

# Automation & Testing Suite for embedded software | AUTOSAR-compatible

# ATS 05.03.003.002

# **Basic Usage Manual**

SCHLEISSHEIMER SOFT- UND HARDWAREENTWICKLUNG GMBH

www.automation-testing-suite.com www.schleissheimer.com



2024

# Contents

Chapter 1. Getting started
1.1. Project creation4
1.2. Opening existing project11
1.3. Removing project12
1.4. Saving project
Chapter 2. Testing files
2.1. Building SimuDLL15
2.2. Adding new test17
2.3. Modifying a test19
2.3.1. Sequences23
2.3.2. Range values25
2.3.3. Special operators25
2.3.4. Structures usage in tests
2.3.5. Class objects usage in tests
2.3.6. Functions mocking
2.4. Running tests
2.4.1. Charts
2.4.2. MC/DC coverage
2.5. Importing/exporting tests
2.6. Modifying SimuDLL project
Chapter 3. Additional features of CPP Tests
3.1. ATS Reports44
Chapter 4. Code Generator
4.1. Scripts Control
4.2. Files Control
List of Figures

# **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.

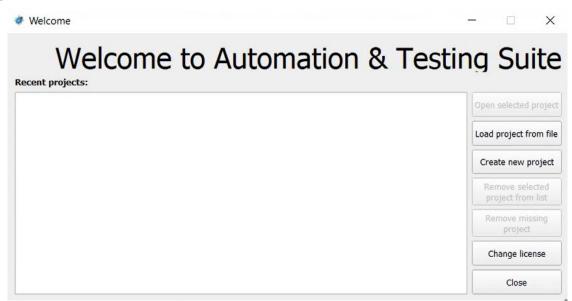
First, run the application. On your screen, the License Tool window will appear (Figure 1). You can import license by inputting the activation key (section on the right side) or by using remote license parameters (on the left). After successfully activating the license, application needs to be restarted.

Product name	License key	Туре	Support	Basic	Code Coverage	Enabled in VM	
ATS Basic(Local)	2.	Perpetual					
	Select licens	e			Refresh		
e License Search Parameters			License Update				
			Input new	activation key			
			input nev	Conduct Rey			
			9******	*-****-****-****	*****		
Accept Cancel				Activate			

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.

	ð ×
File Simulation Code Coverage View Tools Help	
CPP Tests Code Generator	
Subs viewer ØB	
Nama Result	
Name Result	
log	හ ස
vvg 2014/01/10/2017 16	0.0

2024-09-18 08:21:17 View restored. 2024-09-18 08:21:26 Project correctly loaded.

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 1	Commilation	Toola
r igure 4.	Compilation	100is.

Configuration - Compilatio	n tools		×
Type to find	Compilation t	ools	
<ul> <li>Application</li> <li>General</li> </ul>	MsBuild.exe	C:/Program Files/Microsoft Visual Studio/2022/Community/MSBuild/Current/Bin/amd64/MSBuild.e	
Compilation tools Appearance Company C++ Project	Cl.exe	C:/Program Files/Microsoft Visual Studio/2022/Community/VC/Tools/MSVC/14.33.31629/bin/Host	
Project Database CTC Reports Requirements Code Generator			
		Save configuration Car	ncel

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.

🛷 ATS5	×	🔹 ATS5	×
Import files for te	sting	Import files for t	testing
Import: Add files: Import method: Additional includes path: Select language standard: Requirements prefix Please note, that after analy Remember to make sure th append option) so it won't Selected source files tree: Composition of the strees Composition of the strees Compositio	Source Files Append new items to tree  Append new items to tree  Replace existing stubs  C++14 (default)  REQ_TEST  ysis of files is finished the project will be automatically saved. at database configuration is properly set (chosen overwrite or overwrite already saved data!  5/Source/Testing/ATS_CPPProjectTesting/ATS_CPPProjec  5/Source/Testing/ATS_CPPProjectTesting/ATS_CPPProjec	Import: Path: Import method: Additional includes path: Select language standard Requirements prefix Please note, that after an Remember to make sure	Visual Studio Solution/Project
	VisabledConstructor.cpp		< Back Next > Cancel

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.

Requirements		-	×	
Include in raport		Requireme	nt	
V	REQ_INCLUDE1			
V	REQ_INCLUDE2			
	REQ_INCLUDE3			
	REQ_INCLUDE4			
V	REQ_INCLUDE5			
V	REQ_INCLUDE6			
V	REQ_INCLUDE7			
V	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.

Configuration - Requirement	nts				×
Type to find	Requirements				
<ul> <li>Application         <ul> <li>General</li> <li>Compilation tools</li> <li>Appearance</li> </ul> </li> </ul>	Prefix of requirements	REQ_INCLUDE		Requireme	nts list
Company C++ Project Project Database					
CTC Reports Requirements Code Generator					
				Save configuration	Cancel
				care configuration	Carroci

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.

dFloat_Test1					
_					
It: Passed					
unction Code					
	::addFloat(float a, float b)				
<pre>3 return a + b; // REQ_ 4 }</pre>	TEST5 REQ_INCLUDE1				
		🛚 a 🔲 b 🔲 retRcv 🔲 retErr 🔲 ErrorPoir	ts 🔳 Axis Y 🔳 Axis X		
2.00					
4.00					
6.00					
8.00					
0.00					2.
0.0					2.
Category	Туре	Name	Sequence step		
Category	1300	Hume	1	2	
Parameter	float	a	11	2	
Parameter	float	b	11	2	
Expected return	fioat		22	4	
Value returned	float	-	22	4	
			Requi	rements Coverage	
quirements Summary ta	able				
Requirement name	Covera	ıge (%)	Test name	Test status	
			Test name Test status addFloat_Test1 Passed		

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

Figure 9. Tools for adding/removing requirements.

ITS_CppTestingPrj : switchCaseFunction :	switchCaseFunction	_Test1	Sequence Length 1 + *
Name	Type	S1	REQ_TEST1
<ul> <li>Input arguments</li> </ul>			REQ_TEST2
a	int		
b	int		
c	int		
<ul> <li>Expected return value</li> </ul>			
	int		
<ul> <li>Global Variables</li> </ul>			
Input			
Expected value			
<ul> <li>User Variables</li> </ul>			Global Variables Ø
Input			Giobal variables
Expected value			Type Name 🗢
			int globalVariable
			long globalVariable2
			double globalVariable3
			float globalVariable5
			4

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.

elect files		
	TS_CPPProjectTesting	
▼	Header Files	
V		
	ATS_ClassDisabledConstructor.h	
	☐ ATS_ClassDisabledConstructorWorkingParam.h	
	ATS_CPPTesting_Nested.h	
v	」	
v		
	☐ Glasswithkers for finitives.n	
	Source Files	
	++ ATS_CinTests.cpp	
	++ ATS_ClassDisabledConstructor.cpp	
V	** ATS_ClassDisabledConstructorWorkingParam.cpp	
	*+ ATS_CppTestingPrj.cpp	

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 ATS5 with imported project.

