

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

# ATS 05.03.001

# Basic Usage Manual

SCHLEISSHEIMER SOFT- UND HARDWAREENTWICKLUNG GMBH

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



## Contents

Chapter 1. Getting started
1.1. Project creation4
1.2. Opening existing project11
1.3. Removing project12
1.4. Saving project13
Chapter 2. Testing files
2.1. Building SimuDLL15
2.2. Adding new test
2.3. Modifying a test
2.3.1. Sequences24
2.3.2. Range values
2.3.3. Special operators27
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 Reports46
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.

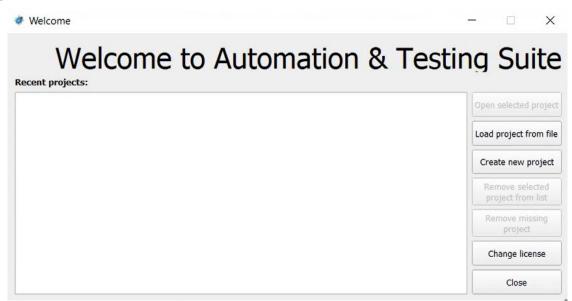
First, run the application. On your screen, the License Tool window will appear (Figure 1). You can import license by inputting the activation key (section on the right side) or by using remote license parameters (on the left). After successfully activating the license, application needs to be restarted.

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	Constant Key		
			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.

	P/Desktop/NewProject.ats5prj		-	σ	$\times$
File Simulatio	on Code Coverage View Tools Help	1			
	Code Generator				
🗩 👸	) DÜL 🐌 👯 🕨 🏷	🗸 🗛 🗛 🖻			
Stubs viewer		86			
Name		Result			
- Control		reade			
Log					0
2022-09-26 18:	15:11 Project correctly loaded. 15:11 2022-09-26 18:15:11 CodeGen: 0	Config report			
2022-09-26 18:1	15:14 Project configuration saved. 15:14 Project tree and tests saved suc				

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.

Configuration - Compilatio	n tools		
e to find	Compilation	cools	
Application General	MsBuild.exe	C:/Program Files/Microsoft Visual Studio/2022/Community/MSBuild/Current/Bin/amd64/MSBuild.e	Suggest
Compilation tools Appearance Company	Cl.exe	C:/Program Files/Microsoft Visual Studio/2022/Community/VC/Tools/MSVC/14.33.31629/bin/Host	Suggest
C++ Project Project Database CTC Reports			
Requirements Code Generator			
		Save configuration	Cancel

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):

ATS5	×	🗱 ATS5	
Import files for te	esting	Import files for te	esting
Import: Path: Import method: Additional includes path: Select language standard: Requirements prefix	Visual Studio Solution/Project <ul> <li>TS_CPPProjectTesting.sln</li> <li></li> <li>Replace existing stubs</li> <li><ul> <li><li><li><li><li><li><li><li><li><li></li></li></li></li></li></li></li></li></li></li></ul></li></ul>	Import: Add files: Import method: Additional includes path: Select language standard: Requirements prefix Selected source files tree: C/projects/at @ ATS_CinTe	Source Files  Append new items to tree  Append new items to tree  Replace existing stubs  C++14 (default)  REQ_INCLUDE  S5/Testing/ATS_CPPProjectTesting/ATS_CPPProjectTesting =  S5/Testing/ATS_CPPProjectTesting =  S5/Testing/ATS_S_CPPProjectTesting =  S5/Testing/ATS_S_CPPProjectTesting =  S5/Testing/ATS_S_S_S_S_S_S_S_S_S_S_S_S_S_S_S_S_S_S_
	< Back Next > Cancel	ATS_Class	sts.h DisabledConstructor.cpp DisabledConstructor.h DisabledConstructorWorkingParam.cpp DisabledConstructorWorkingParam.h Cancel

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).

 $\times$ 

Figure 6. Requirements list.

Requirements	list		-	×
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 General Compilation tools Appearance</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.

addFloat_Test1						
It: Passed						
unction Code						
2 {	::addFloat(float a, float b)					
3 return a + b; // REQ 4 }	_TEST5 REQ_INCLUDE1					
2.00 1		a b retRcv retErr ErrorPoint	s 🔳 Axis Y 🔳 Axis X			
4.00						
6.00						
8.00						
0.00					2.	
			Sequence	a atap		
Category	Туре	Name	1	2		
Parameter	float	a	11	2		
Parameter	float	b	11	2		
Expected return	float		22	- 4		
Value returned	float		22	4		
Tanao Fotannoa	nout	Value returned float - 22 4				
			Requir	ements Coverage		
			Requir	ements Coverage		
quirements Summary t	able		Requir	ements Coverage		
quirements Summary t Requirement name	able	age (%)	<b>Requir</b> Test name	rements Coverage		

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

Figure 9. Tools for adding/removing requirements.

