Hat

Table of contents

English version

Description

Hat

The peculiarity of the Hat browser is that autotests are performed directly in the browser without Selenium and WebDriver.

The built-in Net Framework contains a sufficient number of methods necessary to perform the basic tasks of test automation. The C# programming language and the built-in code editor are used to describe autotest scripts. You can also use Visual Studio as an editor. The user-friendly browser interface displays all the steps of the test with the corresponding description of sales. The verification result is generated in a report and sent to the specified email address. Autotests can be run from the command line of the Windows operating system. This happens when using autotests in popular continuous integration environments such as: Jenkins, TeamCity, GitLab CI/CD.

The official page: https://somovstudio.github.io/

Download the app from GitHub: https://github.com/SomovStudio/Hat

The version of this documentation corresponds to the browser version 1.5.0.3 (update 03/27/2026)

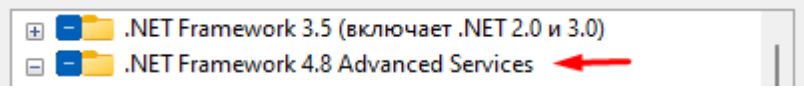
Quick start

System requirements

System requirements

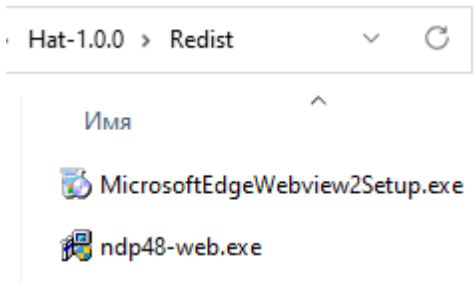
1. The operating system Windows 8, 8.1, 10, 11

2. Package Microsoft .NET Framework 4.8 (download the installer .NET Framework Runtime)



3. Package Microsoft Edge WebView2 (download the installer WebView2 Runtime)

Note: the application archive in the Redist folder contains package installers Microsoft .NET Framework 4.8 и Microsoft Edge WebView2



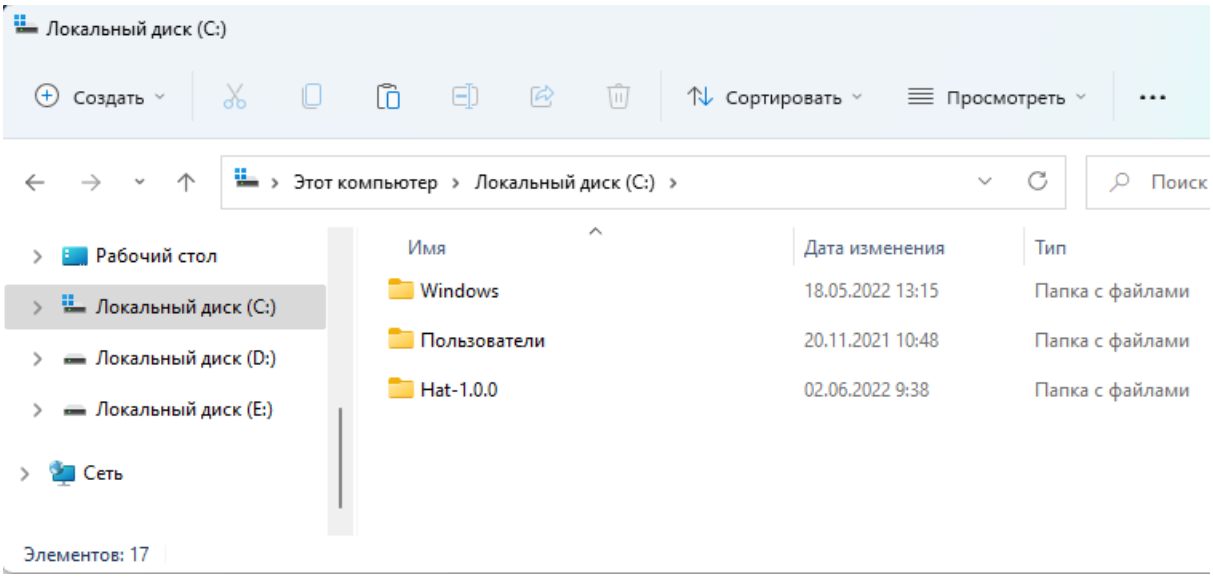
Installation and launch

Installing and launching the browser Hat

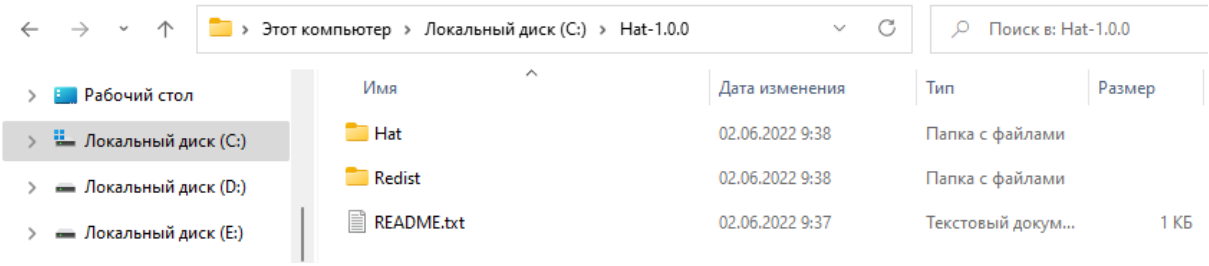
1. Download the browser from the official GitHub page:
https://github.com/SomovStudio/Hat/

Check the archive with an antivirus program and if everything is in order, open the archive properties and turn on the "Unlock" flag so that the system does not ask you for permission every time you start the browser.

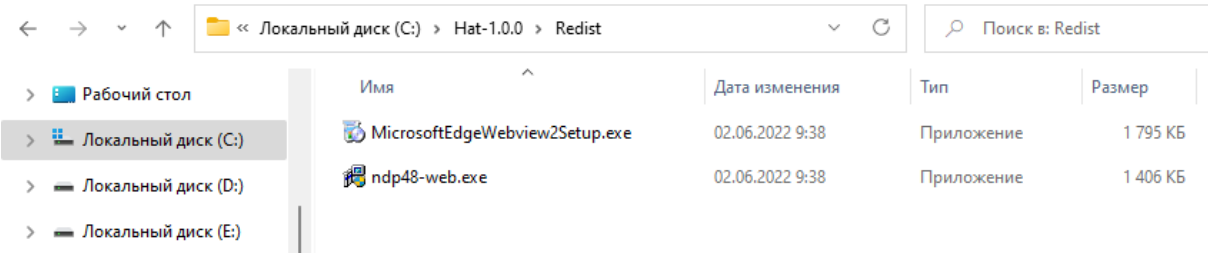
2. The application is completely portable and the downloaded archive can be unzipped to any location on the hard disk.
Tip: it is better not to install in the Program Files folder, otherwise you will have to give the application administrator rights, unzip it to the root of the disk.



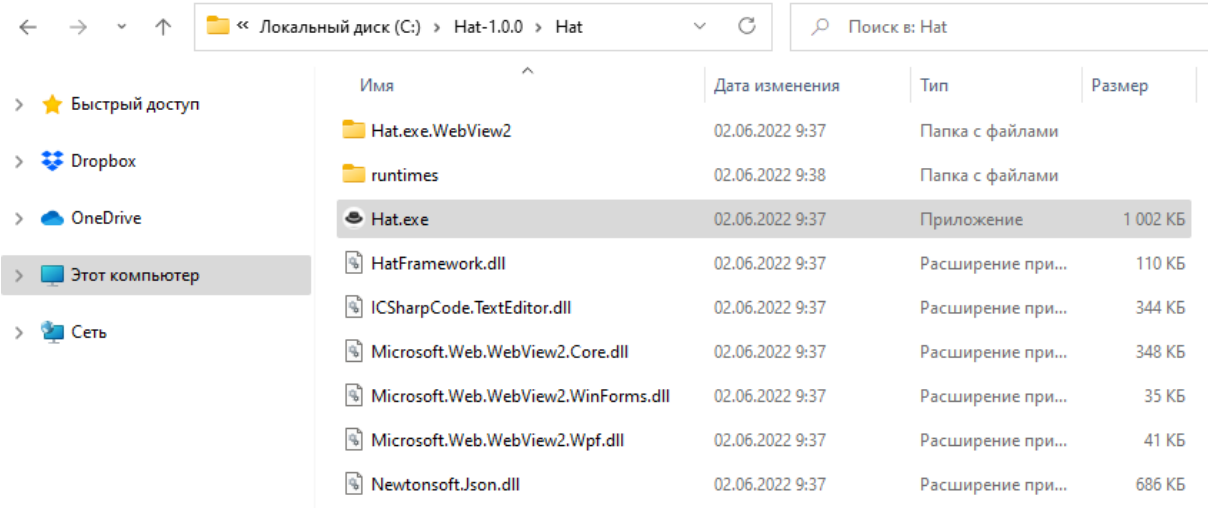
go to the folder Hat-1.0.0



3. The Redist folder contains the package installers Microsoft .NET Framework 4.8 и Microsoft Edge WebView2



4. The application itself is located in the Hat folder.

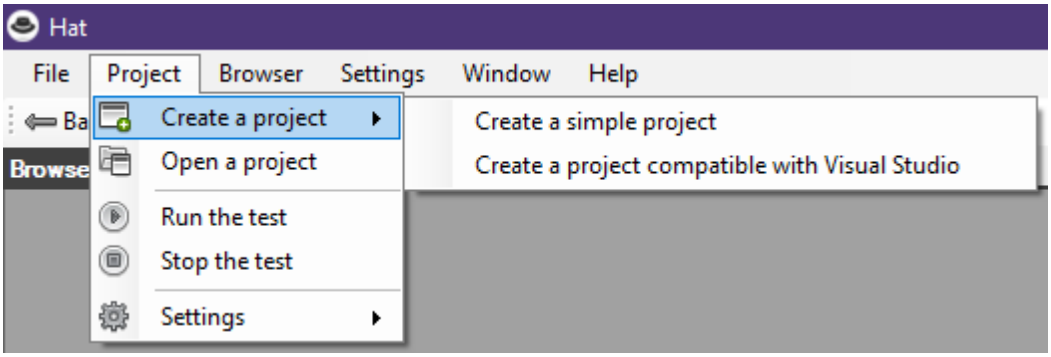


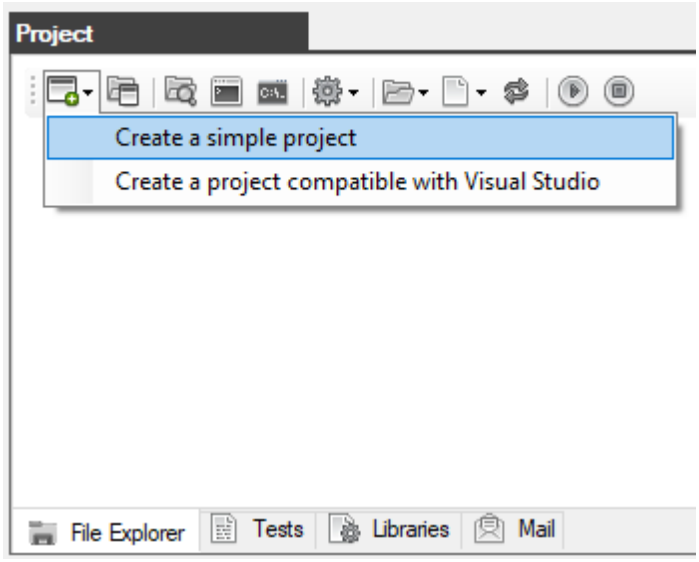
The browser is launched by a file Hat.exe

Creating a simple project

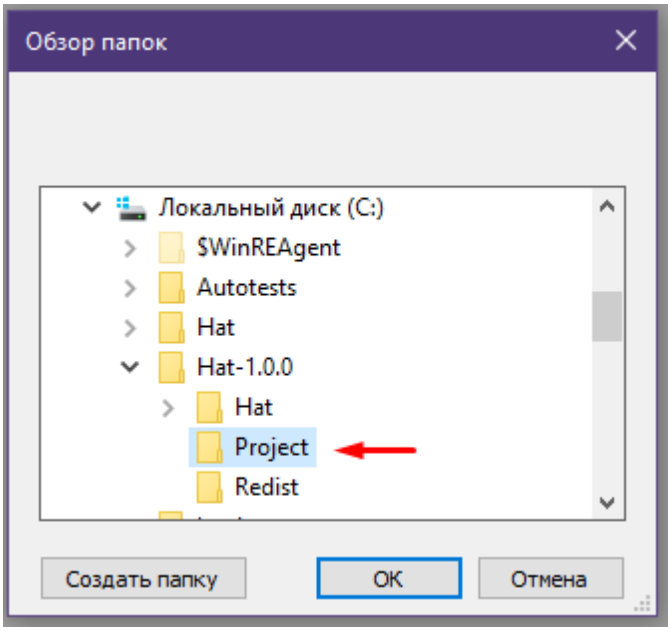
Creating a project and launching an autotest

