



Automation & Testing Suite

for embedded software | AUTOSAR-compatible

ATS 5.2.0

Basic Usage Manual

SCHLEISSHEIMER SOFT- UND HARDWAREENTWICKLUNG GMBH

www.automation-testing-suite.com

www.schleissheimer.com



2024

Contents

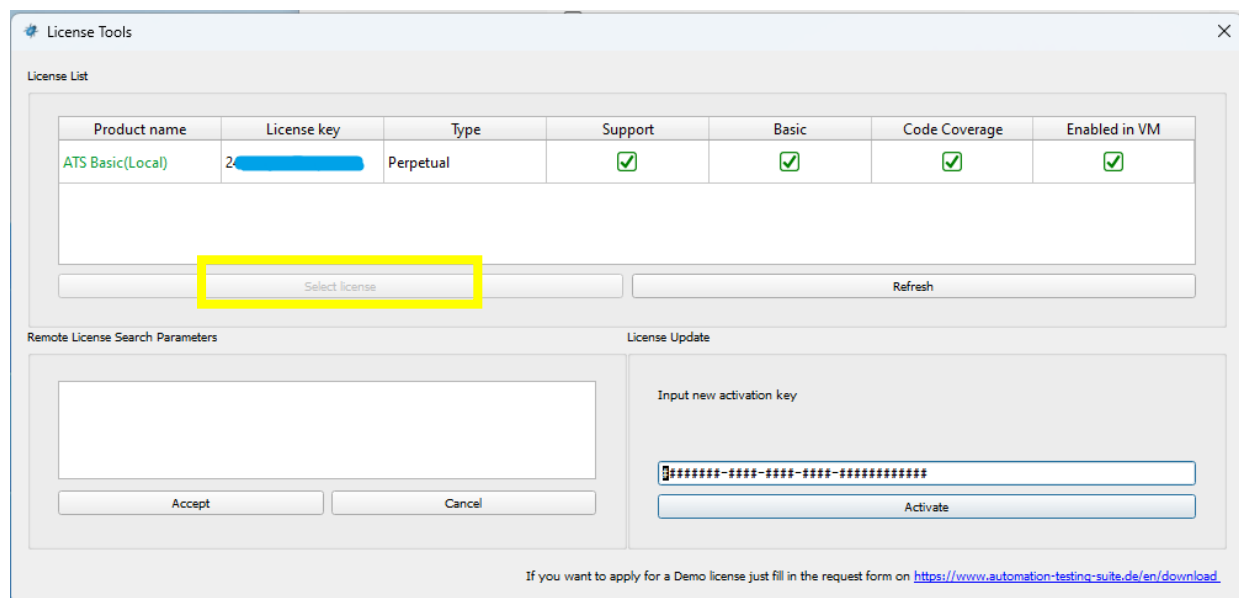
Chapter 1. Getting started	3
1.1. Project creation	4
1.2. Opening existing project.....	11
1.3. Removing project.....	12
1.4. Saving project	13
Chapter 2. Testing files	15
2.1. Building SimuDLL	15
2.2. Adding new test	17
2.3. Modifying a test	18
2.3.1. Sequences.....	22
2.3.2. Range values	23
2.3.3. Special operators.....	24
2.3.4. Structs usage in tests	25
2.4. Running tests.....	27
2.4.1. Charts	30
2.4.2. MC/DC coverage	31
2.5. Importing/exporting tests.....	36
2.6. Modifying SimuDLL project.....	36
Chapter 3. Additional features of CPP Tests	38
3.1. ATS Reports.....	39
Chapter 4. Code Generator	40
4.1. Scripts Control	40
4.2. Files Control.....	41
List of Figures	44

Chapter 1. Getting started

In this chapter, you will be introduced with activating the license, creating new project and opening the existing one (also from a different device), as well as saving it.

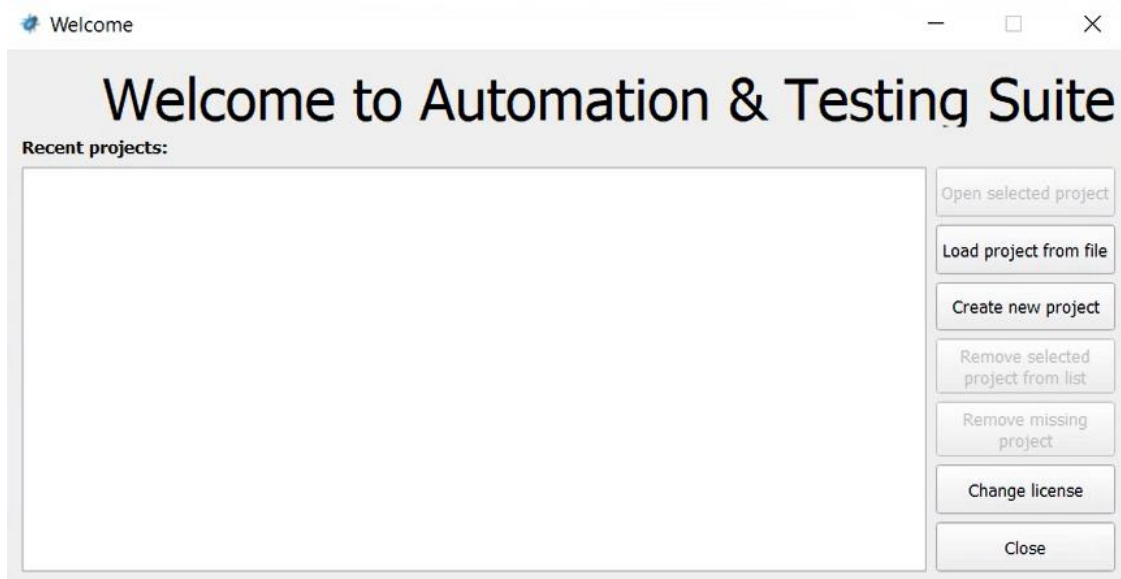
First, run the application. On your screen, the License Tool window will appear (Figure 1). You can import license by inputting the activation key (section on the right side) or by using remote license parameters (on the left). After successfully activating the license, application needs to be restarted.

Figure 1. License Tools



After restarting the application, you will need to select the particular license, which you want to use. To do that, click it and then just click the button „*Select license*”. Now, on your screen there will be showed a logging view. After you log in, you can go further to Welcome Window (Figure 2).

Figure 2. Welcome Window



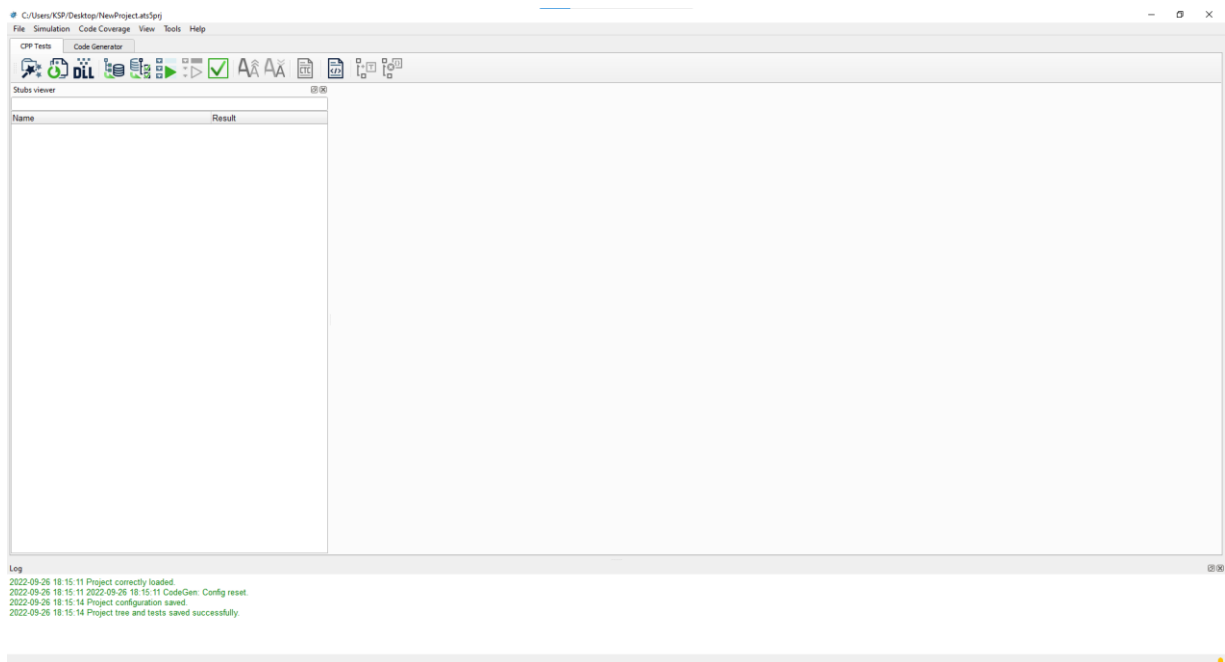
With this window, you are able to:

- open selected project,
- load project from file – it allows to open a project by manual selecting a particular *.ats5prj* file,
- create new project,
- remove selected project from list,
- remove missing project – it removes a project from ATS recent projects list, that is not existing anymore on your computer (for example a project that has been deleted),
- change license – it shows the License Tools
- close application.

1.1. Project creation

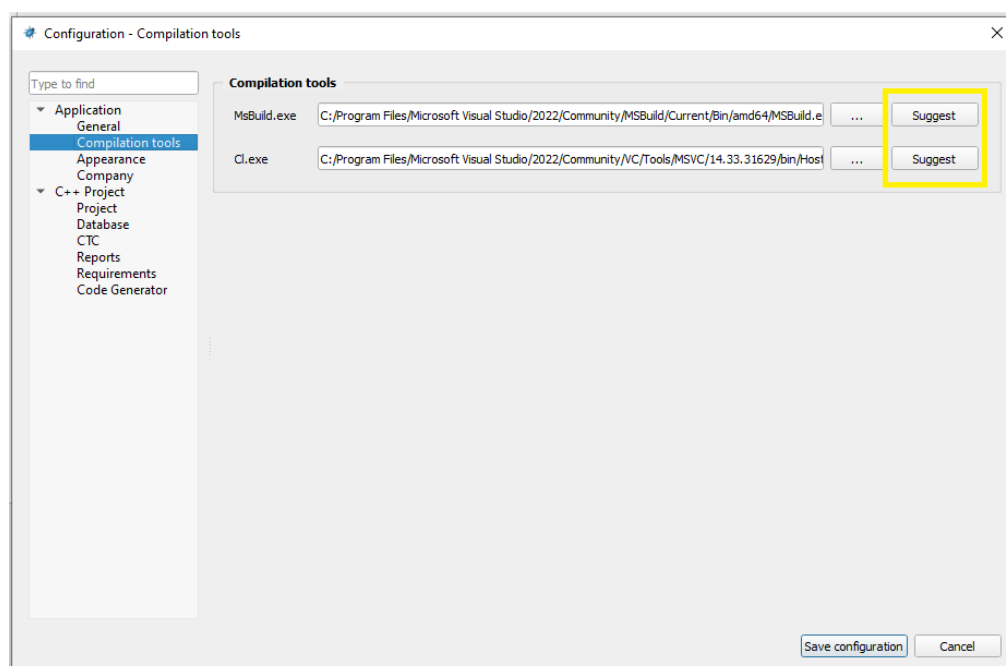
In the first step, select a place for your new project and type in the name. It will be saved as *.ats5prj* file. After creating a project, this is how main view of the application looks like (Figure 3).

Figure 3. Main View of ATS5.



Before you start having the files analyzed by ATS, please make sure, that you have set a path to MSBuild and Cl.exe. To check this, go to *Tools – Configuration – Compilation tools*, as showed on Figure 4. If the fields are empty, use *Suggest* button to set them automatically. In case it does not happen automatically, you will have to set it manually (choose a particular path to those components or install them, if you have not done it yet).

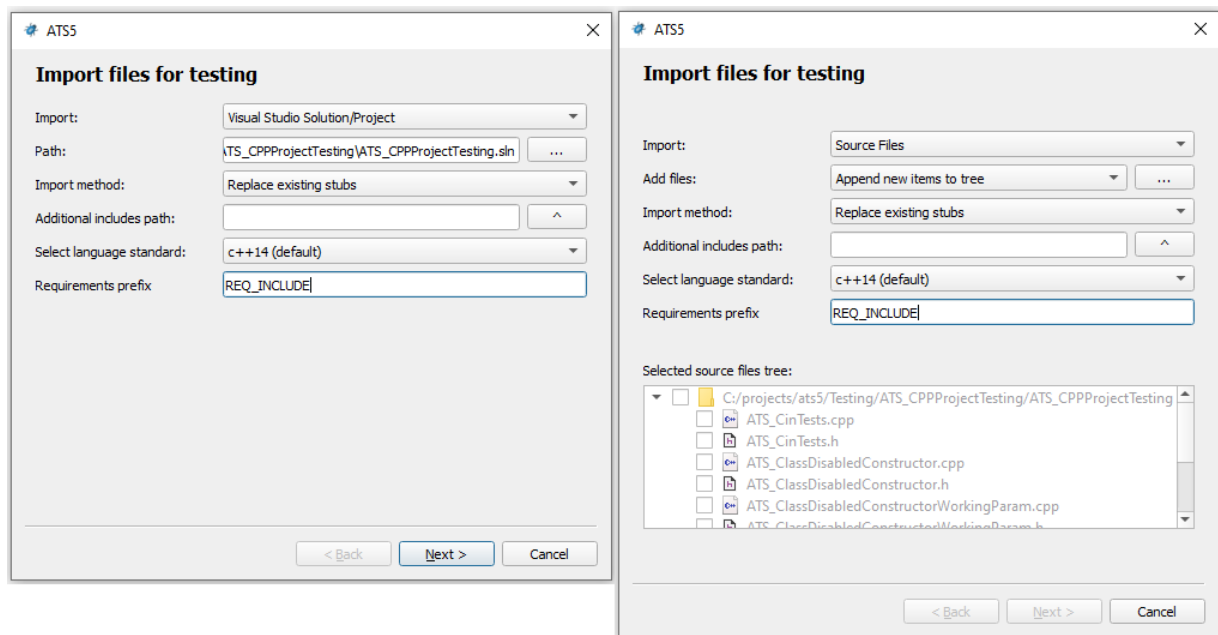
Figure 4. Compilation Tools.



