Automation & Testing Suite

for embedded software / AUTOSAR compatible

# ATS 5.1.0

Basic Usage Manual

SCHLEISSHEIMER SOFT- UND HARDWAREENTWICKLUNG GMBH

www.automation-testing-suite.com

www.schleissheimer.com

H.-J. Schleißheimer

Soft- und Hardware Entwicklung GmbH

2022

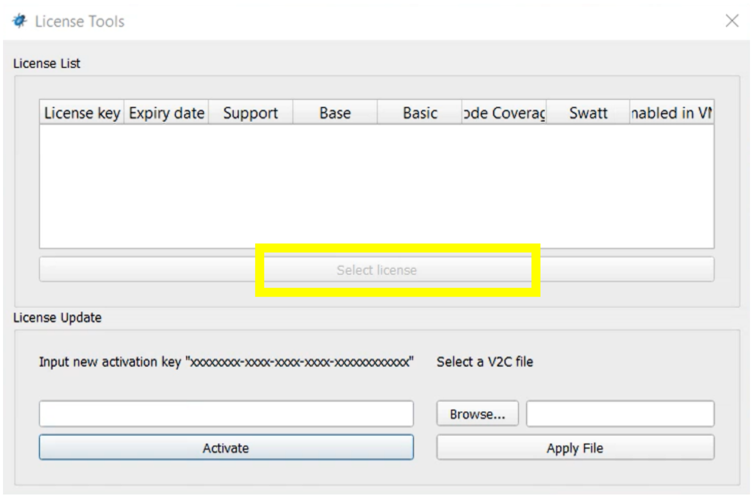
# Contents

# Chapter 1. Getting started

In this chapter, you will be introduced with activating the license, creating new project and opening the existing one, as well as saving it.

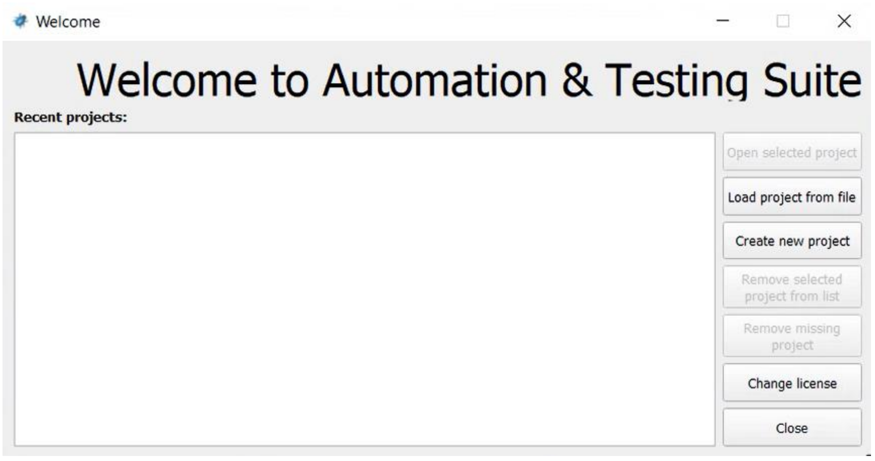
First, run the application. On your screen, the License Tool window will appear (Figure 1). You can import license by loading a V2C file or by inputting the activation key. After successfully activating the license, application needs to be restarted.

*Figure 1. License Tools*



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

*Figure 2. Welcome Window*



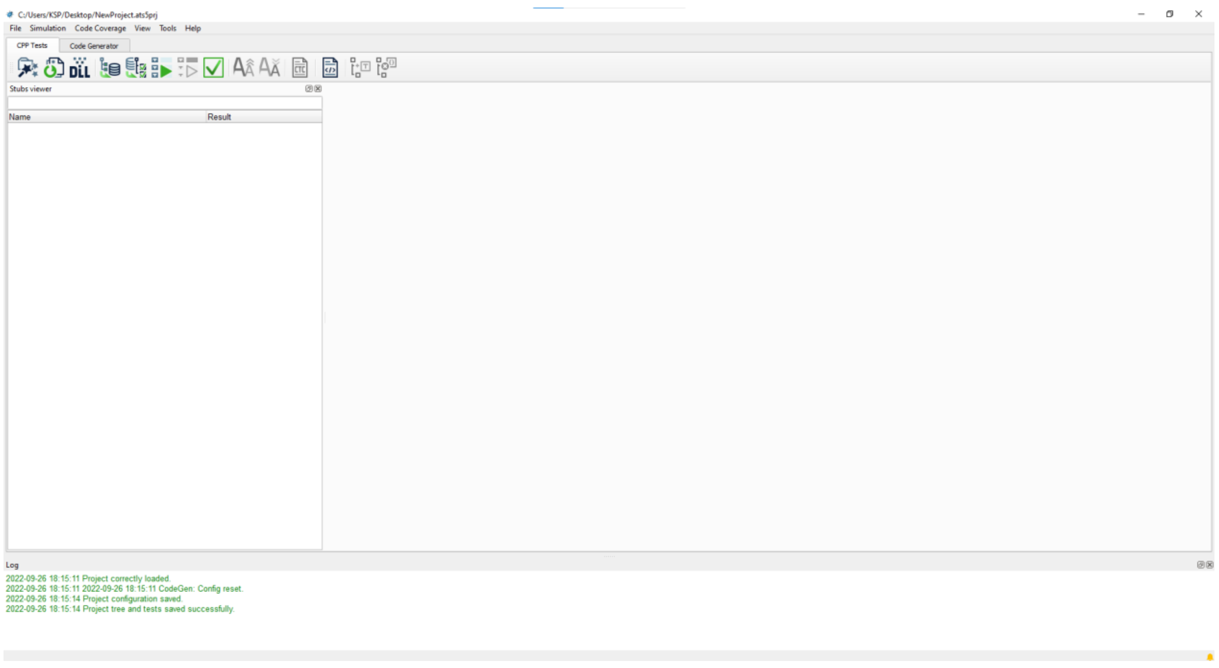
With this window, you are able to:

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

## 1.1. Project creation

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

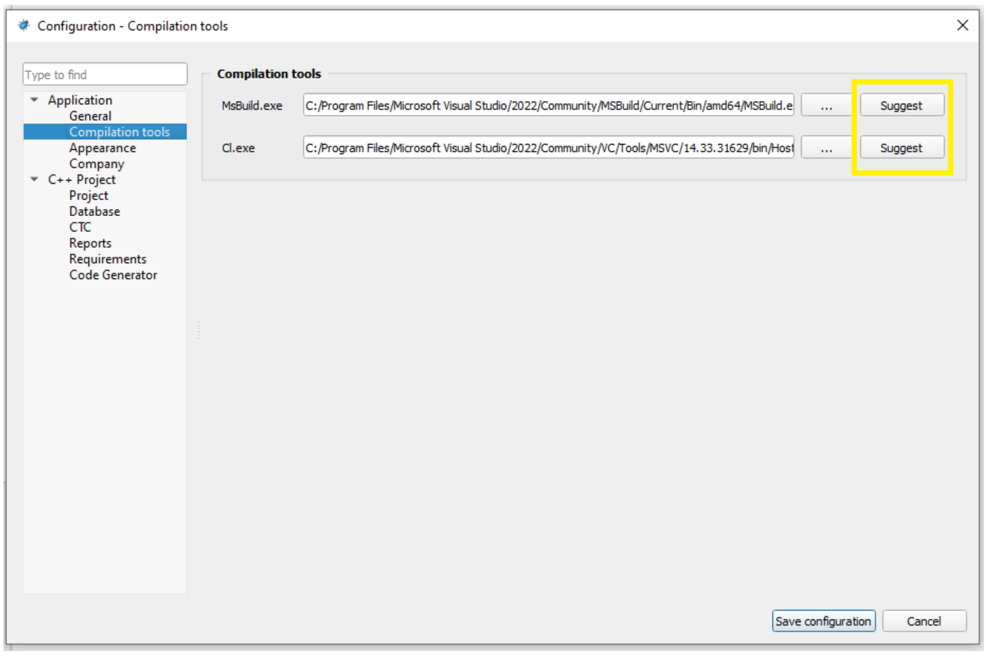
*Figure 3. Main View of ATS5.*



```
2022-09-26 18:15:11 Project correctly loaded.
2022-09-26 18:15:11 2022-09-26 18:15:11 CodeGen: Config reset.
2022-09-26 18:15:14 Project configuration saved.
2022-09-26 18:15:14 Project tree and tests saved successfully.
```

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.*



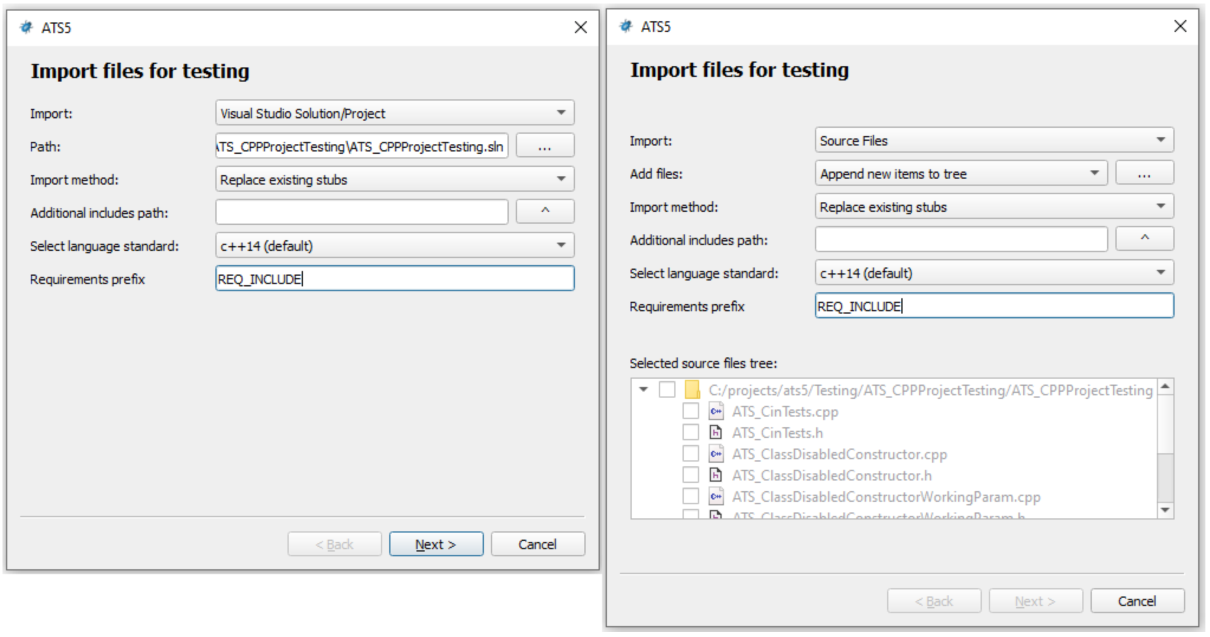
Compilation tools

MsBuild.exe: C:/Program Files/Microsoft Visual Studio/2022/Community/MSBuild/Current/Bin/amd64/MSBuild.e

Cl.exe: C:/Program Files/Microsoft Visual Studio/2022/Community/VC/Tools/MSVC/14.33.31629/bin/Host

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 key shortcut):