1. In the Project window, on the Explorer tab or in the main Create Project menu, click the Create Simple Project button

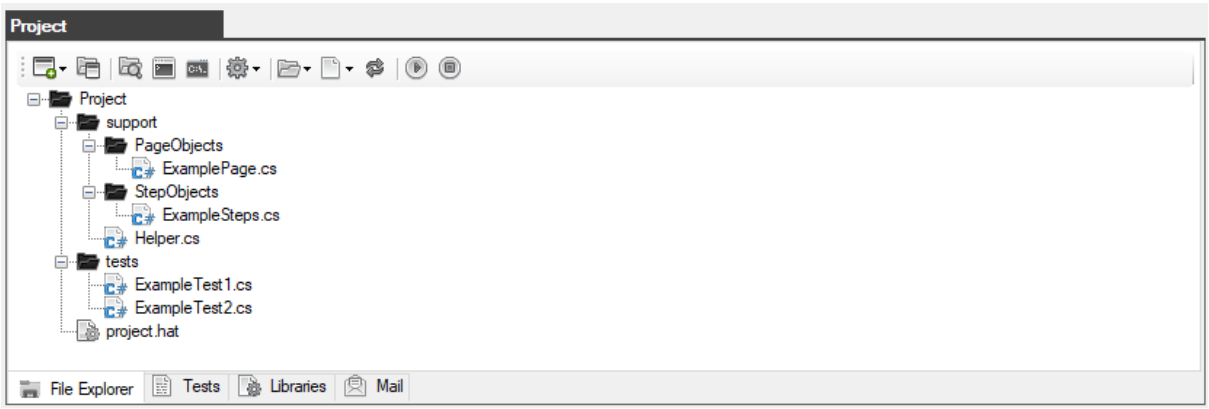




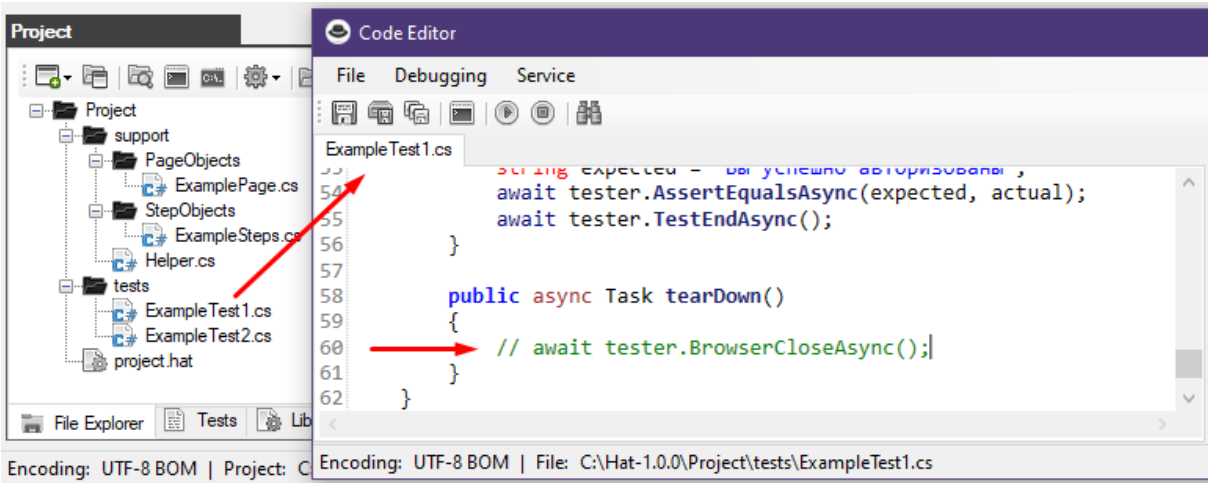
2. Create a folder that will contain the project with any name and anywhere on the disk.
In this case, the Project folder has been created at C:\Hat-1.0.0\
Select the project folder and click OK



3. In the Project window, the contents of the project will be displayed on the Explorer tab.
As you can see, the project is not empty, it contains several demo autotests.



4. Double-click on the ExampleTest1.cs file to open it in the code editor.
In line 60, a line is commented out with a call to the BrowserCloseAsync() method that closes the browser.
This is necessary so that the browser does not automatically close after the completion of the autotest.

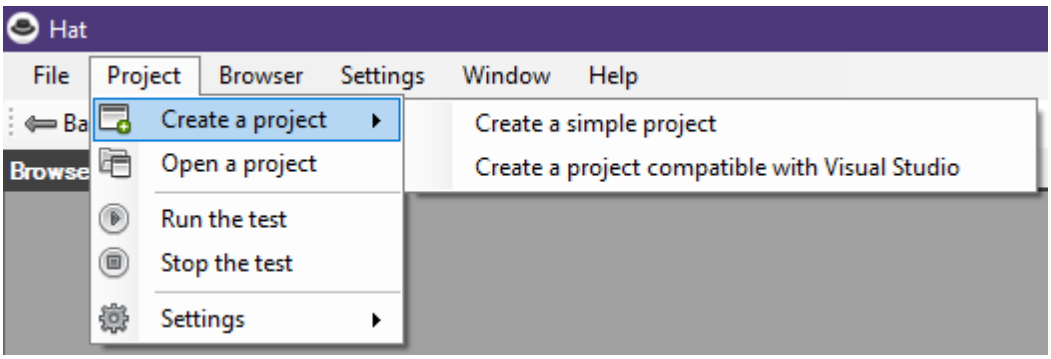


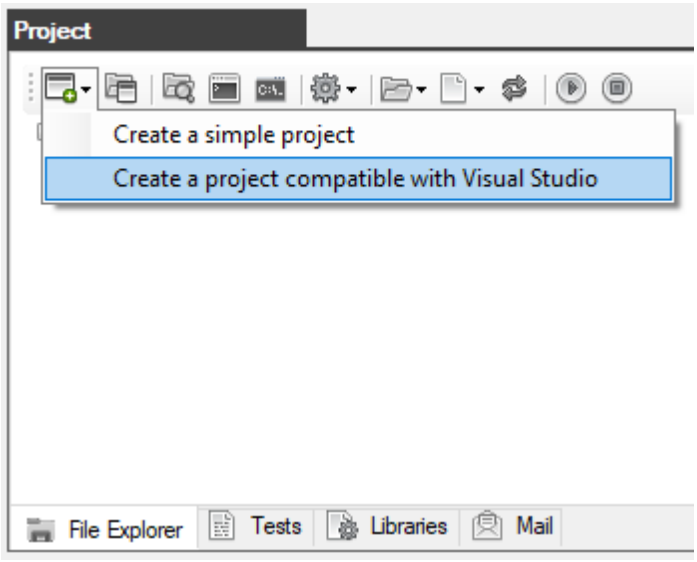
The project has been successfully created.

Creating a project compatible with Visual Studio

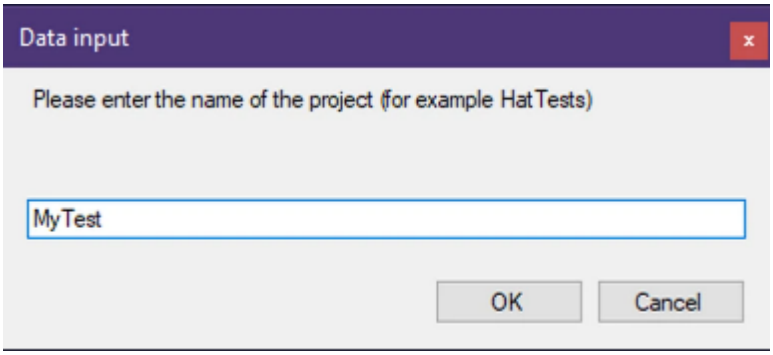
Creating a project compatible with Visual Studio

1. In the Project window, on the Explorer tab or in the main Create Project menu, click the "Creating a project compatible with Visual Studio" button

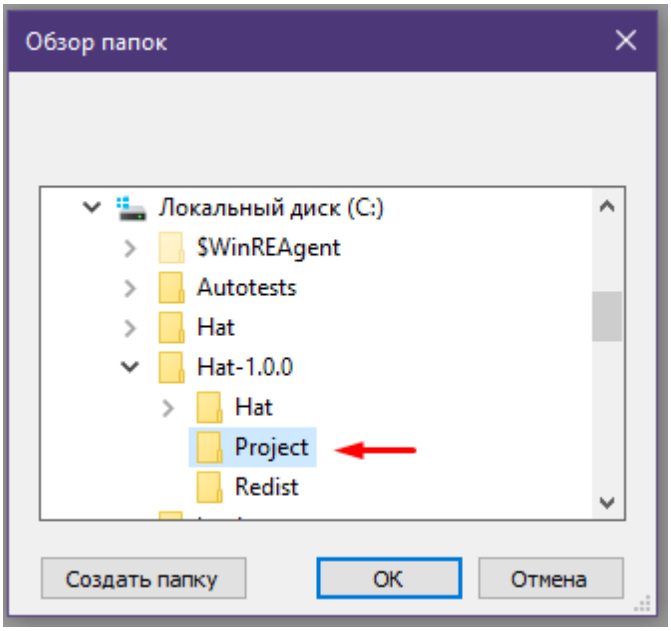




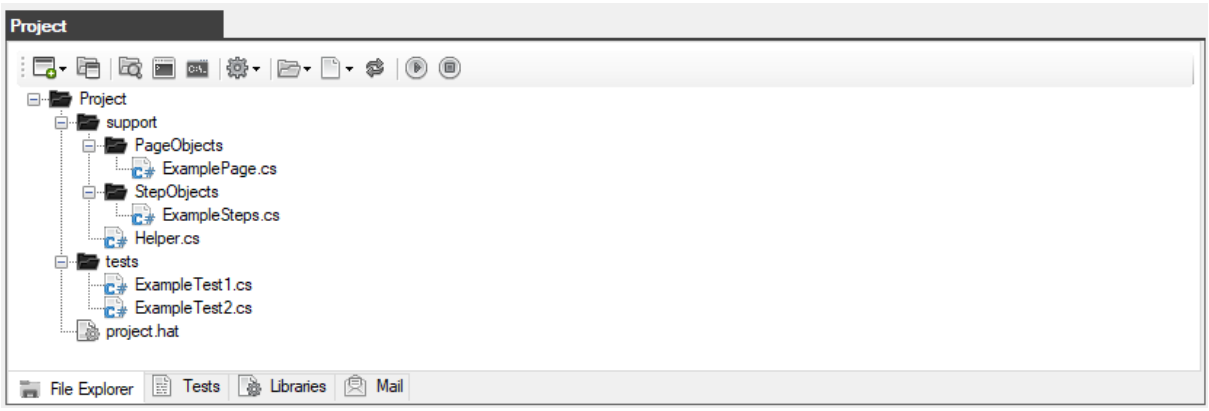
2. A window opens for entering the project name.
Prerequisite the project name must be in Latin letters and without spaces.



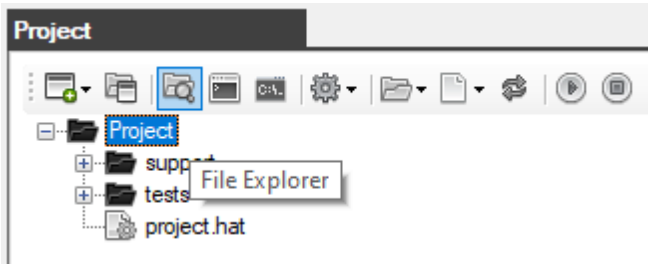
3. Create a folder that will contain the project with any name and anywhere on the disk.
In this case, the Project folder has been created at C:\Hat-1.0.0\
Select the project folder and click OK



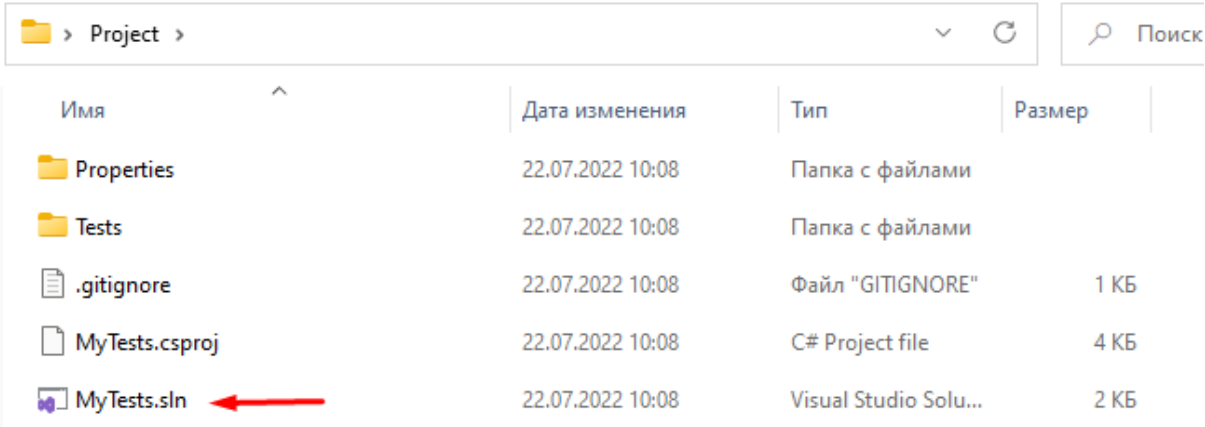
4. In the Project window, the contents of the project will be displayed on the Explorer tab. As you can see, the project is not empty, it contains several demo autotests.