C/Meru/Test/Destsphrew/hojectalSpj*     File Simulation Code Coverage View Tools Help	×
CPP Tests Code Generator	
Ali a	
Subst view     Image       Name     Result       * (1) AFS_Cheabs     ************************************	
log 2024 04-19 08 22 35 United Standards Under 2024 04-19 08 22 35 United Standards Free Andrea 2024 04-19 08 22 34 United Standards Andrea 2024 04-19 08 25 34 United Standards Andrea 2024 04-19 08	
2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.	-

# **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

🄻 Welcome		– 🗆 X
Welcome to Automation Recent projects:	n & Testin	g Suite
test1	9 marzec 2022 22:35:28	Open selected project
C:\Users\Oliwia\Desktop\Nowy folder\test1.ats5prj		Load project from file
		Create new project
		Remove selected project from list
		Remove missing project
		Change license
		Close

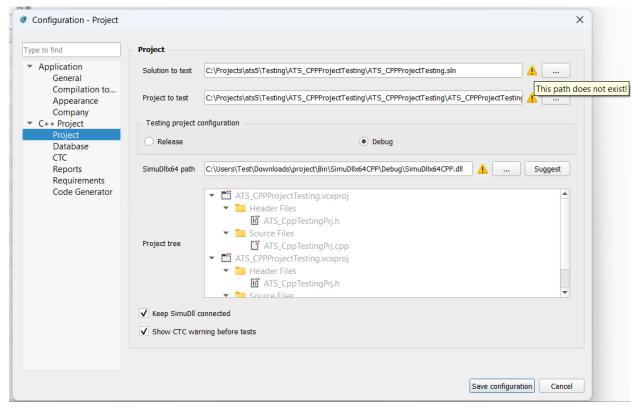
In case of opening existing ATS project from different device, you will need to adjust some paths to be able to use such project. Right after opening it, on the screen appears warning box with information which paths needs to be configured (Figure 13).

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"*.

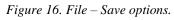
Figure 15. Remove missing project in Welcome Window.

🆸 Welcome	- 🗆 🗙
Welcome to Auton Recent projects:	nation & Testing Suite
emptyProject	14 marzec 2022 09:15:23 Open selected project
C:\Users\Oliwia\Desktop\Nowy folder\emptyProject.ats5prj	Load project from file
test1 C:\Users\Oliwia\Desktop\Nowv folder\test1.ats5orj	11 marzec 2022 07:47:14 Create new project
c. frame frame have been able to a low and the second by the second been able to a low and to a low and the second been able to a low and to a lo	Remove selected project from list
	Remove missing project
	Change license
	Close

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.



File	Simulation	Code Coverage	View	Tools	Help
R	New project	Alt	+N		
2	Load project	Alt	+L		
	Save Project	Alt	+S		
e	Save as	Cti	·I+Alt+S	Shift+S	
۲	Exit				



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.

e to find Application General Compilation tools Appearance	Save method	
Company C++ Project Project Database	Save to file     Database URI	
CTC Reports Requirements Code Generator		TestDB  Add database
Code Generator		Test - Mon, 09 Sep 2024 08:12:07
	Database file	C:\Users\Test\Desktop\TestProject\db
	Append	

Firstly, user has to select the database (or create new one using "*Add database*" button), and then specific database entry with a method of saving:

- append method causes adding new data to selected document,
- *override* method will overwrites selected document with latest changes.

Figure 18. Configuration - Saving data to MongoDb.

<ul> <li>Application General Compilation tools Appearance Company</li> <li>C++ Project Project Potetsbasse</li> <li>Databasse</li> <li>Database URI</li> <li>mongodb://localhost:27017</li> <li>Database name</li> <li>Test DB</li> <li>Database entry (Username, Date)</li> </ul>	
Database         Database URI         mongodb://localhost:27017           CTC         Reports         Database name         TestDB           Code Generator         Code Generator         Code Generator	
Requirements Database name TestDB Code Generator	
	Add database
	24 08:12:07
Method of saving database	
Append     Override	

# **Chapter 2. Testing files**

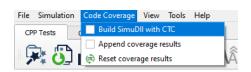
In this chapter, you will get to know how to build DLL, prepare your files for analysis, create tests and how to run them.

### 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 19).

Figure 19. Code Coverage tab.



If you decided to build DLL with CTC enabled, you can set CTC options in *Tools - Configuration – CTC* menu (Figure 20).

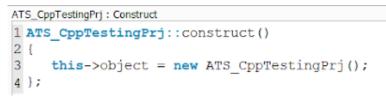
Figure 20. CTC Tools.

Configuration - CTC		×
Type to find	стс	
<ul> <li>Application General Compilation tools Appearance</li> </ul>	Code coverage report type           Image: Statement coverage	Decision/Branch coverage     MC/DC coverage
Company C++ Project Project	CTC report generation Generate TXT report	Auto-open after generation
Database CTC Reports Requirements	Generate XML report	Auto-open after generation
Code Generator	Generate HTML report	Auto-open after generation
	Source code editor coloring option	
	Accurate	Expanded
	CTC additional options	
	Analyse header files	
	CTC report threshold	100 %
		Save configuration Cancel

Last important step to take, is to make sure that all constuctors 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 21.

Figure 21. Defining constructors with non-params.



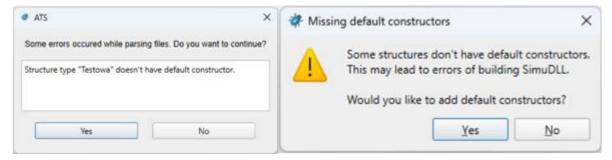
However, in some cases there is a necessity to define them with parameters. In such situations, if the application recognizes it, application will display an information, as shown on Figure 22.

Figure 22. Information while recognizing constructor with parameters.



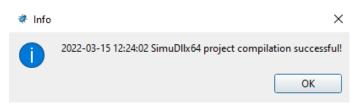
Also if parsed testing project contains structure without defined default constructor, ATS will recognize it and ask if user would like to create such default constructors (see Fig. 23).

Figure 23. Create default constructurs for structures.



After all is set up, successful building the DLL will display a dialog with confirmation (Figure 24). 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 24. 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

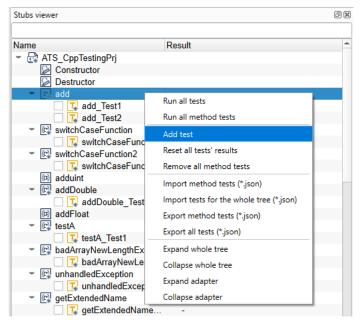
Adding test to adapters (tree items named as class methods or global functions) is possible in three ways.

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 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.