*Figure 5. Importing files to CPP Tests.*



ATS5

**Import files for testing**

| | |
|---|---|
| Import: | Visual Studio Solution/Project |
| Path: | \TS_CPPProjectTesting\ATS_CPPProjectTesting.sln |
| Import method: | Replace existing stubs |
| Additional includes path: | |
| Select language standard: | c++14 (default) |
| Requirements prefix | REQ_INCLUDE |

ATS5

**Import files for testing**

| | |
|---|---|
| Import: | Source Files |
| Add files: | Append new items to tree |
| Import method: | Replace existing stubs |
| Additional includes path: | |
| Select language standard: | c++14 (default) |
| Requirements prefix | REQ_INCLUDE |

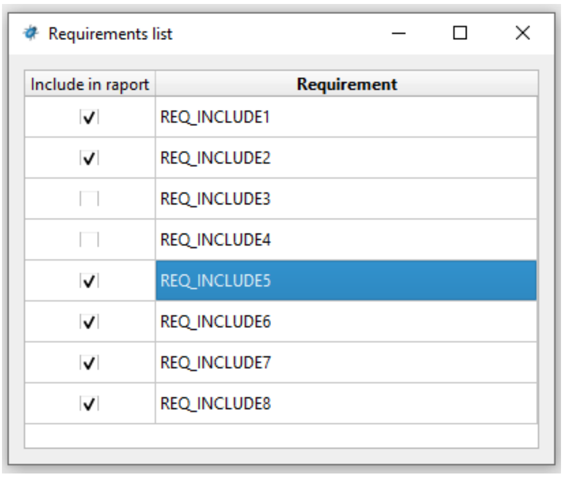
Selected source files tree:

- C:/projects/ats5/Testing/ATS_CPPProjectTesting/ATS_CPPProjectTesting
  - ATS_CinTests.cpp
  - ATS_CinTests.h
  - ATS_ClassDisabledConstructor.cpp
  - ATS_ClassDisabledConstructor.h
  - ATS_ClassDisabledConstructorWorkingParam.cpp

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.*

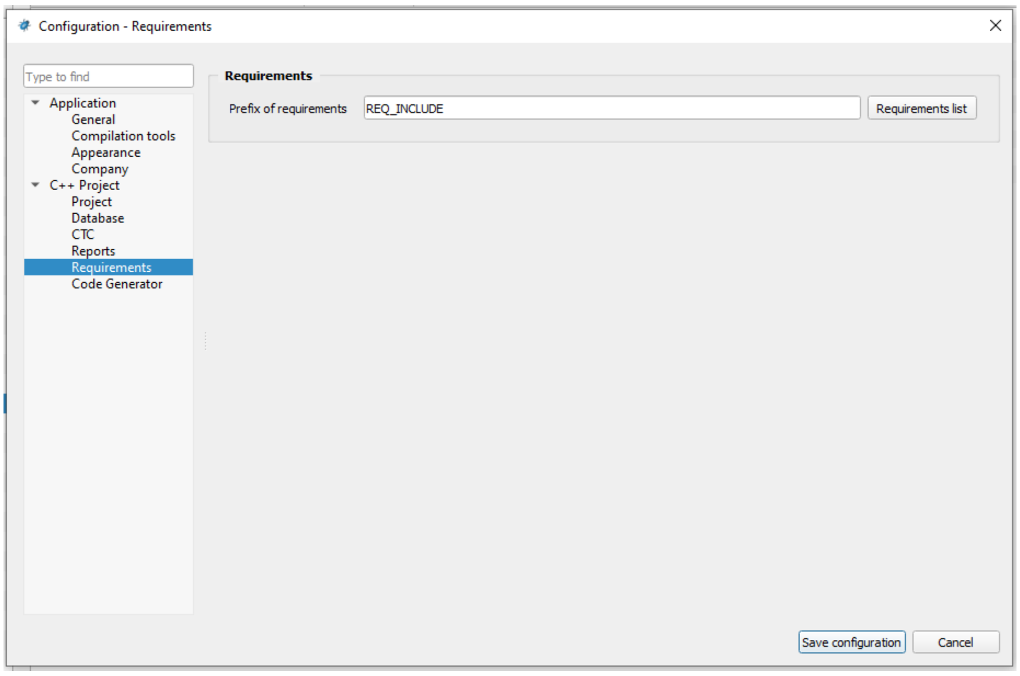


Requirements list

| Include in raport | Requirement |
|---|---|
| ✓ | REQ_INCLUDE1 |
| ✓ | REQ_INCLUDE2 |
| | REQ_INCLUDE3 |
| | REQ_INCLUDE4 |
| ✓ | REQ_INCLUDE5 |
| ✓ | REQ_INCLUDE6 |
| ✓ | REQ_INCLUDE7 |
| ✓ | 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.

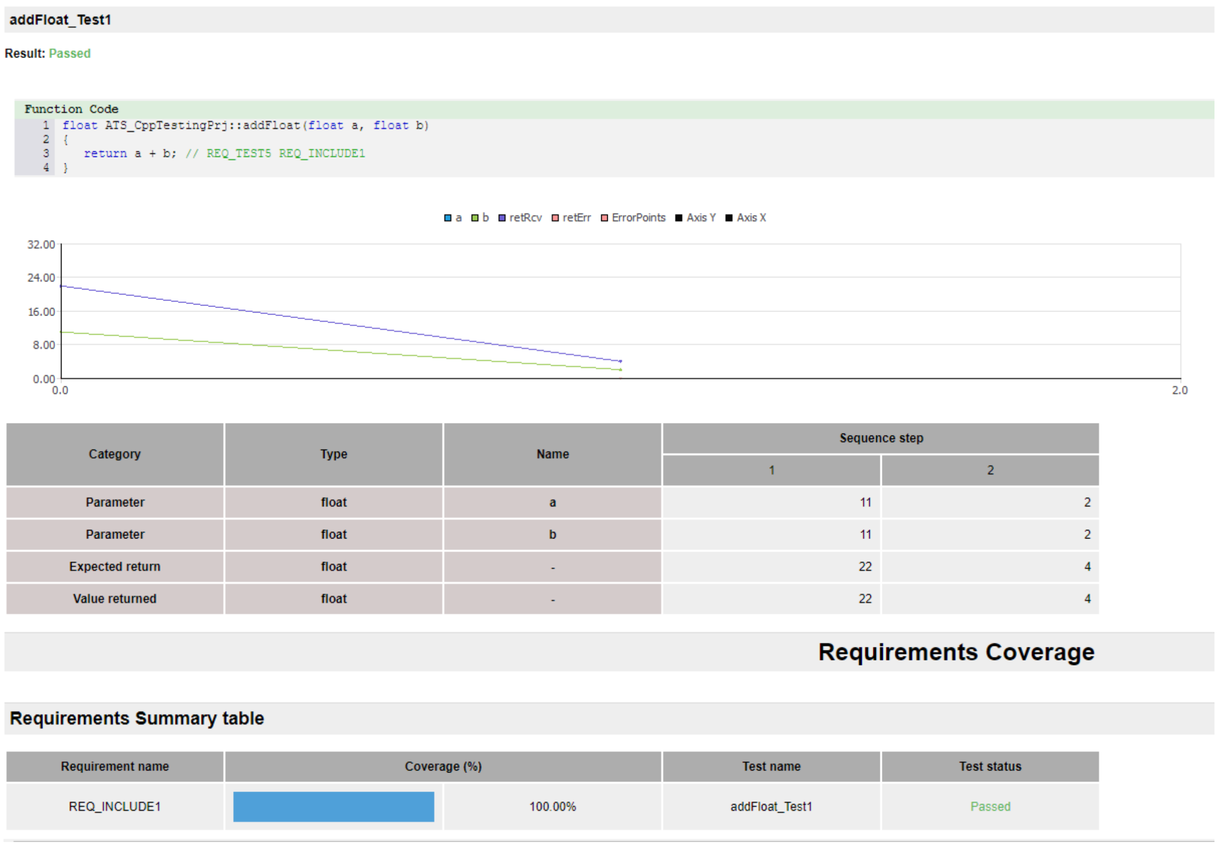
*Figure 7. Requirements in Configuration.*



Prefix of requirements: REQ_INCLUDE

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**

**Result:** Passed

```
Function Code
float ATS_CppTestingPrj::addFloat(float a, float b)
{
   return a + b; // REQ_TEST5 REQ_INCLUDE1
}
```

| Category | Type | Name | Sequence step | |
|---|---|---|---|---|
| | | | 1 | 2 |
| Parameter | float | a | 11 | 2 |
| Parameter | float | b | 11 | 2 |
| Expected return | float | - | 22 | 4 |
| Value returned | float | - | 22 | 4 |

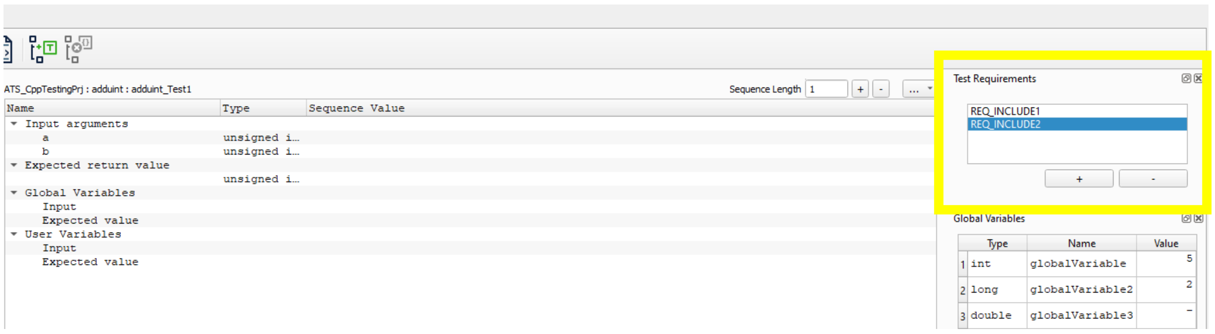
**Requirements Coverage**

**Requirements Summary table**

| Requirement name | Coverage (%) | | Test name | Test status |
|---|---|---|---|---|
| REQ_INCLUDE1 | 🟦 | 100.00% | 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.*



ATS_CppTestingPrj : adduint : adduint_Test1

Sequence Length 1

```
Name                      Type           Sequence Value
▾ Input arguments
     a                    unsigned i...
     b                    unsigned i...
▾ Expected return value
                          unsigned i...
▾ Global Variables
     Input
     Expected value
▾ User Variables
     Input
     Expected value
```

Test Requirements

REQ_INCLUDE1
REQ_INCLUDE2

Global Variables

| | Type | Name | 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.