Now, you can start analysing source files and creating tests. To choose files for analysis, click the first left button on the Toolbar (or use CTRL + W keyboard shortcut):



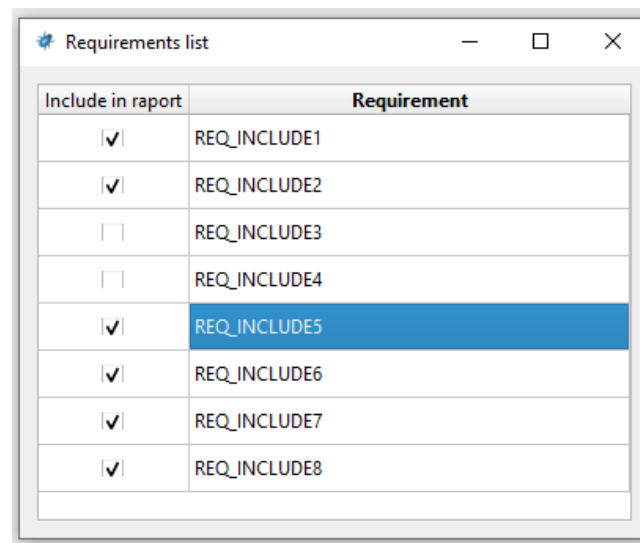
Figure 5. Importing files to CPP Tests.



It will display a dialog window with such features, as showed on Figure 5. In here, you can select:

- a way of importing files (by *Visual Studio Solution/Project*, by *Project Root Folder* or by *Source files*),
- importing method (*Replace existing stubs* or *Append to existing stubs*),
- language standard,
- and also specify requirements prefix. Those requirements are recognized from comments in loaded files and added to the list of requirements (Fig. 6).

Figure 6. Requirements list.

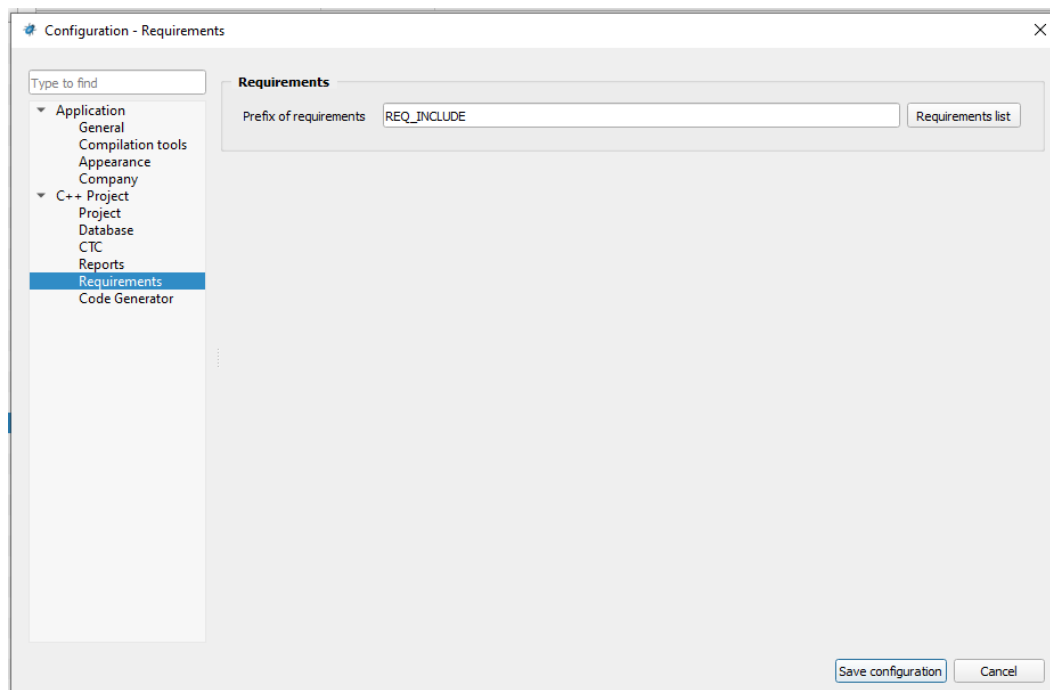


The image shows a dialog box titled "Requirements list" with a table containing 8 rows. Each row has a checkbox in the "Include in report" column and a requirement ID in the "Requirement" column. The 5th row, "REQ_INCLUDE5", is highlighted in blue.

Include in report	Requirement
<input checked="" type="checkbox"/>	REQ_INCLUDE1
<input checked="" type="checkbox"/>	REQ_INCLUDE2
<input type="checkbox"/>	REQ_INCLUDE3
<input type="checkbox"/>	REQ_INCLUDE4
<input checked="" type="checkbox"/>	REQ_INCLUDE5
<input checked="" type="checkbox"/>	REQ_INCLUDE6
<input checked="" type="checkbox"/>	REQ_INCLUDE7
<input checked="" type="checkbox"/>	REQ_INCLUDE8

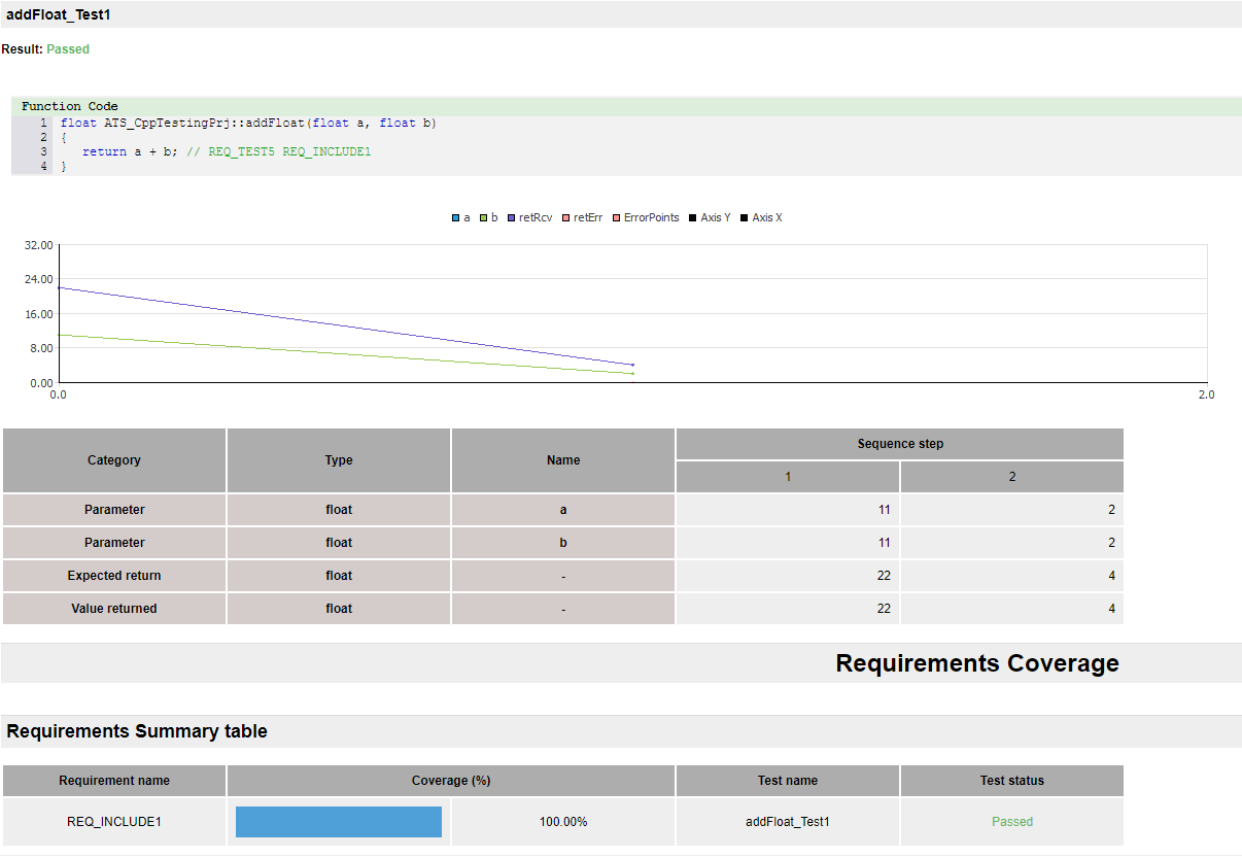
Requirements can be also configured in *Tools - Configuration - Requirements* (Fig. 7), where the prefix can be change or user can select/deselect many requirements to add them (or not) to the analysis.

Figure 7. Requirements in Configuration.



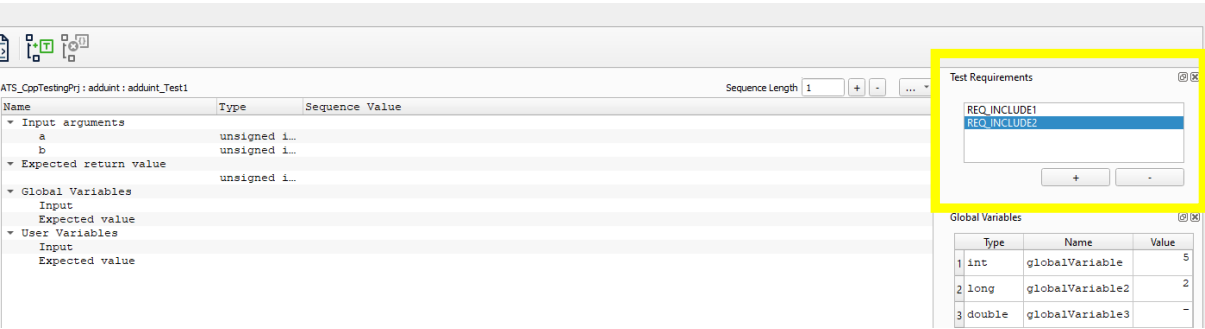
If there are requirements in the files and the prefix had been set, they will be presented in Test Report in form of the table (Fig. 8).

Figure 8. Requirements Summary table.



Requirements can also be added (with button “+”) or removed (with button “-“) for a specific test (Fig. 9).

Figure 9. Tools for adding/removing requirements.



Removing causes that the given requirement will not be displayed in Test Report and will be omitted in analysis. Adding is available when user want to add existing requirement, which was removed or omitted.

Going back to the dialog of importing files to analyse, in here you can also set the path for selected files (if the import way is Visual Studio Solution/Project or Project Root Folder) or select a method for adding files (if the import way is Source Files). The options for that last case are *Append new items to tree*, *Override all items in the tree*.

Warning: since now, you are only able to parse files that are using basic variable types. Any other types will cause and display errors.

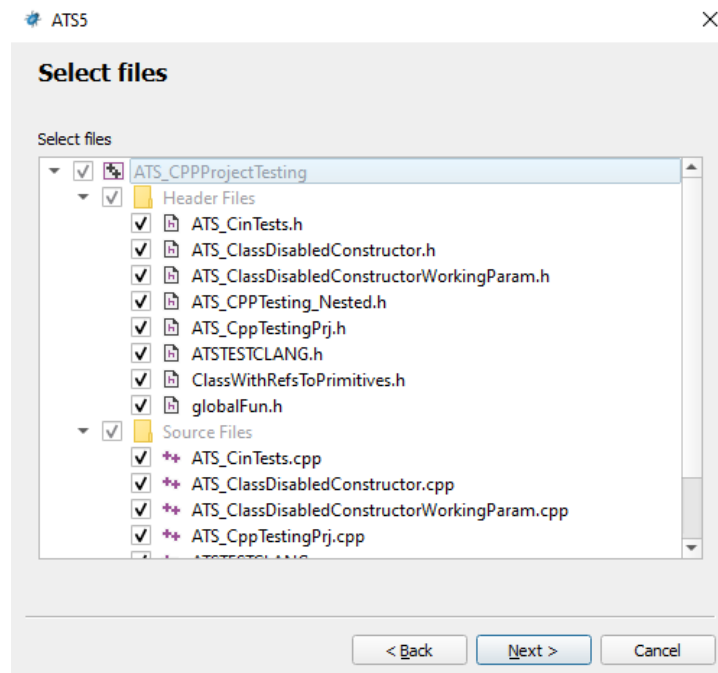
To describe and clarify the ways of importing files, please get familiar with this information:

- Visual Studio Solution/Project – it allows to choose *.sln* or *.vcxproj* files, so you can display files that are included in it.
- Project Root Folder – it allows to choose root folder from which files and subfolders will be displayed for further analysis.
- Source Files – it allows to add source files which a user wants to have displayed in tree section (right side of Figure 5). Adding source files is available multiple times when „*Add files*” option is selected.

Besides that, application allows to set additional includes path – it can be done in two ways. The first method is to simply click the button on the right side of the field and type in the paths, which you need. The second method is to click the „...” button and select the output folders manually. By setting additional includes path, you can specify paths to folders with files that are needed to be included in analysis, and that are placed outside the project.