NTS_CppTestingPrj : adduint : adduint	Test1		Sequence Length 1 +	Test Requirem	ents	Ø
Name	Type	Sequence Value		REQ_INCL	UDE1	
<ul> <li>Input arguments</li> </ul>				REQ_INCL	UDE2	
a	unsigned i					
b	unsigned i					
<ul> <li>Expected return value</li> </ul>						
	unsigned i				+	
<ul> <li>Global Variables</li> </ul>						
Input						
Expected value				Global Variable	es	Ø
<ul> <li>User Variables</li> </ul>				Ture	News	Malain
Input				Туре	Name	Value
Expected value				1 int	globalVariable	5
				2 long	globalVariable2	2
				3 double	globalVariable3	-

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.

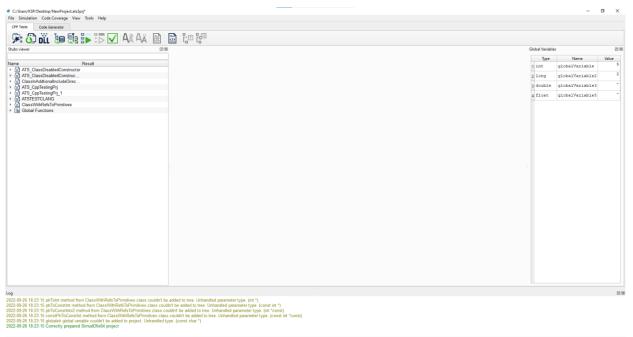
By going *"Next*", the application would show a selection section (Figure 10). In here please choose files, using checkboxes, that you would like to have in your project.

Figure 10. Selection section in CPP Tests.

ATS_CPPProjectTesting	
- Zhang Lang Kang Silan	
✓ 🖻 ATS_CinTests.h	
✓ ** ATS_ClassDisabledConstructorWorkingParam.cpp	
	<ul> <li>ATS_ClassDisabledConstructor.h</li> <li>ATS_ClassDisabledConstructorWorkingParam.h</li> <li>ATS_CCPPTesting_Nested.h</li> <li>ATS_CCPPTestingPrj.h</li> <li>ATS_CCPTEstingPrj.h</li> <li>ATSTESTCLANG.h</li> <li>ClassWithRefsToPrimitives.h</li> <li>ClassWithRefsToPrimitives.h</li> <li>globalFun.h</li> <li>Source Files</li> <li>** ATS_CinTests.cpp</li> <li>** ATS_ClassDisabledConstructor.cpp</li> </ul>

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.



## **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:

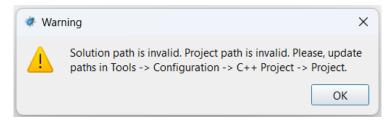
- opening selected project from Recent projects list (Figure 12), or
- loading a project from a file.

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

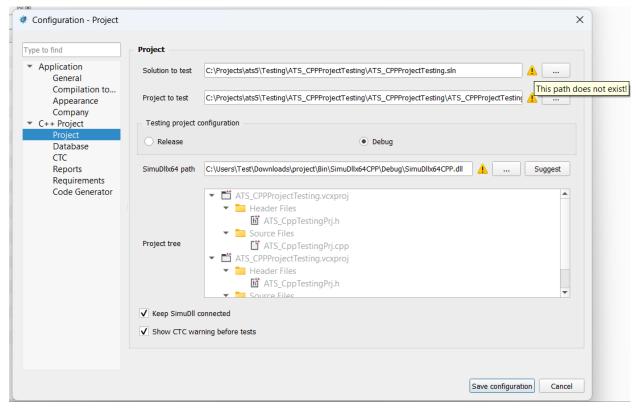
In case of opening existing ATS project from different device, you will need to adjust some paths to be able to use such project. Right after opening it, on the screen appears warning box with information which paths needs to be configured (Figure 13).

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"*.

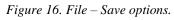
Figure 15. Remove missing project in Welcome Window.

🆸 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 branch ( ) (recenters). )	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.

to find	Database			
Application General Compilation tools Appearance Company	Save method MongoDb Save to file			
C++ Project Project Database CTC	Database URI	mongodb://localhost:27017		
Reports Requirements Code Generator	Database name		V	Add database
Code Generator	Database file	C:\Users\KSP\Desktop\NewProject\db		
	Method of savir			
	Append	C	) Override	

## **Chapter 2. Testing files**

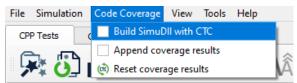
In this chapter, you will get to know how to build DLL, prepare your files for analysis, create tests and how to run them.

#### 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 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 19).

Figure 19. CTC Tools.

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

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 20.

Figure 20. Defining constructors with non-params.