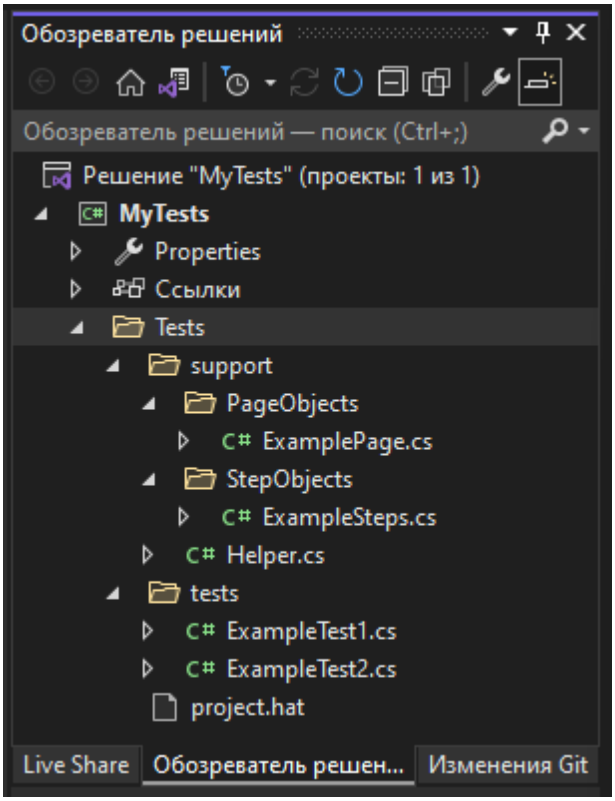
5. This project supports work in Visual Studio (it is assumed that you already have Visual Studio installed)
To open a project in Visual Studio, click on the Explorer button in the Project window



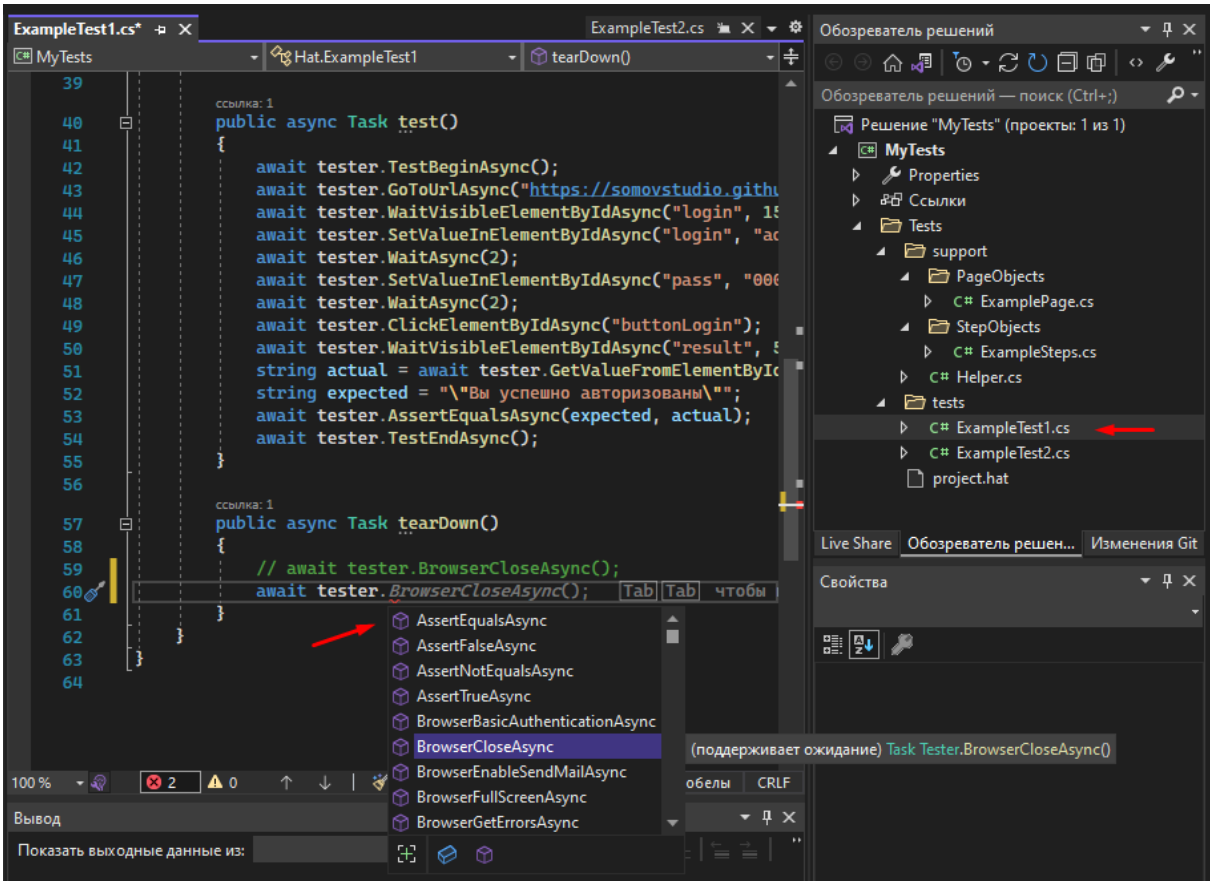
Then double-click on the file with the *.sln extension (in this case, MyTests.sln)



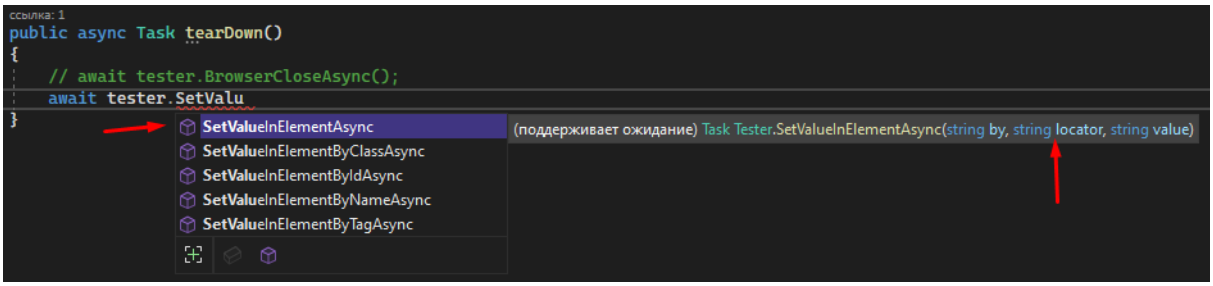
When Visual Studio is loaded, our project will immediately be opened and available for work in the Solution Explorer window



Double-click on the ExampleTest1.cs file to open it in the code editor.
In line 59, a line is commented out with a call to the BrowserCloseAsync() method that closes the browser.
This is necessary so that the browser does not automatically close after the completion of the autotest.



It is very convenient to use Visual Studio because the editor, when entering, shows a drop-down list with all available methods, as well as shows the syntax of the selected method and the variables required by it.

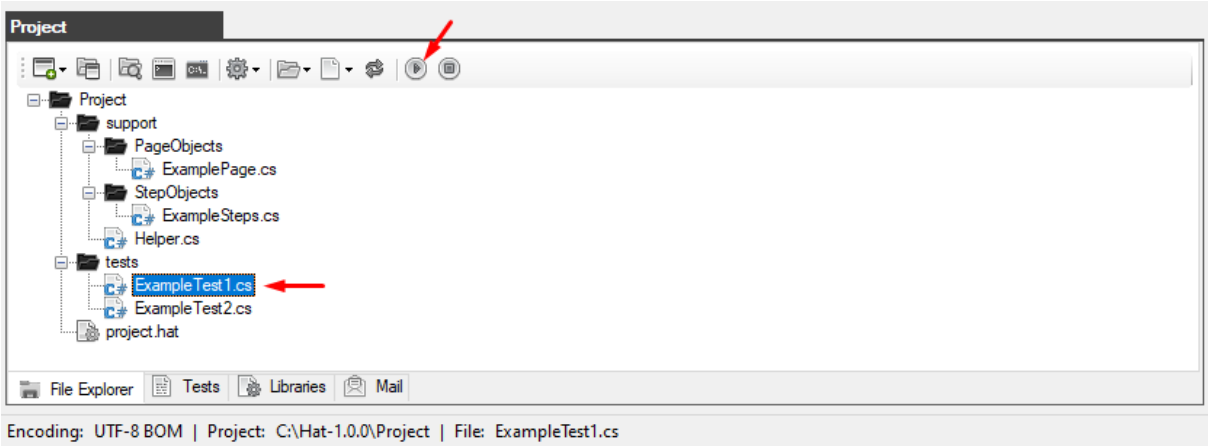


The project has been successfully created.

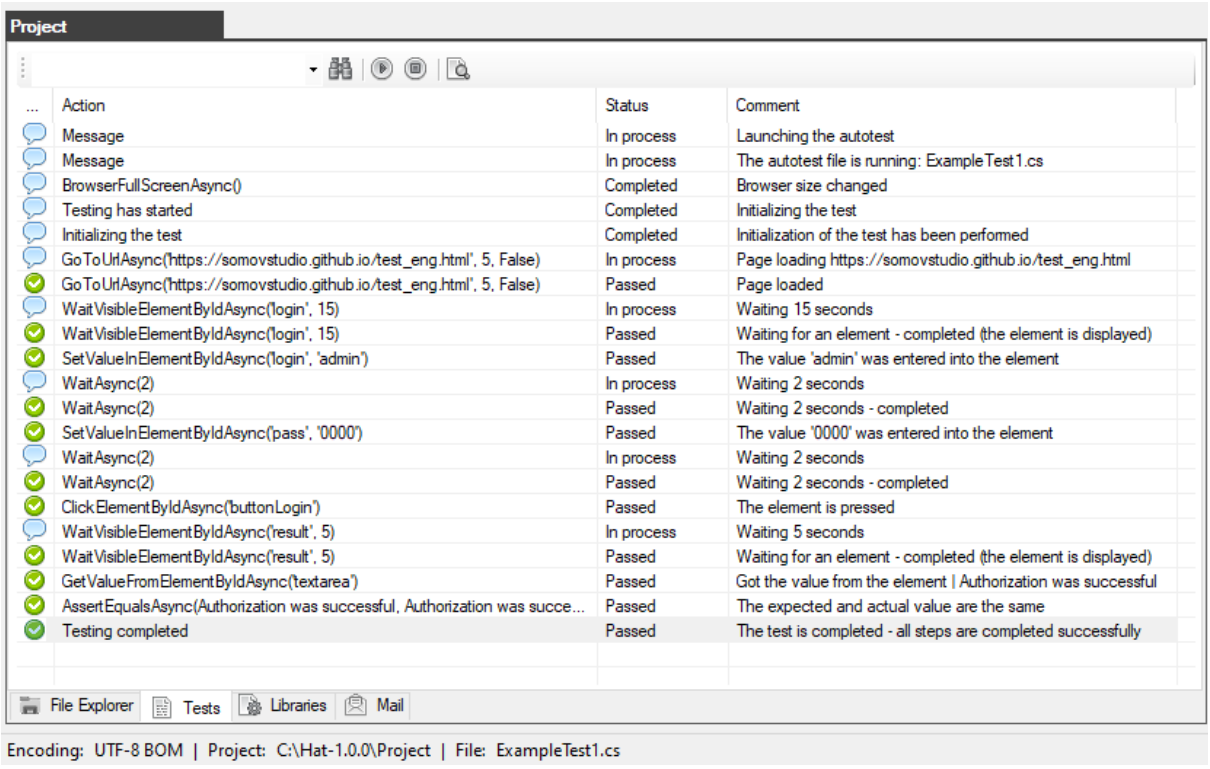
Launching the autotest and result

Launching the autotest

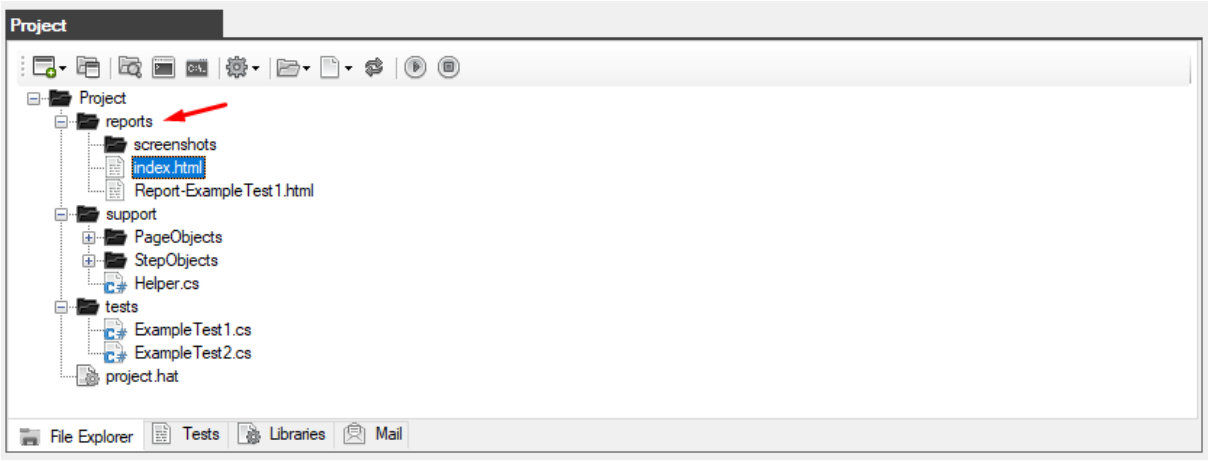
To run the autotest, select it in the tests folder and click on the "Run test" button
In our case, the ExampleTest1.cs file is launched