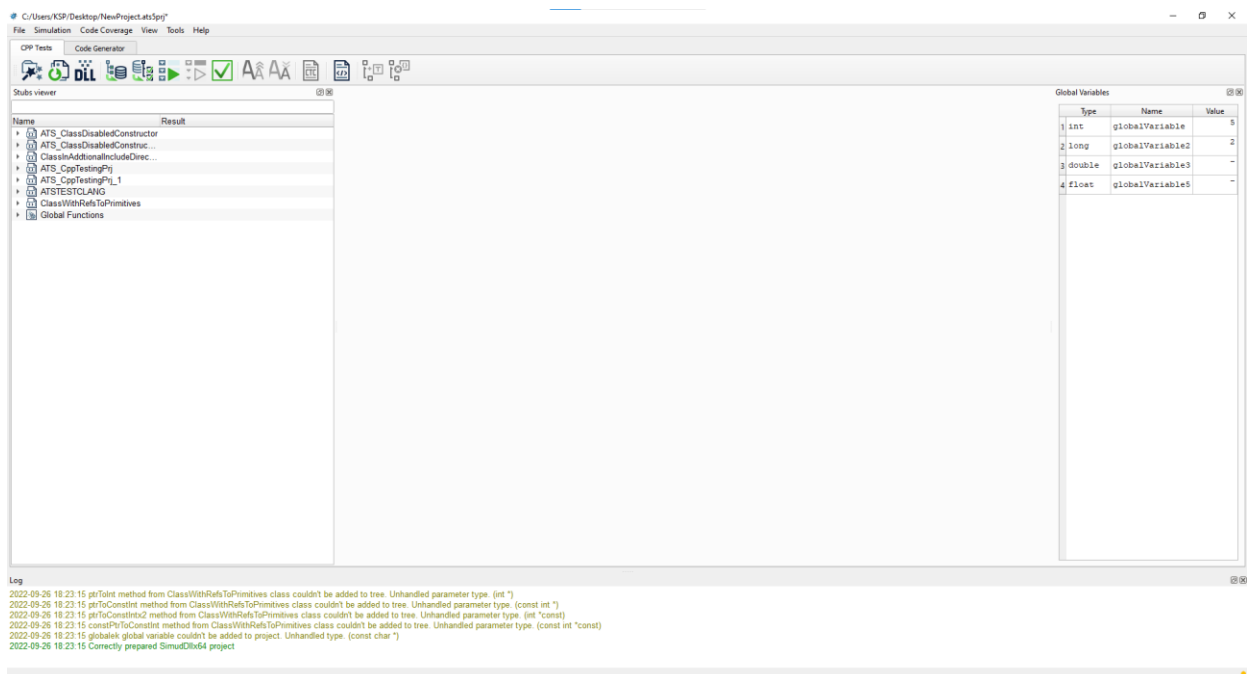
By going „*Next*”, the application would show a selection section (Figure 10). In here please choose files, using checkboxes, that you would like to have in your project.

Figure 10. Selection section in CPP Tests.



The last step is to confirm all selected files. Click „*Finish*” to finalize the process of importing files and to display them in a main view (Figure 11).

Figure 11. Main View of ATSS5 with imported project.

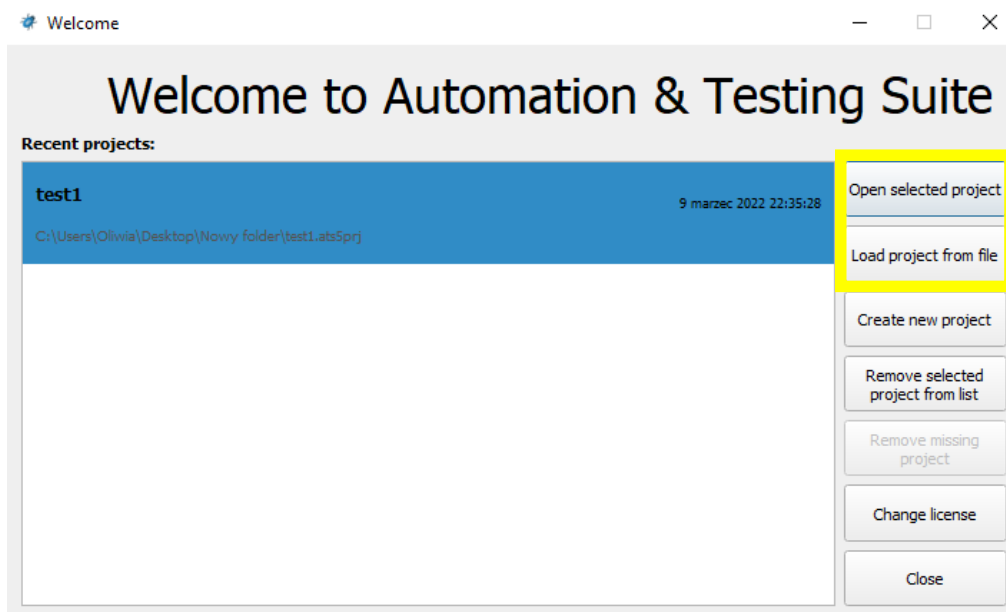


1.2. Opening existing project

If you already have created a project and now you would like to open it, you can do that by:

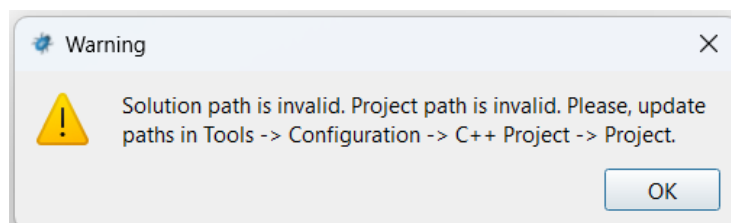
- opening selected project from Recent projects list (Figure 12), or
- loading a project from a file.

Figure 12. Recent projects list in Welcome Window



In case of opening existing ATS project from different device, you will need to adjust some paths to be able to use such project. Right after opening it, on the screen appears warning box with information which paths needs to be configured (Figure 13).

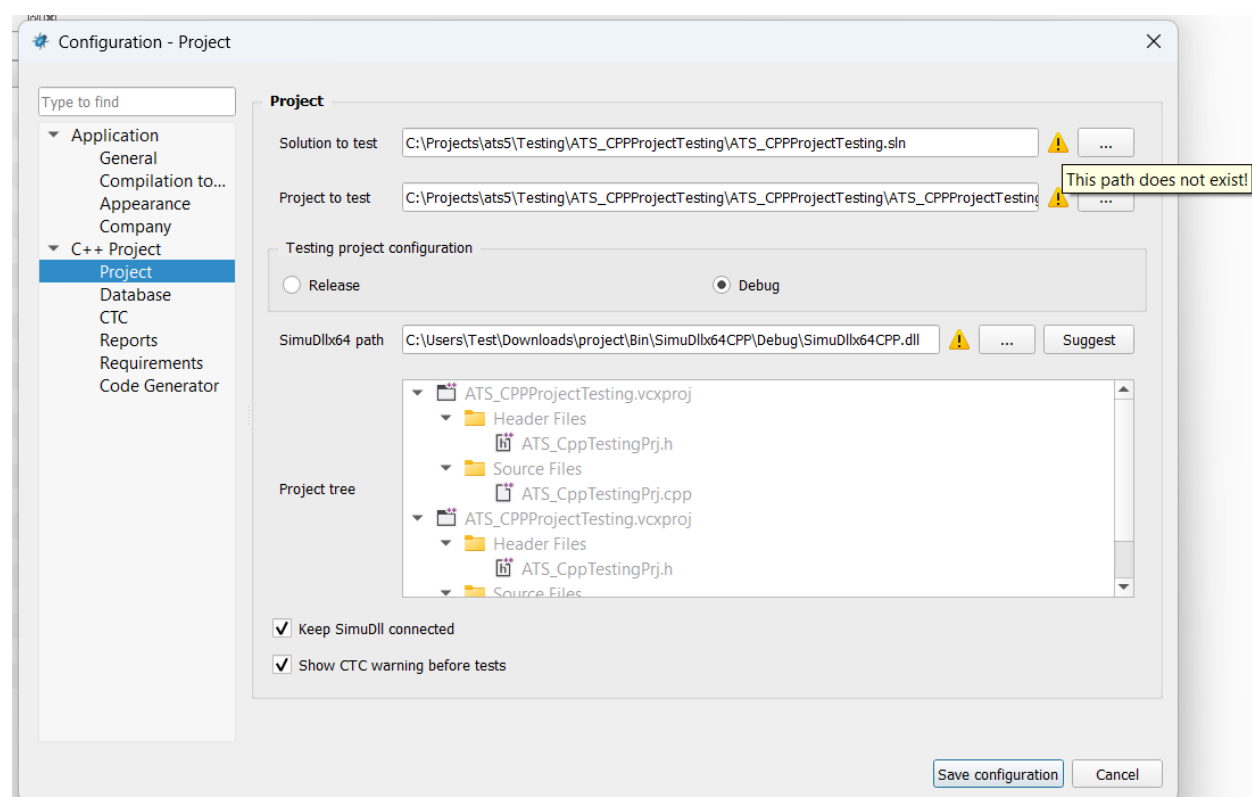
Figure 13. Warning Box - Configure project paths



Following the instructions, if you go to Tools – Configuration – C++ Project – Project section, you will notice some yellow triangles saying „This path

does not exist!”. Those errors occurs because loaded project has different paths set (local paths from different device), so you have to adjust them to be set as yours local paths. For SimuDll path this issue is easy to handle – you can click Suggest button and it will automatically search and set correct path for SimuDll project of this ATS project (Figure 14). However, to set project and solution path you will have to search for correct files manually – press „...” buttons to open browsing dialog and select correct paths.

Figure 14. Configuration - Configure project paths

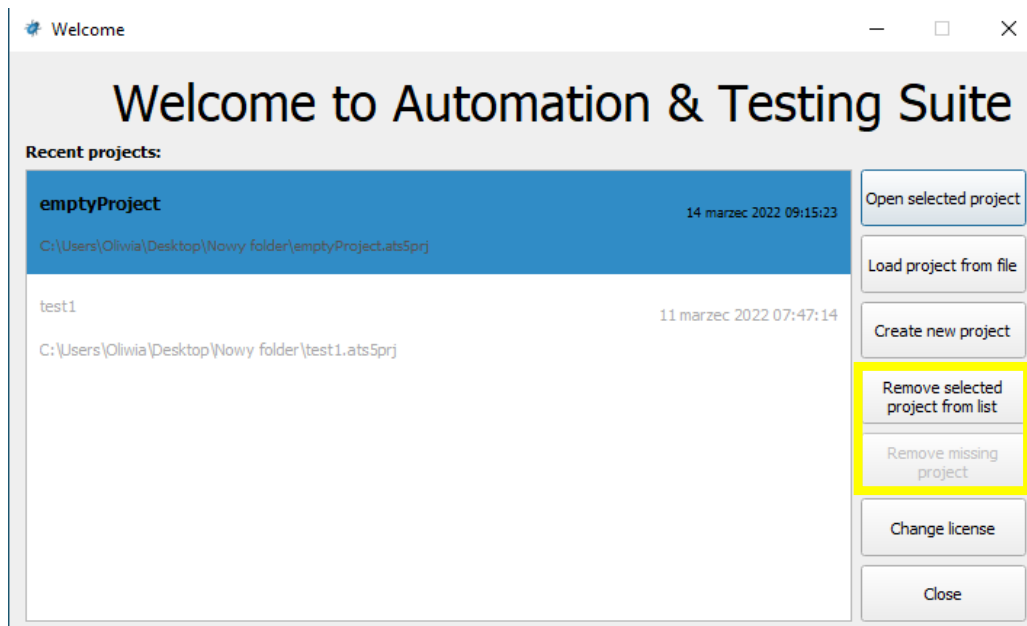


When those paths are fixed, you can save configuration. When you rebuild SimuDll some errors may occur, but those errors must be resolved manually in SimuDll project.

1.3. Removing project

If you have deleted a project, or moved it to other folder, you could see this project as disabled element on the list (Figure 15). To remove that element, simply select this project and then click the button „Remove missing project”.

Figure 15. Remove missing project in Welcome Window.

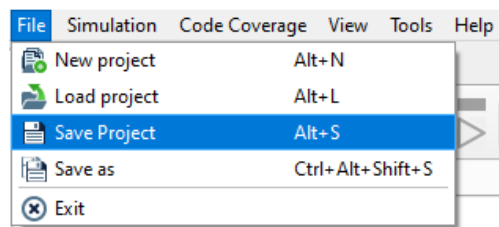


In case you would like to remove a particular project from Recent projects list, you can do this by selecting it and clicking „*Remove selected project from list*”.

1.4. Saving project

To save a project you can go to *File* and select „*Save project*” (if your intent is to overwrite the existing project file) or „*Save as*” to save but simultaneously create new project file. There are also dedicated keyboard shortcuts for both actions.

Figure 16. File – Save options.



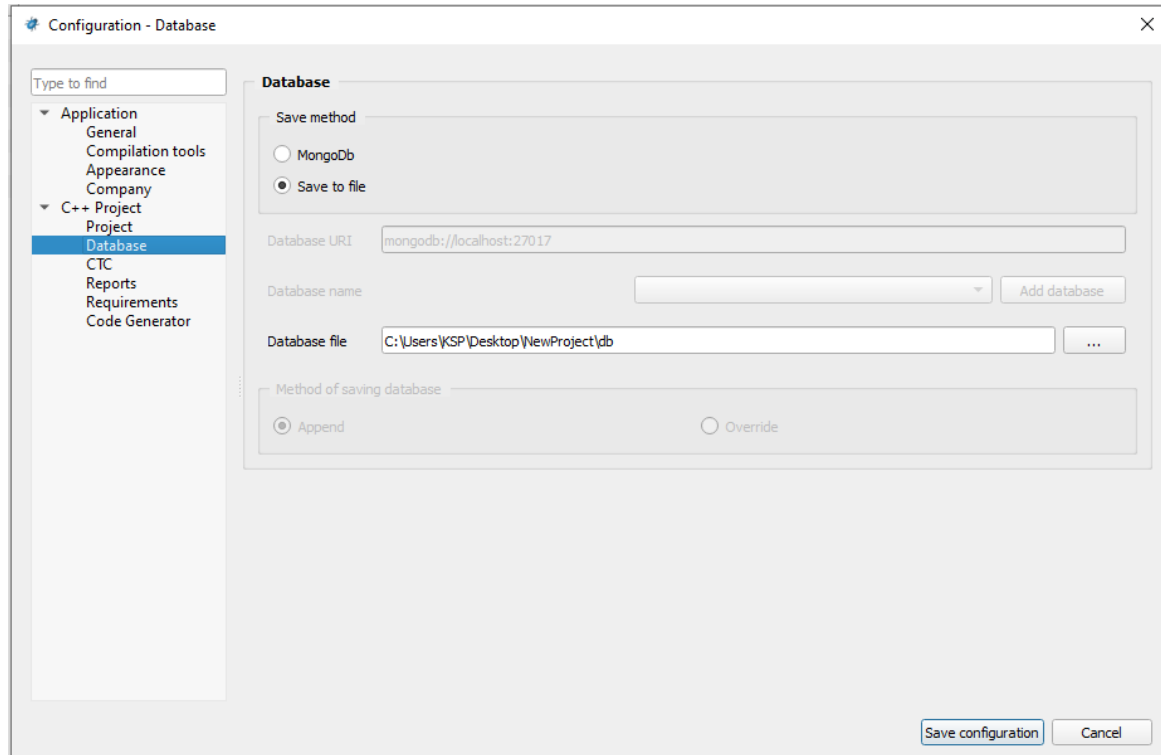
Another way to save project is by using this button from Toolbar:



A user can specify a saving method in *Tools – Configuration – Database*. The options are: saving tests as JSON files or saving them in MongoDB database

(Figure 17). In this second case, it is required to have MongoDB software to save tree in database.