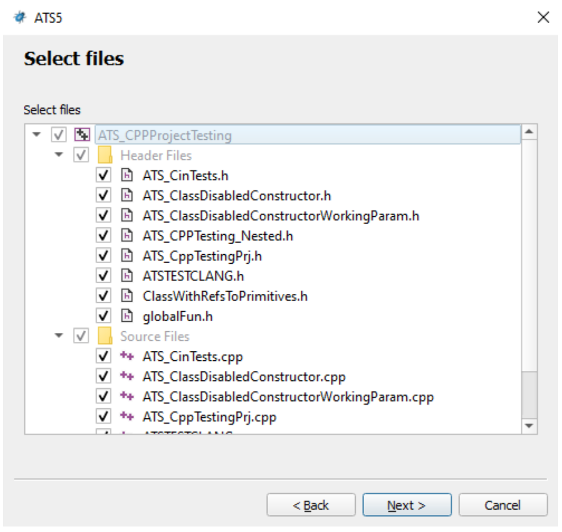
To describe and clarify the ways of importing files, please get familiar with this information:

- 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.*



ATS5

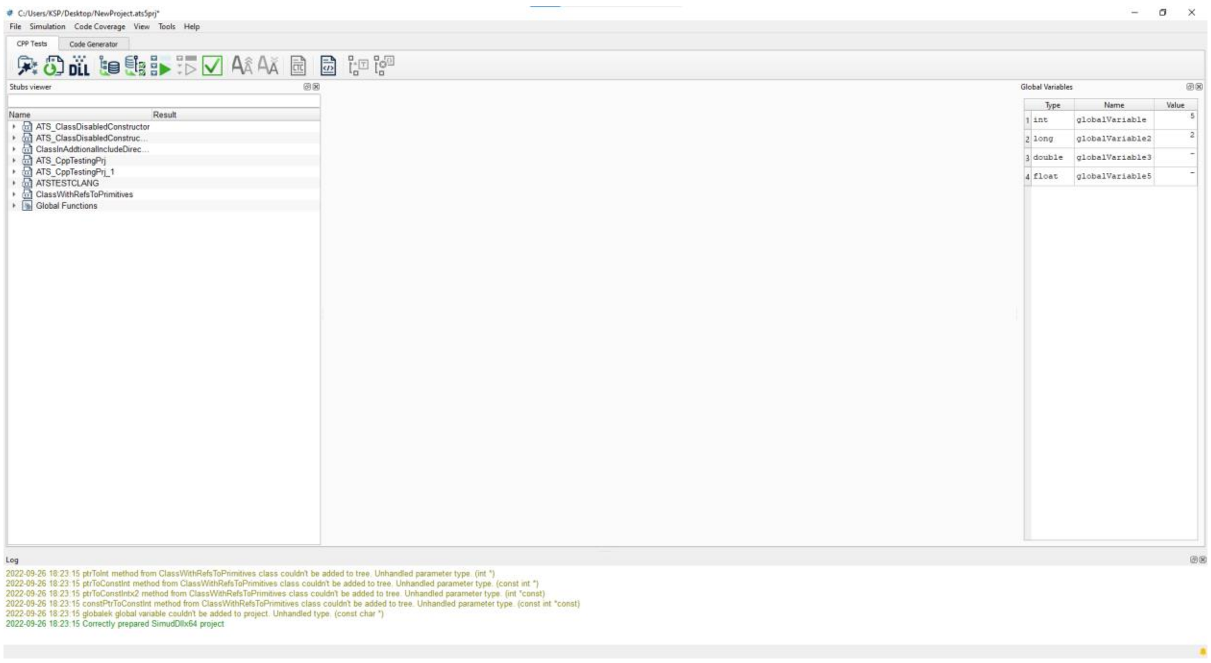
**Select files**

Select files

- ATS_CPPProjectTesting
  - Header Files
    - ATS_CinTests.h
    - ATS_ClassDisabledConstructor.h
    - ATS_ClassDisabledConstructorWorkingParam.h
    - ATS_CPPTesting_Nested.h
    - ATS_CppTestingPrj.h
    - ATSTESTCLANG.h
    - ClassWithRefsToPrimitives.h
    - globalFun.h
  - Source Files
    - ATS_CinTests.cpp
    - ATS_ClassDisabledConstructor.cpp
    - 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:/Users/KSP/Desktop/NewProject.ats5prj*

Stubs viewer

| Name | Result |
|---|---|
| ATS_ClassDisabledConstructor | |
| ATS_ClassDisabledConstruc... | |
| ClassInAddtionalIncludeDirec... | |
| ATS_CppTestingPrj | |
| ATS_CppTestingPrj_1 | |
| ATSTESTCLANG | |
| ClassWithRefsToPrimitives | |
| Global Functions | |

Global Variables

| | Type | Name | Value |
|---|---|---|---|
| 1 | int | globalVariable | 5 |
| 2 | long | globalVariable2 | 2 |
| 3 | double | globalVariable3 | - |
| 4 | float | globalVariable5 | - |

Log

```
2022-09-26 18:23:15 ptrToInt method from ClassWithRefsToPrimitives class couldn't be added to tree. Unhandled parameter type. (int *)
2022-09-26 18:23:15 ptrToConstInt method from ClassWithRefsToPrimitives class couldn't be added to tree. Unhandled parameter type. (const int *)
2022-09-26 18:23:15 ptrToConstIntx2 method from ClassWithRefsToPrimitives class couldn't be added to tree. Unhandled parameter type. (int *const)
2022-09-26 18:23:15 constPtrToConstInt method from ClassWithRefsToPrimitives class couldn't be added to tree. Unhandled parameter type. (const int *const)
2022-09-26 18:23:15 globalek global variable couldn't be added to project. Unhandled type. (const char *)
2022-09-26 18:23:15 Correctly prepared SimudDllx64 project
```

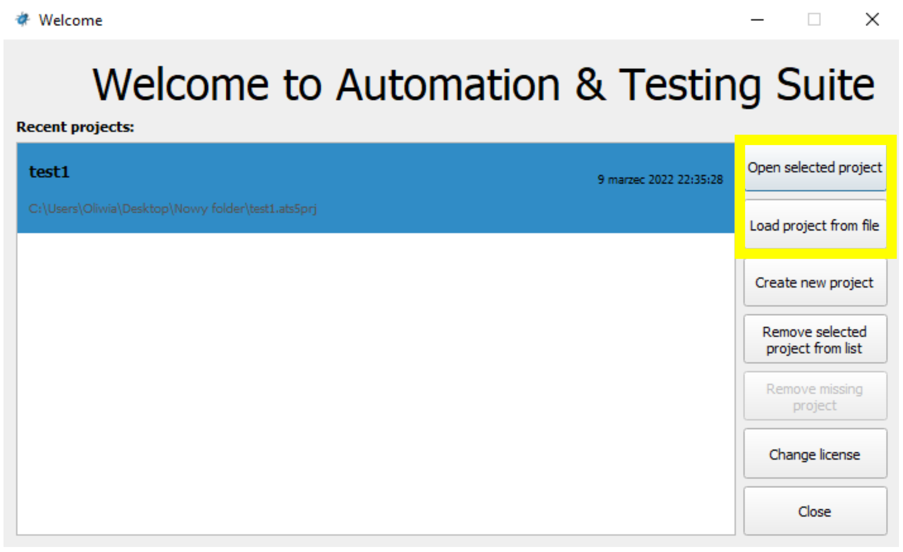
## 1.2. Opening existing project

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.

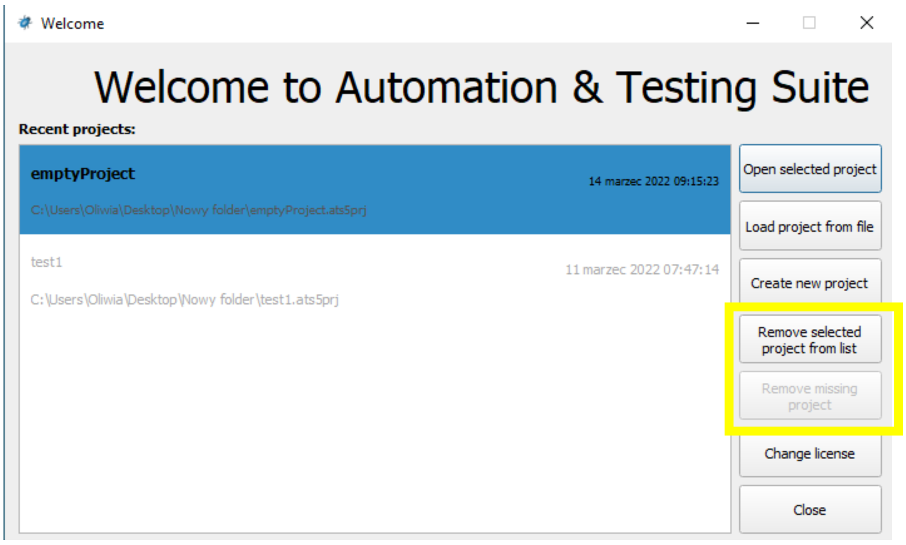
*Figure 12. Recent projects list in Welcome Window*



## 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 13). To remove that element, simply select this project and then click the button „*Remove missing project*".

*Figure 13. Remove missing project in Welcome Window.*

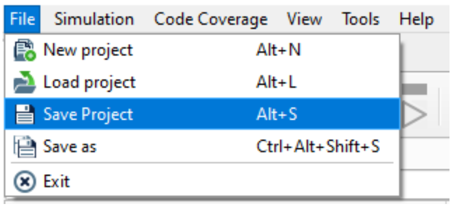


In case you would like to remove a particular project from Recent projects list, you can do this by selecting it and clicking „*Remove selected project from list*".

## 1.4. Saving project

To save a project you can go to *File* and select „*Save project*” (if your intent is to overwrite the existing project file) or „*Save as*” to save but simultaneously create new project file.

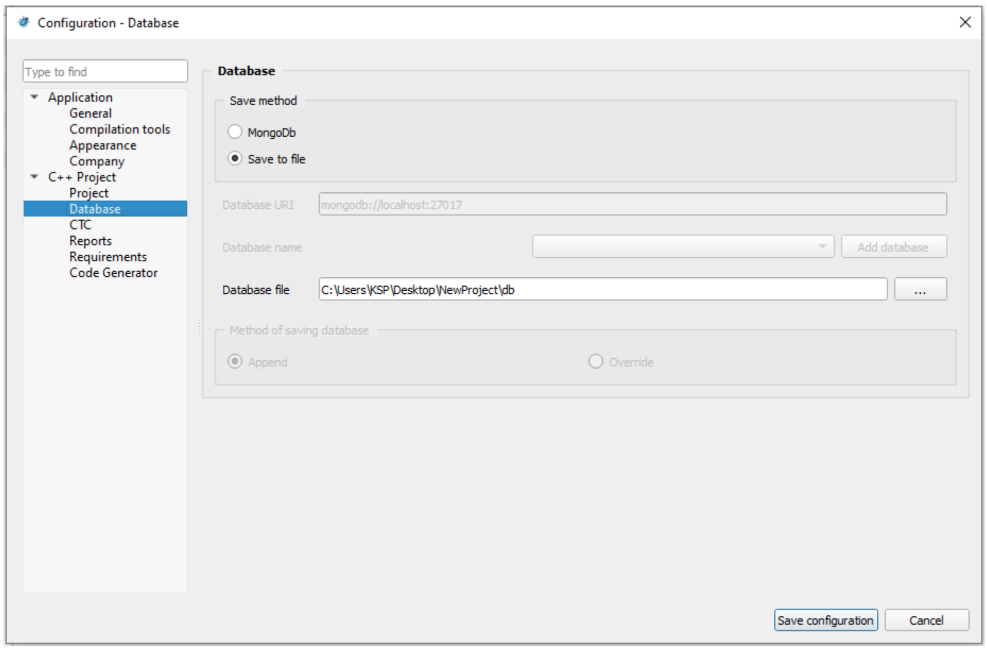
*Figure 14. File – Save options.*



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

A user can specify a saving method in *Tools – Configuration – Database*. The options are saving tests as JSON files or saving them in database MongoDB (Figure 15). In this second case, it is required to have MongoDB software to save tree in database.