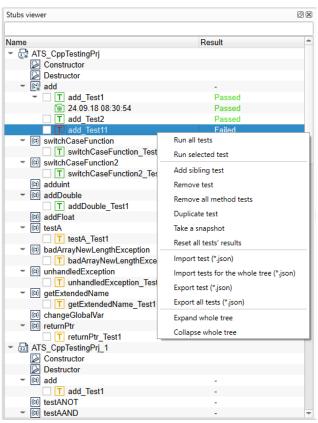
Third option is to use context menu on adapter by pressing right mouse button on it (Figure 25). Please note, launching a context menu in Stubs viewer is only possible when right-clicked on "Name" column area – invoking context menu won't work if it was done on "Result" column.

Figure 25. Adding tests via context menu.



Application allows you to rename test by double-clicking on it. Also, there is a possibility to remove single/multiple tests or remove all tests from method/class, duplicate it, add sibling test, take a snapshot of it, run it and reset its results (Figure 26).

Figure 26. Additional options for test in Stubs Viewer



#### 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 27).

Figure 27. Main View of ATS5 with added tests.

D:/projects/ats5/ATS5_DemoProject/ATS5_DemoProject	ct.ats5prj												-	0 ×
File Simulation Code Coverage View Tools Help	P													
CPP Tests Code Generator														
🗭 🖏 🛍 🐌 🗱 🍺	V 4â 4ă		1 1 1											
Stubs viewer		28									Global Variables			28
			ATS_CppTestingPrj : switchCaseFunction : :						Sequence Length 4	• • u*	-			
Name	Result	-	Name	Type	S1	S2	S3	54			Type		ame lobalVariable	
- ATS_CppTestingPrj	Result		<ul> <li>Input arguments</li> </ul>								long	9	lobalVariable	2
Constructor				int	[1,10	ų.					double	ġ	lobalVariable	3
			0	int	4						float		lobalVariable	5
Destructor     is add			- Expected return value								const char *		lobalek	
T add Test1	Passed			int		10					ATS_CppTesti ATS_CppTesti		lobalClass1 lobalClass2	
			<ul> <li>Global Variables</li> </ul>										ointerInteger	
24.09.18 08:30:54	Passed		Input								LINE -		lohal@rmor	-
add_Test2	Passed		<ul> <li>Expected value</li> <li>User Variables</li> </ul>								4			•
add_Test11	Failed		- Oser Variables											
<ul> <li>switchCaseFunction</li> </ul>			Expected value								User Variables			28
switchCaseFunction_Test1	Passed													
<ul> <li>switchCaseFunction2</li> </ul>														• •
T switchCaseFunction2_Test1	Passed													
adduint											Name Typ	e		Value
<ul> <li>addDouble</li> </ul>											varInt int			5
T addDouble_Test1	Passed										varIntPtr int			new int(1)
🖹 addFloat	-										structSample Str	ukturaStr (int, st	d:string, unsign	(varint,"sample".
- In testA														
T testA Test1														
<ul> <li>Im badArrayNewLengthException</li> </ul>														
T badArrayNewLengthException Tes	st1 -													
<ul> <li>Im unhandledException</li> </ul>														
Internet unbandledException_Test1														
<ul> <li>migetExtendedName</li> </ul>														Þ
T getExtendedName Test1											1			
ChangeGlobalVar											Test Description			08
* Ini returnPtr											rest pescription			6.6
T returnPtr Test1											This is sample descrip	tion for method	switchCaseFunctio	~0
											ing is sumple descrip		annun coper anneo	~
<ul> <li>ATS_CppTestingPrj_1</li> </ul>														
Constructor														
Destructor														
- 🕅 add														
T add_Test1	-													
iei testANOT	-		Input stream											
✓ istAAND		*									Test Requirements	Test Descripti	on	
Log 2024-09-18 08:31:11 Start test: ATS_CopTestingPri : a	addDouble : addDouble :	Test1												28
2024-09-18 00:31:11 Test result. Passed The test wa 2024-09-18 00:31:21 Test aid_al_exit1: has been duplied 2024-09-18 00:31:34 Contencied to server 2024-09-18 00:31:34 Gatt test. AfS. CopFictingPij. 2024-09-18 00:32:35 Contencied to server 2024-09-18 00:32:35 Contencied to server 2024-09-18 00:32:35 Contencied to server 2024-09-18 00:32:36 Test result. Passed The test wa 2024-09-18 00:32:36 Test result. Passed The test wa 2024-09-18 00:32:36 Test result. Passed The test wa 2024-09-18 00:34:19 Project tree and tests saved suc	is performed in: 1ms cated. add : add_Test11 ; performed in: 2ms switchCaseFunction : sw is performed in: 22ms		ction_Test1											

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 28).

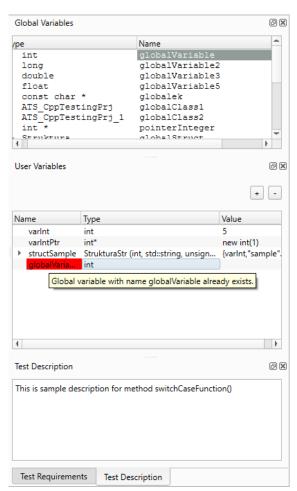
*Figure 28. Setting wrong data type for a test parameter* 

lame	Type	S1 S2	S3	S4
<ul> <li>Input arguments</li> </ul>				
a	int	[1,10]	2.12	
b	int	3 '8'		
c	int	4		
<ul> <li>Expected return value</li> </ul>				
	int	* !0		
<ul> <li>Global Variables</li> </ul>				
- Input				
globalVariable2	long	234132		
<ul> <li>Expected value</li> </ul>				
globalVariable	<ul> <li>double</li> </ul>	0.03		
<ul> <li>User globalVariable</li> </ul>	<b></b>			
In globalVariable2				
Ex globalVariable3				
globalVariable5				
globalek				
globalClass1				
globalClass2				
pointerInteger				
globalStruct				
globalStructe	-			

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 29). It is a docking widget so you can always undock this and place anywhere else to let it be more comfortable to use.

Figure 29. Add user variable section



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 click minus "-" button. In case you create user variable with same name as already existing global variable, application will recognize it as error and mark such variable on red background.

When defining structure types for user variables, it is possible to select which constructor should be used – it can be selected via combobox list. All default constructors without parameters are named same as structure name (without arguments in parenthesis) and all custom constructors with different arguments are listed as well:

Figure 30. User variable structures constructor selection.

