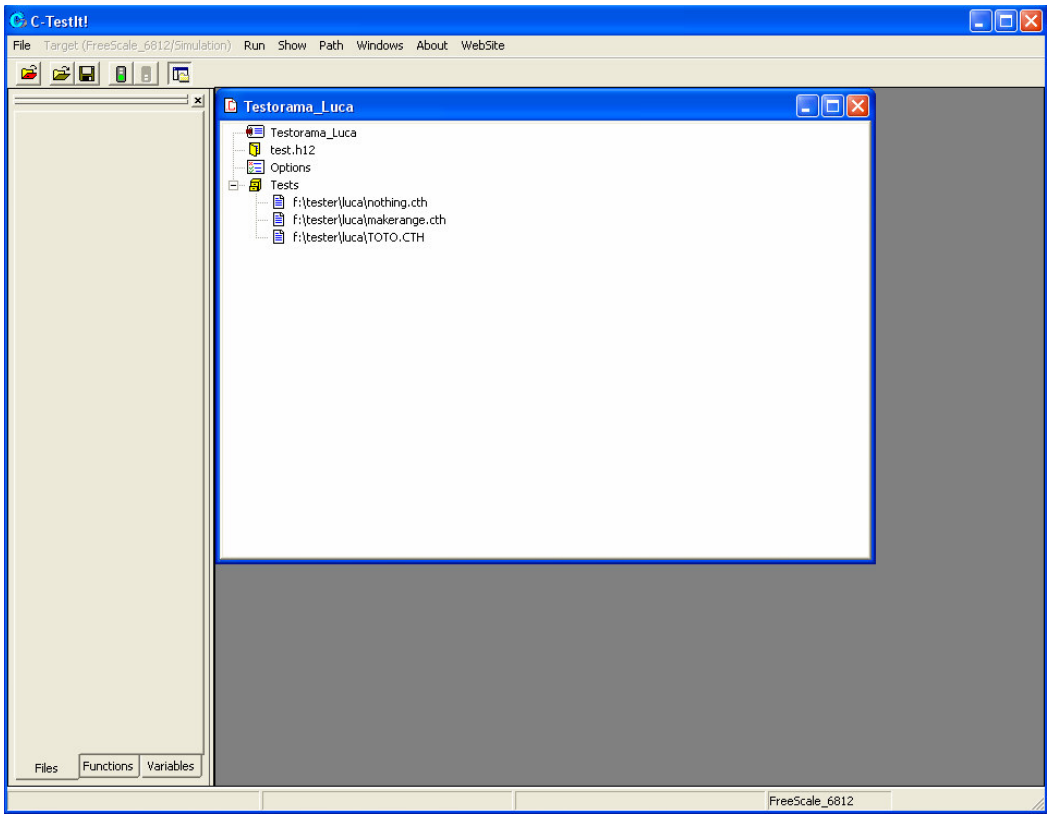


The testorama window show:

- The Testorama name. You can update this by right clicking on it.
- The Icon for the executable file that the testorama will use. This will be updated when you add a test in the Testorama

- The Options Icon will allow to specify options for the testorama
- The set of test files for this Testorama.

Here is what the screen may look like once you have started to specify the components of the testorama:

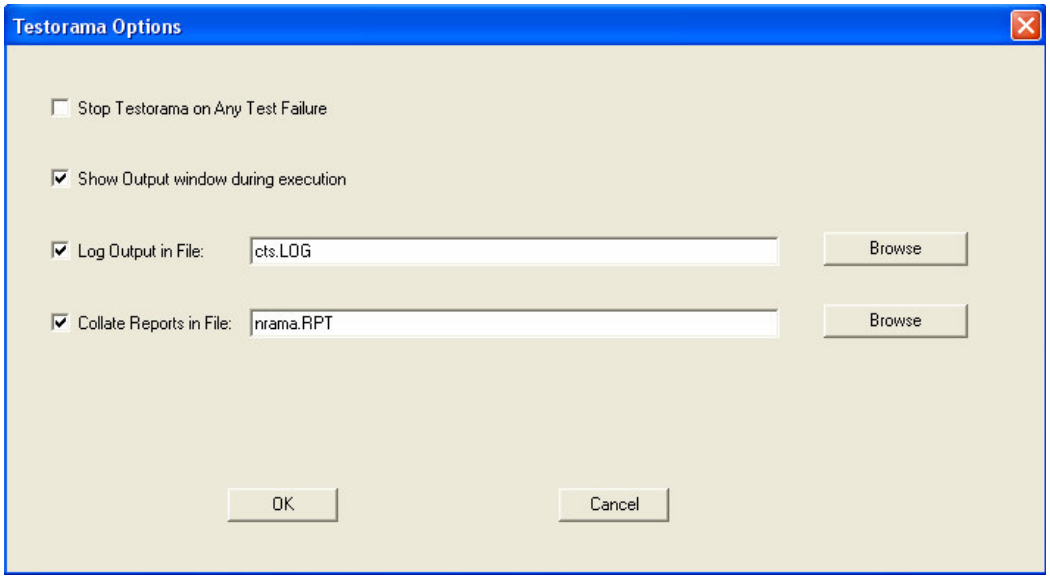


This testorama is based on executable file test.h12, and it will execute the following tests is sequence:

f:\tester\luca\nothing.cth
f:\tester\luca\makerange.cth
f:\tester\luca\toto.cth

preliminary

The options of a Testorama can be viewed/specified by right clicking, the Options Icon. The following dialog is then displayed:

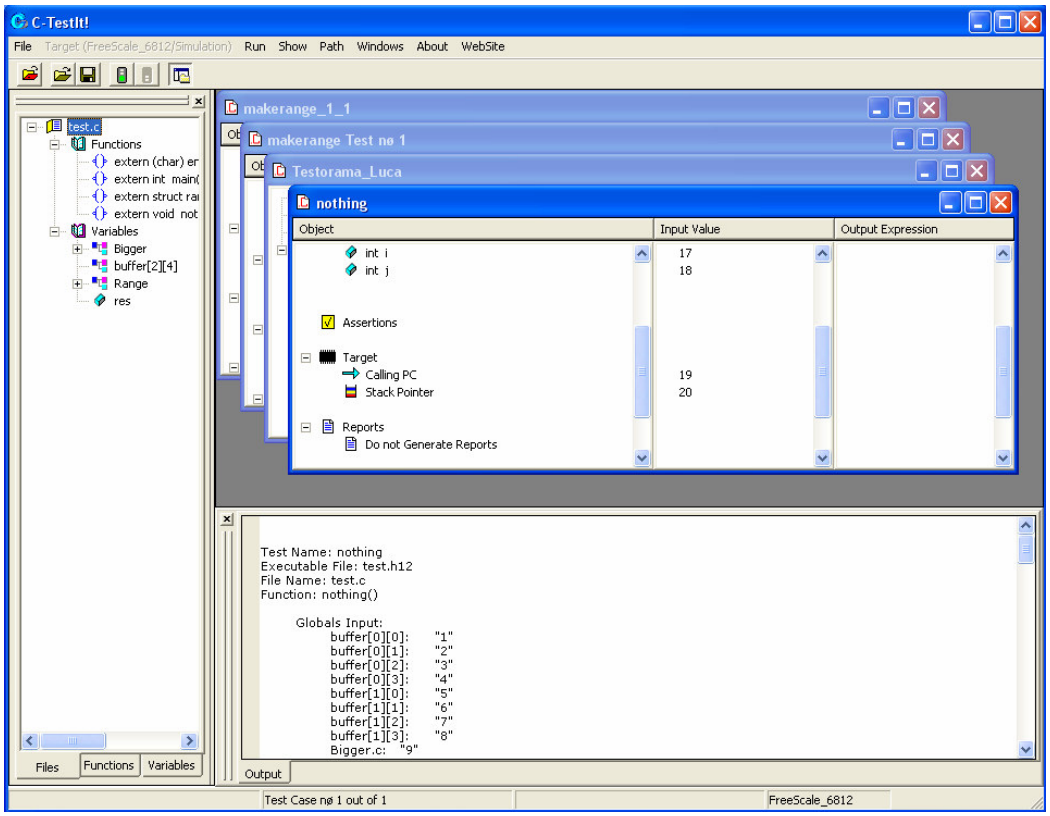


The options are:

- Stop Testorama on Any Test Failure
- Show Output window during execution. This will open the output window to show the tests execution.
- Log Output To File. This allows to duplicate the output produced in the output window into a file, which can be archived or examined later
- Collate Report Files, this allows to collate all the report files produced by the individual tests in a unique file.

Run a Testorama

Once a Testorama is completely specified, you can launch the test, and the result will be something like:



A test window is opened for every test specified in the testorama, that test window show results for that particular test.

Save a Testorama

To save a Testorama, select the appropriate menu option under the File Menu. You will have to enter a name for the saved testorama. Testoramas are saved in .CTS files, and can then be later reloaded for execution or editing/updating.