*Figure 15. Configuration - Database.*



Save method: ( ) MongoDb [✓ Save to file]

Database URI: mongodb://localhost:27017

Database name:

Database file: C:\Users\KSP\Desktop\NewProject\db

Method of saving database: [✓ Append] ( ) Override

# Chapter 2. Testing files

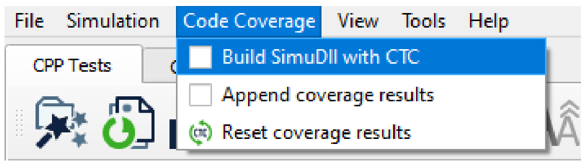
In this chapter, you will get to know how to build DLL, prepare your files for analysis, create tests and how to run them.

## 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:

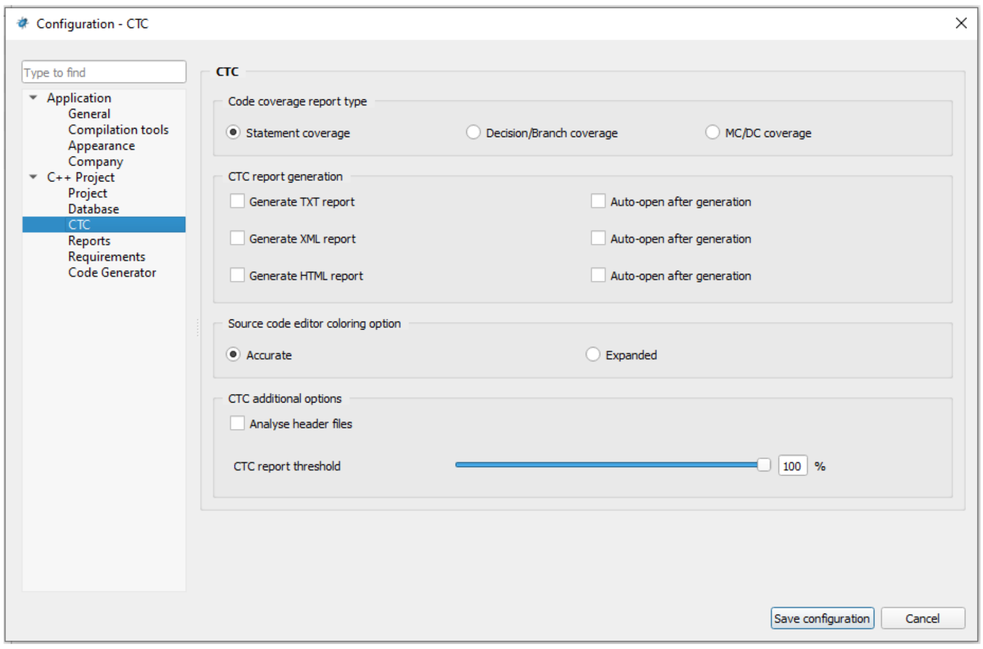
However, before building it, you should decide whether you would like to have it built with CTC enabled or not. If yes, go to the Code Coverage tab and tick the checkbox „*Build SimuDLL with CTC*" (Figure 16).

*Figure 16. Code Coverage tab.*



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

*Figure 17. CTC Tools.*



Configuration - CTC

Type to find

- Application
  - General
  - Compilation tools
  - Appearance
  - Company
- C++ Project
  - Project
  - Database
  - CTC
  - Reports
  - Requirements
  - Code Generator

CTC

Code coverage report type

[✓ Statement coverage] Decision/Branch coverage MC/DC coverage

CTC report generation

- [ ] Generate TXT report — [ ] Auto-open after generation
- [ ] Generate XML report — [ ] Auto-open after generation
- [ ] Generate HTML report — [ ] Auto-open after generation

Source code editor coloring option

[✓ Accurate] Expanded

CTC additional options

- [ ] Analyse header files

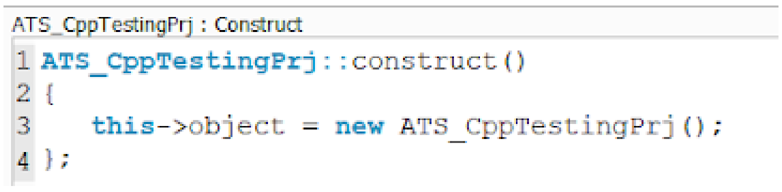
CTC report threshold 100 %

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 18.

*Figure 18. Defining constructors with non-params.*

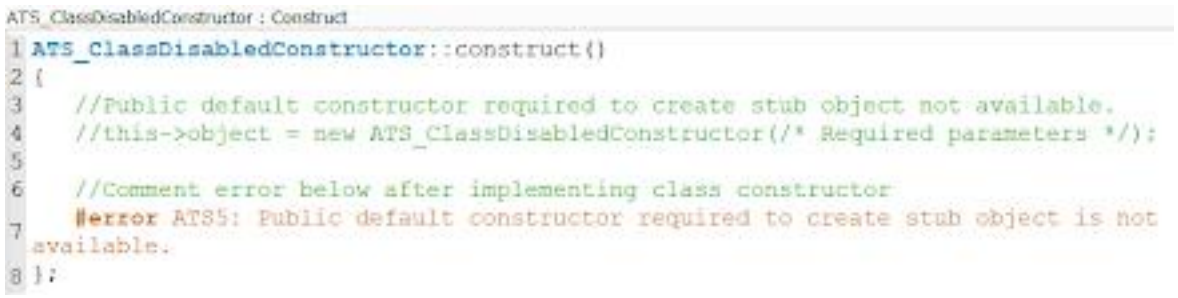


ATS_CppTestingPrj : Construct

```
ATS_CppTestingPrj::construct()
{
    this->object = new ATS_CppTestingPrj();
};
```

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 19.

*Figure 19. Information while recognizing constructor with parameters.*



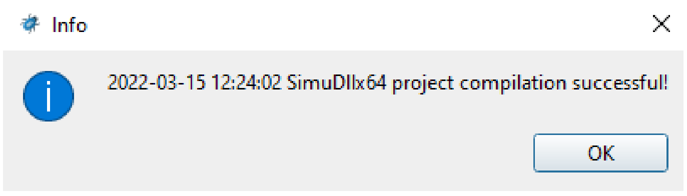
ATS_ClassDisabledConstrutor : Construct

```
ATS_ClassDisabledConstructor::construct()
{
   //Public default constructor required to create stub object not available.
   //this->object = new ATS_ClassDisabledConstructor(/* Required parameters */);

   //Comment error below after implementing class constructor
   #error ATS5: Public default constructor required to create stub object is not
available.
};
```

After all is set up, successful building the DLL will display a dialog with confirmation (Figure 20). 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 20. Successfully built DLL notification.*



Info

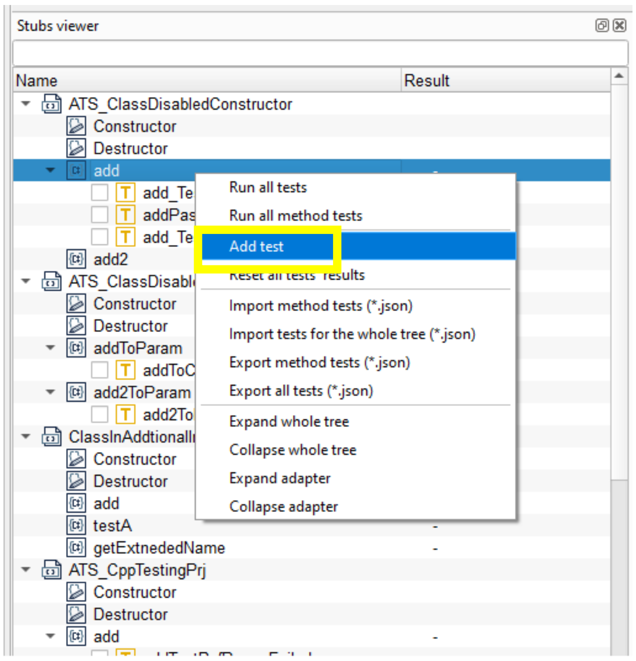
2022-03-15 12:24:02 SimuDllx64 project compilation successful!

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 21).

*Figure 21. Adding tests via context menu.*



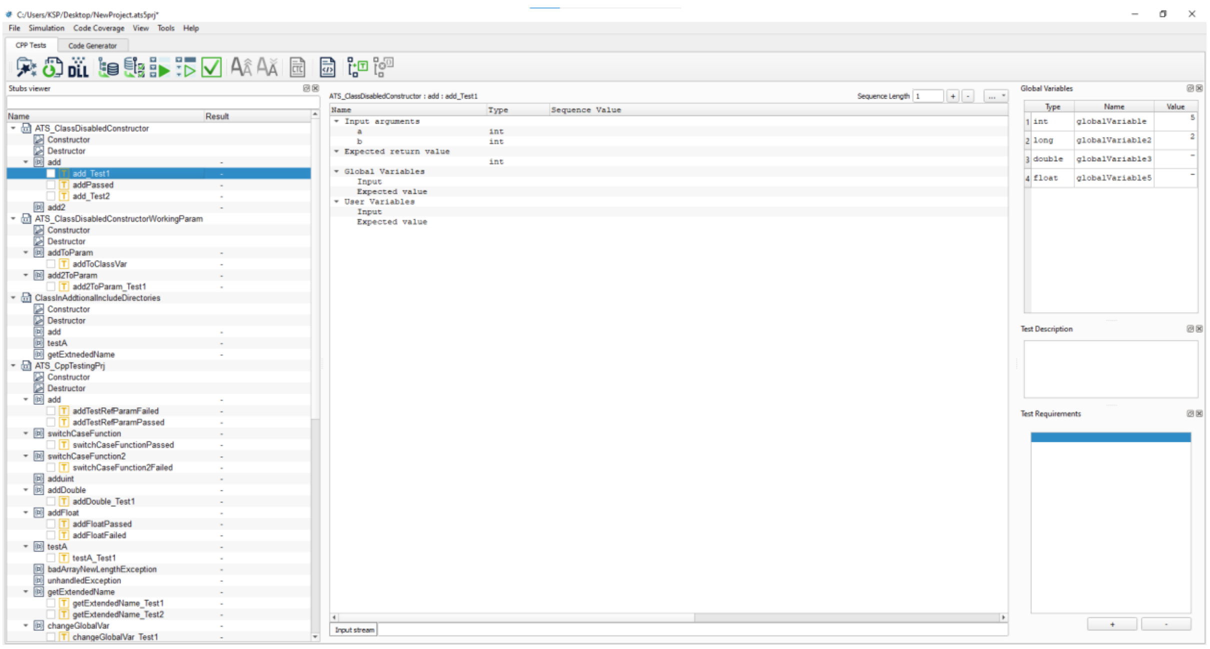
And the last, third option is to use the second button from the right side of a Toolbar:

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

## 2.3. Modifying a test

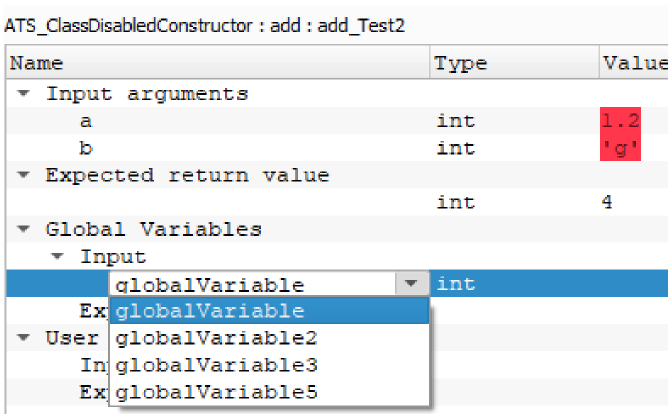
Clicking on test (tree item) shows a new window, that allows user to specify values for input arguments of methods/functions as well as expected return values (Figure 22).