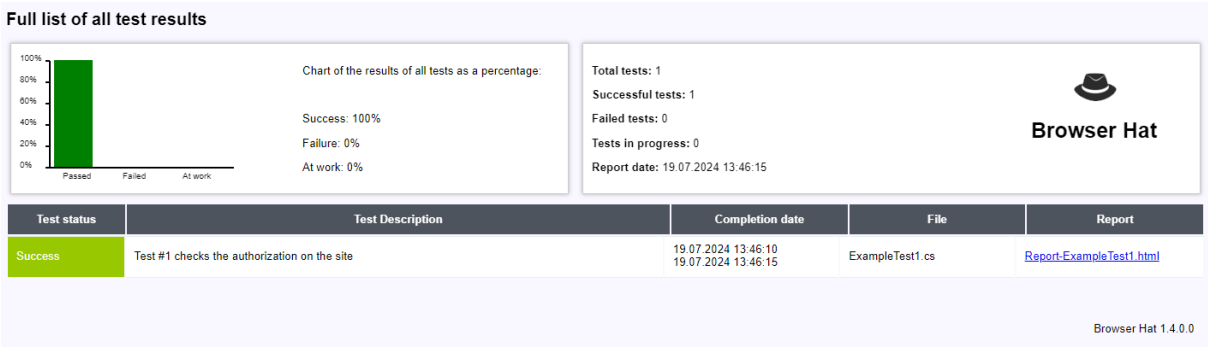
The verification process is displayed on the "Test" tab



Upon completion of the autotest, a folder "report" will be created in the root of the project, which contains all reports and screenshots with the results of the performed check.



Double-clicking on the report file (for example: index.html or Report-ExampleTest1.html) you will receive a report with a full description of the progress of the autotest.
For more information about reports and sending reports by mail, see the relevant sections.



Full list of all test results

100% 80% 60% 40% 20% 0%
Passed Failed At work

Chart of the results of all tests as a percentage:

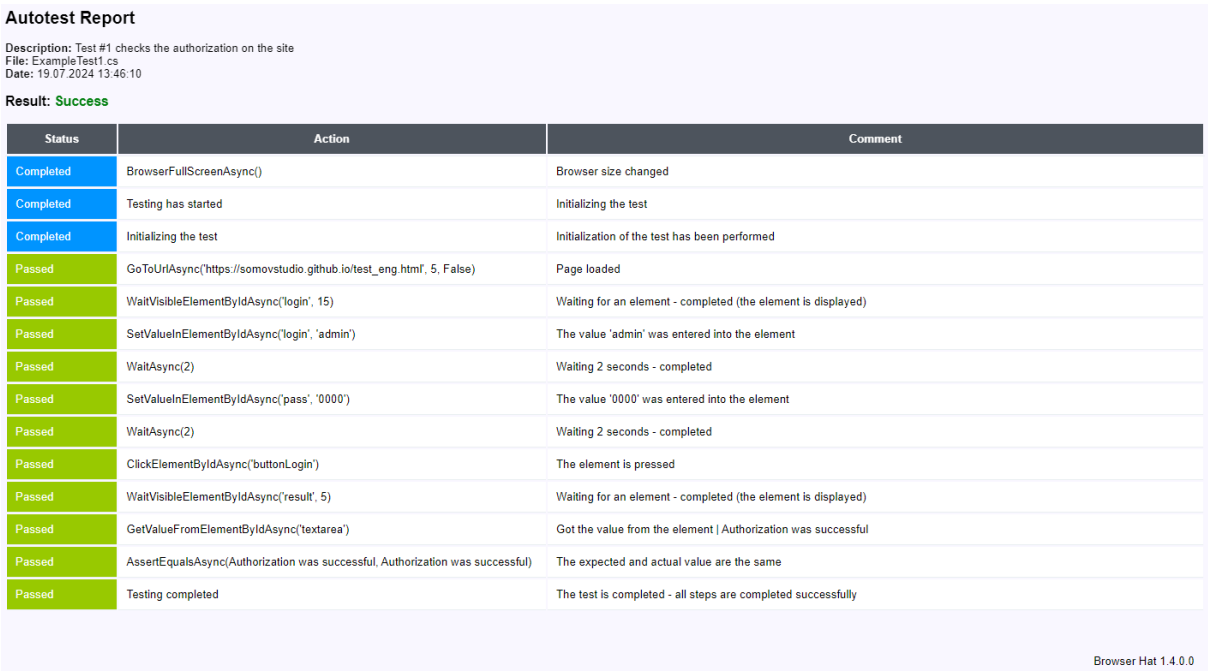
Success: 100%
Failure: 0%
At work: 0%

Total tests: 1
Successful tests: 1
Failed tests: 0
Tests in progress: 0
Report date: 19.07.2024 13:46:15

Browser Hat

Test status	Test Description	Completion date	File	Report
Success	Test #1 checks the authorization on the site	19.07.2024 13:46:10 19.07.2024 13:46:15	ExampleTest1.cs	Report-ExampleTest1.html

Browser Hat 1.4.0.0



Autotest Report

Description: Test #1 checks the authorization on the site
File: ExampleTest1.cs
Date: 19.07.2024 13:46:10

Result: Success

Status	Action	Comment
Completed	BrowserFullScreenAsync()	Browser size changed
Completed	Testing has started	Initializing the test
Completed	Initializing the test	Initialization of the test has been performed
Passed	GoToUrlAsync('https://somovstudio.github.io/test_eng.html', 5, False)	Page loaded
Passed	WaitVisibleElementByIdAsync('login', 15)	Waiting for an element - completed (the element is displayed)
Passed	SetValueInElementByIdAsync('login', 'admin')	The value 'admin' was entered into the element
Passed	WaitAsync(2)	Waiting 2 seconds - completed
Passed	SetValueInElementByIdAsync('pass', '0000')	The value '0000' was entered into the element
Passed	WaitAsync(2)	Waiting 2 seconds - completed
Passed	ClickElementByIdAsync('buttonLogin')	The element is pressed
Passed	WaitVisibleElementByIdAsync('result', 5)	Waiting for an element - completed (the element is displayed)
Passed	GetValueFromElementByIdAsync('textarea')	Got the value from the element | Authorization was successful
Passed	AssertEqualsAsync(Authorization was successful, Authorization was successful)	The expected and actual value are the same
Passed	Testing completed	The test is completed - all steps are completed successfully

Browser Hat 1.4.0.0

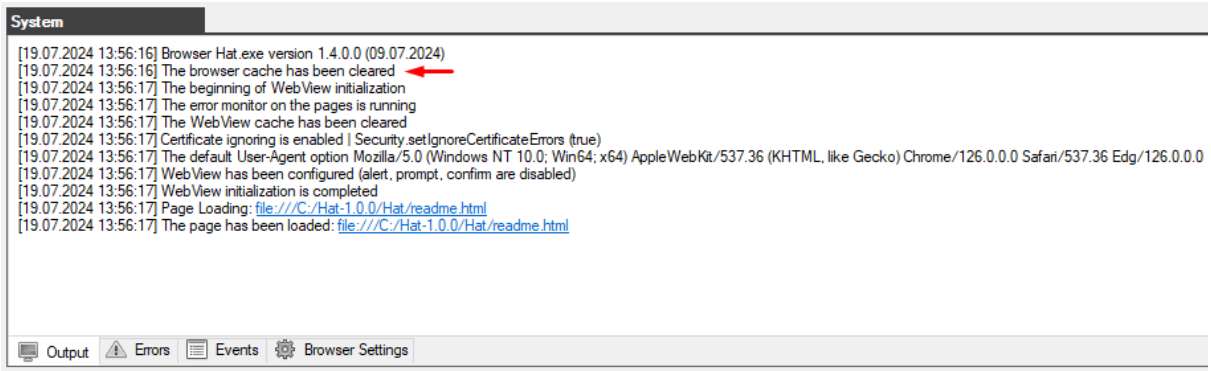
As a result, the project was created, the autotest was launched and a report was received.

Browser Cache

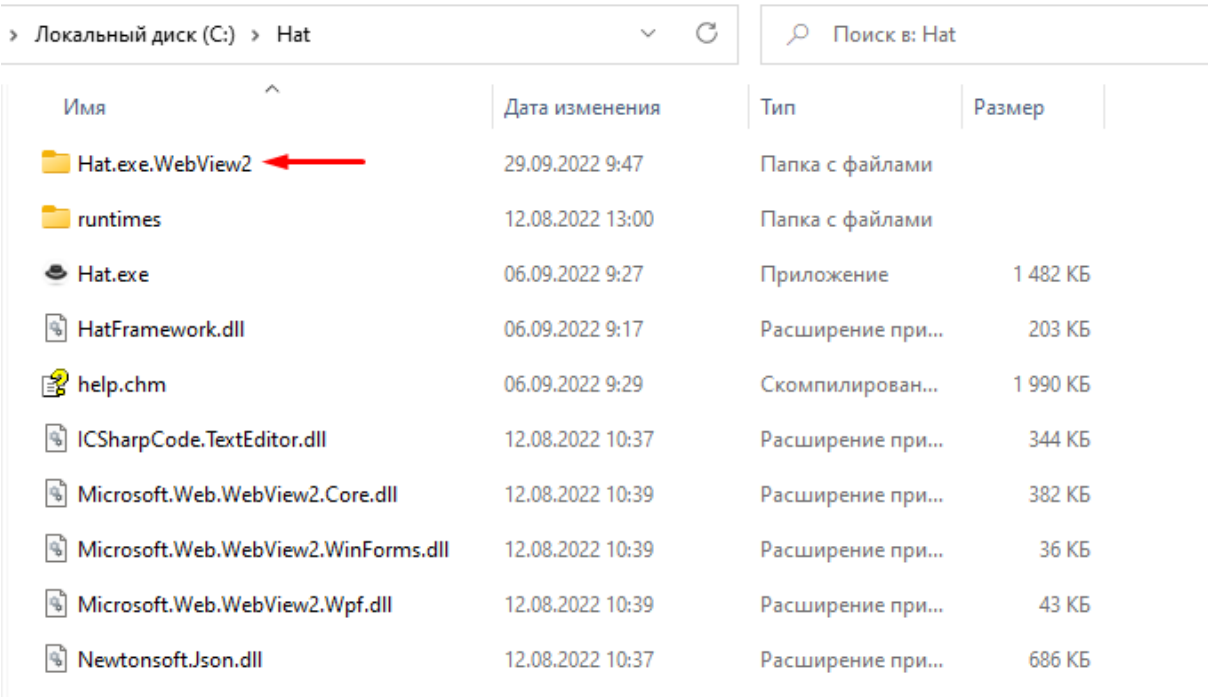
Clearing

Кэш браузера - очистка

The browser automatically cleans the cache before each launch and reports this to the console



You can also clear the cache manually.
To do this, you need to close the browser and delete the "Hat.exe.WebView2" folder



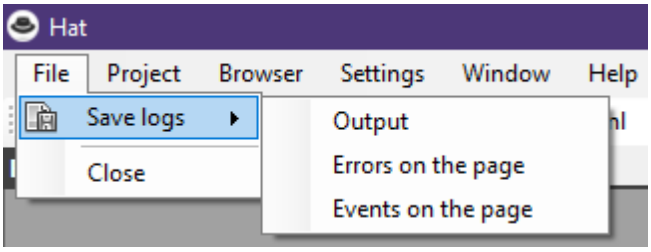
The entire cache is stored in the folder "Hat.exe.WebView 2", after you delete it and launch the browser, this folder will be created again and a new cache of the current session will be stored in it.