User Variables		Ø
		+ -
Name	Туре	Value
varInt	int	5
varIntPtr	int*	new int(1)
<ul> <li>structSample</li> </ul>	StrukturaStr (int, std::string, unsign	{varInt, "sample", 2334432, {, 3}}
x	int	varInt
b	std::string	"sample"
te	unsigned long long	2334432
▶ n	nestedStructTwoDeepth (testStruct	{,3}
testStr	ATS CooTestinaPri	
	ATS_CppTestingPrj ATS_CppTestingPrj_1 Class1 ATS_ClassDisabledConstructor ATS_String ATS_StructPointer ATSTESTCLANG	
	testStruct (int, int) testStruct UserStruct (int, float)	:t (int, int)

If some parametrized constructor was selected, user is able to define values for structure fields – those values will be then automatically placed in the main node of structure inside {} brackets.

Defining class objects variables is also done via selection in above combobox element – user has to simply choose which class type should be used for specific variable. For both structure and class objects variables it is allowed to use pointers by changing selected type using "\*" character at the end of the expression. SimuDll needs to be rebuilt to allow usage of such created user variables and to allow you to select them from combobox placed in the test definition section (Figure 31).

Figure 31. User variables usage in test

ATS_CppTestingPrj : switchCaseFunction : s	witchCaseFunction_Test	2	Sequence Length 1 + *	Global Variables			Ø
Name	Type	S1		Туре		Name	Value
<ul> <li>Input arguments</li> </ul>				int		globalVariable	5
a	int	45		long		globalVariable2	
b	int	4		double		globalVariable3	
c	int	4		float		globalVariable5	
<ul> <li>Expected return value</li> </ul>				const cha		globalek	&"asd"[0]
	int	4		ATS_CppTe		globalClass1	
<ul> <li>Global Variables</li> </ul>					stingPrj_1	globalClass2	
Input				int *		pointerInteger	-
Expected value				1		al and 19 tough	
<ul> <li>User Variables</li> </ul>							
- Input				User Variables			0
varInt	int	*varIntPtr					0.
<ul> <li>structSample</li> </ul>	StrukturaStr	{21, "sampleStr", 23431554, {{3,4},2}}					
x	int	21					+ -
b	std::string	"sampleStr"					
te	unsigned lo	23431554		Name	Type	V	alue
mest	nestedStruc	{{3,4},2}		varint	int		
▼ st	testStruct	{3,4}		varint	int*	2	ew int(1)
x	int	3					
У	int	4		▶ structSample			/arint,"sample",2334432,{,3}}
x	int	2		testStr	ATS_CppTestir	ngPrj	
<ul> <li>Expected value</li> </ul>							
varInt	<ul><li>int</li></ul>	5					
varInt							
varIntPtr							
structSample	(int)varInt						
testStr							

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 32).

Figure 32. Reference to global variable in test's parameter

Jame	Type	Sequence	Value	
<ul> <li>Input arguments</li> </ul>				
ref	int &	globalVa	riable	
<ul> <li>Expected return value</li> </ul>				
	int	4		
Global Variables	Name		Value	
Global Variables	Name globalVaria	able	Value	5

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

Name	Type	Sequence Value
<ul> <li>Input arguments</li> </ul>		
a	int &	*myIntPtr
b	int	2
с	int	3
<ul> <li>Expected return value</li> </ul>		
	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 34).

Figure 34. Setting or getting pointer user variable

<u>r</u>		
🔻 User Variables		
<ul> <li>Input</li> </ul>		
ptr[0]	int*	1
<ul> <li>Expected value</li> </ul>		
*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 35). Also, you can modify the amount of sequences by putting its length number in the textfield.

Figure 35.	Test sequences
------------	----------------

ATS_CppTestingPrj : switchCaseFunction : switch	hCaseFunction_Test8				Sequence Length 4 +
Name	Type	S1	S2	53	S4
<ul> <li>Input arguments</li> </ul>					
a	int	2			
b	int	421		34	
c	int	33			
<ul> <li>Expected return value</li> </ul>					
	int	22			
<ul> <li>Global Variables</li> </ul>					
<ul> <li>Input</li> </ul>					
globalVariable	int	6			
<ul> <li>Expected value</li> </ul>					
globalVariable	int	7			
<ul> <li>User Variables</li> </ul>					
<ul> <li>Input</li> </ul>					
▼ uu	Struktura	{4,3,2,nullptr}	{,,,}	{,,,}	{,,,}
x	int	4			
b	float	3			
te	unsigned long	2			
nest	nestedStruct *	nullptr			
<ul> <li>Expected value</li> </ul>					
▼ 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: 2, 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 36).

Test Description രിയ This is test description. User Variables Test Description Global Variables ØX Name Type int globalVariab globalVariab long globalVariab double globalVariab float const char \* globalek globalStruct Struktura StrukturaDwa globalStruct globalStr2 StrukturaStr Tabs globalTabs UserStruct globalUserSt hestedStruct globalNested nestedStructTwoDeepth globalNested Þ

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.

Figure 36. Test description

#### 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 37.	Ranges	in tests '	parameters
------------	--------	------------	------------

Name	Type	S1	S2
<ul> <li>Input arguments</li> </ul>			
a	int	66	3
b	int	[1,5,1]	3
с	int	2	4
<ul> <li>Expected return value</li> </ul>			
	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 38. Special operators in ranges

_	_				
Name	Type	S1	S2	S3	S4
<ul> <li>Input arguments</li> </ul>					
a	int	3	3		
b	int	4	-2		[-3,10,3
c	int	33	[1,10,2]		
<ul> <li>Expected return value</li> </ul>					
	int	! 3	>0	*	<=82

Usage of these special operators is presented on Figure 38. 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. Structures usage in tests

It is possible to use structures within tests and to define user variables of such type. In ATS5 this test will be displayed and handled a little bit different than regular test with primitive types (Figure 39).

Figure 39. Empty struct fields

Name	Type	S1
<ul> <li>Input arguments</li> </ul>		
a	int	
▼ b	Struktura	{ , , , }
x	int	
b	float	
te	unsigned long.	
nest	nestedStruct	*
<ul> <li>Expected return value</li> </ul>		
	void	

As you can see, the general row of such structure 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 40).

Figure 40. Struct usage in test

Name	Type	S1
<ul> <li>Input arguments</li> </ul>		
a	int	3
▼ b	Struktura	{3,21.3,53,nullptr]
x	int	3
b	float	21.3
te	unsigned long lo	ng 53
nest	nestedStruct *	nullptr
<ul> <li>Expected return value</li> </ul>		
	void	
- Clabel Venichles		

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 41).