4	ATS_CppTestingPrj : Construct
	1 ATS CppTestingPrj::construct()
	2 {
	<pre>3 this-&gt;object = new ATS_CppTestingPrj();</pre>
	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()
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 available.
8 };
```

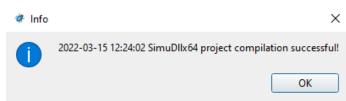
Also if parsed testing project contains structure without defined default constructor, ATS will recognize it and ask if user would like to create such default constructors (see Fig. 22).

Figure 22. Create default constructurs for structures.

ATS X	✤ Missing default constructors
Some errors occured while parsing files. Do you want to continue?	Some structures don't have default constructors.
Structure type "Testowa" doesn't have default constructor.	This may lead to errors of building SimuDLL.
	Would you like to add default constructors?
Yes No	<u>Y</u> es <u>N</u> o

After all is set up, successful building the DLL will display a dialog with confirmation (Figure 23). 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 23. Successfully built DLL notification.

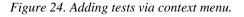


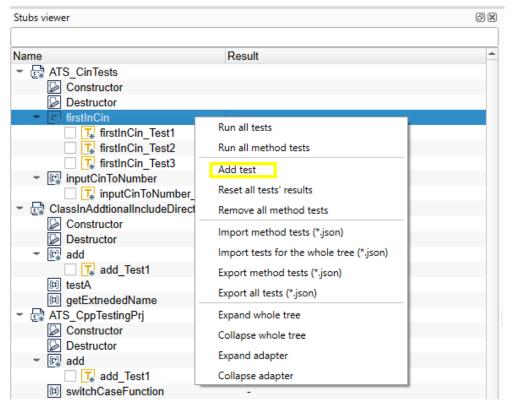
After choosing files to analyze and compile the DLL now you are ready to test them.

#### 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 24).



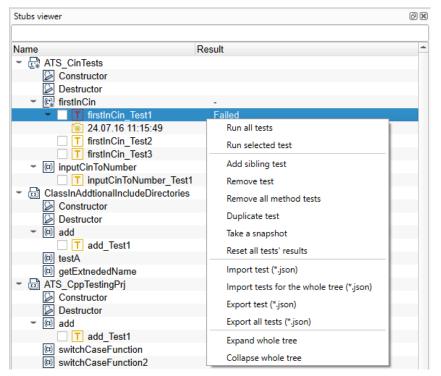


And the last, third option is to use the second button from the right side of a Toolbar "Add a new test to the method":

<mark>ני</mark>י⊡

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

Figure 25. 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 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 25).

#### 2.3. Modifying a test

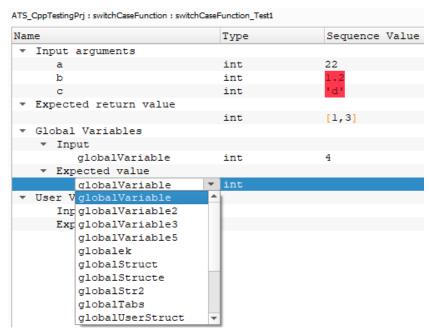
Clicking on test (tree item) shows a new window, that allows user to specify values for input arguments of methods/functions as well as expected return values (Figure 26).

Figure 26. Main View of ATS5 with added tests.

CPP Tests Code Generator								
🔎 🕂 🚺 🛍 🐌 🕅								
abs viewer	08					Global Variable		ß
los viewer	04	ATS_ClassDisabledConstructor : add : add	_Test1		Sequence Length 1 + *			
	Result	Name	Type	Sequence Value		Туре	Name	Value
me ATS_ClassDisabledConstructor	Result	* Input arguments				1 int	globalVariable	
Constructor		a .	int				globalVariable2	
Destructor		* Expected return value	int			2 long	globalvariable2	
Destructor		* Expected return value	int			3 double	globalVariable3	
T add Test1	-	- Global Variables	Inc			-		
		Input				4 float	globalVariable5	
T addPassed	-	Expected value						
B add2		<ul> <li>User Variables</li> </ul>						
ATS_ClassDisabledConstructorWorkingParam	-	Input						
Constructor		Expected value						
Destructor								
✓ III addToParam								
addToClassVar								
* IPI add2ToParam								
add2ToParam_Test1								
ClassInAddtionalIncludeDirectories								
CassimudutionalincibuleDirectories								
Constructor Destructor								
P add						Test Descriptio	0	6
add     letstA								
D getExtnededName								
ATS_CppTestingPrj								
Constructor								
Destructor								
Destructor								
T addTestRefParamFailed								
T addTestRefParamPassed						Test Requirem	ents	
* P switchCaseFunction								
T switchCaseFunctionPassed								
* I switchCaseFunction2								
T switchCaseFunction2Failed								
adduint								
▼ Ø addont								
T addDouble Test1								
* D addFloat								
T addFloatPassed								
T addFloatFailed								
* PitestA								
T testA_Test1								
badArrayNewLengthException								
bi unhandledException								
* P getExtendedName								
T getExtendedName_Test1								
T getExtendedName Test2								

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 27).

Figure 27. Setting wrong data type for a test parameter



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

Figure 28. Add user variable section

Туре	Name	Value	-
int	globalVariab	le	5
long	globalVariab	le2	2
double	globalVariab		-
float	globalVariab	le5	-
const char *	globalek		&"asd"[0]
int *	pointerInteg		-
Struktura	globalStruct		-
StrukturaDwa	globalStruct	e	-
StrukturaStr	globalStr2		-
hestedStruct	globalNested		-
nestedStruct	IwoDeepth globalNested	Struct2Deptn	- +
Test Requirements	Global Variables		
User Variables			രി
User Variables			@(
User Variables			Ø + -
	Туре	Value	
Name myName	int	4	
Name myName classTypePtr	int ATS_CinTests*		
Name myName classTypePtr myStructDefault	int ATS_CinTests* testStruct	4	
Name myName classTypePtr	int ATS_CinTests*	4 nullptr {3,2}	
Name myName classTypePtr myStructDefault	int ATS_CinTests* testStruct	4 nullptr {3,2} 3	
Name myName classTypePtr myStructDefault myStructParams x y	int ATS_CinTests* testStruct testStruct (int, int) int int	4 nullptr {3,2} 3 2	