Figure 17. Configuration - Database.



Chapter 2. Testing files

In this chapter, you will get to know how to build DLL, prepare your files for analysis, create tests and how to run them.

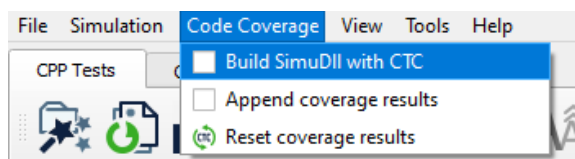
2.1. Building SimuDLL

Now, when you have opened a project or created a new one, there is only one more step to do before testing your files. This step is to build the DLL. It can be done by clicking the third button on the left:



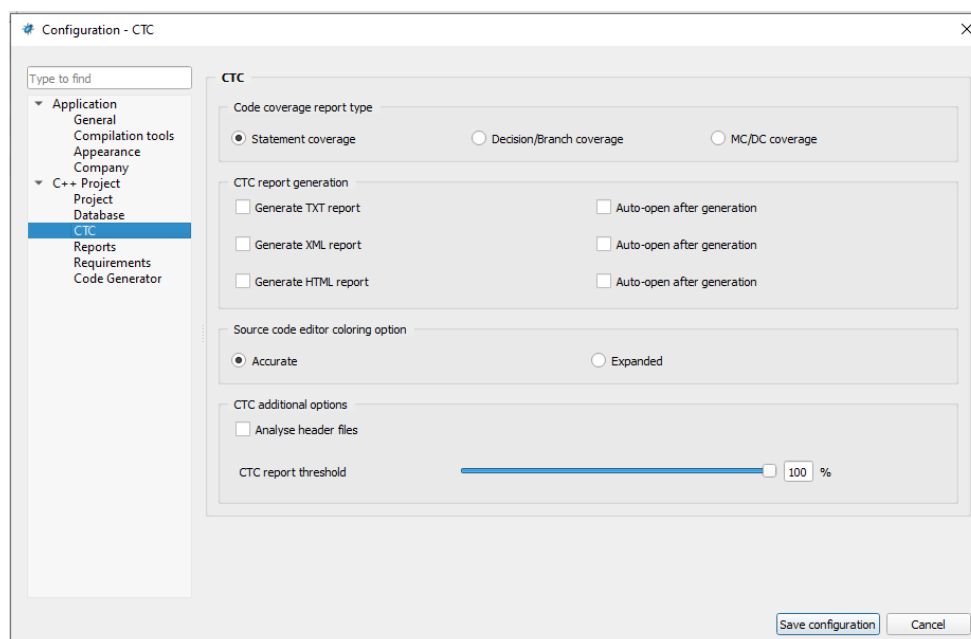
However, before building it, you should decide whether you would like to have it built with CTC enabled or not. If yes, go to the Code Coverage tab and tick the checkbox „Build SimuDLL with CTC” (Figure 18).

Figure 18. Code Coverage tab.



If you decided to build DLL with CTC enabled, you can set CTC options in *Tools - Configuration – CTC* menu (Figure 17).

Figure 19. CTC Tools.



Last important step to take, is to make sure that all constructors and destructors are defined correctly.

Constructor and destructor methods are methods, which are used to create and destroy objects of the class with tests. By default, these methods are defined without any parameters, in the way showed on Figure 20.

Figure 20. Defining constructors with non-params.

```
ATS_CppTestingPrj : Construct
1 ATS_CppTestingPrj::construct()
2 {
3     this->object = new ATS_CppTestingPrj();
4 };
```

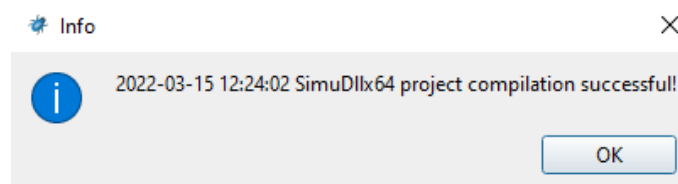
However, in some cases there is a necessity to define them with parameters. In such situations, if the application recognizes it, application will display an information, as shown on Figure 21.

Figure 21. Information while recognizing constructor with parameters.

```
ATS_ClassDisabledConstructor : Construct
1 ATS_ClassDisabledConstructor::construct()
2 {
3     //Public default constructor required to create stub object not available.
4     //this->object = new ATS_ClassDisabledConstructor(/* Required parameters */);
5
6     //Comment error below after implementing class constructor
7     #error ATS5: Public default constructor required to create stub object is not
8     available.
9 };
```

After all is set up, successful building the DLL will display a dialog with confirmation (Figure 22). On the other hand, if something fails you will get errors displayed in a log window at the bottom of application with details – what went wrong.

Figure 22. Successfully built DLL notification.



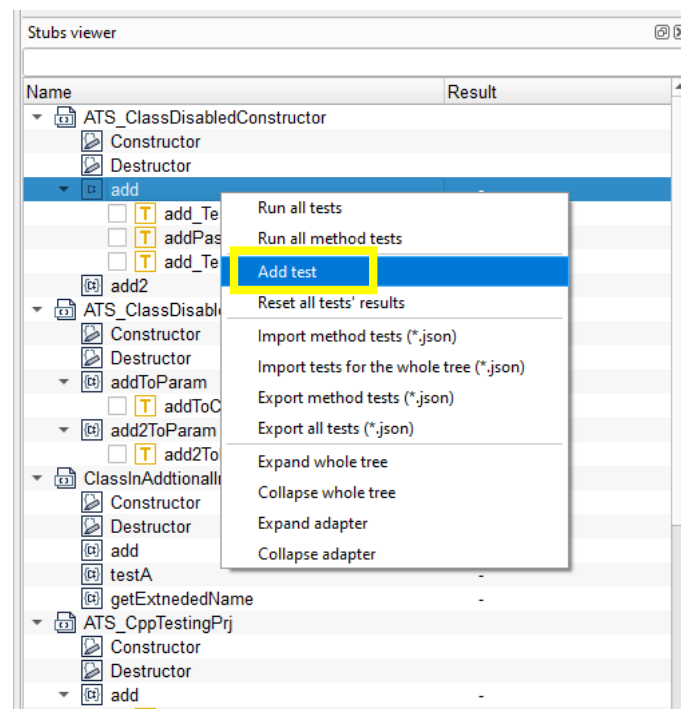
After choosing files to analyze and compile the DLL now you are ready to test them.

2.2. Adding new test

Adding test to adapters (tree items named as class methods or global functions) is possible in three ways.

First one is to simply double-click on adapter (this option is available only when adapter does not contain any tests yet). Second one is to use context menu on adapter by pressing right mouse button on it (Figure 23).

Figure 23. Adding tests via context menu.

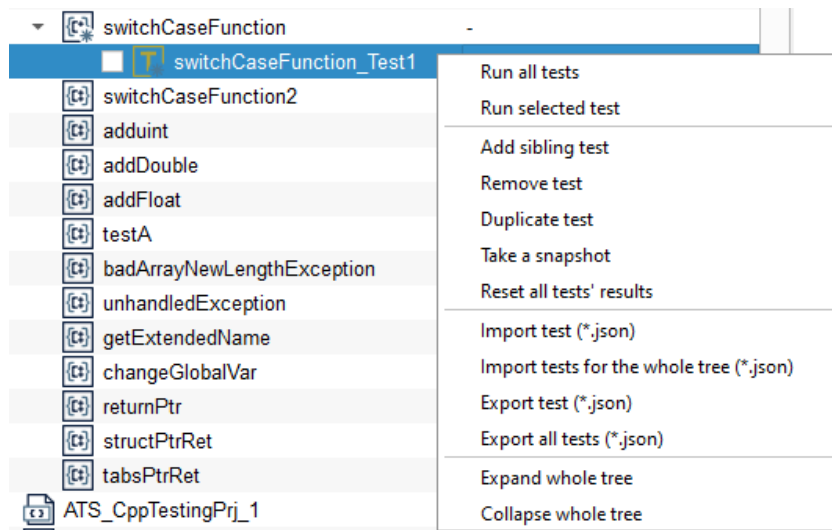


And the last, third option is to use the second button from the right side of a Toolbar „Add a new test to the method”:



To create a test with above button, you have to select a target method first – it will be added directly for this method.

Figure 24. Additional options for test in Stubs Viewer

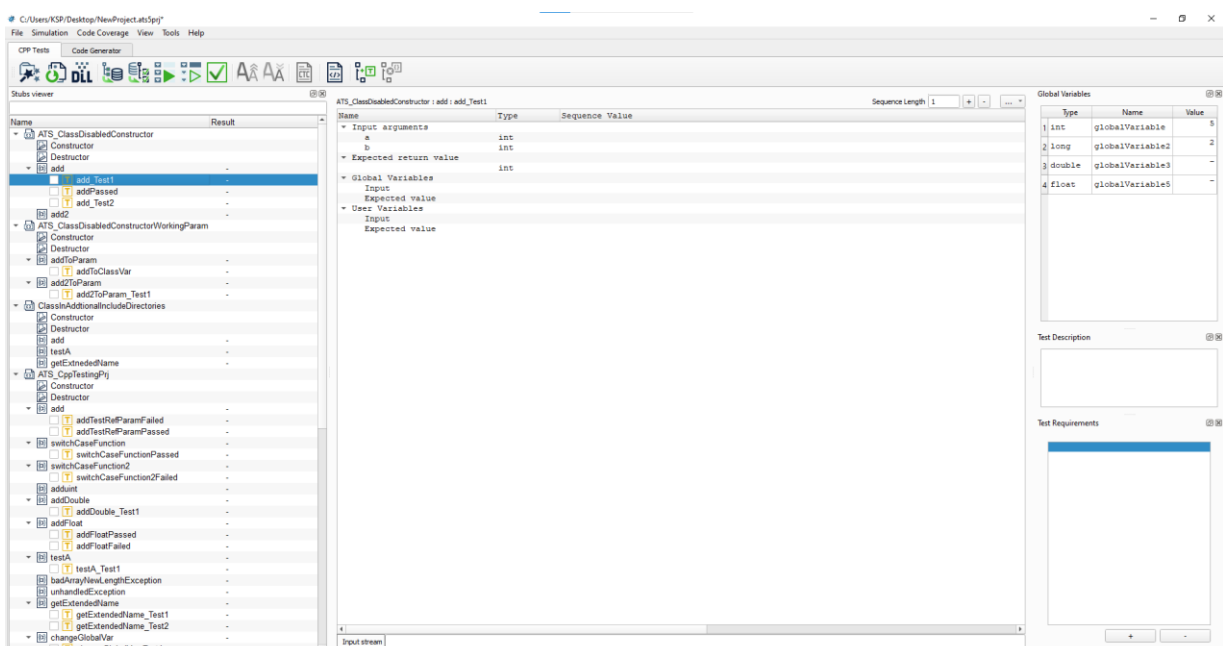


Application allows you to rename test by double-clicking on it. Also, there is a possibility to remove test, duplicate it, add sibling test, take a snapshot of it, run it and reset its results (Figure 24).

2.3. Modifying a test

Clicking on test (tree item) shows a new window, that allows user to specify values for input arguments of methods/functions as well as expected return values (Figure 25).

Figure 25. Main View of ATS5 with added tests.



Application allows user to input only parameters that are used in a specific method/function. For example, for *switchCaseFunction* method, which returns integer and its parameters could be also only integer numbers, there will be error (marked as red background), if user tries to input other data types (Figure 26).

Figure 26. Setting wrong data type for a test parameter

ATS_CppTestingPrj : switchCaseFunction : switchCaseFunction_Test1

Name	Type	Sequence Value
▼ Input arguments		
a	int	22
b	int	1.2
c	int	'd'
▼ Expected return value		
	int	[1,3]
▼ Global Variables		
▼ Input		
globalVariable	int	4
▼ Expected value		
globalVariable	int	
▼ User Variables		
Input		
Expected		
globalVariable2		
globalVariable3		
globalVariable5		
globalek		
globalStruct		
globalStructe		
globalStr2		
globalTabs		
globalUserStruct		

Global variables can be added by selecting them from the expanding list (Figure above). You can select many different global variables in single test but once used global variable in input argument or expected value cannot be duplicated.

To use a user variable, firstly you need to create it in the view of class definition or while having selected any method/test of the class which you would like to create user variable for. By default, user variables creation section is located on the right side of ATS application just under Test Description (Figure 27). It is a docking widget so you can always undock this and place anywhere else to let it be more comfortable to use.

Figure 27. Add user variable section

The screenshot shows a window with two main sections: 'Global Variables' and 'User Variables'.

Global Variables:

Type	Name
int	globalVariable
long	globalVariable2

Test Description:

User Variables:

Name	Type	Value
myName	int	4
myStruct	UserStruct	{4,3,2}
x	int	4
b	float	3.2
myStruct2	Tab	{2,2.0,new int[2],nullptr}
x	int	2
b	float	2.0
pointer	int *	new int[2]
nest	UserStruct *	nullptr

Press plus „+” button to create new user variable and specify name, type and value for it, then push Enter to confirm your inputs. Removing already created user variable is done after you select it from the list and then clicking minus „-” button. SimuDll needs to be rebuild to allow you to use such created user variables and select them from combobox placed in the test definition section (Figure 28).

Figure 28. User variables usage in test

The screenshot shows a window with a 'User Variables' section. On the left, there is a tree view showing the test structure. On the right, there is a 'User Variables' table.