Interface

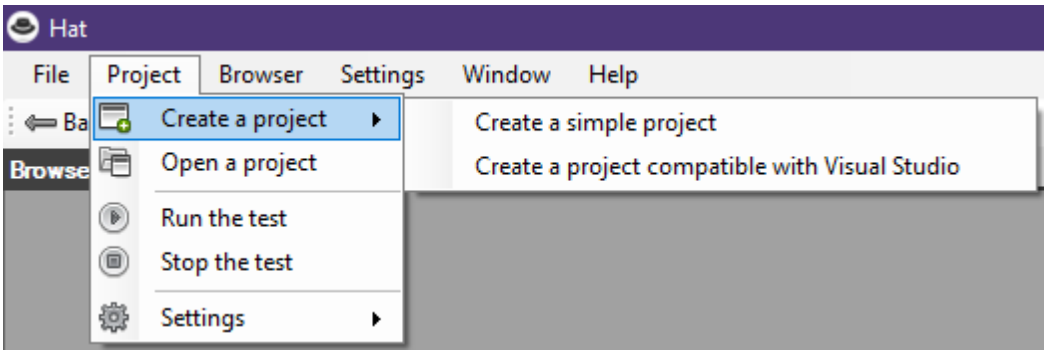
The main menu

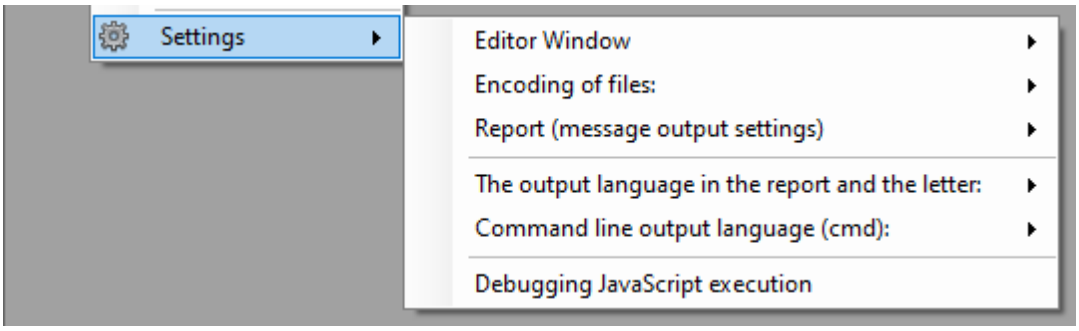
The main menu

Menu "File" - allows you to save logs to txt files.

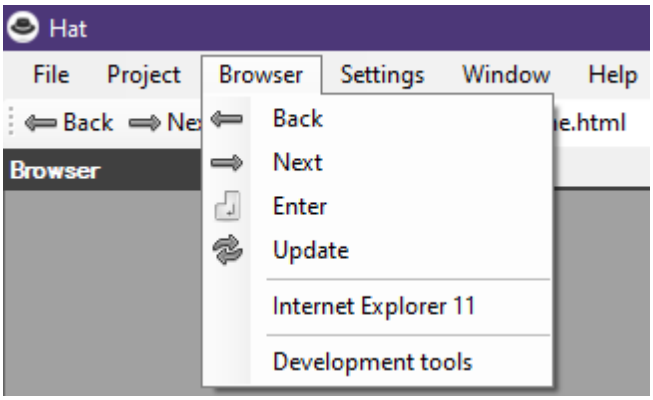


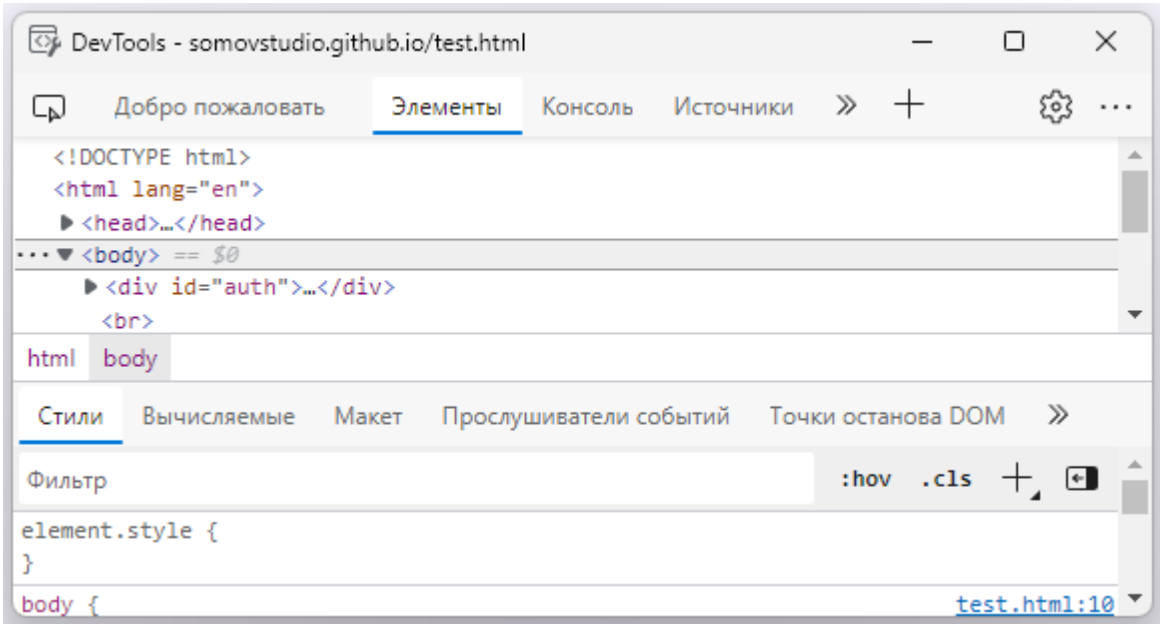
Menu "Project" - creating and opening autotest projects, performing and stopping autotests, as well as settings.



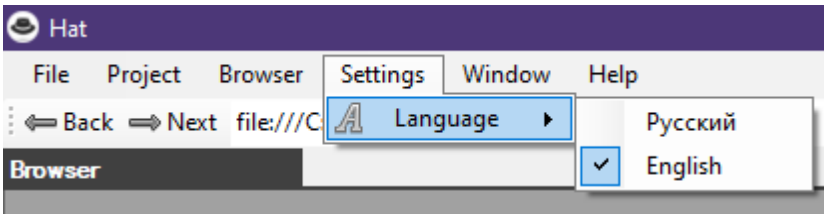


Menu "Browser" - it serves as a browser navigation, as well as allows you to open the developer dashboard "Development Tool" and open the Internet Explorer 11 window

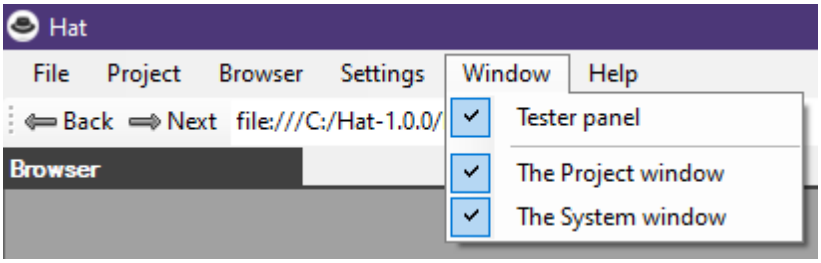




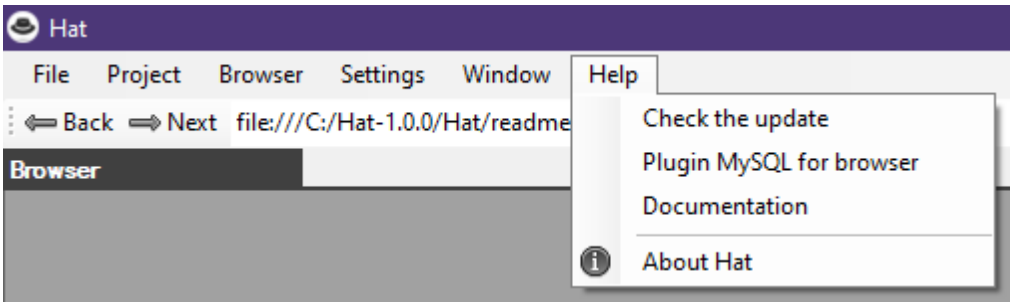
Menu "Settings" - setting the program language



Menu "Window" - allows you to hide or display the system browser windows.



Menu "Help" - contains background information and links to the update and plugin



The Toolbar

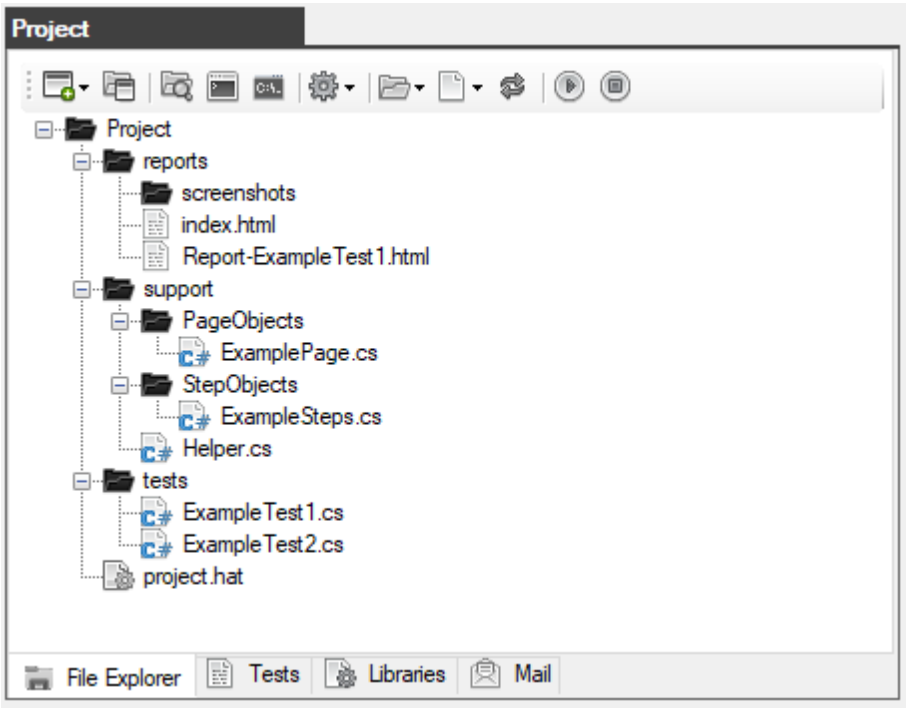
The Toolbar

Contains a field for entering the site address and browser navigation buttons.

The Project window - is the Explorer tab

The Project window - is the Explorer tab

The main window for working with the project.

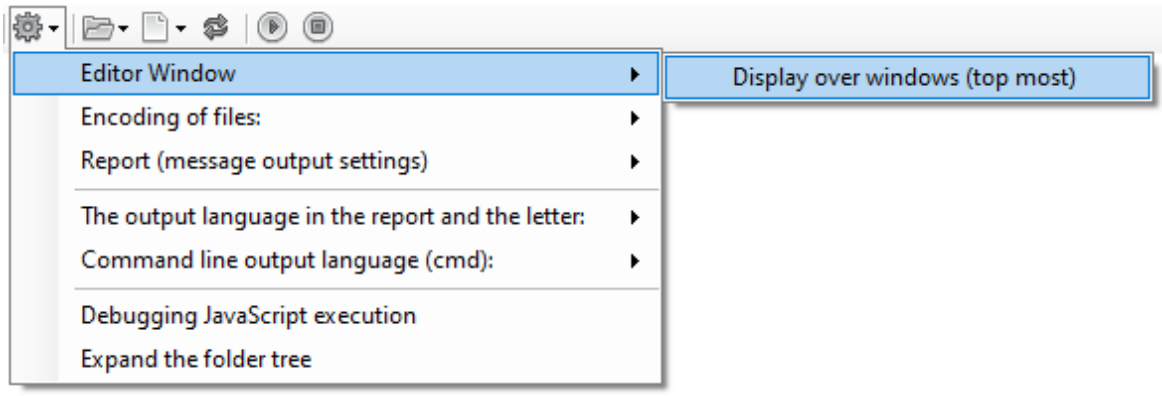


The toolbar contains the following buttons:

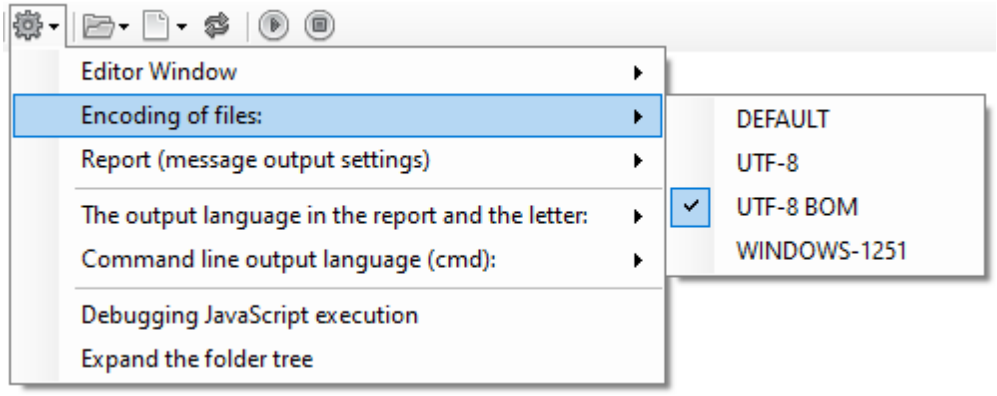
- Create a project - creates a new autotest project (simple or compatible with Visual Studio)
- Open a project - opens a previously created autotest project
- Explorer - opens the project folder in Windows Explorer
- Form a startup command - opens a window with an already formed autotest startup command for the Windows command prompt (cmd)
- Command Prompt (cmd) - opens the Windows system console
- Settings - the settings menu for working with the project
- Folders - menu for working with folders
- Files - menu for working with files
- Update - updating information in the folder and file tree
- Run the test - runs the selected autotest
- Stop the test - stops the running autotest