Name myName classTypePtr myStructDefault myStructParams x	int ATS_CinTests* testStruct testStruct (int, int) int	4 nullptr {3,2} 3 2 &myStructDefault	0 + -
Name myName classTypePtr myStructDefault myStructParams x y structPtr classType	int ATS_CinTests* testStruct testStruct (int, int) int int	4 nullptr {3,2} 3 2	
Name myName classTypePtr myStructDefault myStructParams x y structPtr	int ATS_CinTests* testStruct testStruct (int, int) int int testStruct*	4 nullptr {3,2} 3 2 &myStructDefault	
Name myName classTypePtr myStructDefault myStructParams x y structPtr classType globalVariable	int ATS_CinTests* testStruct testStruct (int, int) int int testStruct* ATS_CinTests	4 nullptr {3,2} 3 2 &myStructDefault	•
Name myName classTypePtr myStructDefault myStructParams x y structPtr classType	int ATS_CinTests* testStruct testStruct (int, int) int int testStruct* ATS_CinTests	4 nullptr {3,2} 3 2 &myStructDefault classTypePtr[0]	•
Name myName classTypePtr myStructDefault myStructParams x y structPtr classType globalVariable	int ATS_CinTests* testStruct testStruct (int, int) int int testStruct* ATS_CinTests int	4 nullptr {3,2} 3 2 &myStructDefault classTypePtr[0]	

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:

			0
U	Jser Variables		@ (
			+ -
Ν	lame	Туре	Value
	myName	int	4
	classTypePtr	ATS_CinTests*	nullptr
	myStructDefault	testStruct	
1	<ul> <li>myStructParams</li> </ul>	testStruct (int, int)	{3,2}
L	x	int	3
	у	int	2
L	structPtr	testStruct*	&myStructDefault
	classType	ATS_CinTests	classTypePtr[0]
1	<ul> <li>nestedStr</li> </ul>	nestedStruct (int, nestedStructTwo	{3,,43}
	x	int	3
	ns	nestedStructTwoDeepth (char)	
	S	nestedStructTwoDeepth (char)	43
		nestedStructTwoDeepth	]

Figure 29. User variable structures constructor selection.

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 30).

Figure 30. User variables usage in test

			Туре		Name	
ame	Type	Sequence Value	int		globalVariable	-
Input arguments			long		globalVariable2	
a	int		4		groburtariabies	Þ
b	int					
c	int					_
Expected return value			Test Description			6
	int					
Global Variables						
Input						
Expected value						
User Variables						
* Input						
myName	int	4	User Variables			6
<ul> <li>myStruct</li> </ul>	UserStruct	{4,5}				
x	int	4			-	•
b	float	5		-		
<ul> <li>Expected value</li> </ul>			Name	Туре	Value	
myName	💌 int		myName	int	4	_
myName			<ul> <li>myStruct</li> </ul>	UserStruct	{4,3.2}	
myStruct			x	int	4 3.2	
myStruct2			b ▼ myStruct2	float Tabs	3.2 {2,2.0,new int[2],nullpt	
			* mystruct2	int	{2,2.0, new int[2], nullpt	1
			h	float	2.0	
			pointer	int*	new int[2]	
			nest	UserStruct *	nullptr	
			nex	e se se de la contra		

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 31).

Figure 31. Reference to global variable in test's parameter

Туре	Sequence V	Value
int &	globalVari	able
int	4	
Name		Value
Name globalVaria		Value 5
	int &	int & globalVari

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

Туре	Sequence Value
int &	*myIntPtr
int	2
int	3
int	4
	int & int int

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 33).

Figure 33. Setting or getting pointer user variable

r		
🔻 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 34). Also, you can modify the amount of sequences by putting its length number in the textfield.

Figure 34	Test	sequences
-----------	------	-----------

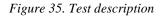
ATS_CppTestingPrj : switchCaseFunction : switch	hCaseFunction_Test8				Sequence Length 4 +
Name	Type	S1	S2	53	S4
<ul> <li>Input arguments</li> </ul>					
a	int	2			
b	int	421		34	
c	int	33			
<ul> <li>Expected return value</li> </ul>					
	int	22			
<ul> <li>Global Variables</li> </ul>					
<ul> <li>Input</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: 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 35).



	Description	(	5) 🗙
This is	s test description.		
Uro	r Variables Test Description		
Glob	al Variables	(	5) 🗙
Туре		Name	*
	int	globalVariab	
	int long	globalVariab globalVariab	
		-	
	long	globalVariab	
1 1 1 1	long double	globalVariab globalVariab	
9 9 9 9 9 9	long double float	globalVariab globalVariab globalVariab	
•	long double float const char *	globalVariab globalVariab globalVariab globalek	
•	long double float const char * Struktura	globalVariab globalVariab globalVariab globalek globalStruct	
* * *	long double float const char * Struktura StrukturaDwa	globalVariab globalVariab globalVariab globalek globalStruct globalStruct	
* * * *	long double float const char * Struktura StrukturaDwa StrukturaStr	globalVariab globalVariab globalVariab globalek globalStruct globalStruct globalStr2	
* * * *	long double float const char * Struktura StrukturaDwa StrukturaStr Tabs	globalVariab globalVariab globalVariab globalek globalStruct globalStruct globalStr2 globalTabs	