*Figure 22. Main View of ATS5 with added tests.*



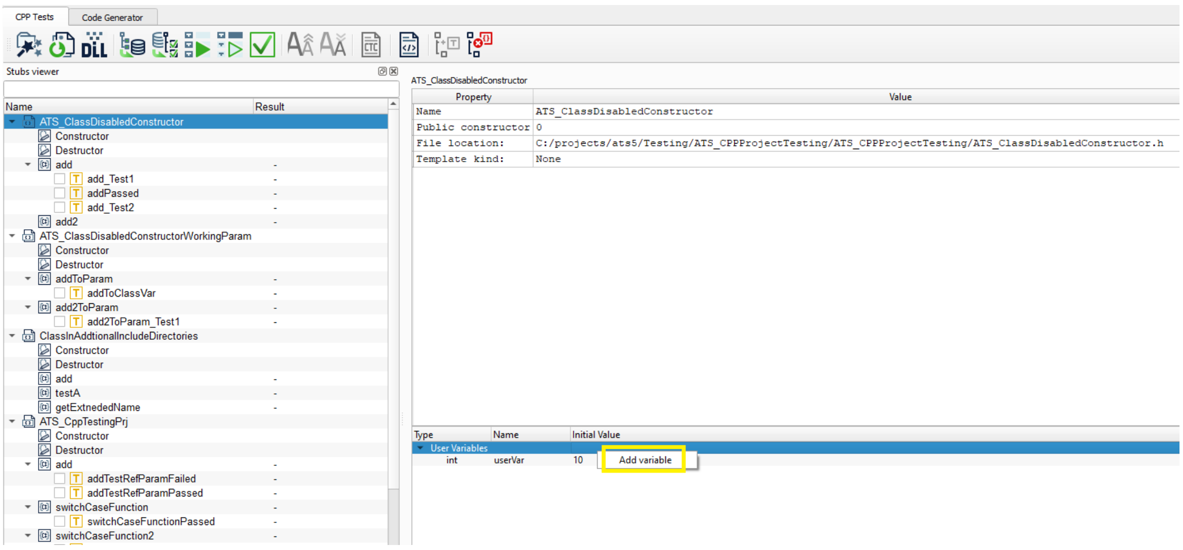
Application allows user to input only parameters that are used in a specific method/function. For example, for *add* method, which returns integer and its parameters could be also only integer number, there will be error (marked as red background), if user tries to input other data types (Figure 23).

*Figure 23. Setting wrong data type for a test parameter.*



Global variables can be added by selecting them from the expanding list (Figure above). To use a user variable, first you need to create it in the window with class definition (Figure 24). In the bottom section, double-click User Variables field or right-click it to open an option „*Add variable*”. Now, define the type, name and value of this variable, then push Enter to confirm your inputs. Now, to use such variables, the SimuDLL needs to be rebuild.

*Figure 24. Adding user variable section.*



```
Name                                          Result
▾ ATS_ClassDisabledConstructor
      Constructor
      Destructor
    ▾ add                                       -
          add_Test1                             -
          addPassed                             -
          add_Test2                             -
      add2                                      -
▾ ATS_ClassDisabledConstructorWorkingParam
      Constructor
      Destructor
    ▾ addToParam                                -
          addToClassVar                         -
    ▾ add2ToParam                               -
          add2ToParam_Test1                     -
▾ ClassInAddtionalIncludeDirectories
      Constructor
      Destructor
      add                                       -
      testA                                     -
      getExtnededName                           -
▾ ATS_CppTestingPrj
      Constructor
      Destructor
    ▾ add                                       -
          addTestRefParamFailed                 -
          addTestRefParamPassed                 -
    ▾ switchCaseFunction                        -
          switchCaseFunctionPassed              -
    ▾ switchCaseFunction2                       -
```

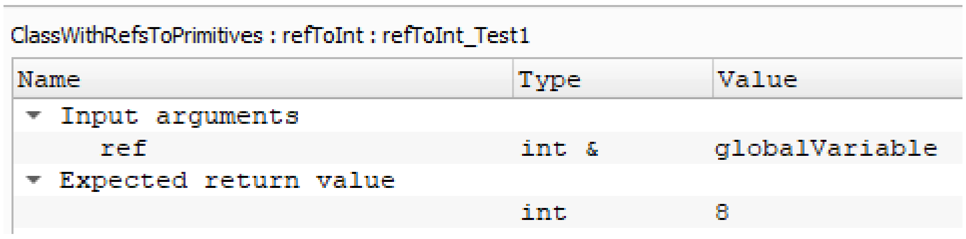
ATS_ClassDisabledConstructor

| Property | Value |
|---|---|
| Name | ATS_ClassDisabledConstructor |
| Public constructor | 0 |
| File location: | C:/projects/ats5/Testing/ATS_CPPProjectTesting/ATS_CPPProjectTesting/ATS_ClassDisabledConstructor.h |
| Template kind: | None |

| Type | Name | Initial Value |
|---|---|---|
| ▾ User Variables | | |
| int | userVar | 10 |

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 25).

*Figure 25. Reference to global variable in test's parameter.*

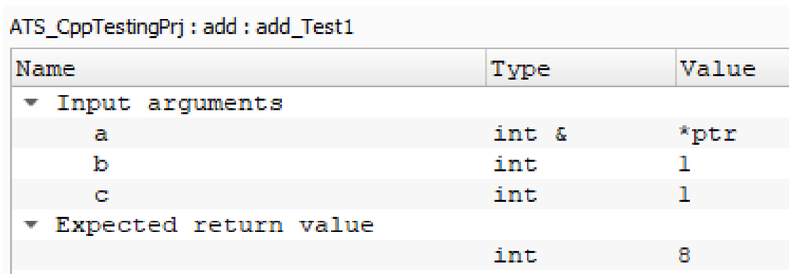


ClassWithRefsToPrimitives : refToInt : refToInt_Test1

| Name | Type | Value |
|---|---|---|
| ▾ Input arguments | | |
| ref | int & | globalVariable |
| ▾ Expected return value | | |
| | int | 8 |

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



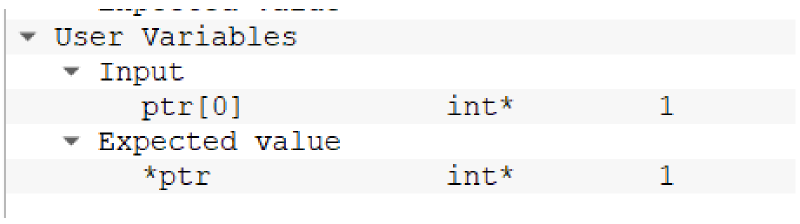
ATS_CppTestingPrj : add : add_Test1

```
Name                          Type       Value
▾ Input arguments
      a                       int &      *ptr
      b                       int        1
      c                       int        1
▾ Expected return value
                              int        8
```

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 27. Setting or getting pointer user variable.*

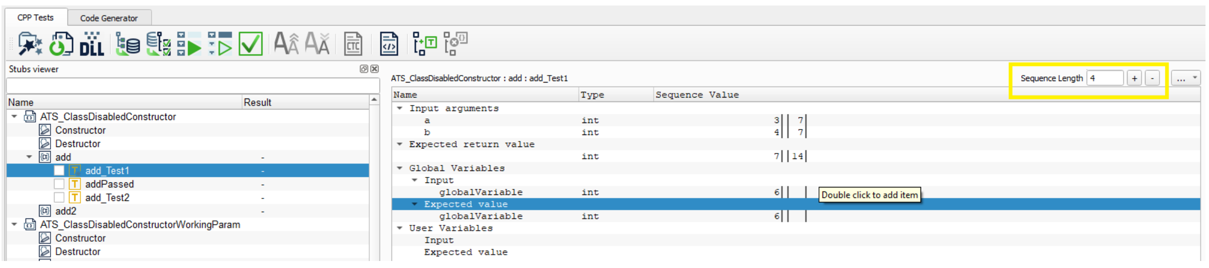


```
▾ User Variables
   ▾ Input
        ptr[0]          int*        1
   ▾ Expected value
        *ptr            int*        1
```

## 2.3.1. Sequences

A particular test can be run in sequences. To add new sequence, click the „+" button on the right side of the fields with test params (Figure 28). Also, you can modify the amount of sequences by putting its length by number.

*Figure 28. Test sequences.*



```
Name                                         Result
▾ ATS_ClassDisabledConstructor
     Constructor
     Destructor
   ▾ add                                       -
        add_Test1                              -
        addPassed                              -
        add_Test2                              -
     add2                                      -
▾ ATS_ClassDisabledConstructorWorkingParam
     Constructor
     Destructor
```

ATS_ClassDisabledConstructor : add : add_Test1

```
Name                          Type    Sequence Value
▾ Input arguments
      a                       int              3| | 7|
      b                       int              4| | 7|
▾ Expected return value
                              int              7| |14|
▾ Global Variables
   ▾ Input
         globalVariable       int              6| |  |
   ▾ Expected value
         globalVariable       int              6| |  |
▾ User Variables
     Input
     Expected value
```

Sequences will be added as well, if you input values in parameters field, separated by semicolon (;). For example, when user types “1;2;3” into value column, it means that the test will be executed three times. Adding additional semicolons without values after them means that we take previous sequence value. Thus, having “1;2;3;;;” means that the test includes 6 sequences, where 3 last sequences will consists of input values equal 3 (1;2;3;3;3;3).

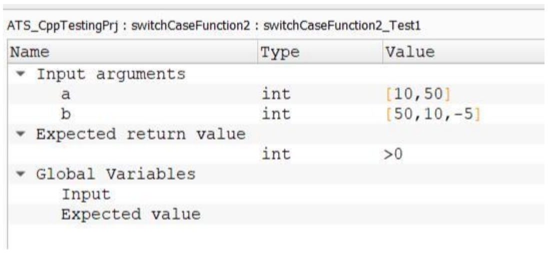
To sum it up, if you do not define parameters in the following sequences, they will be automatically set as values of earlier defined parameters. For example, as presented on Figure 28, expected return value of user variable *userVarTest* will be always equal 2 in every sequence. But as input for *userVarTest* the value will be „1” for the first and the second sequence, and then in the third and the fourth sequence the value will be „3”.

In test, you can also add comments, which will be displayed in generated reports.