Settings Menu

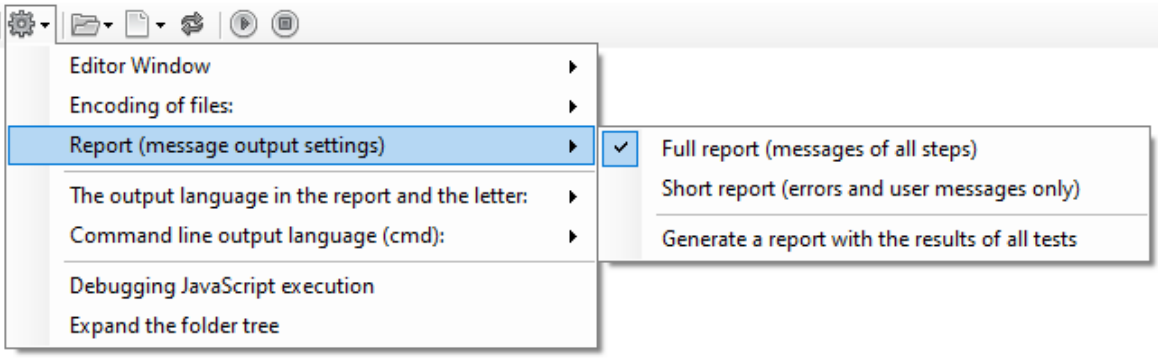
- Choosing the way to display the editor window



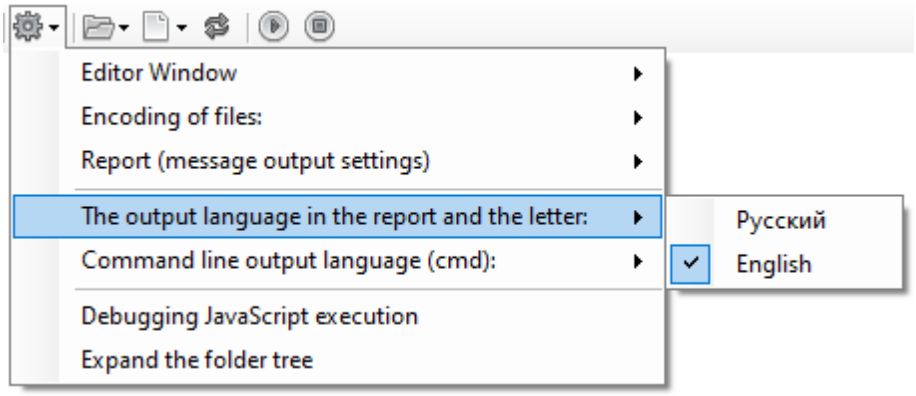
- Choosing the encoding for the project files



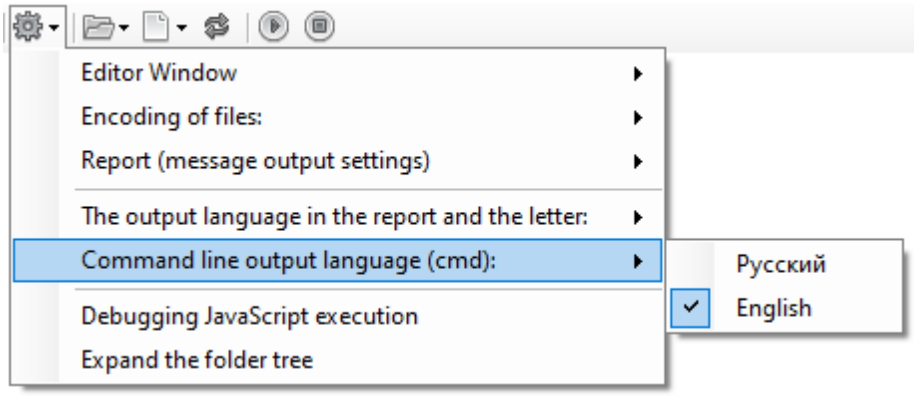
- Configuring the output of messages in the report, as well as generating a report with all the results of the checks.



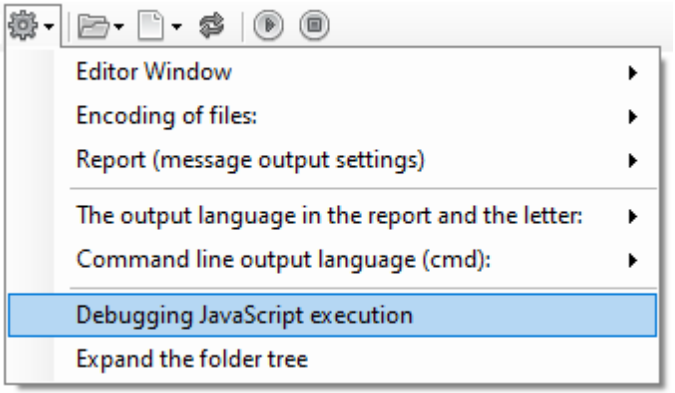
o Selecting the output language in the report and letter



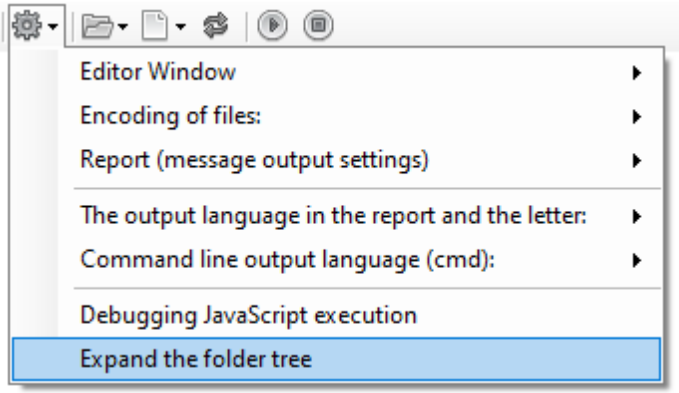
o Selecting the output language on the command line (cmd)



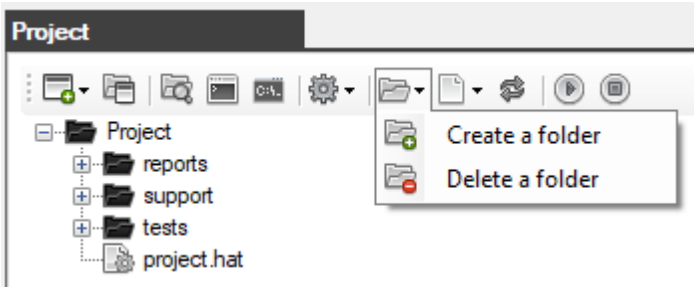
o Enabling debugging when executing JavaScript code



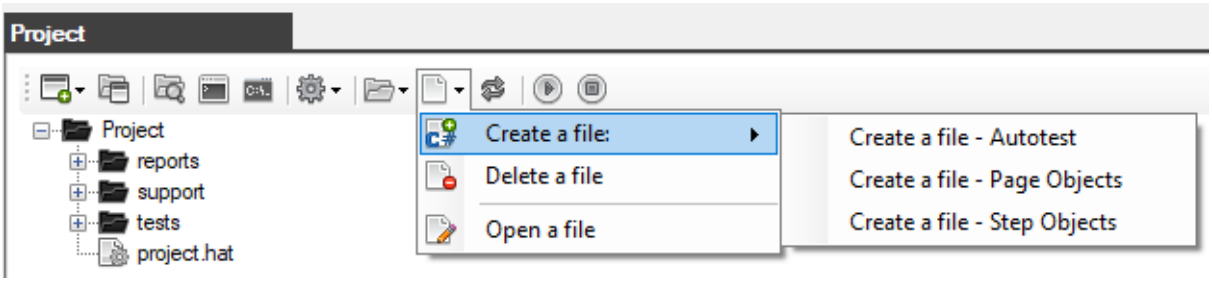
- Expand the folder and file tree in Explorer



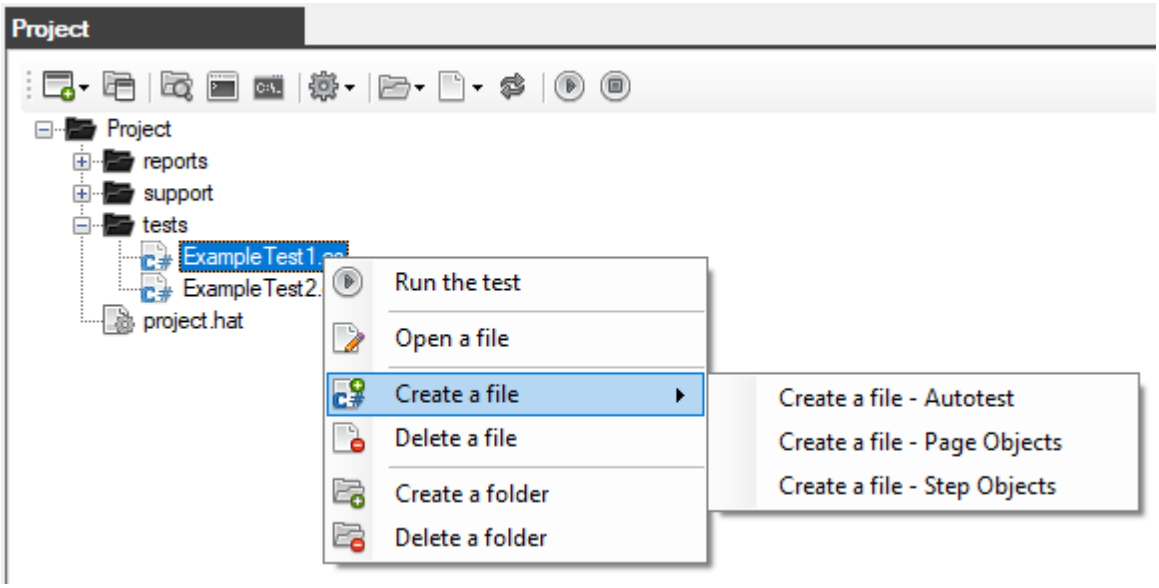
The folder menu allows you to create and delete folders



The file menu allows you to create new autotest files, open and delete existing files