* * * * *	long double float const char * Struktura StrukturaDwa StrukturaStr Tabs UserStruct	globalVariab globalVariab globalVariab globalstruct globalStruct globalStr2 globalTabs globalUserSt globalNested	

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).

ATS_CppTestingPrj : switchCaseFunction : switchCaseFunction_Test9								
Name	Type	S1	S2					
<ul> <li>Input arguments</li> </ul>								
a	int	66	3					
b	int	[1,5,1]	3					
c	int	2	4					
<ul> <li>Expected return value</li> </ul>								
	int	<100	[1,50]					

Figure 36. Ranges in tests' parameters

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 37. 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 37. 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 38).

Figure 38. 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 37).

Figure 39. Struct usage in test

Nar	ne		Type	S1
-	Input	arguments		
	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
-	Expect	ed return value		
			void	
_	(C1 - 1 1	17		

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 39).

Figure 40. Struct examples in test

ne	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		
<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}
<pre>v nestnest</pre>	nestedStructTwoD	{'h'}
х	char	'h'
x	int	34
User Variables		
<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	{31,2,2,globalNestedStructPtr
x	int	31
b	float	2
te	unsigned long long	2

Additionally, user can set a value for current structure using option from context menu "Set variable" (Figure 41).

Figure 41. Set struct variable

Jame	Type	S1	
<ul> <li>Input arguments</li> </ul>			
a	int	3	
▼ b	Struktura	globalStruct	
x	int	-	Expand until non-empty sequence
b	float	-	Expand all empty sequences
te	unsigned long	-	Furnered and analyzes all assures and
nest	nestedStruct *	-	Expand and replace all sequences
<ul> <li>Expected return value</li> </ul>			Сору
	void		Paste
<ul> <li>Global Variables</li> </ul>			
<ul> <li>Input</li> </ul>			Set Variable
<ul> <li>globalUserStruct</li> </ul>	UserStruct	globalUserStruct	Restore default value
x	int		
b	float	-	
<ul> <li>Expected value</li> </ul>			
<ul> <li>globalNestedStruct</li> </ul>	nestedStruct	{{'h'},34}	
<ul> <li>nestnest</li> </ul>	nestedStructT	{'h'}	
x	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 42).

Figure 42. 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 41) – 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.

ATS_CinTests : firstInCin : firstInCin_Test2		Sequ	uence Length 3	+
Name	Туре	S1	S2	S3
Input arguments				
<ul> <li>Expected return value</li> </ul>				
	char	'h'		
👻 Global Variables				
Input				
Expected value				
<ul> <li>User Variables</li> </ul>				
- Input				
classType	ATS_CinTests	classTypePtr[0]		classType
<ul> <li>Expected value</li> </ul>	-			
classTypePtr	ATS_CinTests*	nullptr	*	

Figure 43. Class objects usage in tests

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:

Mock function	ns													-		×
		Enable	ed	•			t	oMoc	:k (v	void)						
Mock function	testt	✓			1 2				re	eturn	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												Save	e chan	ges	Disc	ard

Figure 44. Mock functions widget.

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 45).

Figure 45. Additional includes for mock functions.

🛷 ATS5	-		×	🛷 ATS5	-		×
#include <math.h></math.h>				#include			
				#include <math.h></math.h>			
+ - Sa	ive	Ok		+ - Sav	e	OK	۲

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 46. 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 48).

Figure 47. Example of mocked method.

ATSTESTCLANG : function : function_Test1 Name		S1							
Input arguments	Type	51			Enabled	0 int toMock(void)			
<ul> <li>Expected return value</li> </ul>	int	0	Mock function	testt	V	1 { 2 return	99:		
<ul> <li>Global Variables</li> <li>Input</li> </ul>	Inc	0	Method (Class)	nestedAlsoHere (ATSTESTCLANG)		3 }	,		
Expected value • User Variables			Method (Class)	nestShouldBeHere (ATSTESTCLANG)					
Input Expected value			Mock function	toMock	V				
Expected Value			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 ch	Disca	

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

Figure 48. Mock function result.



Catanani	Tune	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.

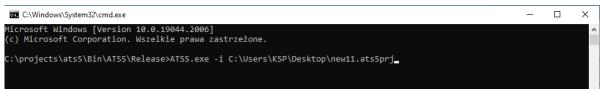
To select many tests from different classes and to make them execute, user can select particular checkboxes and then use the button to run them:

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. 49):

#### 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 49. 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 50).