### 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 29.

*Figure 29. Ranges in tests' parameters.*



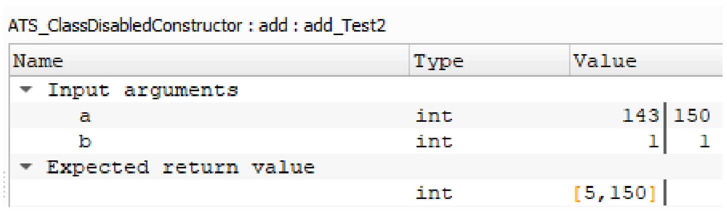
ATS_CppTestingPrj : switchCaseFunction2 : switchCaseFunction2_Test1

| Name | Type | Value |
|---|---|---|
| ▾ Input arguments | | |
| a | int | [10,50] |
| b | int | [50,10,-5] |
| ▾ Expected return value | | |
| | int | >0 |
| ▾ Global Variables | | |
| Input | | |
| Expected value | | |

Another example – range specified as [5,10,2] will run test with given values “5;7;9”. If user puts two ranges for the same input argument and in the same 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]).

Important information about ranges is that they differ as a parameter for return values. For them (return values), you can only specify [min, max] params (without step). It means that return values specified in a range (e.g. [5,150]), will take every value from that range as positively passed in a test.

*Figure 30. Ranges in return values.*



ATS_ClassDisabledConstructor : add : add_Test2

| Name | Type | Value | |
|---|---|---|---|
| Input arguments | | | |
| a | int | 143 | 150 |
| b | int | 1 | 1 |
| Expected return value | | | |
| | int | [5,150] | |

As showed on Figure 30, test from the first sequence will be passed (the result is 144, so it contains in given range), but test from the second sequence will fail.

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

“<” - values less than;

“>” - values bigger than;

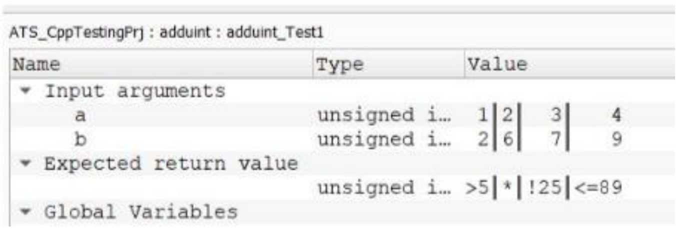
“<=” - values less and equal to;

“>=” - 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 31. Special characters in ranges.*



ATS_CppTestingPrj : adduint : adduint_Test1

| Name | Type | Value |
|---|---|---|
| ▾ Input arguments | | |
| a | unsigned i… | 1\|2\| 3\| 4 |
| b | unsigned i… | 2\|6\| 7\| 9 |
| ▾ Expected return value | | |
| | unsigned i… | >5\|*\|!25\|<=89 |
| ▾ Global Variables | | |

Usage of these special characters is presented on Figure 31.

## 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 the button or again – use a context menu.

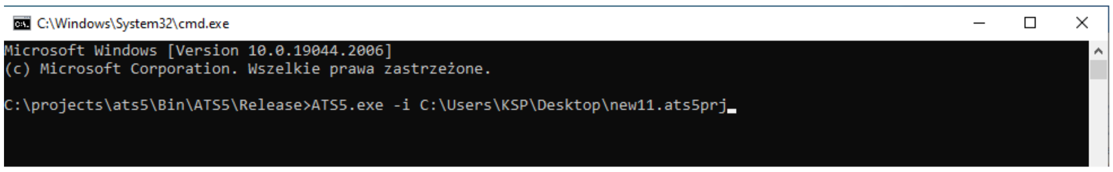
To select many tests from different classes and to make them execute, user can select particular checkboxes and then use the button to run them:

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. 32):

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 32. Automatic mode for running tests.*



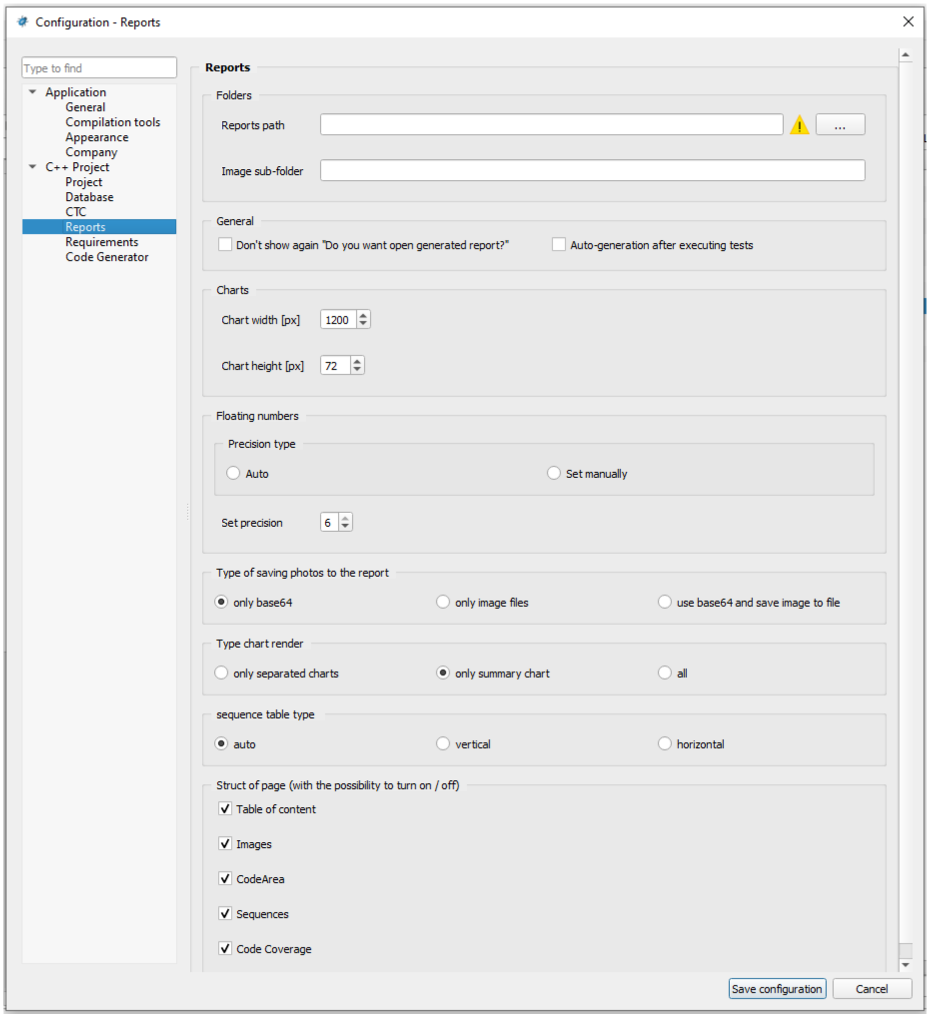
```
Microsoft Windows [Version 10.0.19044.2006]
(c) Microsoft Corporation. Wszelkie prawa zastrzeżone.

C:\projects\ats5\Bin\ATS5\Release>ATS5.exe -i C:\Users\KSP\Desktop\new11.ats5prj_
```

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 33).

*Figure 33. Reports Tools.*



Configuration - Reports

Type to find

- Application
  - General
  - Compilation tools
  - Appearance
  - Company
- C++ Project
  - Project
  - Database
  - CTC
  - Reports
  - Requirements
  - Code Generator

**Reports**

Folders

Reports path: [ ] ...

Image sub-folder: [ ]

General

[ ] Don't show again "Do you want open generated report?"    [ ] Auto-generation after executing tests

Charts

Chart width [px]: 1200

Chart height [px]: 72

Floating numbers

Precision type: ( ) Auto    ( ) Set manually

Set precision: 6

Type of saving photos to the report: [✓ only base64]    only image files    use base64 and save image to file

Type chart render: only separated charts    [✓ only summary chart]    all

sequence table type: [✓ auto]    vertical    horizontal

Struct of page (with the possibility to turn on / off)

[✓] Table of content

[✓] Images

[✓] CodeArea

[✓] Sequences

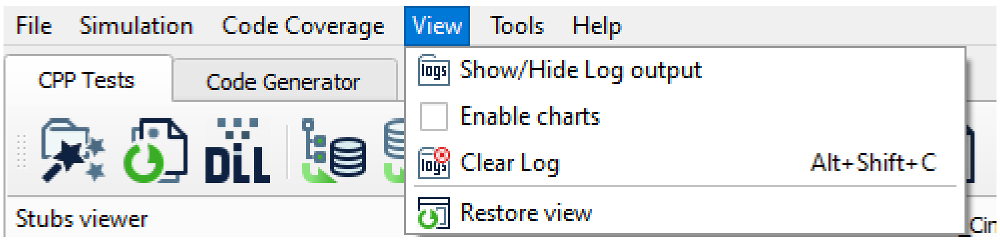
[✓] Code Coverage

Save configuration    Cancel

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 34).

*Figure 34. View tab.*

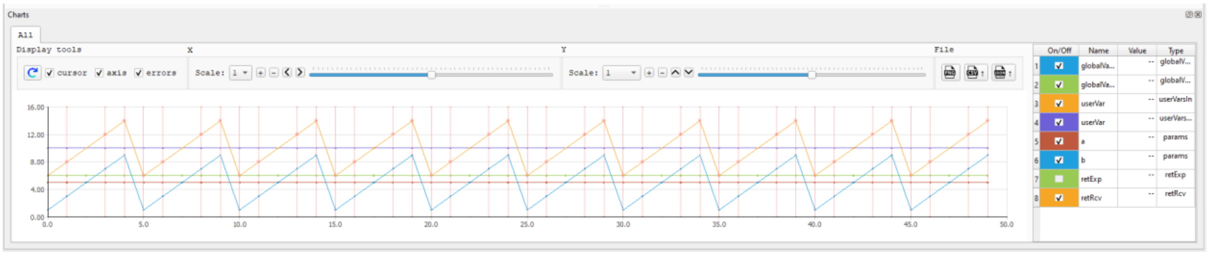


After setting this on, every executed test will automatically show charts with results.

### 2.4.1. Charts

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

*Figure 35. Charts section.*



| | On/Off | Name | Value | Type |
|---|---|---|---|---|
| 1 | ✓ | globalVa... | -- | globalV... |
| 2 | ✓ | globalVa... | -- | globalV... |
| 3 | ✓ | userVar | -- | userVarsIn |
| 4 | ✓ | userVar | -- | userVars... |
| 5 | ✓ | a | -- | params |
| 6 | ✓ | b | -- | params |
| 7 | | retExp | -- | retExp |
| 8 | ✓ | retRcv | -- | retRcv |

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.

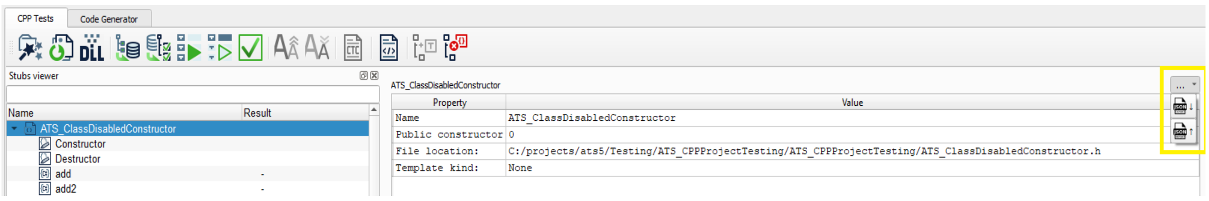
The following two sections concerning axis X and axis Y include tools for changing the scale of displayed chart – you can zoom it in or zoom it out. Also, there is a slider for moving the graph to right or left (for X axis) and to up and down (for Y axis). Moving the chart is also possible without the 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.