Figure 41.	Struct	examples	in test
------------	--------	----------	---------

ATS_CppTestingPrj : structFun : structFun_Test1		
Name	Туре	S1
<ul> <li>Input arguments</li> </ul>		
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
<ul> <li>Expected return value</li> </ul>		
	void	
▼ Global Variables		
<ul> <li>Input</li> </ul>		
globalUserStruct	UserStruct	{5,0.5}
x	int	5
b	float	0.5
<ul> <li>Expected value</li> </ul>		
<ul> <li>globalNestedStruct</li> </ul>	nestedStruct	{{'h'},34}
<ul> <li>nestnest</li> </ul>	nestedStructTwoD	{'h'}
x	char	'h'
x	int	34
<ul> <li>User Variables</li> </ul>		
<ul> <li>Input</li> </ul>		
▼ uu	Struktura	{4,5,44,nullptr}
x	int	4
b	float	5
te	unsigned long long	44
nest	nestedStruct *	nullptr
<ul> <li>Expected value</li> </ul>		
▼ WW	Struktura	<pre>{31,2,2,globalNestedStructPtr}</pre>
x	int	31
b	float	2
te	unsigned long long	
nest	nestedStruct *	globalNestedStructPtr

Additionally, user can set a value for current structure using option from context menu "Set variable" (Figure 42).

Figure 42. Set struct variable

Nar	me	Туре	S1	
-	Input arguments			
	a	int	3	
	▼ b	Struktura	globalStruct	and the second
	x	int	-	Expand until non-empty sequence
	b	float	-	Expand all empty sequences
	te	unsigned long	-	Evened and contact all convenees
	nest	nestedStruct *	-	Expand and replace all sequences
*	Expected return value			Сору
		void		Paste
-	Global Variables			
	<ul> <li>Input</li> </ul>			Set Variable
	<ul> <li>globalUserStruct</li> </ul>	UserStruct	globalUserStruct	Restore default value
	x	int	- 4	
	b	float	-	
	<ul> <li>Expected value</li> </ul>			
	<ul> <li>globalNestedStruct</li> </ul>	nestedStruct	{{'h'},34}	
	<ul> <li>nestnest</li> </ul>	nestedStructT	{'h'}	
	х	char	'h'	
	x	int	34	

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 43).

Figure 43. Set Variable for struct

🏘 Select variable		×
myStructDefault myStructParams structPtr[0]		
Set index for pointer	Ok	Cancel

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 42) – they contain "-" symbol. The only option to change it, is to restore the value by selecting option from context menu "Restore default value".

#### 2.3.5. Class objects usage in tests

As it was mentioned before, it is possible to use class objects in tests – as well as global or user variables (see Figure below) or input arguments/expected return values in tests as well as class pointers.

Figure 44. Class objects usage in tests

ATS_CinTests : firstInCin : firstInCin_Test2		Seq	uence Length 3	+
Name	Type	S1	S2	S3
Input arguments				
<ul> <li>Expected return value</li> </ul>				
	char	'h'		
<ul> <li>Global Variables</li> </ul>				
Input				
Expected value				
<ul> <li>User Variables</li> </ul>				
<ul> <li>Input</li> </ul>				
classType	ATS CinTests	classTypePtr[0]		classType
<ul> <li>Expected value</li> </ul>	_			
classTypePtr	ATS_CinTests*	nullptr	*	

To set argument as class object, it is required to use "Set variable" option from right-click context menu opened on specific cell in sequence column.

#### **2.3.6. Functions mocking**

Mocking functionality is placed under a mock widget button placed in the

toolbar:



In this window there are listed all mock functions recognized from testing project – in the parenthesis are defined classes which those mock functions are involved in, and on the left side of the parenthesis is written the name of function or method that mocked function is changed in:

Figure 45. Mock functions widget.

Mock function	ns										-		×
		Enabled	+			to	Mocl	(void)					
Mock function	testt	<b>v</b>		2	ł			return	99;				
Method (Class)	nestedAlsoHere (ATSTESTCLANG)			3	}								
Method (Class)	nestShouldBeHere (ATSTESTCLANG)												
Mock function	toMock	•											
Method (Class)	function (ATSTESTCLANG)												
Mock function	setFlag_1												
Method (Class)	mcdcTest (Mcdc)												
Mock function	setFlag_2												
Method (Class)	mcdcTest (Mcdc)												
Mock function	setFlag_3												
Method (Class)	mcdcTest (Mcdc)												
Mock function	setFlag_4												
Method (Class)	mcdcTest (Mcdc)												
Mock function	setFlag_5		-										
Add includes	1									Save c	hanges	Disc	card

To use such mock function, user needs to accept the checkbox in column "Enabled". If function is not a void type, specify the appropriate return type in code editor placed on right side after clicking on specific mock function. After user has defined all mock functions, it is required to click "*Save changes*" to apply this code edits. To run tests with mock functions usage, SimuDLL has to be rebuilt first.

If mock functions use some components from additional sources or libraries, it is allowed to add includes which will exist in a file where mocked functions are defined, by clicking on "*Add include*" button – then new window will be displayed (Figure 46).

😻 ATS5	_		×	😻 ATS5	_		×
#include <math.h></math.h>				#include			
				#include <math.h></math.h>			
+ -	Save	Ok	C	+ -	Save	OK	:

Figure 46. Additional includes for mock functions.

To add new include, type in the component and confirm with "+" button – it will be then append to the list. Click "*Save*" to confirm the action and "OK" to quit.

Below is an example of behavior for mocked function and created test – originally it is supposed to return "1" value:

Figure 47. Original method definition before mocking.

ATSTESTCLANG : toMock : ATSTESTCLANG.h

51 int toMock()
52 {
53 return 1;
54 }

but according to enabled toMock() function, the output will be different (see Figure 49).

Figure 48. Example of mocked method.

ATSTESTCLANG : function : function_Te	st1		Mock function	ns			-		×
Name	Type	S1			Enabled *				
Input arguments				1	chabica				
<ul> <li>Expected return value</li> </ul>	int	0	Mock function	testt	✓	2 return 99;			
- Global Variables	100	U	Method (Class)	nestedAlsoHere (ATSTESTCLANG)		1 { 2 return 99; 3 }			
Input Expected value									
<ul> <li>User Variables</li> </ul>			Method (Class)	nestShouldBeHere (ATSTESTCLANG)					
Input Expected value			Mock function	toMock	<b>v</b>				
Ingeotea faiae			Method (Class)	function (ATSTESTCLANG)					
			Mock function	setFlag_1					
			Method (Class)	mcdcTest (Mcdc)					
			Mock function	setFlag_2					
			Method (Class)	mcdcTest (Mcdc)					
			Mock function	setFlag_3					
			Method (Class)	mcdcTest (Mcdc)					
			Mock function	setFlag_4					
			Method (Class)	mcdcTest (Mcdc)					
			Mock function	setFlag_5					
			Add includes				ve changes	Disca	ard

The test output is "99" value instead of "1":

Figure 49. Mock function result from test report.

Catavary	Ture	Name	Sequence step
Category	Туре	Name	1
Expected return	int	-	0
Value returned	int	•	99

#### 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.

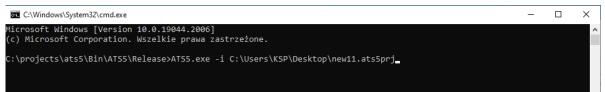
To select many tests from different classes and to make them execute, user can select particular checkboxes and then use the button to run them:

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. 50):

#### 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 50. 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 51).