User Variables Table:

Name	Type	Value
myName	int	4
myStruct	UserStruct	{4,3,2}
x	int	4
b	float	3.2
myStruct2	Tab	{2,2.0,new int[2],nullptr}
x	int	2
b	float	2.0
pointer	int *	new int[2]
nest	UserStruct *	nullptr

A dropdown menu is open, showing the following options:

- myName
- myStruct
- myStruct2

Similarly to global variables, you can use multiple user variables in single test, but they cannot be duplicated.

In tests, where a parameter can be a reference (e.g. `int &`), application allows you to use only user (local) variables or global variables (Figure 29).

Figure 29. Reference to global variable in test's parameter

ClassWithRefsToPrimitives : refToInt : refToInt_Test1

Name	Type	Sequence Value
▼ Input arguments		
ref	int &	globalVariable
▼ Expected return value		
	int	4

Global Variables

Type	Name	Value
int	globalVariable	5
long	globalVariable2	2

If you declare user variable as a pointer, for example in this way: *(type)* `int* (name) ptr (value) nullptr`, it can be then used in test's parameters like this:

Figure 30. User variable as a pointer used in test's parameter

ATS_CppTestingPrj : add : add_Test1

Name	Type	Sequence Value
▼ Input arguments		
a	int &	*myIntPtr
b	int	2
c	int	3
▼ Expected return value		
	int	4

There is also a possibility to use pointers in arguments that are not using references. For example, you can define user variable as a pointer to integer and then use it as a parameter in argument of `int` type – simply use `*userVar` or `userVar[0]`.

On the other hand, if you want to set value or set expected value of user variable which is pointer type, you can only do that by typing „*” before variable's name or array index after name like `ptrVar[0]` (Figure 31).

Figure 31. Setting or getting pointer user variable

▼ User Variables		
▼ Input		
ptr[0]	int*	1
▼ Expected value		
ptr	int	1

2.3.1. Sequences

A particular test can be run in sequences. To add new sequence, click the „+” button on the right side of the fields with test params (Figure 32). Also, you can modify the amount of sequences by putting its length number in the textfield.

Figure 32. Test sequences

ATS_CppTestingPrj : switchCaseFunction : switchCaseFunction_Test8 Sequence Length + - ...

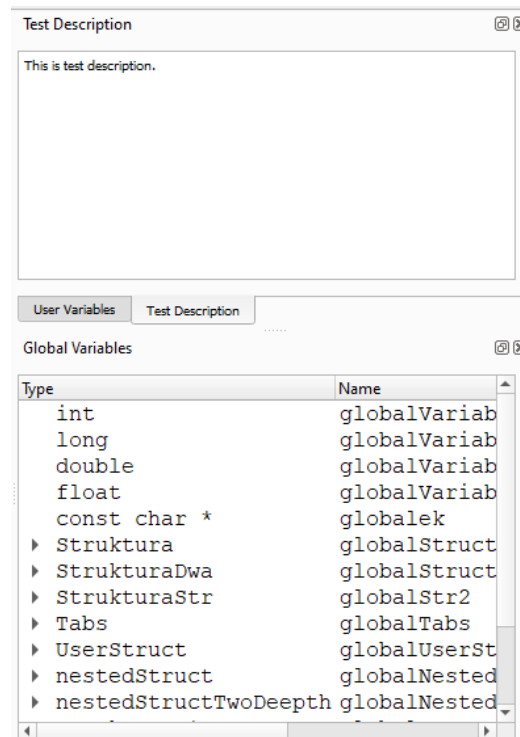
Name	Type	S1	S2	S3	S4
▼ Input arguments					
a	int	2			
b	int	421		34	
c	int	33			
▼ Expected return value	int	22			
▼ Global Variables					
▼ Input					
globalVariable	int	6			
▼ Expected value					
globalVariable	int	7			
▼ User Variables					
▼ Input					
▼ uu	Struktura	{4,3,2,nullptr}	{,,,}	{,,,}	{,,,}
x	int	4			
b	float	3			
te	unsigned long...	2			
nest	nestedStruct *	nullptr			
▼ Expected value					
▼ uu	Struktura	{1,2,3,*}	{,,,}	{,,,}	{,,,}
x	int	1			
b	float	2			
te	unsigned long...	3			
nest	nestedStruct *	*			

Each column „S1”, „S2” and so on, is a separated sequence. So for first sequence of this test for input arguments were provided values: 3, 421, 33 and for expected return value 22. Due to empty cells in second sequence, all values from previous sequence (S1) will be extended also to second sequence. It means that values for second sequence are exactly the same as for S1. You can notice that value for b parameter has changed in third sequence to 34. So all other empty cells will automatically expand values from previous sequence besides value for b parameter. About expected return value, in above case it will always be equal to 22 in each sequence.

To sum it up, if you do not define parameters in the following sequences, they will be automatically set as values of earlier defined parameters. Sequences work the same also for global and user variables.

In test, you can also add description, which will be displayed in generated reports (Figure 33).

Figure 33. Test description



By default, this section is placed on the right side of ATS application under Global Variables section, but it is a dockable widget so you can always dock it anywhere else.

2.3.2. Range values

ATS5 allows users to create tests with range values in parameters. Range is specified as [min, max, step]. To use a range, you have to put your values between square brackets „[” and „]”, with comas as separator for min and max value and a step (which is optional, by default it will be set to 1). Ranges are presented on Figure 34. Important information about ranges is that they differ for return values. In such case, you can only specify [min, max] params (without step). It means that return values specified in a range (e.g. [1,50]), will take every

value from that range as positively passed in a test (see expected return value in sequence S2 of below example).

Figure 34. Ranges in tests' parameters

ATS_CppTestingPrj : switchCaseFunction : switchCaseFunction_Test9

Name	Type	S1	S2
▼ Input arguments			
a	int	66	3
b	int	[1, 5, 1]	3
c	int	2	4
▼ Expected return value			
	int	<100	[1, 50]

Another example – range specified as [5,10,2] will run test with given values 5, 7, 9 – so there will be created 3 sequences additionally for purpose of this range. If user will provide two ranges in separated parameters within single sequence, application will combine them, using Cartesian product operation. It is also possible to have range with a negative step. This requires putting a bigger value as a minimum parameter than maximum parameter (e.g. [15, 2, -3] or [-5, -1, 1]). As shown on Figure 34, after execution of this test, its result will passed (the received return value is 2 for S1 and 4 for S2, so it passes both conditions).

2.3.3. Special operators

Moreover, application allows you to use special operators for specifying return value. Those operators are:

- „<” - values less than;
- „>” - values bigger than;
- „<=” - values less and equal to;
- „>=” - values bigger and equal to;
- „!” - negation (it means that user can expect every value except the ones given in return range if exclamation mark was added);
- „*” - all values are correct.

Figure 35. Special operators in ranges

ATS_CppTestingPrj : switchCaseFunction : switchCaseFunction_Test10					Sequence Length
Name	Type	S1	S2	S3	S4
▼ Input arguments					
a	int	3	3		
b	int	4	-2		[-3,10,3]
c	int	33	[1,10,2]		
▼ Expected return value					
	int	!3	>0	*	<=82

Usage of these special operators is presented on Figure 35. It is not allowed to use those operators for input arguments, but for all expected return values (also for global and user variables) it is completely correct.

2.3.4. Structs usage in tests

It is possible to use structs within tests and to define user variables of struct type. In ATS5 such test will be displayed and handled a little bit different than regular test with primitive types (Figure 36).

Figure 36. Empty struct fields

ATS_CppTestingPrj : structFun : structFun_Test1		
Name	Type	S1
▼ Input arguments		
a	int	
▼ b		
x	int	
b	float	
te	unsigned long...	
nest	nestedStruct *	
▼ Expected return value		
	void	

As you can see, the general row of such struct shows what type is this, and after filling out the values in the below cells, this general row will be updated in real-time inside curly brackets { } with each value separated by a comma (Figure 37).

Figure 37. Struct usage in test

ATS_CppTestingPrj : structFun : structFun_Test1		
Name	Type	S1
▼ Input arguments		
a	int	3
▼ b		
x	int	{3,21.3,53,nullptr}
b	float	21.3
te	unsigned long long	53
nest	nestedStruct *	nullptr
▼ Expected return value		
	void	

There are some rules to follow while defining values of struct fields in a test. Firstly, it is forbidden to use ranges as input arguments of struct – it is only possible to use range as expected return value. When a test contains global variable of struct type, it is forbidden to define its field with usage of user variable as its value. But on the other hand, if user variable is used in a test and it is a struct type, it is possible to use global variable as its value (Figure 38).

Figure 38. Struct examples in test

ATS_CppTestingPrj : structFun : structFun_Test1		
Name	Type	S1
▼ Input arguments		
a	int	3
▼ b	Struktura	{3,21.3,53,nullptr}
x	int	3
b	float	21.3
te	unsigned long long	53
nest	nestedStruct *	nullptr
▼ Expected return value	void	
▼ Global Variables		
▼ Input		
▼ globalUserStruct	UserStruct	{5,0.5}
x	int	5
b	float	0.5
▼ Expected value		
▼ globalNestedStruct	nestedStruct	{{'h'},34}
▼ nestnest	nestedStructTwoD...	{'h'}
x	char	'h'
x	int	34
▼ User Variables		
▼ Input		
▼ uu	Struktura	{4,5,44,nullptr}
x	int	4
b	float	5
te	unsigned long long	44
nest	nestedStruct *	nullptr
▼ Expected value		
▼ ww	Struktura	{31,2,2,globalNestedStructPtr}
x	int	31
b	float	2
te	unsigned long long	2
nest	nestedStruct *	globalNestedStructPtr

Additionally, user can set a value for current struct using option from context menu „Set variable” (Figure 39).

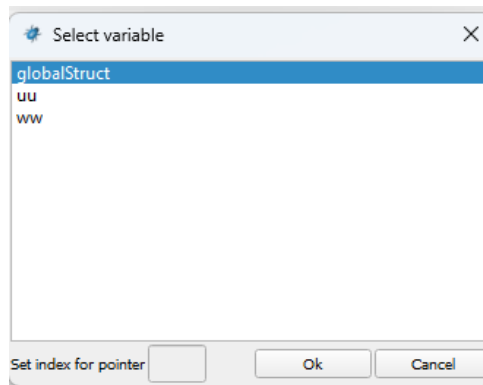
Figure 39. Set struct variable

ATS_CppTestingPrj : structFun : structFun_Test1			Sequ
Name	Type	S1	
▼ Input arguments			
a	int	3	
▼ b	Struktura	globalStruct	
x	int	-	
b	float	-	
te	unsigned long...	-	
nest	nestedStruct *	-	
▼ Expected return value	void		
▼ Global Variables			
▼ Input			
▼ globalUserStruct	UserStruct	globalUserStruct	
x	int	-	
b	float	-	
▼ Expected value			
▼ globalNestedStruct	nestedStruct	{{'h'},34}	
▼ nestnest	nestedStructT...	{'h'}	
x	char	'h'	
x	int	34	

- Expand until non-empty sequence
- Expand all empty sequences
- Expand and replace all sequences
- Copy
- Paste
- Set Variable
- Restore default value

Set variable option will open a new dialog with list of all global and user variables of the same type as currently selected struct (Figure 40).

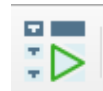
Figure 40. Set Variable for struct



In case of pointers, it is possible to set an index. When variable is selected and confirmed, the fields with values cannot be modified (Figure 39) – they contain „-“ symbol. The only option to change it, is to restore the value by selecting option from context menu „Restore default value”.

2.4. Running tests

After filling in all params that you need for your tests, now you can run them. To start one selected test – click the button in Toolbar:



If your mouse's focus will be set to *class* or *adapter*, clicking *Run Selected Test* will cause running all tests from the selected class/adapter. Running selected test is also possible using context menu, after right-clicking tree item in the Stubs Viewer. If you would like to run all created tests, simply use button or again – use a context menu.



To select many tests from different classes and to make them execute, user can select particular checkboxes and then use the button to run them:

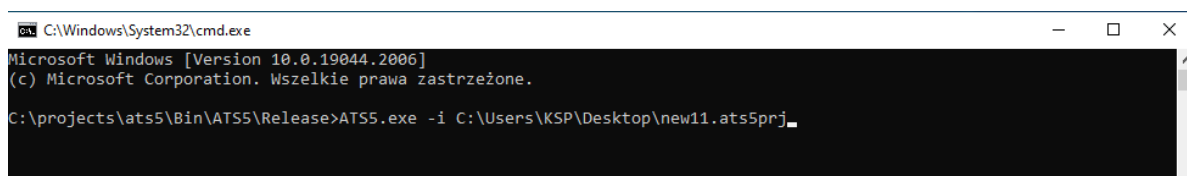


There is also a possibility to execute all created tests with automatic mode from command line. To have it done, open a command line from the folder with *ATS5.exe* file. Then, type in the following instructions (Fig. 41):

ATS5.exe -i *PATH*

PATH is a path to your created *.ats5prj* file which includes tests, that user want to execute. After running above command, ATS5 automatically generates HTML reports for done tests.