## 2.5. 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 36. Importing/exporting buttons in class definition.*

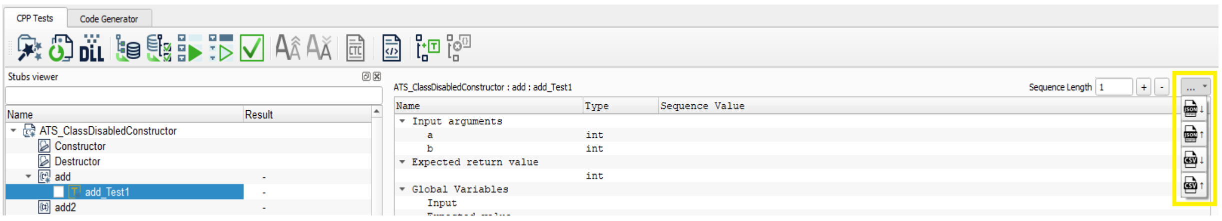


ATS_ClassDisabledConstructor

| Property | Value |
|---|---|
| Name | ATS_ClassDisabledConstructor |
| Public constructor | 0 |
| File location: | C:/projects/ats5/Testing/ATS_CPPProjectTesting/ATS_CPPProjectTesting/ATS_ClassDisabledConstructor.h |
| Template kind: | None |

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 37. Importing/exporting buttons in tests' parameters section.*



ATS_ClassDisabledConstructor : add : add_Test1

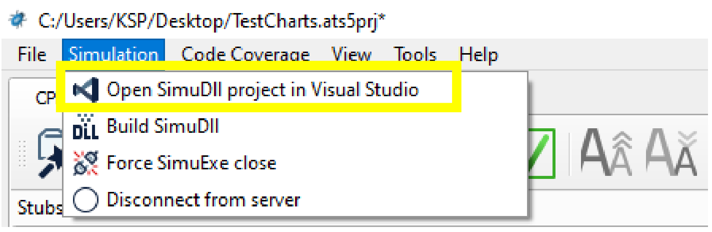
| Name | Type | Sequence Value |
|---|---|---|
| Input arguments | | |
| a | int | |
| b | int | |
| Expected return value | | |
| | int | |
| Global Variables | | |
| Input | | |

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

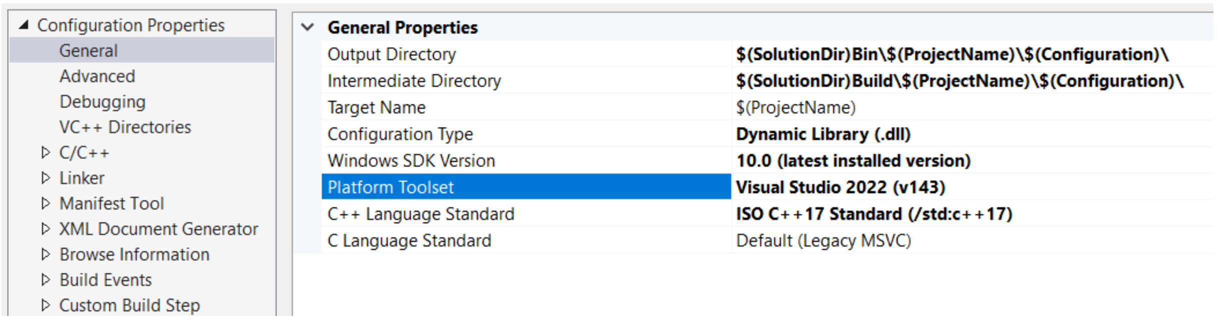
In case there will appear any error during SimuDLL project compilation, user can open SimuDLL project via ATS application by clicking *Simulation* menu, then „*Open SimuDLL project in Visual Studio*". After that, *.vcxproj* file containing SimuDLL will be opened.

*Figure 38. Simulation – Open SimuDLL in VS.*



Proper SimuDLL project configuration looks like this:

*Figure 39. SimuDLL Configuration.*



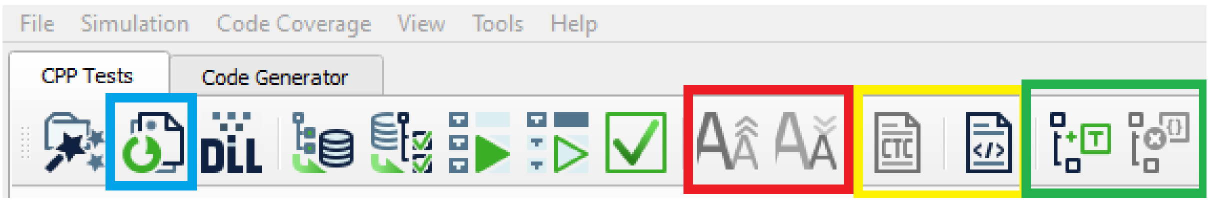
| General Properties | |
|---|---|
| Output Directory | **$(SolutionDir)Bin\$(ProjectName)\$(Configuration)** |
| Intermediate Directory | **$(SolutionDir)Build\$(ProjectName)\$(Configuration)** |
| Target Name | $(ProjectName) |
| Configuration Type | **Dynamic Library (.dll)** |
| Windows SDK Version | **10.0 (latest installed version)** |
| Platform Toolset | **Visual Studio 2022 (v143)** |
| C++ Language Standard | **ISO C++17 Standard (/std:c++17)** |
| C Language Standard | Default (Legacy MSVC) |

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

*Figure 40. 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.

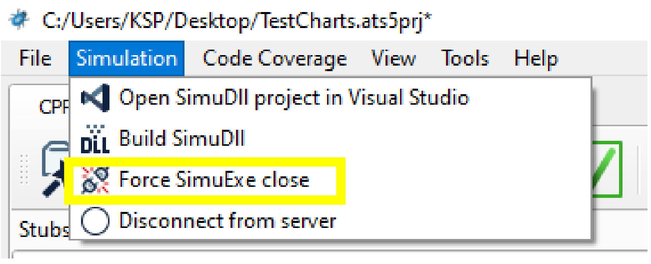
Buttons in red area are related to increasing/decreasing font size for constructor and destructor methods' definitions.

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).

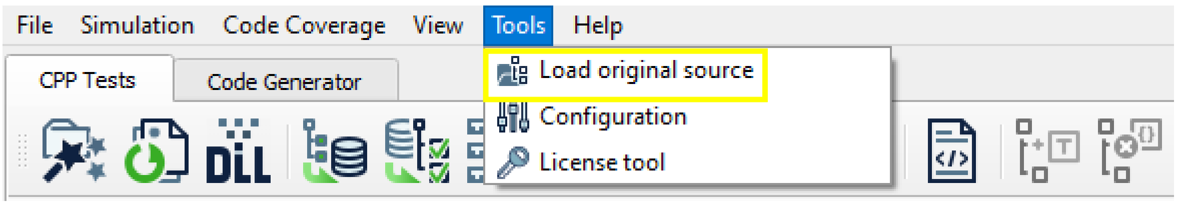
In case of a problem with executing tests, you can force SimuDLL to close. To do that, go to *Simulation – Force SimuExe close* (Figure 41).

*Figure 41. 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 42. Tools tab.*



## 3.1. ATS Reports

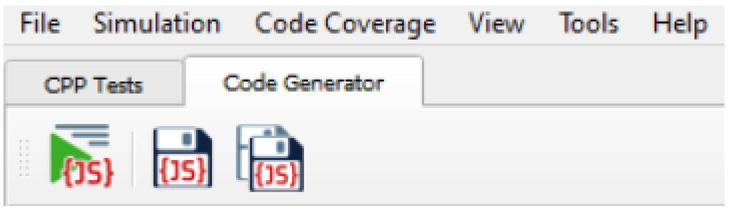
Those reports are generated as HTML file. They include Table of contents in the top of the page, then the titles of classes that contain done tests, names of the methods with their test results and their definition titled as Function Code.

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

# Chapter 4. Code Generator

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

*Figure 43. Code Generator basic button.*

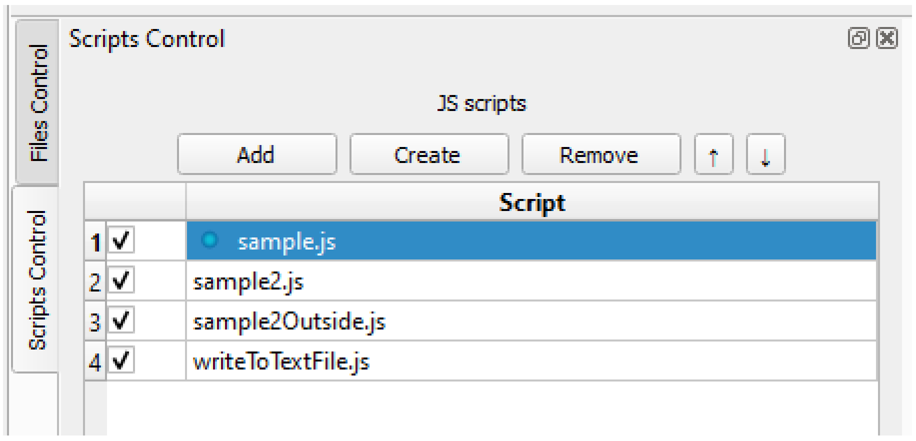


In Code Generator tab, there are 3 basic buttons. The first on the left is used to run all JS scripts, loaded to a project. The second in the middle is used to save single, selected script. And the last one saves all created or modified scripts.

## 4.1. Scripts Control

In this tab you are able to load existing JS scripts and open them in application (Figure 44). To load scripts from your computer, click *Add* button. To create new JavaScript script, select *Create*.