Figure 51. Reports Tools

e to find	Reports		
Application General	Folders		
Compilation tools Appearance Company	Reports path		<u>^</u>
C++ Project Project Database	Image sub-folder		
CTC Reports	General		
Requirements Code Generator	Don't show again "Do you war	nt open generated report?" Auto-	generation after executing tests
	Charts		
	Chart width [px] 1200 🗘		
	Chart height [px] 72 🗘		
	Floating numbers		
	Precision type		
	O Auto	⊖ Set ma	anually
	Set precision 6		
	Type of saving photos to the repo	ort	
	only base64	<ul> <li>only image files</li> </ul>	<ul> <li>use base64 and save image to file</li> </ul>
	Type chart render		
	<ul> <li>only separated charts</li> </ul>	only summary chart	🔾 al
	sequence table type		
	<ul><li>auto</li></ul>	vertical	horizontal
	Struct of page (with the possibility	y to turn on / off)	
	✓ Table of content		
	✓ Images		
	✓ CodeArea		
	Sequences		
	✓ Code Coverage		

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 52. View tab

File Simulatio	on Code Coverage	View	Tools	Help		
CPP Tests	Code Generator	lings S	how/Hic	le Log outpu	t	
0			nable ch	arts		
🖌 🔆 🗸	) dil 🤃	🕞 C	lear Log		Alt+Shift+C	
Stubs viewer		<u>ເ</u>	estore vi	iew		Cir

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

It is possible to disable specific classes or global functions from tests execution – to do that, right-click on selected stub node (class or global function) in Stubs viewer and select option "*Disable class*" or "*Disable global function*" – it will make this element greyed out in the tree and all tests created for this stub will not be executed, neither they will be added to test report. If there are already some executed tests with results, all data will remain visible in the tree after this class was disabled.

Stubs viewer		ð
Name	Result	-
ATS_CppTestingPrj   ATS_CppTestingPrj_1  Enable this  Constructor  Destructor		
<ul> <li>♥ add</li> <li>□ T add_Test1</li> <li>♥ testANOT</li> </ul>	Failed	
E testAAND     T testAAND_Test1     G Class1	- Passed	
Constructor  Destructor  returnClass	-	
<ul> <li>□ T returnClass_Test1</li> <li>▼ fill returnClassPtr</li> <li>□ T returnClassPtr Test1</li> </ul>	Passed - Passed	
ATS_ClassDisabledConstructor     Constructor     Destructor		
<ul> <li>CB add</li> <li>T add_Test1</li> <li>(ii) add2</li> </ul>	- Passed	
T add2 Test1	- Passed	

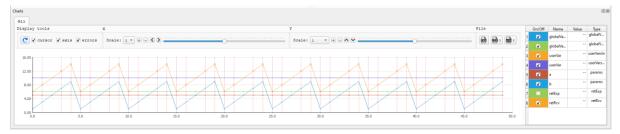
Figure 53. Disabled class in stubs viewer.

To restore elements again, just right-click on this stub in the tree and press *"Enable this class/global function"*.

#### **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 54. 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 55).

Figure 55.	Project	configuration	with	CTC options
------------	---------	---------------	------	-------------

e to find	Project		
Application General Compilation tools	Solution to test		<b>A</b>
Appearance Company C++ Project	Project to test		<b>^</b>
Project Database	Testing project	configuration	
CTC Reports Requirements	Release	O Debug	
Code Generator	SimuDllx64 path	C:\Users\Tests\Desktop\MCDCTests	Suggest
	Project tree		
	✔ Keep SimuDII co	nected	
	Show CTC warr	ing before tests	

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 56), 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. 57) 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 under this below figure.

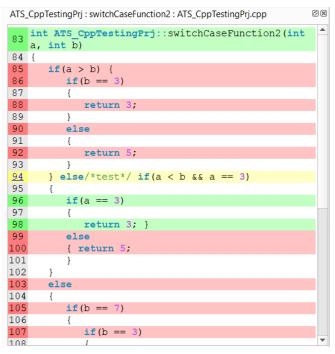
Figure 56. Main ATS5 window with CTC feature

CPP Tests Code Generator									
🗩 🖏 🛍 🐚 🕼 🕨		<b>A</b> Â A	Ă						
ubs viewer			ØX	ATS_CppTestingPrj : mcdcTest2			 ATS_CppTe	estingPrj : mcdcTest2 : A	TS @
ame	Result		Coverage *	Property Name	mcdcTest2	Value	 164 int	CppTestingPrj::: (int a, int b, int	mcdcT
Constructor				Method access	public		int		
Destructor				Desugerated return ty	pe int		165 {		
(tt) add	-	-	-	Function type	classMethod			if (a < b && b :	> c)
▼ I switchCaseFunction	-		40%	isConst	no		167 168	{ return b;	
switchCaseFunct.					a, int		169	}	
switchCaseFunct.				Parameters:	b, int c, int		170 }	·	
switchCaseFunct			1001	Parameters:	d, int				
<ul> <li>switchCaseFunction2</li> </ul>	-		19%		a, inc				
witchCaseFunct	Failed								
<ul> <li>mcdcTest1</li> <li>mcdcTest1</li> </ul>		2	42%						
T test toPassed	- Passe		2.70				Global Var	lables	
test_toFailed	Failed						Type	Name	Value
<ul> <li>mcdcTest2</li> </ul>	-		100%				1 int	globalVariable	
T test toPassed	Passe	d					1 And	groburturiubic	
T test1_toFailed	Failed						2 long	globalVariable2	2
T test2_toFailed	Failed						a dau	globalVariable3	
mcdcTest2WithParenthes	-		-				3 dou	grobarvariabres	<b>`</b>
💷 mcdcTest3	-	-	-				4 flo	globalVariable5	5
theckANDthenOR	-	-							
mcdcTest4	-		-						
(a) mcdcTest4WithParenthes									
	-								
III oneORinsideAND			-						
fourORbool									
	-	-	-						

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 57. 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 58) – 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.

	-						
	True	False			Condition		_
1 :		0	Тб				_
2	9	1	Τŵ	& F			_
3	9	8	F &	&			
	scription		Pairs				_
M	C/DC (c	ond 1):	1+3				
	C/DC (c	ond 2):	1 + 2	]			
M							
M							
M							

Figure 58. 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.</li>
- 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 59).

Figure 59. Example of MC/DC dialog – minus symbols..

⋪ MC/D	C Mcdc.cp	p:62			_	×
if (a    ( b	9 && c ))					
True	False			Condition		*
1 1	0	т II (_ &	<u>د _</u> )			
2 1	0	F    (T &	£Т)			
3 0	0	F    (T &	EF)			
40	0	F    (F &	<u>د _</u> )			
						*
Descripti	on P	Pairs				
		1-3 1-4				
	(cond 2):					
	(cond 3):					
						*

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 60. Importing/exporting buttons in class definition.