```
Figure 50. 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	rt	
	only base64	$\bigcirc$ only image files	$\bigcirc$ use base64 and save image to file
	Type chart render		
	<ul> <li>only separated charts</li> </ul>	$\textcircled{\begin{subarray}{c} \bullet \end{array}}$ only summary chart	) al
	sequence table type		
	• auto	vertical	<ul> <li>horizontal</li> </ul>
	Struct of page (with the possibility	r to turn on / off)	
	✓ Table of content		
	✓ Images		
	✔ CodeArea		
	V 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 51. View tab

File Simulation Code Coverage	View Tools Help	
CPP Tests Code Generator	Show/Hide Log output	
	Enable charts	
🗡 🙆 DİL 🔚	🖥 🕼 Clear Log 🛛 🛛 Alt+Shift+	c 📘
Stubs viewer	Restore view	Cir

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 52. Charts section

Char	ts									68
	111 isplay tools	x		¥			File	On/Off	Name	Value Type
	C v cursor v axis v errors	Scale: 1 * + - < >		Scale:	1 * + - • •			1 🔽	globalVa	globalV
								2 🖌	globalVa	globalV
	16.00								userVar	userVarsIn
	12.00	$\Lambda$							userVar	uservars
								5 🔽	a	params
	8.00							6 🗸	o retExp	retExp
	4.00								retRcv	retRcv
	0.00								i conce	
	0.0 5.0	10.0 15.0	20.0	25.0	30.0 35.0	40.0 45	0 50.0			

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 53).

Figure 53. Project configuration with CTC options

🏘 Configuration - Project				×
Type to find	Project			
<ul> <li>Application</li> <li>General</li> </ul>	Solution to test			<u> </u>
Compilation tools Appearance Company C++ Project	Project to test			Â
Project Database	Testing project	configuration		
CTC Reports Requirements	Release		O Debug	
Code Generator	SimuDllx64 path	C:\Users\Tests\Desktop\MCDCTests		Suggest
	Project tree			
	✓ Keep SimuDI cor			
	Show CTC warn	ing before tests		
				Save configuration Cancel

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 54), when CTC option is set on, there are added some new features, such as:

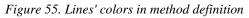
- In the *Stubs' viewer* (on the left side of the screen below in the yellow frame) there is added a new column "Coverage" which displays the percentage of code coverage. If it shows 100% it means that all lines of the code have been tested.
- Coloring the lines (Fig. 55) in the method's definition on the right side (yellow frame), the colors of the lines have different meanings. Explanation of them will be given under this below figure.

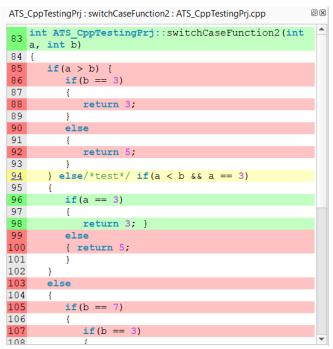
Figure 54. Main ATS5 window with CTC feature

CPP Tests Code Generator								
🗩 🖏 dii 🐚 🕼 🌗		Aâ Aă 🖻						
ubs viewer		03	ATS_CppTestingPrj : mcdcTest2			 ATS_CppTe	estingPrj : mcdcTest2 : Al	TS @
			December		Value	 int		
ime	Result	Calls Coverage	Name	mcdcTest2	T G ± G G	164 ATS	CppTestingPrj:::	mcdc
ATS_CppTestingPrj		12 24%	Method access	public			(int a, int b, in	nt c
Destructor			Desugerated return 1	•		int	d)	
Destructor						165 { 166	if (a < b & & b :	> a)
(III) add	-		Function type	classMethod		167	I (a C D && D )	× C)
<ul> <li>switchCaseFunction</li> </ul>	-	3 40%	isConst	no		168	return b;	
switchCaseFunct				a, int b, int		169	}	
switchCaseFunct	Failed		Parameters:	c, int		170 }		
SwitchCaseFunct			Parameters:	d, int				
<ul> <li>switchCaseFunction2</li> </ul>	-	1 📕 19%		a, inc				
SwitchCaseFunct	Failed							
(t) mcdcTest	-	2 42%						
▼ Image: weight wei	-					Global Var	lables	
test_toPassed	Passe	d				Type	Name	Valu
test_toFailed	Failed	3 100%						Val
mcdcTest2 T test toPassed	- Passe					1 int	globalVariable	
test1 toFailed	Failed	U				2 long	globalVariable2	
T test2 toFailed	Failed					2 20119	growartariabiob	·
mcdcTest2WithParenthes	ralled					3 dou	globalVariable3	1
mcdcTest3	-							
CheckANDthenOR						4 110	globalVariable5	1
(t) mcdcTest4	-							
mcdcTest4WithParenthes								
oneORinsideAND								
(#) fourORbool								
(iii) adduint	-							
▼ ttp addDouble		3 100%						
T test1 withoutMCDC	Failed							

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).