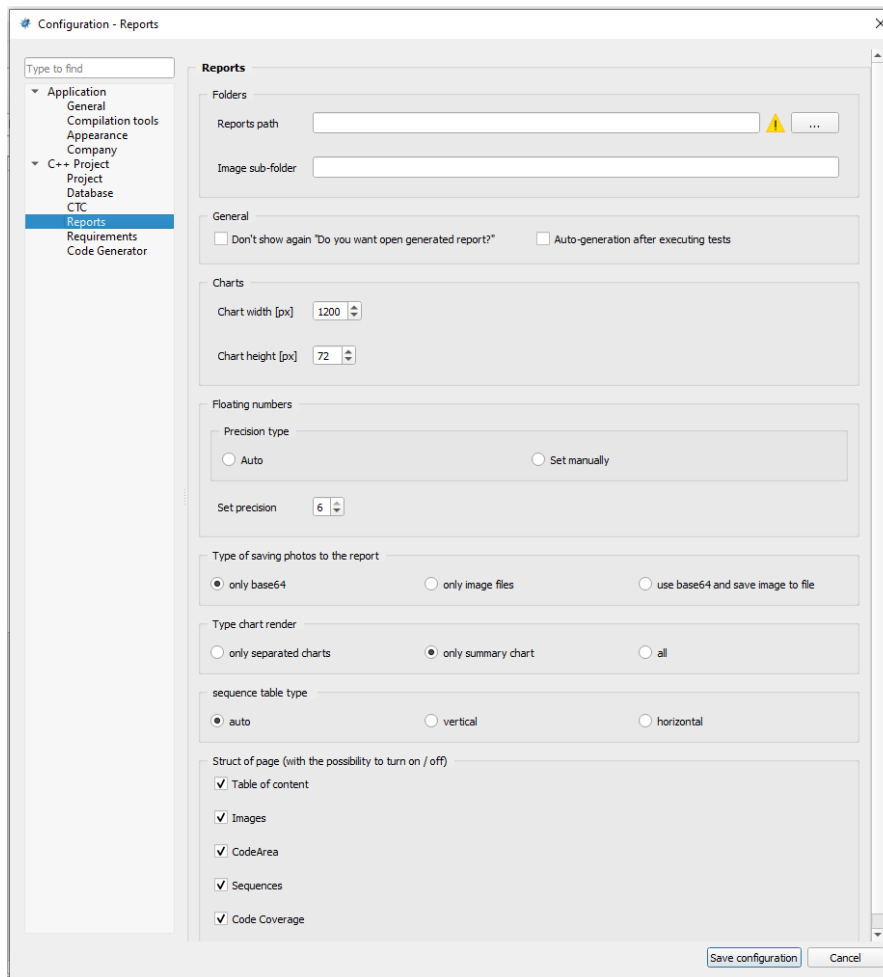
Figure 41. Automatic mode for running tests



After running tests with methods described above, an informational dialog will appear. It includes such information as: numbers of tests done correctly and incorrectly, name of executed test, status of the test result (Passed/Failed), time in which the test was performed. In case that some test is not executable (for example due to incorrect data types in params), this dialog will also include that information. Also, status of executed test is shown in column „*Result*” in the Stubs Viewer tree.

A view with the results of executed tests could be different – it depends on configurations that were set in Tools. Settings concerning generating reports can be checked in *Tools – Configuration – Reports* (Figure 42).

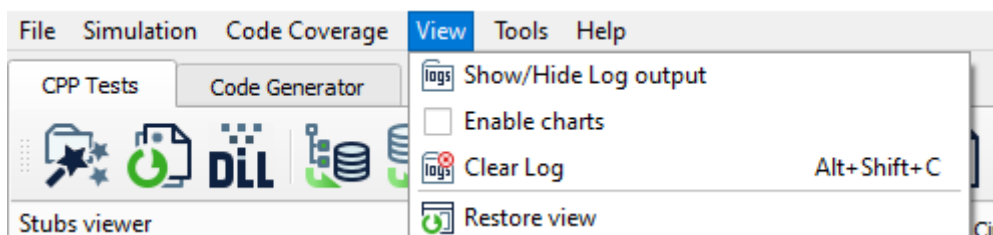
Figure 42. Reports Tools



In here, you can choose paths for reports, as well as for other images, and establish where they should be stored. By using checkboxes you can decide whether to auto-generate a report after every test execution or not. There is plenty of settings to choose, that will allow you to individualize ATS5.

If you would like to always show charts after test execution, you will find that option in a tab, called View. There is a checkbox „Enable charts” (Figure 43).

Figure 43. View tab

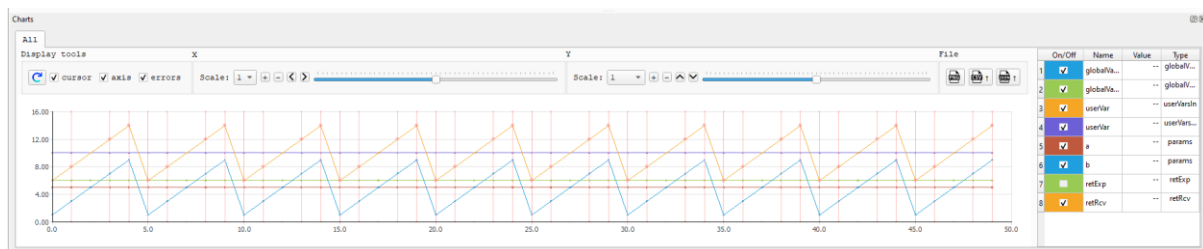


After setting this on, every executed test will automatically show charts with results.

2.4.1. Charts

Charts are presenting test's results – they can be very simple or pretty complicated, depending on given parameters value and number of sequences. Charts consist of input arguments, expected return values for variables and actual received return values.

Figure 44. Charts section



In the middle part of the *Charts* widget, you can find generated chart, tools for manipulating the chart and buttons for exporting the results. On the right side of it, there is a table with parameters' values. These values will be changing in real-time when your mouse will be hovering points on the chart.

The display tools consist of a button for resetting the view and three checkboxes to turn on/off displaying cursor, errors and axis on the chart.

The following two sections concerning axis X and axis Y include tools for changing the scale of displayed chart – you can zoom it in or zoom it out. Also, there is a slider for moving the graph to right or left (for X axis) and to up and down (for Y axis). Moving the chart is also possible without the toolbar - in such case user has to click and hold on the chart and then move the mouse in any direction. In addition, zooming in and out the chart is allowed by using mouse wheel (axis X) and using mouse wheel while holding SHIFT (axis Y).

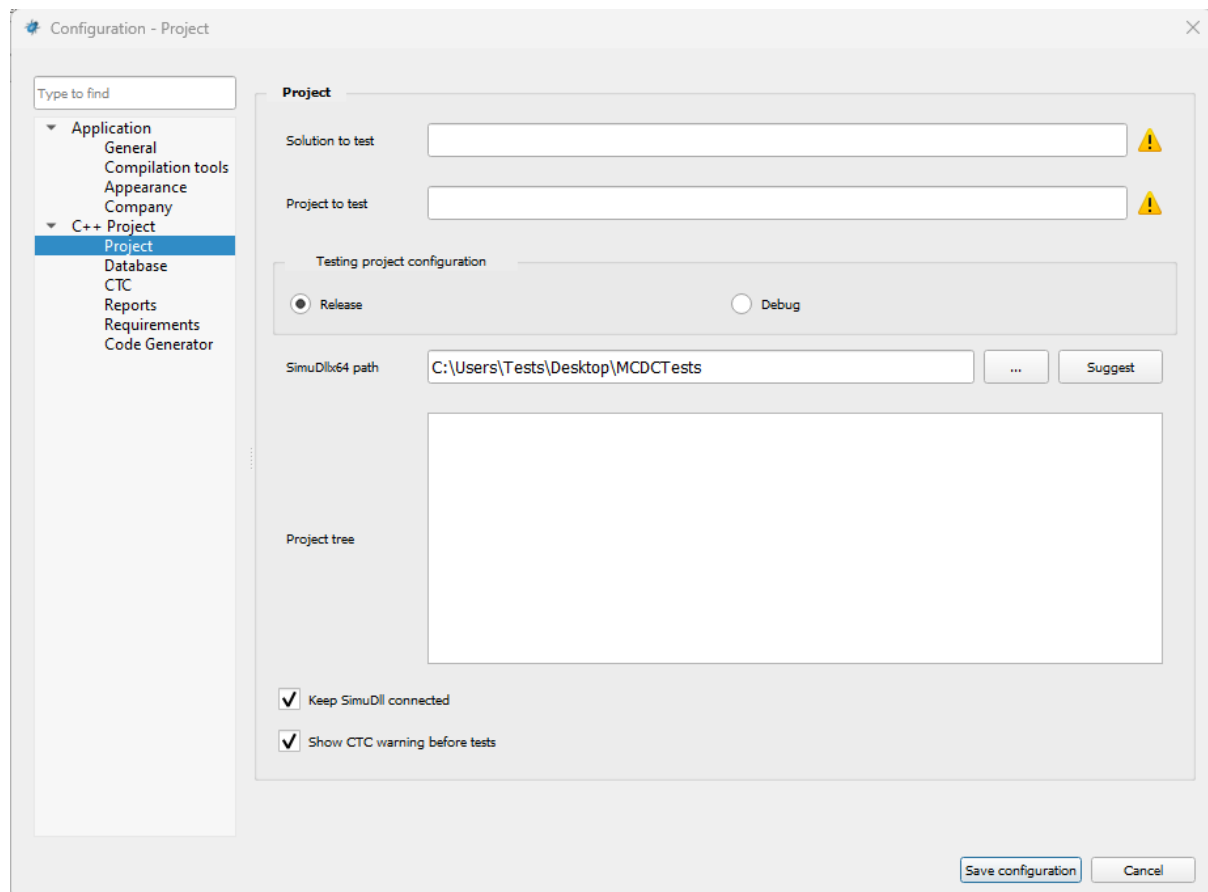
Furthermore, in the section called File, there are three buttons for exporting. The first one is used to generate PNG file with the displayed chart, the second

one is used to export the results to CSV file, and the same happens, when user clicks the last button, with the only difference that the file with results will be in a format of JSON.

2.4.2. MC/DC coverage

As it was mentioned before, settings concerning CTC options can be defined in *Tools – CTC*, but there are also some other important decisions to make, when you would like to generate test report with MC/DC (Modified Condition/Decision Coverage) coverage included. Those decisions can be made in *Tools – Project* section (see Figure 45).

Figure 45. Project configuration with CTC options



The checkboxes in the bottom of the dialog allow you to decide whether you would like to show CTC warning dialogs when executing tests and whether you would like to keep connection with SimuDLL. The last one needs to be turned

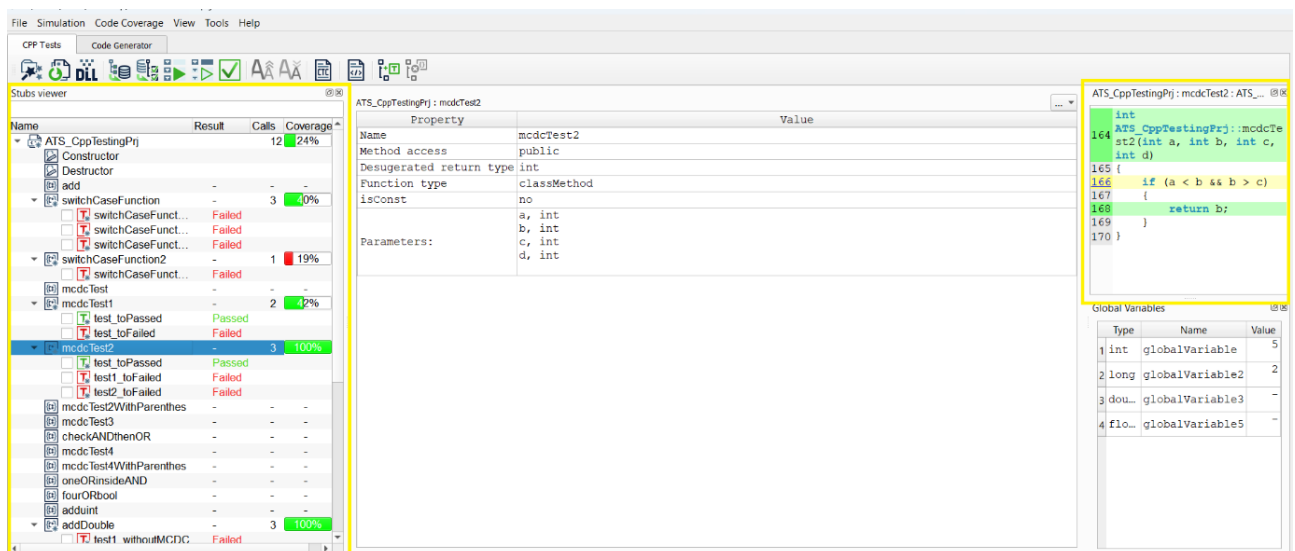
off if user wants to generate CTC report, so if you would have this checkbox set on and execute test, you will get a warning dialog about it.

To enable generating MC/DC report, go to *Tools – CTC* and select „MC/DC coverage” in *Code coverage report type*. Furthermore, select which type of report you would like to generate – TXT, XML or HTML report. Now, if you save your configuration, you are ready to execute tests with MC/DC feature.

In the general overview (Figure 46), when CTC option is set on, there are added some new features, such as:

- In the *Stubs’ viewer* (on the left side of the screen below in the yellow frame) there is added a new column „Coverage” which displays the percentage of code coverage. If it shows 100% it means that all lines of the code have been tested.
- Coloring the lines (Fig. 47) – in the method’s definition on the right side (yellow frame), the colors of the lines have different meanings. Explanation of them will be given below the Figure 46.

Figure 46. Main ATSS window with CTC feature

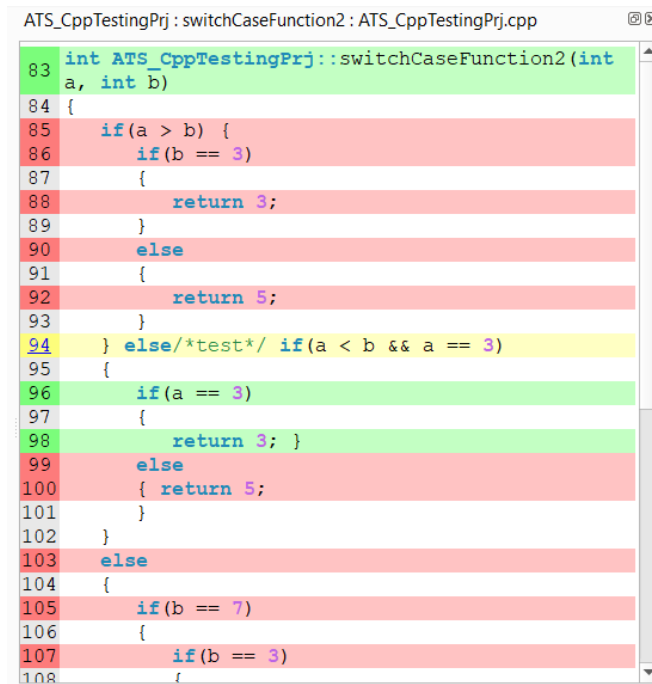


Line which is marked:

- green – means that this line was used and executed during the test,

- yellow – means that there was MC/DC recognized in this line,
- red – means that this line of code has not been executed and used during the test (the conditions were false, so the program did not go inside the lines).