CPP Tests Code Generator				
🗩 🛈 🛍 🐚 🕼 🕨 🗄		i 🕼 🖫 🕼		
Stubs viewer		ATS_CppTestingPrj		 S_CppTestingPrj : ATS_CppTestingPrj.h @ 8
Name ATS_CppTestingPrj	Result	* Property Name Public construct	Value ATS_CppTestingPrj orl	class ATS_CppTestingFrj { public: //#ifdef TEST DEFINE
Constructor Destructor		File location: Template kind:	D:/projects/ats5/ATS5_DemoFroject/ATS5_DemoFroject/CopiedFiles/ATS_CppTestingFrj.h None	 // int invisibleFun(int a); //#endif
<ul> <li>T add_Test1</li> <li>24.09.18 08:30:54</li> </ul>	Passed			<pre>8 9 int add(int6 a, int b, int c); 10 int switchCaseFunction(int a, int b, int c);</pre>
T add_Test2     T add_Test11     witchCaseFunction	Failed			<pre>11 int switchCaseFunction2(int a, int b); 12 unsigned int adduint(unsigned int5 a, unsigned int b);</pre>
T switchCoseFunction Test	Danied			

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 61. Importing/exporting buttons in tests' parameters section.

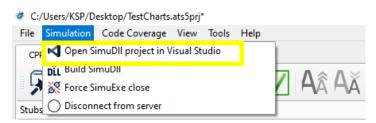
CPP Tests Code Generator											
🕞 👸 🛍 🗓 🖓 🕨		π									
Stubs viewer		88	ATS_CppTestingPrj : add : add_Test1				Sequence Length 2 + -		Global Variables		Ø
			Name	Type	S1	60			Type	Name	Value
Name	Result	-	- Input arguments	- 35-0		0.5		- e	int	globalVariable	5
- G ATS CppTestingPrj			a a a a a a a a a a a a a a a a a a a	int s	warTnt	*varIntPtr		- e	long	globalVariable2	2
Constructor			b	int	1	0			double	globalVariable3	
Destructor			c	int	1	0		60 L	float	globalVariable5	
Destructor     add			* Expected return value						const char *	globalek	6"asd"[0]
- (e) add	-	_		int	10	<=30		<b>60</b> †	ATS_CppTestingPrj	globalClass1	1
<ul> <li>T add_Test1</li> </ul>	Passed		<ul> <li>Global Variables</li> </ul>						ATS_CppTestingPrj_1	globalClass2 pointerInteger	1
24.09.18 08:30:54	Passed		- Input						int *	poincerinceder	1
T add_Test2	Passed		globalVariable	int	4				4		F
T add Test11	Failed		<ul> <li>Expected value</li> </ul>								-
- Isl sudichCareFunction			globalVariable	int	16				User Variables		2

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 62. Simulation – Open SimuDLL in VS.



#### Proper SimuDLL project configuration looks like this:

Configuration Properties	✓ General Properties	
General	Output Directory	<pre>\$(SolutionDir)Bin\\$(ProjectName)\\$(Configuration)\</pre>
Advanced	Intermediate Directory	<pre>\$(SolutionDir)Build\\$(ProjectName)\\$(Configuration)\</pre>
Debugging	Target Name	\$(ProjectName)
VC++ Directories	Configuration Type	Dynamic Library (.dll)
▷ C/C++	Windows SDK Version	10.0 (latest installed version)
▷ Linker	Platform Toolset	Visual Studio 2022 (v143)
▷ Manifest Tool	C++ Language Standard	ISO C++17 Standard (/std:c++17)
XML Document Generator	C Language Standard	Default (Legacy MSVC)
<ul> <li>Browse Information</li> <li>Build Events</li> <li>Custom Build Step</li> </ul>		

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 64 there are buttons marked in red, yellow, green and blue.

Figure 64. 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.

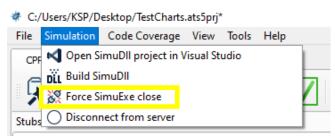
Buttons in red area are related to increasing/decreasing font size for constructor and destructor methods' definitions.

Buttons in yellow frame concern generating CTC and ATS reports.

Button in green frame concern adding new test for selected method/global function.

In case of a problem with executing tests, you can force SimuDLL to close. To do that, go to *Simulation – Force SimuExe close* (Figure 65).

Figure 65. 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 66. Tools tab

🏘 D:/projects/a	ts5/ATS5_DemoProject/	ATS5	_DemoProject.ats5prj*	
File Simulatio	n Code Coverage V	liew	Tools Help	
CPP Tests	Code Generator		📙 Load original source	
			Configuration	× E
🧏 🔆	DIL 🔚 👯	2	🔎 License tool	M m

#### **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.

Automation & Testing Suite for embedded software / AutoDate compatible All 2024.07.16		8 br case : 0 case : 0 case : 1 br 2 case : 3 if 4 ( 5 ) 7 el: 8 ( 9 ) 1 br 2 defaul 2 a: 5 ) 6 if (b ) 7 (	<pre>10; mk; i &gt; 2) a = 30; me // Comments 5 a = 30; mk; ib; id;</pre>							
_	21	9 ) O else :	f(b>a)							
Table of contents	33 33 33	2 27	= a - b;							
1 AT3_CopTestingPi 1 standtrCadeFundson 1.1.ftandtrCadeFundson_Test1	3	4 else 5 ( 6 rv 7 )	= a - a;							
ATS_CppTestingPrj	41	9 if(rv 0 { 1 if;	> 0) (zv > c)							
	4	2 {	zv = zv - c;							
switchCaseFunction	41	6 ) 5 el: 6 (								
	41	7	IN = C - IN;							
switchCaseFunction_Test1	41 51 51	0 ret	ourn rv;							
Result: Failed	5 5 5 5 5 5 5	2 clas	ourn e;							
<pre>Function Code i in \$22_pypErsing(py):====================================</pre>	54.00 40.25 24.50 8.75 -7.00	7 3		18	<b>.</b>	dlov 🖨 relia 🗋 rel 3.3	Err 🖬 ErrorPoints	53		10
15 a = 30; 16 )			-				Sequ	ince step		
16 ) 17 elle // Commans REQ_TEST2 18 (	C	ategory	Туре	Name	1	2	3	4 5	6	7
20 ) 21 break;	P	arameter	int	а	1	3	5	1 3	5	3
22 default: 23 a = 40;	P	arameter	int	ь	3	3	3	3 3	3	3
20 ) 21 default; 22 default; 23 a = = 0; 24 ] 25 ] 26 ] 27 ] 27 ] 28 ] 29 ] 29 ] 20	P	arameter	int	¢	2	2	2	2 2	2	-4
27 ( 28 zw = b - a;		ected return	int		10	10	10	• •		55
29 ) 20 else if( b> a)		e returned	int	-	1	2	2	1 2	2	-4
31 {	4.0									-

Figure 67. Report example.