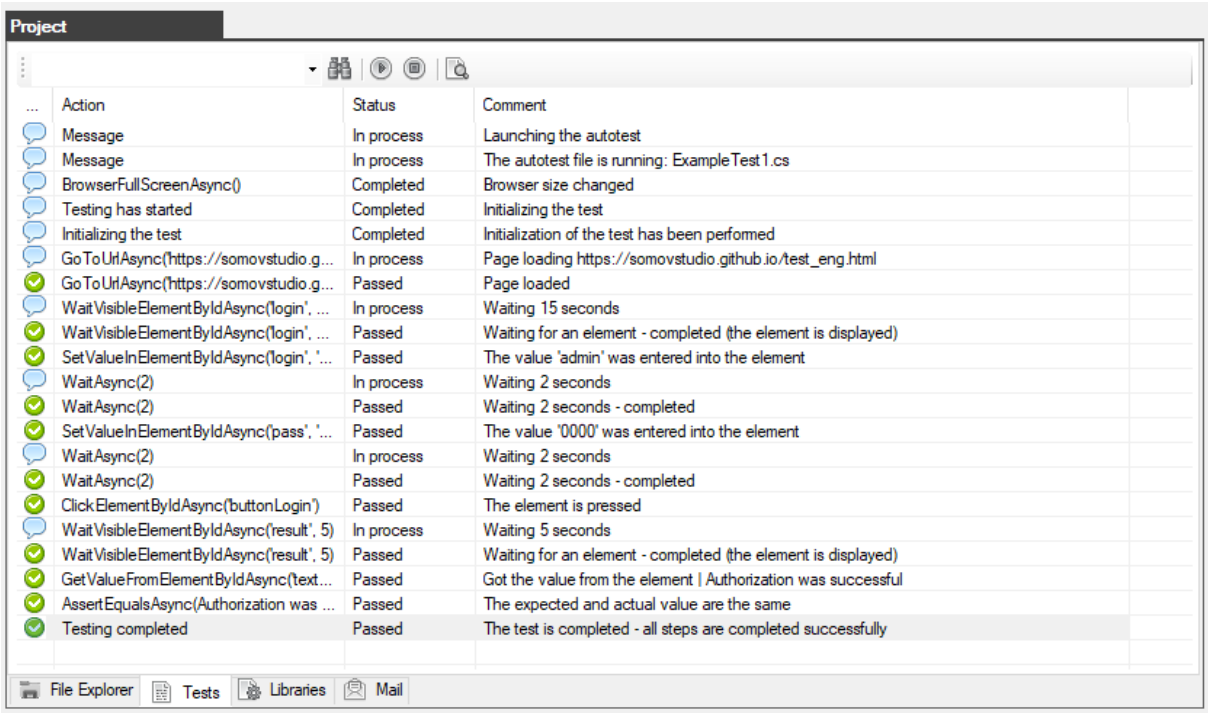
The tree of project folders and files

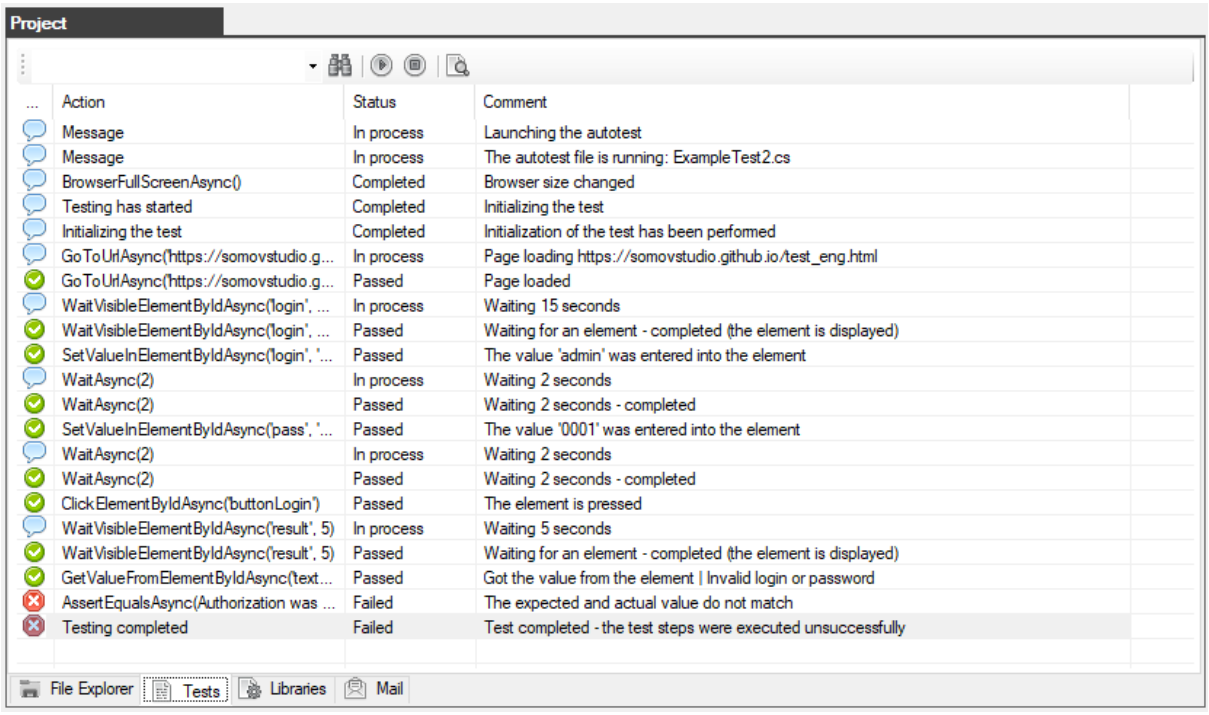


The Project window - is the Test tab

The Project window - is the Test tab

Table of autotest steps.





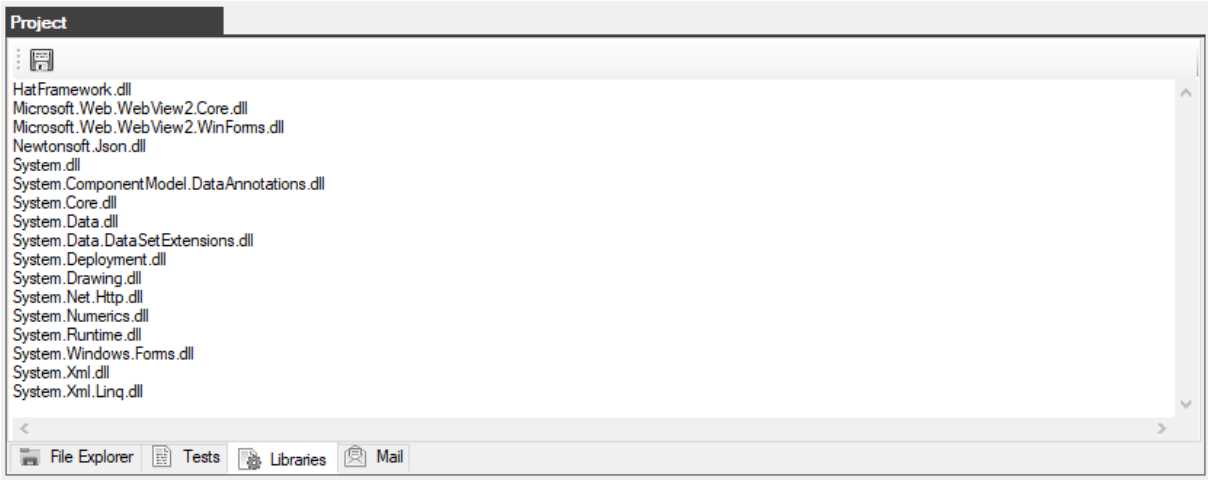
The toolbar contains the following buttons:

- Field and Search button - creates a new autotest project
- Open a project - opens a previously created autotest project
- Run the test - runs the selected autotest
- Stop the test - stops the running autotest
- Detailed information about the step - opens a window with a description of the step

The Project window - the Library tab

The Project window - the Library tab

List of plug-in libraries required to perform autotests



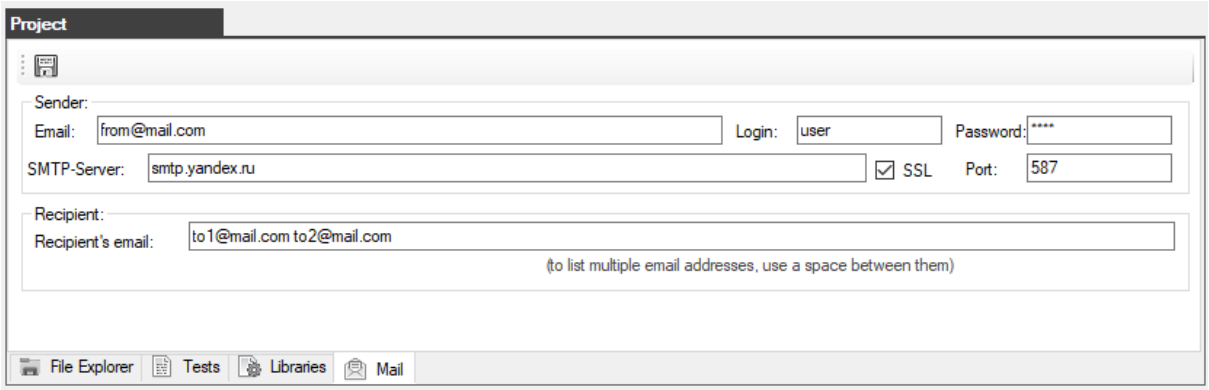
The toolbar contains the following buttons:

- Save - saves the list of libraries to the project.hat project file

The Project window - the Mail tab

The Project window - the Mail tab

Mail server connection settings for sending emails with reports on the results of passing autotests



The toolbar contains the following buttons:

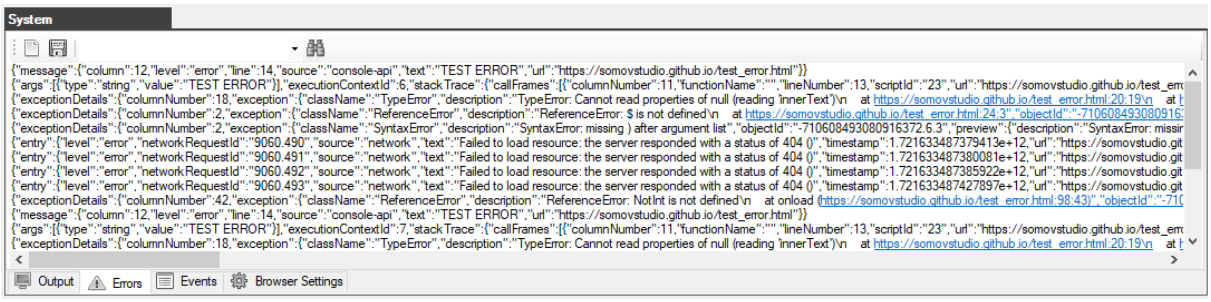
- Save - saves the settings to the project.hat project file

The system window

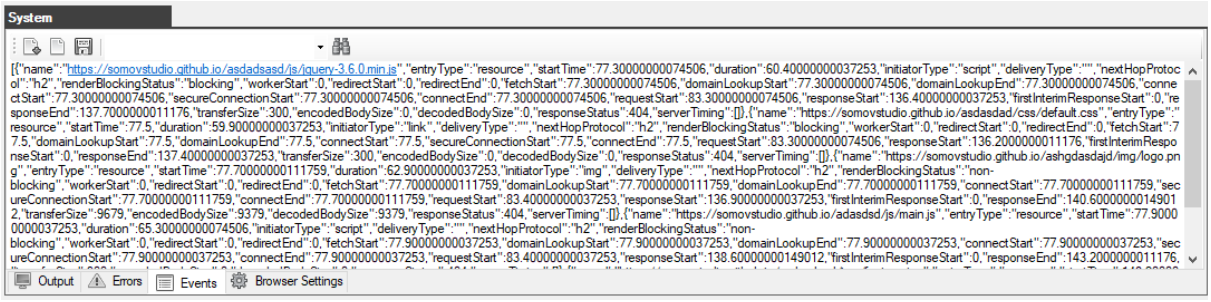
The system window

The "Output" tab displays all events that occur when the application is running.

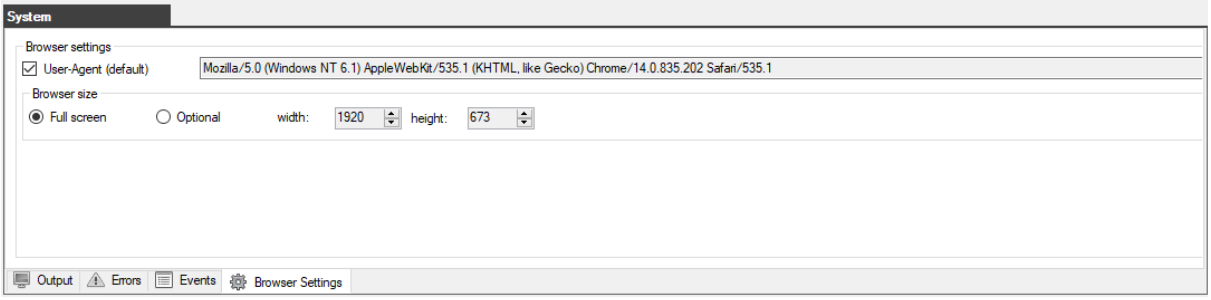
The "Errors" tab displays all errors (console) occurring on loaded pages.



The "Events" tab - displays all events (network) occurring on loaded pages.



The "Browser Settings" tab is the main browser settings.

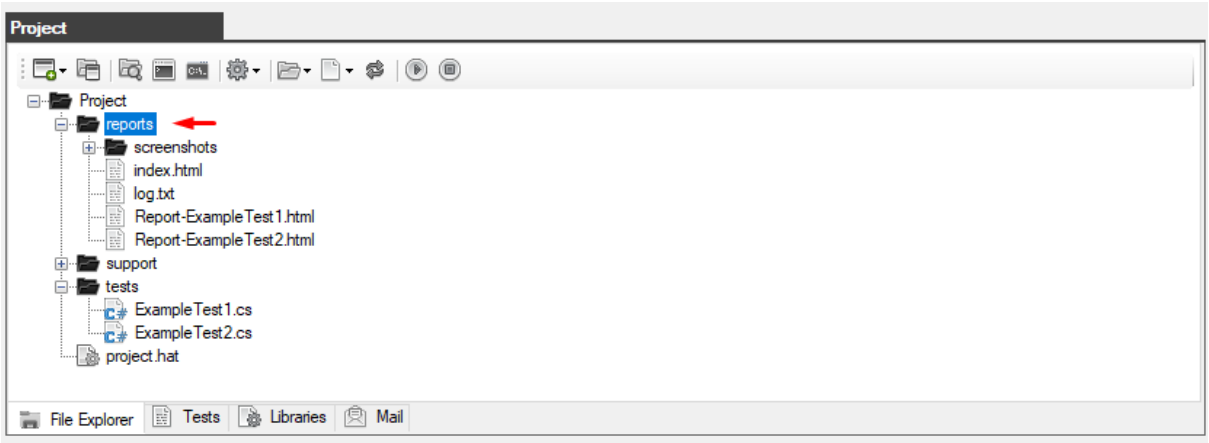


Reports

Autotest Report

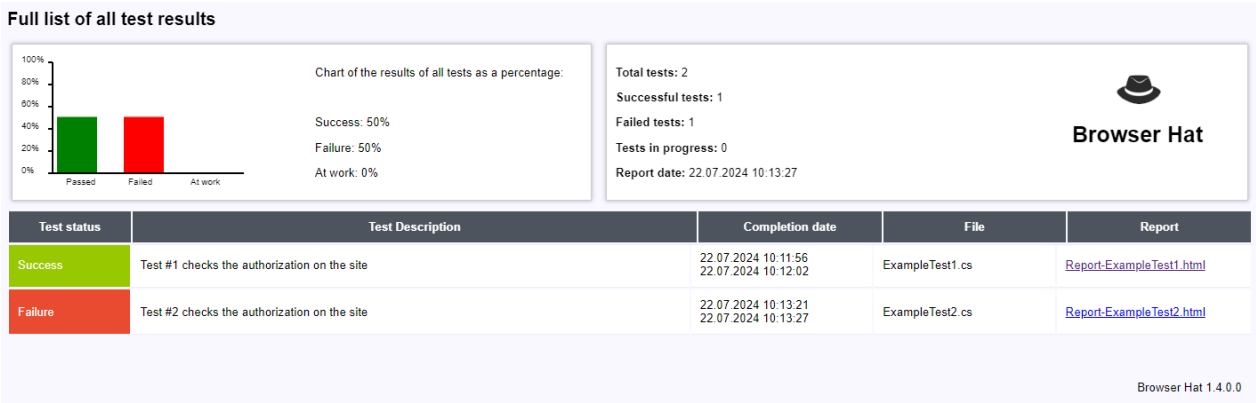
Autotest Report

When you run the autotest, a new reports folder will be created in the project folder if it does not already exist.
After completion of the autotest, a file with a report in html format will be created.
The name of the report file corresponds to the name of the autotest file with the prefix Report
For example, when performing the ExampleTest1.cs autotest, a report will be created with the name Report-ExampleTest1.html
To view the results for all completed autotests, a file is created and updated index.html