Figure 47. Lines' colors in method definition

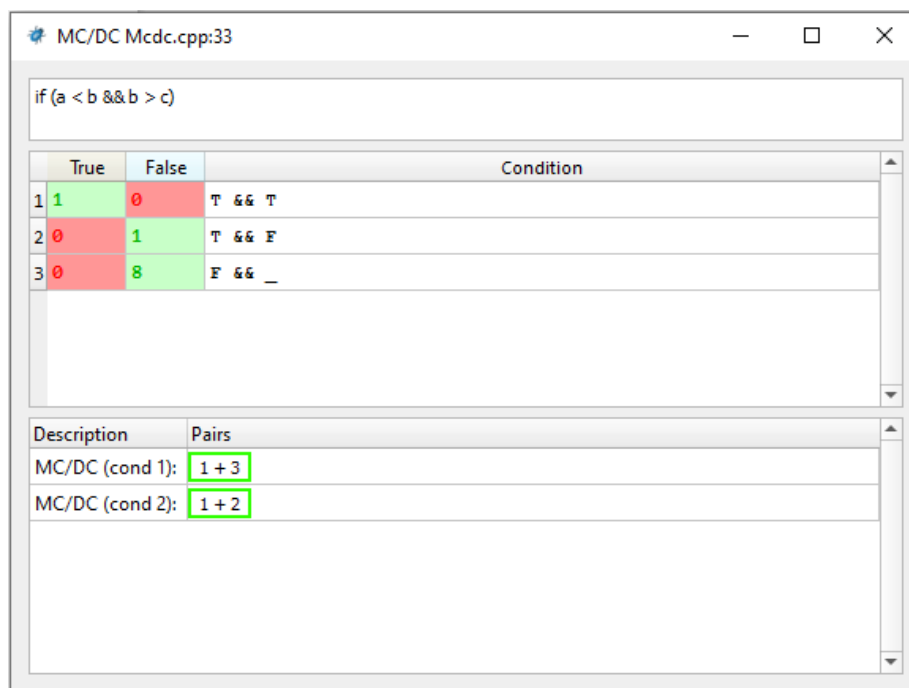


To open MC/DC details dialog, click the yellow line's number (it is underscored) in method's definition. It has 3 main sections (Figure 48) – on the top of the MC/DC dialog there is located analyzed condition from already executed test. Then, there is a table with all the conditions listed and their actual amount of execution – so value '5' in the first row of *True* column means that this condition was obtained 5 times and of course was successful (the result of *True && True* is always *True*). In the second row you can see that this condition was obtained and executed only once, and its result is *False*.

Underscore ("_") in the „Condition” column means any boolean value (*True/False*) as this value will not affect the result anyway.

The last section of this dialog is the description. It shows if the current condition (leaf-level Boolean expression) is independent from other conditions' results. The independence of a condition concerns that only one condition changes at a time. The symbol plus (“+”) or minus (“-“) placed between conditions' numbers in the description, indicates which pair of conditions were achieved (plus) and which were not (minus). If there is at least one pair with plus, it means MC/DC was fulfilled.

Figure 48. Example of MC/DC dialog – plus symbols.

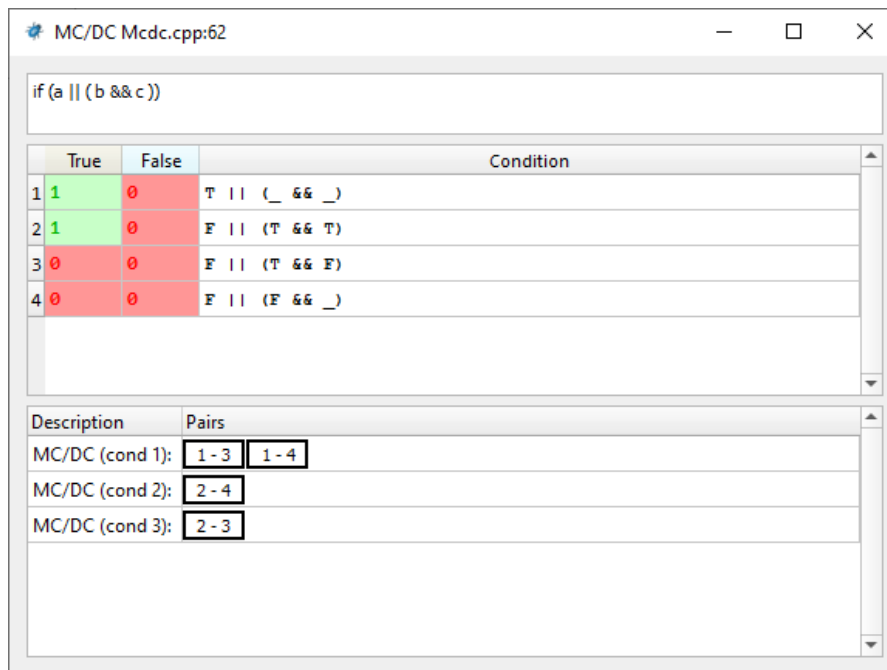


Interpretation for above example can be: to check the condition's independence, there needs to be executed a pair of condition 1 (True AND True) and 3 (False AND _), and also a pair of 1 and 2 – and they all have already been achieved. In other words:

- The first condition (T && T) and the third condition (F && _) demonstrate that 'a < b' can independently affect the outcome decision.
- The first condition (T && T) and the second condition (T && F) demonstrate that 'b > c' can independently affect the outcome decision.

Let's have a look at opposite situation with minuses (Figure 49).

Figure 49. Example of MC/DC dialog – minus symbols..



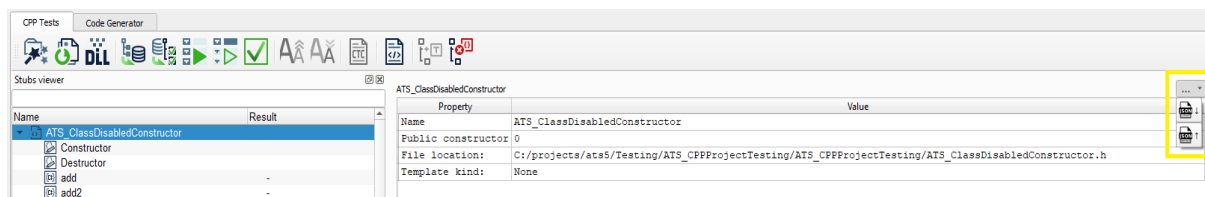
Example description interpretation for above is – to check independence of the condition there is a need to execute:

- *The first condition (T || (_ && _)) and the third condition (F || (T && F)) or the first condition and the fourth condition (F || (F && _)).* They all demonstrate that ‘a’ (from analyzed expression) can independently affect the outcome decision.
- The second condition (F || (T && T)) and the fourth condition (F || (F && _)). They all demonstrate that ‘b’ (from analyzed expression) can independently affect the outcome decision.
- The second condition (F || (T && T)) and the third condition (F || (T && F)). They all demonstrate that ‘c’ (from analyzed expression) can independently affect the outcome decision.

2.5. Importing/exporting tests

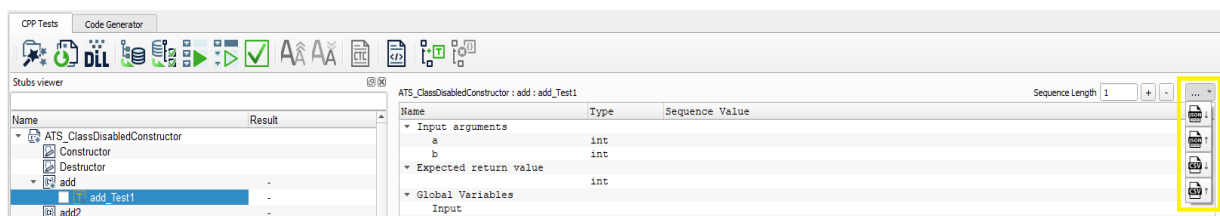
Application allows you to import ready tests (in a format of .JSON files) to the project. It can be done by right-clicking a class or method in Stubs viewer and selecting the option „*Import method tests*”/ „*Import tests for the whole tree*”. Other way to import tests is to select a button „...”, that is placed next to Sequence Length in tests parameters field or in class/method definition.

Figure 50. Importing/exporting buttons in class definition.



Exporting tests is equally simple – you can find this option in context menu of tree items or – after selected a specific test – export it via button, placed next to Sequence Length. As you can see, application allows you to export tests as CSV and also as JSON files.

Figure 51. Importing/exporting buttons in tests' parameters section.



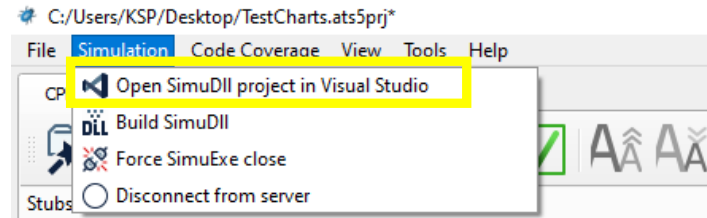
The difference between importing/exporting files in format of JSON or CSV, is that while using .JSON files, the test is imported as a new test, and exported test is also exported as a whole element. Importing by CSV file will cause loading only tests' values, and exporting as CSV file will save only tests' values.

2.6. Modifying SimuDLL project

In case there will appear any error during SimuDLL project compilation, user can open SimuDLL project via ATS application by clicking *Simulation* menu,

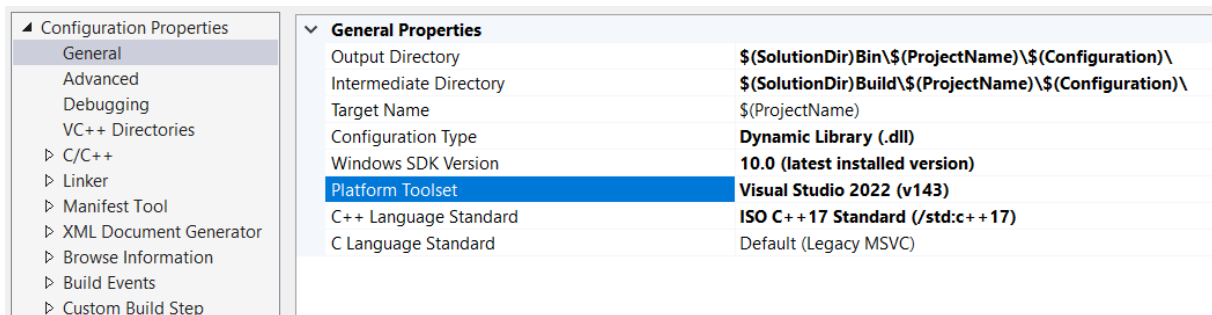
then „*Open SimuDLL project in Visual Studio*”. After that, .vcxproj file containing SimuDLL will be opened.

Figure 52. Simulation – Open SimuDLL in VS.



Proper SimuDLL project configuration looks like this:

Figure 53. SimuDLL Configuration.



Most important is to specify „*Platform Toolset*” to „*Visual Studio 2022 (v143)*”. Otherwise, there may appear errors during compilation. One of the common errors that appear (if „Platform Toolset” is not specified) is that our application cannot find included system headers in files that we are trying to analyse.

Chapter 3. Additional features of CPP Tests

Besides main features that were described before, ATS5 has some other functionalities. On the Figure 54 there are buttons marked in red, yellow, green and blue.

Figure 54. Toolbar additional features



Button in blue frame concerns refreshing project files. It will work, if the application finds any changes in files, that user is currently using in a project (in .h or .cpp files). After clicking the button, if there had been any changes made to the files, the application will update them, remaining all the created tests by user.

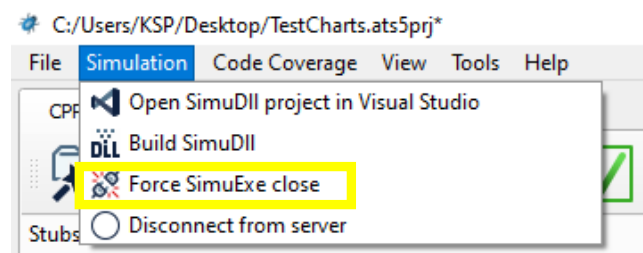
Buttons in red area are related to increasing/decreasing font size for constructor and destructor methods' definitions.

Buttons in yellow frame concern generating CTC and ATS reports.

Buttons in green frame concern adding new test (on the left) and deleting selected stub (on the right).

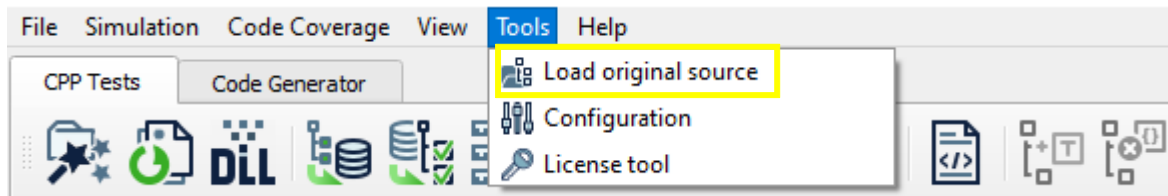
In case of a problem with executing tests, you can force SimuDLL to close. To do that, go to *Simulation – Force SimuExe close* (Figure 55).

Figure 55. Force SimuExe close



Additionally, in Tools tab, you can find an option called „*Load original source*”. It is used for restoring imported file to its original version – without any added tests, variables or snapshots. Snapshots are used to make shots of a test, which cannot be modified but they can be used to restore its values.

Figure 56. Tools tab



3.1. ATS Reports

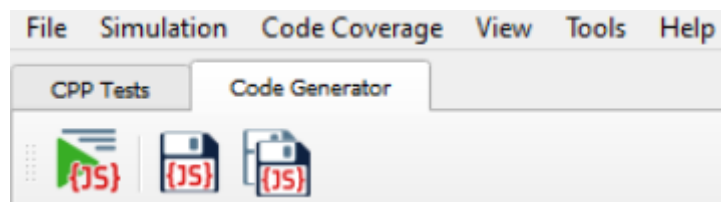
Those reports are generated as HTML file. They include Table of contents in the top of the page, then the titles of classes that contain done tests, names of the methods with their test results and their definition titled as Function Code.