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 56) – 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 56. Example of MC/DC dialog – plus symbols.

	MC/DC	Mcdc.cp	p:33		-	×
if	(a < b &&	kb > c)				
	True	False		Condition		
1	1	0	таат			
2	0	1	T && F			
3	0	8	F && _			
						-
	escriptio		airs			•
N	IC/DC (c	n P ond 1): ond 2):	1 + 3			

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 57).

Figure 57. Example of MC/DC dialog – minus symbols..

⋪ MC/D	C Mcdc.cp	p:62		_	$\times$
if (a    ( b	&& c ))				
True	False		Condition		
1 1	0	T    (_ && _)			
2 1	0	F    (T && T)			
3 0	0	F    (T && F)			
4 0	0	F    (F && _)			
					*
Descripti	on P	airs			
MC/DC (	(cond 1):	1-3 1-4			
MC/DC (	(cond 2):	2 - 4			
MC/DC (	(cond 3):	2 - 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 58. Importing/exporting buttons in class definition.

CPP Tests Code Generator				
🔎 💭 📖 🐚 💱 🖥	🖌 🔽 🗛 🖾 🖻			
Stubs viewer	ØX	ATS ClassDisabledConstructor		*
N	Dk	Property	Value	- 📾 i )
Name	Result	Name	ATS_ClassDisabledConstructor	
<ul> <li>ATS_ClassDisabledConstructor</li> </ul>		Public constructor	0	
Constructor		File location:	C:/projects/ats5/Testing/ATS CPPProjectTesting/ATS CPPProjectTesting/ATS ClassDisabledConstructor.h	
Destructor				_
add [1]		Template kind:	None	
add2	-			

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

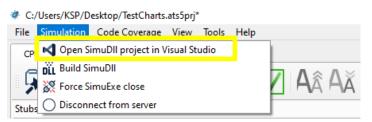
CPP Tests Code Generator							
🗩 🖏 🛍 🌬 🐘 🔽 🗚 🗚 🗟 🖆 🕼							
Stubs viewer Ø 🖉	ATS_ClassDisabledConstructor : add : ad	ld_Test1		Sequence Length 1 +	_		
Name Result	Name	Type	Sequence Value		ài		
	<ul> <li>Input arguments</li> </ul>						
<ul> <li>ATS_ClassDisabledConstructor</li> </ul>	a	int			n n		
Constructor	b	int					
☑ Destructor ✓ [2] add -	* Expected return value			6	ĝ i 🛛		
* 🚱 add -		int					
add Test1	<ul> <li>Global Variables</li> </ul>				Ø1		
(II) add2 -	Input				-		

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 60. Simulation – Open SimuDLL in VS.



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

Figure 61. SimuDLL Configuration.