## **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.

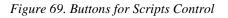
Figure 68. Code Generator basic button

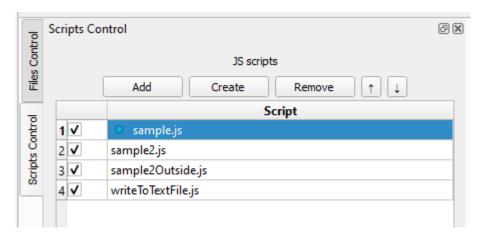


In Code Generator tab, there are 3 basic buttons. The first on the left is used to run all JS scripts, loaded to a project. The second in the middle is used to save single, selected script. And the last one saves all created or modified scripts.

#### 4.1. Scripts Control

In this tab you are able to load existing JS scripts and open them in application (Figure 69). To load scripts from your computer, click *Add* button. To create new JavaScript script, select *Create*.





If you would like to remove loaded .js file from the Script list, select checkbox near desired item and then click *Remove* button – please note, it will remove all elements which have set checkboxes to true. All scripts on a list will

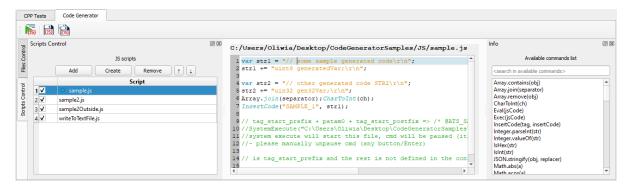
be executed in ascending order (from 1 to n). Down and up arrows buttons allow you to place particular .js file lower or higher on a list, which will cause changes in scripts execution order.

Besides removing actions, checkboxes are used also to enable or disable which files from the list should be executed while running the scripts – 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 70) 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 70) 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 70. 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

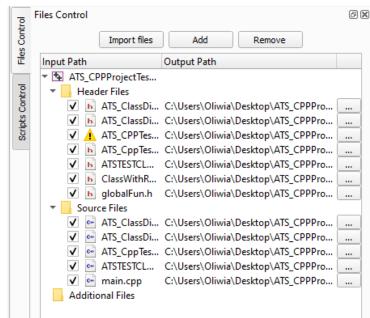
Files Control basic buttons are used for importing files to the project, adding new, single files or removing the selected one (Figure 70). Files to import can be selected from Visual Studio projects or added separately by user.

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

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.





Clicking Import files button opens new window with selecting VCXProj

Figure 72. Selecting file to import in Files Control

file.

		×
←		
Select VCXProj file		
Path to Visual Studio project:		
	<u>N</u> ext Cancel	

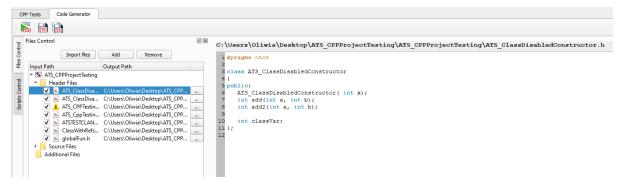
By going *"Next*", there is a window that allows you to choose files, that you would like to include in a project (Figure 73) – you can select single elements or whole folders. To include them, select the item and click the right arrow. It will move the content of the selected item to the right side *"included files*".

Figure 73. Selecting particular files to include in Files Control

iles to include:	Included files:
<ul> <li>ATS_CPPProjectTesting</li> <li>Header Files</li> <li>ATS_ClassDisabledConstruct</li> <li>ATS_ClassDisabledConstruct</li> <li>ATS_CPPTesting_Nested.h</li> <li>ATS_CPpTestingPrj.h</li> <li>ATSESTCLANG.h</li> <li>ClassWithRefsToPrimitives.h</li> <li>globalFun.h</li> </ul>	<ul> <li>ATS_CPPProjectTesting</li> <li>Source Files</li> <li>ATS_ClassDisabledConstruct</li> <li>ATS_ClassDisabledConstruct</li> <li>ATS_CppTestingPrj.cpp</li> <li>ATSTESTCLANG.cpp</li> <li>main.cpp</li> </ul>

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 74).

Figure 74. 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. Configuration - Saving data to MongoDb.	14
Figure 19. Code Coverage tab.	15
Figure 20. CTC Tools.	15
Figure 21. Defining constructors with non-params.	16
Figure 22. Information while recognizing constructor with parameters	16
Figure 23. Create default constructurs for structures	16
Figure 24. Successfully built DLL notification.	17
Figure 25. Adding tests via context menu.	
Figure 26. Additional options for test in Stubs Viewer	
Figure 27. Main View of ATS5 with added tests.	19
Figure 28. Setting wrong data type for a test parameter	19
Figure 29. Add user variable section	20
Figure 30. User variable structures constructor selection	21
Figure 31. User variables usage in test	22
Figure 32. Reference to global variable in test's parameter	

Figure 33. User variable as a pointer used in test's parameter	22
Figure 34. Setting or getting pointer user variable	23
Figure 35. Test sequences	23
Figure 36. Test description	24
Figure 37. Ranges in tests' parameters	25
Figure 38. Special operators in ranges	
Figure 39. Empty struct fields	26
Figure 40. Struct usage in test	27
Figure 41. Struct examples in test	27
Figure 42. Set struct variable	
Figure 43. Set Variable for struct	
Figure 44. Class objects usage in tests	29
Figure 45. Mock functions widget	
Figure 46. Additional includes for mock functions.	
Figure 47. Original method definition before mocking	
Figure 48. Example of mocked method	
Figure 49. Mock function result from test report	
Figure 50. Automatic mode for running tests	32
Figure 51. Reports Tools	
Figure 52. View tab	
Figure 53. Disabled class in stubs viewer.	
Figure 54. Charts section	35
Figure 55. Project configuration with CTC options	
Figure 56. Main ATS5 window with CTC feature	
Figure 57. Lines' colors in method definition	
Figure 58. Example of MC/DC dialog – plus symbols	
Figure 59. Example of MC/DC dialog – minus symbols	40
Figure 60. Importing/exporting buttons in class definition.	41
Figure 61. Importing/exporting buttons in tests' parameters section.	41
Figure 62. Simulation – Open SimuDLL in VS.	42
Figure 63. SimuDLL Configuration.	42
Figure 64. Toolbar additional features	43
Figure 65. Force SimuExe close	43
Figure 66. Tools tab	44

Figure 67. Report example	44
Figure 68. Code Generator basic button	45
Figure 69. Buttons for Scripts Control	45
Figure 70. Scripts Control view in Code Generator	46
Figure 71. Basic button in Files Control	47
Figure 72. Selecting file to import in Files Control	47
Figure 73. Selecting particular files to include in Files Control	48
Figure 74. Files Control view in Code Generator	48