Optionally there could be included comments section. In the middle and the bottom of the page there is a chart with a legend of params, and below that the report includes a table with the values.

Chapter 4. Code Generator

Code Generator is a functionality that allows you to load JavaScript files, modify them, create new one, and then use them to generate dynamic code between customizable tags in .hpp, .cpp and .h files.

Figure 57. Code Generator basic button

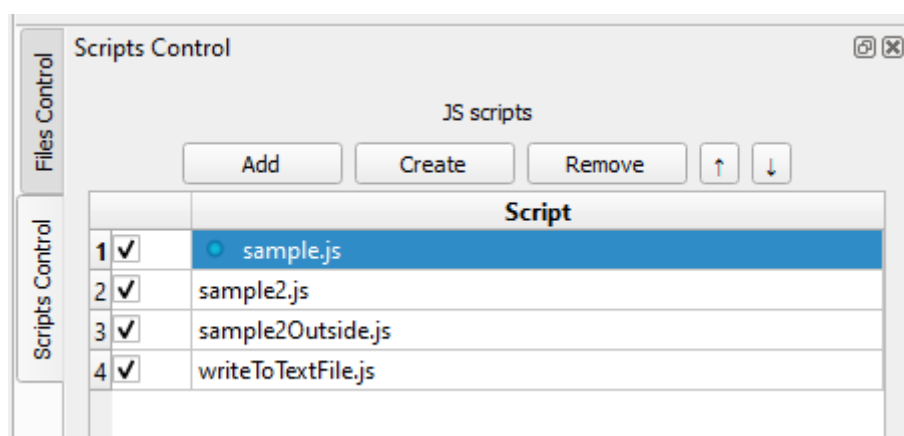


In Code Generator tab, there are 3 basic buttons. The first on the left is used to run all JS scripts, loaded to a project. The second in the middle is used to save single, selected script. And the last one saves all created or modified scripts.

4.1. Scripts Control

In this tab you are able to load existing JS scripts and open them in application (Figure 58). To load scripts from your computer, click *Add* button. To create new JavaScript script, select *Create*.

Figure 58. Buttons for Scripts Control



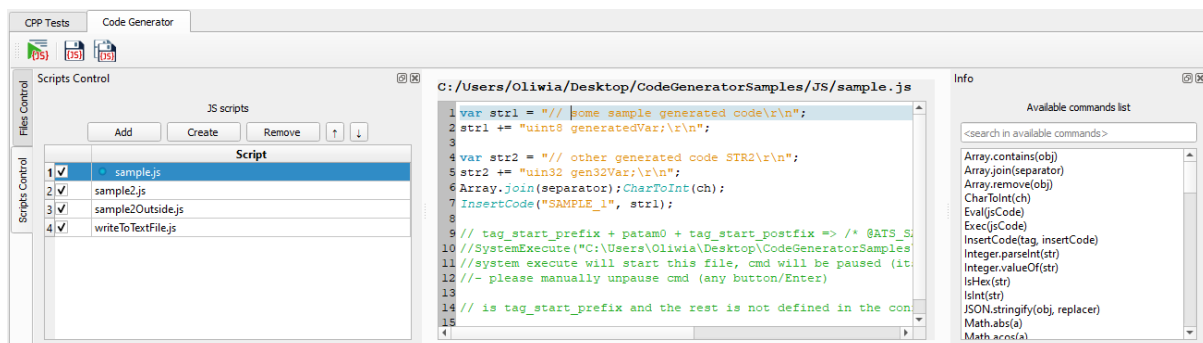
If you would like to remove loaded .js file from the Script list, select the item and then click *Remove* button. All scripts on a list will be executed in ascending order (from 1 to n). Down and up arrows buttons allow you to place

particular .js file lower or higher on a list, which will cause changes in scripts execution order. Checkboxes are used to enable or disable files from the list, without removing them – if you do not want some script to be executed but you want to save it on a list, simply uncheck the box. In this case, it will not be executed.

The blue circle (Figure 58) placed near the name of a file means that this file has been changed and stays unsaved. It will disappear after saving the script.

Going further, in the middle of the screen (Figure 59) there is a modifiable field with JS code. You can add commands from *Available commands list*, which is placed on the right side – just click the line and area, where you would like to have the command inserted and double-click the needed item from the list.

Figure 59. Scripts Control view in Code Generator



To sum up, by modifying JS files there is a possibility to interact with all the selected files and generate code from custom templates (by tags).

4.2. Files Control

Files Control basic buttons are used for importing files to the project, adding new, single files or removing the selected one (Figure 60). Files to import can be selected from Visual Studio projects or added separately by user.

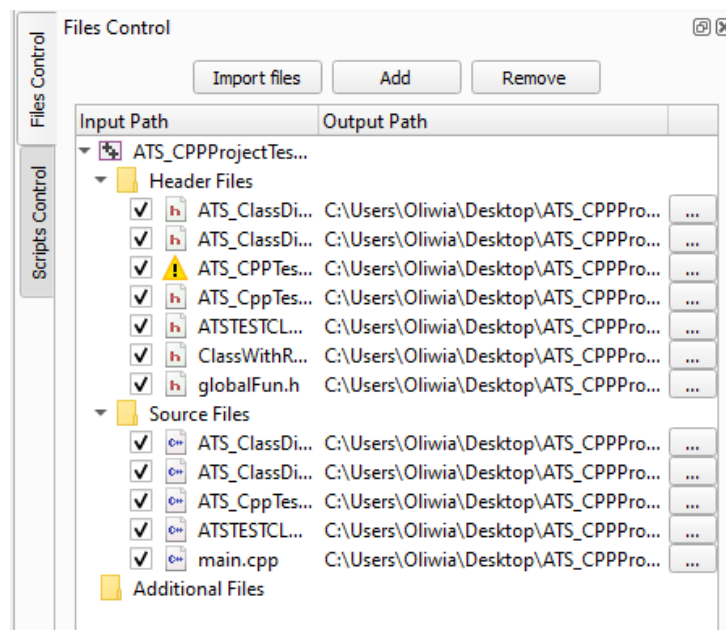
By using *add* button, you can add .h, .cpp or .hpp files to the project.

Remove button allows to delete a single file but also to delete entire loaded folder or a project.

In the tree with imported files, you can find output path and a button „...” that allows you to manually specify an output path for JS methods (InsertCode, ReplaceCode). When these methods will make any changes to the files, those changes will be saved just in this output path. It is set by default to the path of the imported file.

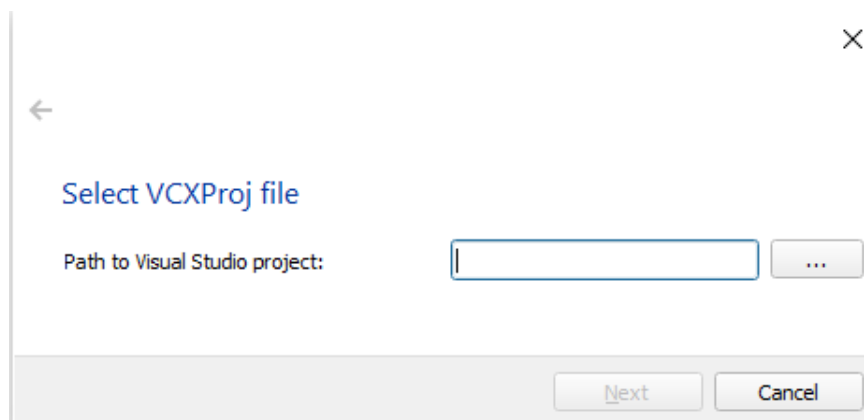
Yellow triangle with an exclamation mark inside informs about warning – a file cannot be found.

Figure 60. Basic button in Files Control



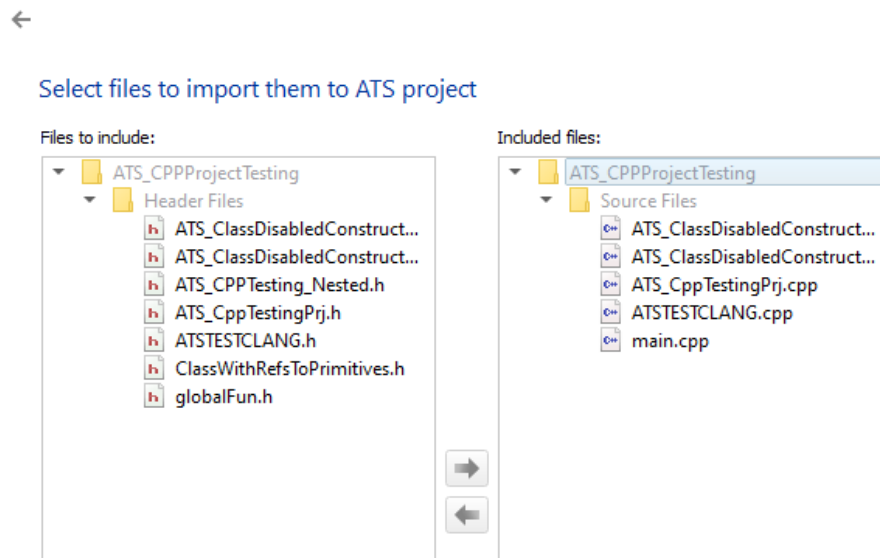
Clicking *Import files* button opens new window with selecting VCXProj file.

Figure 61. Selecting file to import in Files Control



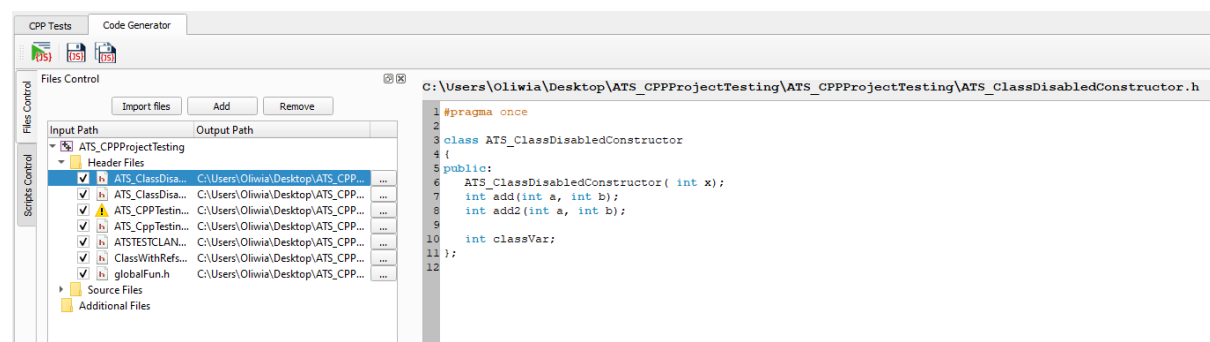
By going „*Next*”, there is a window that allows you to choose files, that you would like to include in a project (Figure 62) – you can select single elements or whole folders. To include them, select the item and click the right arrow. It will move the content of the selected item to the right side „*included files*”.

Figure 62. Selecting particular files to include in Files Control



At the end of the importing process there should be displayed a window with information that the importing was successful. Such imported files will be displayed in a tree and selecting one of its items displays the code of the file in the field on the left (Figure 63).

Figure 63. Files Control view in Code Generator



List of Figures

Figure 1. License Tools.....	3
Figure 2. Welcome Window	4
Figure 3. Main View of ATS5.	5
Figure 4. Compilation Tools.	5
Figure 5. Importing files to CPP Tests.	6
Figure 6. Requirements list.	7
Figure 7. Requirements in Configuration.	7
Figure 8. Requirements Summary table.	8
Figure 9. Tools for adding/removing requirements.	8
Figure 10. Selection section in CPP Tests.	10
Figure 11. Main View of ATS5 with imported project.	10
Figure 12. Recent projects list in Welcome Window	11
Figure 13. Warning Box - Configure project paths	11
Figure 14. Configuration - Configure project paths	12
Figure 15. Remove missing project in Welcome Window.....	13
Figure 16. File – Save options.	13
Figure 17. Configuration - Database.....	14
Figure 18. Code Coverage tab.	15
Figure 19. CTC Tools.	15
Figure 20. Defining constructors with non-params.	16
Figure 21. Information while recognizing constructor with parameters.	16
Figure 22. Successfully built DLL notification.	16
Figure 23. Adding tests via context menu.	17
Figure 24. Additional options for test in Stubs Viewer	18
Figure 25. Main View of ATS5 with added tests.	18
Figure 26. Setting wrong data type for a test parameter	19
Figure 27. Add user variable section	20

Figure 28. User variables usage in test	20
Figure 29. Reference to global variable in test's parameter	21
Figure 30. User variable as a pointer used in test's parameter	21
Figure 31. Setting or getting pointer user variable	21
Figure 32. Test sequences	22
Figure 33. Test description	23
Figure 34. Ranges in tests' parameters	24
Figure 35. Special operators in ranges	25
Figure 36. Empty struct fields.....	25
Figure 37. Struct usage in test.....	25
Figure 38. Struct examples in test.....	26
Figure 39. Set struct variable	26
Figure 40. Set Variable for struct.....	27
Figure 41. Automatic mode for running tests	28
Figure 42. Reports Tools.....	29
Figure 43. View tab.....	29
Figure 44. Charts section	30
Figure 45. Project configuration with CTC options	31
Figure 46. Main ATS5 window with CTC feature	32
Figure 47. Lines' colors in method definition	33
Figure 48. Example of MC/DC dialog – plus symbols.	34
Figure 49. Example of MC/DC dialog – minus symbols..	35
Figure 50. Importing/exporting buttons in class definition.	36
Figure 51. Importing/exporting buttons in tests' parameters section.	36
Figure 52. Simulation – Open SimuDLL in VS.	37
Figure 53. SimuDLL Configuration.	37
Figure 54. Toolbar additional features.....	38
Figure 55. Force SimuExe close	38
Figure 56. Tools tab	39

Figure 57. Code Generator basic button	40
Figure 58. Buttons for Scripts Control.....	40
Figure 59. Scripts Control view in Code Generator	41
Figure 60. Basic button in Files Control.....	42
Figure 61. Selecting file to import in Files Control.....	42
Figure 62. Selecting particular files to include in Files Control.....	43
Figure 63. Files Control view in Code Generator.....	43