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     Browse Information	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 62 there are buttons marked in red, yellow, green and blue.

```
Figure 62. 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).

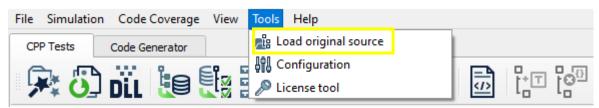
In case of a problem with executing tests, you can force SimuDLL to close. To do that, go to *Simulation – Force SimuExe close* (Figure 63).

C:/Users/KSP/Desktop/TestCharts.ats5prj\*
 File Simulation Code Coverage View Tools Help
 CPF Open SimuDII project in Visual Studio
 Build SimuDII
 Force SimuExe close
 Stubs O Disconnect from server

Figure 63. 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 64. 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.

	4	8 bre 9 case 2	2; k; // REQ_TI 10;	8371						
ALS	l	11 bre 12 case 1 13 if 14 (	ak; 2: (a > 2) a = 20; ce // Comment	REQ_TEST2						
Automation & Testing Suite for embedded software / AUTOSAR compatible	L	20 ) 21 bre 22 defaul 23 a = 24 25 )	40;							
All_2024.07.16		26 if(b) 27 { 28 rv 29 }	> a) = b - a; if ( b> a)							
Table of contents		31 ( 32 rv 22 )	= = - b;							
14TE_CopTextrePH 1 studtoCastFordion 1.11sectasd=minion_Text	L	34 else 35 ( 36 rv 37 ) 38 39 if(rv	= a - a;							
ATS_CppTestingPrj	L	40 { 41 if 42 { 43	<pre>xx = xx - c; (xx &gt; c) &gt; 0)</pre>							
switchCaseFunction		45 el: 46 ( 47								
switchCaseFunction_Test1		49 50 ret 51 )	burn rv;							
Result: Failed		48 } 19 50 ret 51 } 52 else 53 { 54 ret 55 } 56	turn c;							
Partian Code 1 inn Mi2-CppTestingPrj::ewithClaseFunction(inn a, inn b, inn o) 2 ( 2 int or on o; 4 int option ()		57 }			<b>B a D b E c E m</b>	no aretop are	Dr 🖬 DroiPoints			
<pre>5 ( 6 case 1: 7</pre>		56.00 40.25 24.50 8.75								
10 * * 10; 11 * * 10; 12 * * 10; 13 * * 10; 14 * ( * 2) * ( * 2); 14 * ( * 2); 15 * ( * 2); 15 * ( * 2); 16 * ( * 2); 17 * ( * 2); 18 * ( * 2); 19 * ( * 2); 19 * ( * 2); 19 * ( * 2); 10 * ( * 2); 1		-7.00		1.8		3.5		53		7.0
14 ( 15 a = 30; 16 3							Seque	nce step		_
17 else // Comment REQ_TEST2 18 (		Category	Туре	Name	1	2		4 5	8	7
19 a = 30; 20 ) 21 break;		Parameter	int	а	1	3	5	1	3 5	3
22 default: 23 a = 40;		Parameter	int	ь	3	3	3	3	3 3	3
25 )		Parameter	int	c	2	2	2	2	2 2	-4
27 (		Expected return	int	-	10	10	10	•	• •	55
29 ) 30 elemif(b>a)	цI	Value returned	int		1	2	2	1	2 2	-4
31 ( All 2024 07 16 html		4								

Figure 65. 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.

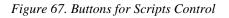
Figure 66. Code Generator basic button

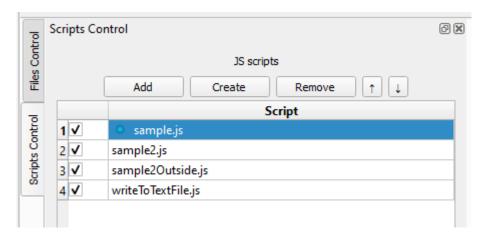


In Code Generator tab, there are 3 basic buttons. The first on the left is used to run all JS scripts, loaded to a project. The second in the middle is used to save single, selected script. And the last one saves all created or modified scripts.

#### 4.1. Scripts Control

In this tab you are able to load existing JS scripts and open them in application (Figure 67). To load scripts from your computer, click *Add* button. To create new JavaScript script, select *Create*.





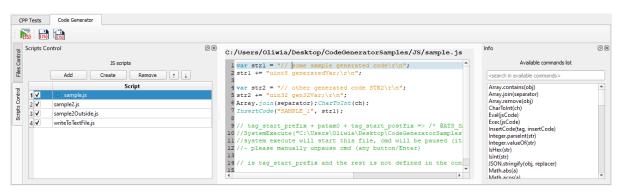
If you would like to remove loaded .js file from the Script list, select the item and then click *Remove* button. All scripts on a list will be executed in ascending order (from 1 to n). Down and up arrows buttons allow you to place

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 67) 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 68) 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 68. 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 69). 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.

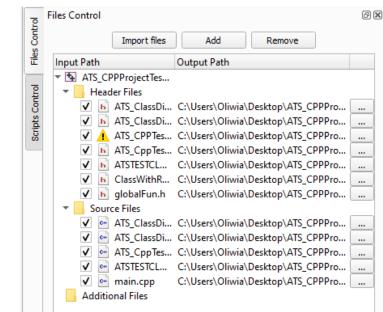


Figure 69. Basic button in Files Control

Clicking Import files button opens new window with selecting VCXProj

file.

Figure 70. Selecting file to import in Files Control

	×
~	
Select VCXProj file	
Path to Visual Studio project:	
	Next Cancel

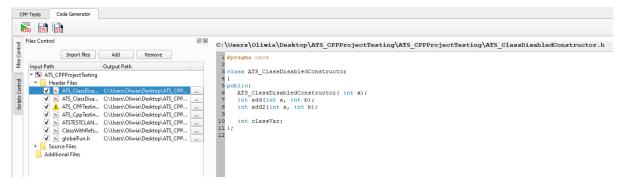
By going *"Next*", there is a window that allows you to choose files, that you would like to include in a project (Figure 71) – you can select single elements or whole folders. To include them, select the item and click the right arrow. It will move the content of the selected item to the right side *"included files*".

Figure 71. 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 72).

Figure 72. 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.	16
Figure 20. Defining constructors with non-params.	16
Figure 21. Information while recognizing constructor with parameters	17
Figure 22. Create default constructurs for structures	17
Figure 23. Successfully built DLL notification.	17
Figure 24. Adding tests via context menu.	18
Figure 25. Additional options for test in Stubs Viewer	19
Figure 26. Main View of ATS5 with added tests.	20
Figure 27. Setting wrong data type for a test parameter	20
Figure 28. Add user variable section	21
Figure 29. User variable structures constructor selection	21
Figure 30. User variables usage in test	22
Figure 31. Reference to global variable in test's parameter	23
Figure 32. User variable as a pointer used in test's parameter	23

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

Figure 67. Buttons for Scripts Control	.47
Figure 68. Scripts Control view in Code Generator	.48
Figure 69. Basic button in Files Control	.49
Figure 70. Selecting file to import in Files Control	.49
Figure 71. Selecting particular files to include in Files Control	.50
Figure 72. Files Control view in Code Generator	.50