*Figure 44. Buttons for Scripts Control.*



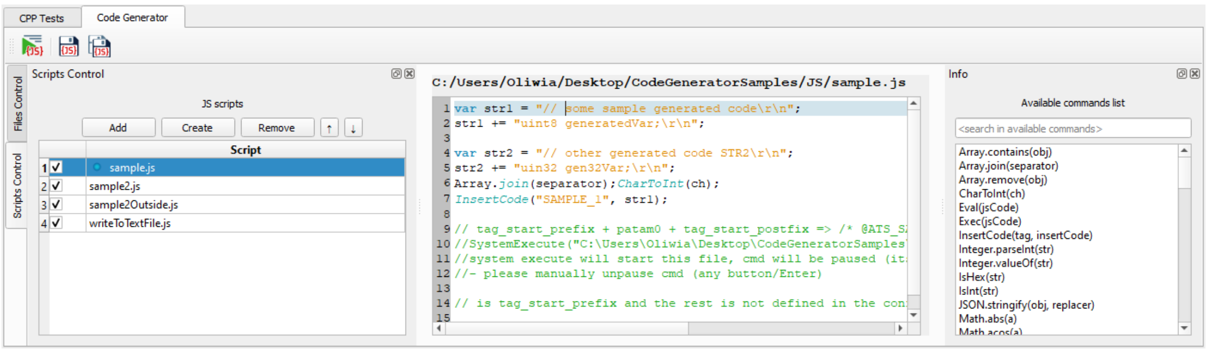
|  |  | Script |
|---|---|---|
| 1 | ✓ | sample.js |
| 2 | ✓ | sample2.js |
| 3 | ✓ | sample2Outside.js |
| 4 | ✓ | writeToTextFile.js |

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 44) 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 45) 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 45. Scripts Control view in Code Generator.*



|  |  | Script |
|---|---|---|
| 1 | ✓ | sample.js |
| 2 | ✓ | sample2.js |
| 3 | ✓ | sample2Outside.js |
| 4 | ✓ | writeToTextFile.js |

C:/Users/Oliwia/Desktop/CodeGeneratorSamples/JS/sample.js

```
var str1 = "// some sample generated code\r\n";
str1 += "uint8 generatedVar;\r\n";

var str2 = "// other generated code STR2\r\n";
str2 += "uin32 gen32Var;\r\n";
Array.join(separator);CharToInt(ch);
InsertCode("SAMPLE_1", str1);

// tag_start_prefix + patam0 + tag_start_postfix => /* @ATS_S
//SystemExecute("C:\Users\Oliwia\Desktop\CodeGeneratorSamples
//system execute will start this file, cmd will be paused (it
//- please manually unpause cmd (any button/Enter)

// is tag_start_prefix and the rest is not defined in the con

```

```
Array.contains(obj)
Array.join(separator)
Array.remove(obj)
CharToInt(ch)
Eval(jsCode)
Exec(jsCode)
InsertCode(tag, insertCode)
Integer.parseInt(str)
Integer.valueOf(str)
IsHex(str)
IsInt(str)
JSON.stringify(obj, replacer)
Math.abs(a)
Math.acos(a)
```

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 46). Files to import can be selected from Visual Studio projects or added separately by user.

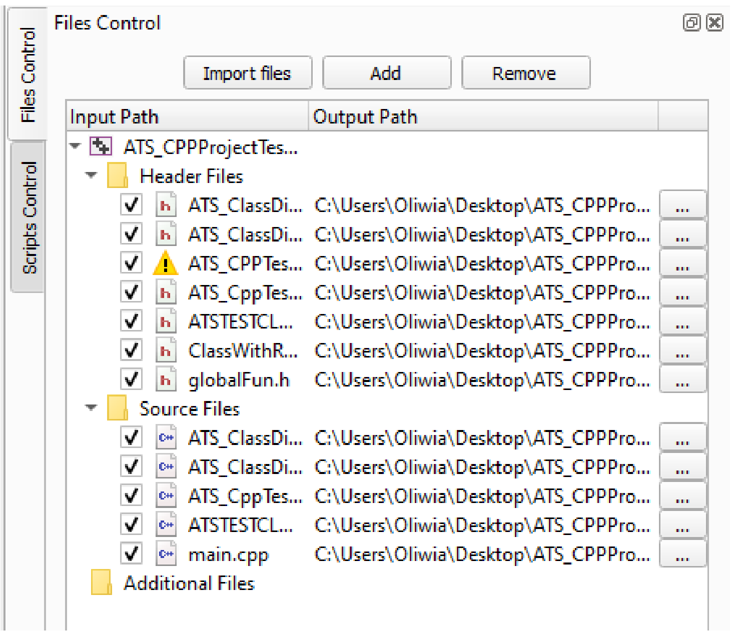
By using *add* button, you can add .h, .cpp or .hpp files to the project.

*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.

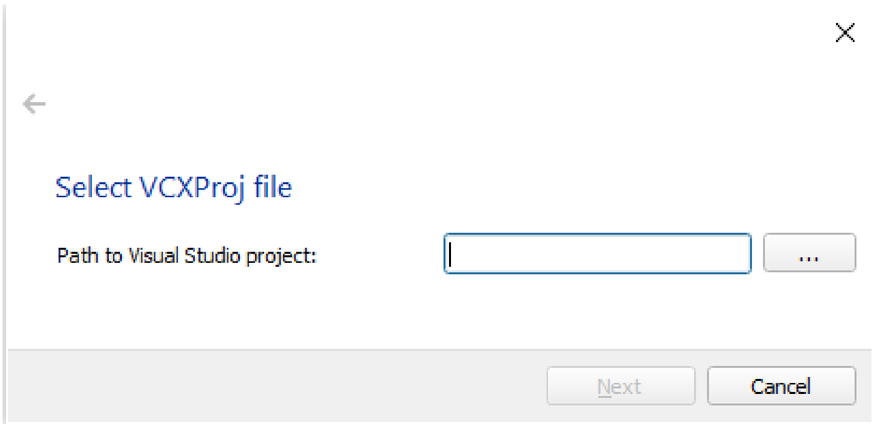
*Figure 46. Basic button in Files Control.*



| Input Path | Output Path | |
|---|---|---|
| ATS_CPPProjectTes... | | |
| Header Files | | |
| ✓ ATS_ClassDi... | C:\Users\Oliwia\Desktop\ATS_CPPPro... | ... |
| ✓ ATS_ClassDi... | C:\Users\Oliwia\Desktop\ATS_CPPPro... | ... |
| ✓ ATS_CPPTes... | C:\Users\Oliwia\Desktop\ATS_CPPPro... | ... |
| ✓ ATS_CppTes... | C:\Users\Oliwia\Desktop\ATS_CPPPro... | ... |
| ✓ ATSTESTCL... | C:\Users\Oliwia\Desktop\ATS_CPPPro... | ... |
| ✓ ClassWithR... | C:\Users\Oliwia\Desktop\ATS_CPPPro... | ... |
| ✓ globalFun.h | C:\Users\Oliwia\Desktop\ATS_CPPPro... | ... |
| Source Files | | |
| ✓ ATS_ClassDi... | C:\Users\Oliwia\Desktop\ATS_CPPPro... | ... |
| ✓ ATS_ClassDi... | C:\Users\Oliwia\Desktop\ATS_CPPPro... | ... |
| ✓ ATS_CppTes... | C:\Users\Oliwia\Desktop\ATS_CPPPro... | ... |
| ✓ ATSTESTCL... | C:\Users\Oliwia\Desktop\ATS_CPPPro... | ... |
| ✓ main.cpp | C:\Users\Oliwia\Desktop\ATS_CPPPro... | ... |
| Additional Files | | |

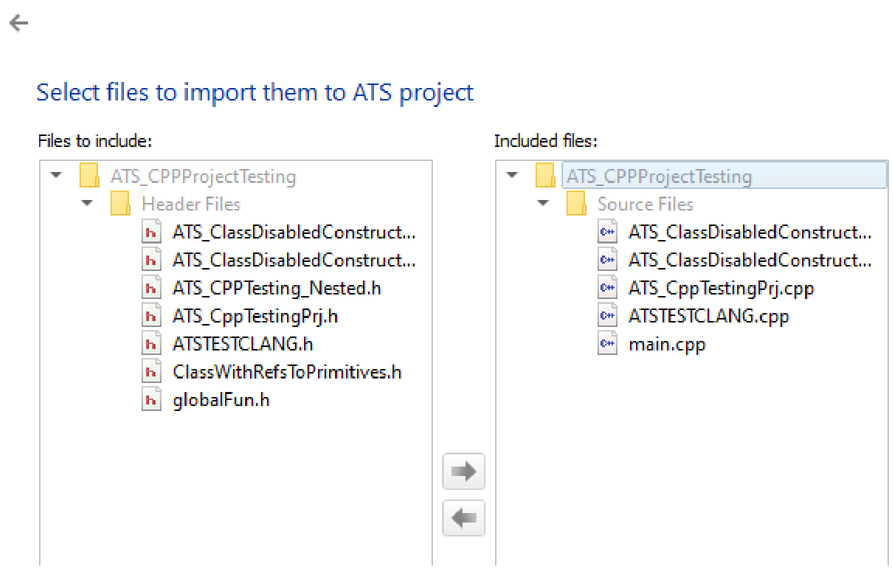
Clicking *Import files* button opens new window with selecting VCXProj file (Figure 47).

*Figure 47. Selecting file to import in Files Control.*



By going „*Next*", there is a window that allows you to choose files, that you would like to include in a project (Figure 48) – 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 48. Selecting particular files to include in Files Control.*



Select files to import them to ATS project

Files to include:

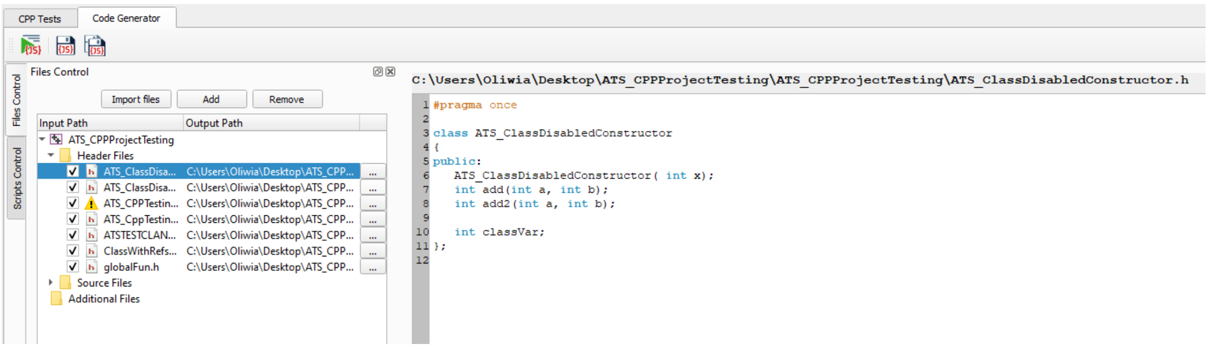
- ATS_CPPProjectTesting
  - Header Files
    - ATS_ClassDisabledConstruct...
    - ATS_ClassDisabledConstruct...
    - ATS_CPPTesting_Nested.h
    - ATS_CppTestingPrj.h
    - ATSTESTCLANG.h
    - ClassWithRefsToPrimitives.h
    - globalFun.h

Included files:

- ATS_CPPProjectTesting
  - Source Files
    - ATS_ClassDisabledConstruct...
    - ATS_ClassDisabledConstruct...
    - ATS_CppTestingPrj.cpp
    - ATSTESTCLANG.cpp
    - main.cpp

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 49).

*Figure 49. Files Control view in Code Generator.*



| Input Path | Output Path |
|---|---|
| ATS_CPPProjectTesting | |
| Header Files | |
| ATS_ClassDisa... | C:\Users\Oliwia\Desktop\ATS_CPP... |
| ATS_ClassDisa... | C:\Users\Oliwia\Desktop\ATS_CPP... |
| ATS_CPPTestin... | C:\Users\Oliwia\Desktop\ATS_CPP... |
| ATS_CppTestin... | C:\Users\Oliwia\Desktop\ATS_CPP... |
| ATSTESTCLAN... | C:\Users\Oliwia\Desktop\ATS_CPP... |
| ClassWithRefs... | C:\Users\Oliwia\Desktop\ATS_CPP... |
| globalFun.h | C:\Users\Oliwia\Desktop\ATS_CPP... |
| Source Files | |
| Additional Files | |

C:\Users\Oliwia\Desktop\ATS_CPPProjectTesting\ATS_CPPProjectTesting\ATS_ClassDisabledConstructor.h

```
#pragma once

class ATS_ClassDisabledConstructor
{
public:
   ATS_ClassDisabledConstructor( int x);
   int add(int a, int b);
   int add2(int a, int b);

   int classVar;
};
```

# List of Figures