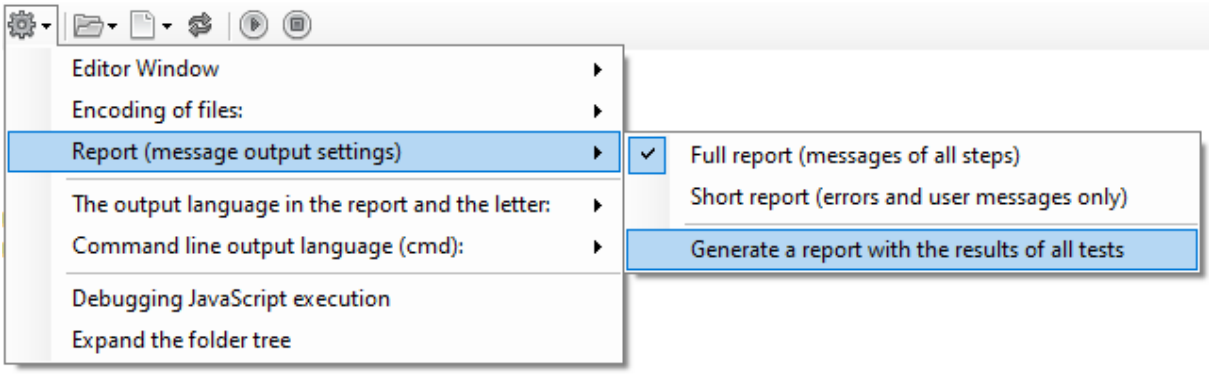


The resulting report opens in any browser

Full report on all completed inspections (index.html) it looks like this:



This report (index.html) is created or updated after the completion of any autotest.



This report can be generated or updated manually via the project settings menu.

The report with the successful completion of all steps and checks looks like this:

Autotest Report

Description: Test #1 checks the authorization on the site
File: ExampleTest1.cs
Date: 22.07.2024 10:11:56

Result: Success

Status	Action	Comment
Completed	BrowserFullScreenAsync()	Browser size changed
Completed	Testing has started	Initializing the test
Completed	Initializing the test	Initialization of the test has been performed
Passed	GoToUrlAsync('https://somovstudio.github.io/test_eng.html', 5, False)	Page loaded
Passed	WaitVisibleElementByIdAsync('login', 15)	Waiting for an element - completed (the element is displayed)
Passed	SetValueInElementByIdAsync('login', 'admin')	The value 'admin' was entered into the element
Passed	WaitAsync(2)	Waiting 2 seconds - completed
Passed	SetValueInElementByIdAsync('pass', '0000')	The value '0000' was entered into the element
Passed	WaitAsync(2)	Waiting 2 seconds - completed
Passed	ClickElementByIdAsync('buttonLogin')	The element is pressed
Passed	WaitVisibleElementByIdAsync('result', 5)	Waiting for an element - completed (the element is displayed)
Passed	GetValueFromElementByIdAsync('textarea')	Got the value from the element | Authorization was successful
Passed	AssertEqualsAsync(Authorization was successful, Authorization was successful)	The expected and actual value are the same
Passed	Testing completed	The test is completed - all steps are completed successfully

Browser Hat 1.4.0.0

The report on the failed test looks like this:

Autotest Report

Description: Test #2 checks the authorization on the site
File: ExampleTest2.cs
Date: 22.07.2024 10:13:21

Result: Failure

Status	Action	Comment
Completed	BrowserFullScreenAsync()	Browser size changed
Completed	Testing has started	Initializing the test
Completed	Initializing the test	Initialization of the test has been performed
Passed	GoToUrlAsync('https://somovstudio.github.io/test_eng.html', 5, False)	Page loaded
Passed	WaitVisibleElementByIdAsync('login', 15)	Waiting for an element - completed (the element is displayed)
Passed	SetValueInElementByIdAsync('login', 'admin')	The value 'admin' was entered into the element
Passed	WaitAsync(2)	Waiting 2 seconds - completed
Passed	SetValueInElementByIdAsync('pass', '0001')	The value '0001' was entered into the element
Passed	WaitAsync(2)	Waiting 2 seconds - completed
Passed	ClickElementByIdAsync('buttonLogin')	The element is pressed
Passed	WaitVisibleElementByIdAsync('result', 5)	Waiting for an element - completed (the element is displayed)
Passed	GetValueFromElementByIdAsync('textarea')	Got the value from the element | Invalid login or password
Failed	AssertEqualsAsync(Authorization was successful, Invalid login or password)	The expected and actual value do not match
Failed	Testing completed	Test completed - the test steps were executed unsuccessfully
Screenshot	File: image-22-07-2024-10-13-26.jpeg	

If any critical errors occur during execution, they will be reflected in the report as follows:

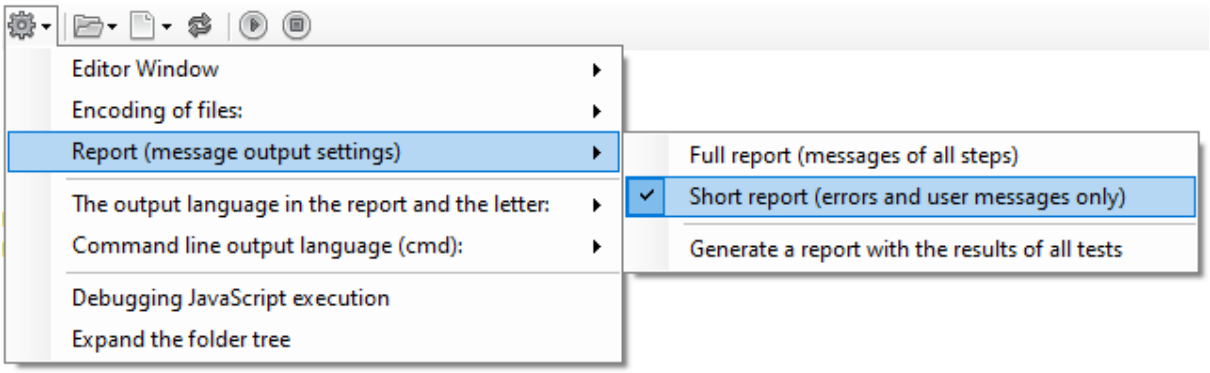
Отчет о работе автотеста

Файл: ExampleTest1.cs

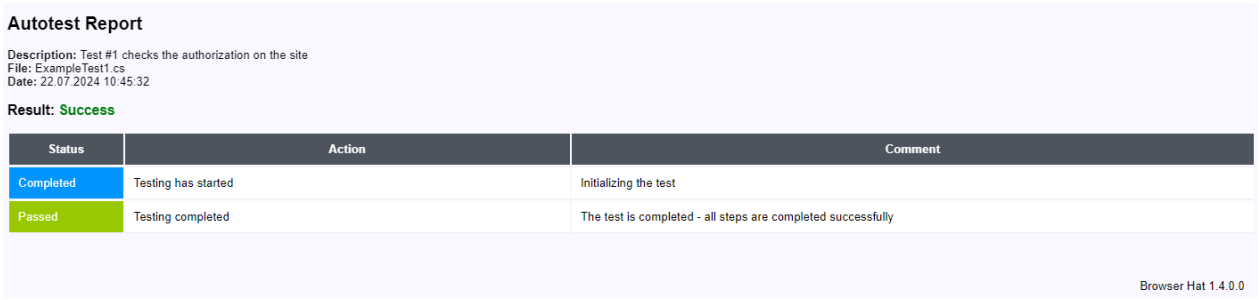
Результат: Провально

Статус	Действие	Комментарий
ОШИБКА		c:\Users\Catfish\Desktop\Hat\project\tests\ExampleTest1.cs(50,19) : error CS1502: Наиболее подходящий перегруженный метод для "HatFramework.Tester.ClickElementByIdAsync(string)" имеет несколько недопустимых аргументов
ОШИБКА		c:\Users\Catfish\Desktop\Hat\project\tests\ExampleTest1.cs(50,48) : error CS1503: Аргумент "1": преобразование типа из "int" в "string" невозможно
ОШИБКА		c:\Users\Catfish\Desktop\Hat\project\tests\ExampleTest1.cs(58,27) : warning CS1998: У этого асинхронного метода нет операторов "await", поэтому он будет выполняться синхронно. Рассмотрите возможность использования оператора "await" для ожидания неблокирующих вызовов API или конструкции "await Task.Run(...)" для выполнения фонового потока, ограниченного производительностью процессора.
ОШИБКА		c:\Users\Catfish\Desktop\Hat\project\tests\ExampleTest2.cs(53,27) : warning CS1998: У этого асинхронного метода нет операторов "await", поэтому он будет выполняться синхронно. Рассмотрите возможность использования оператора "await" для ожидания неблокирующих вызовов API или конструкции "await Task.Run(...)" для выполнения фонового потока, ограниченного производительностью процессора.
ОШИБКА		c:\Users\Catfish\Desktop\Hat\project\tests\ExampleTest3.cs(64,27) : warning CS1998: У этого асинхронного метода нет операторов "await", поэтому он будет выполняться синхронно. Рассмотрите возможность использования оператора "await" для ожидания неблокирующих вызовов API или конструкции "await Task.Run(...)" для выполнения фонового потока, ограниченного производительностью процессора.
Ошибок:	5	

Using the project settings menu, you can switch the report type to "Summary Report", in which case only user messages and error messages will be displayed in the report.



So the reports will look like this:



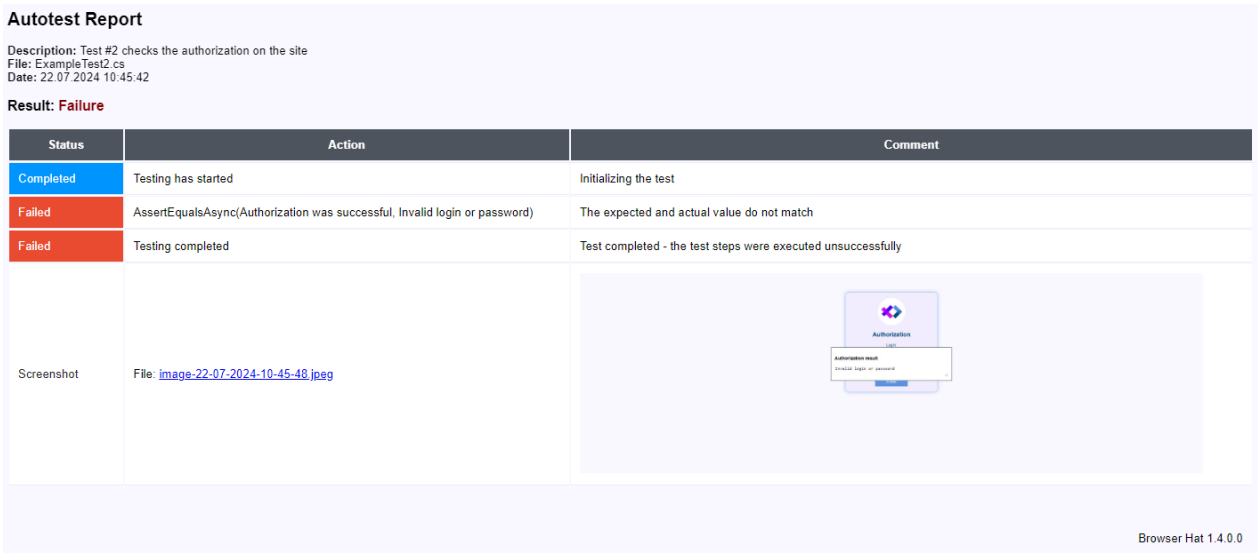
Autotest Report

Description: Test #1 checks the authorization on the site
File: ExampleTest1.cs
Date: 22.07.2024 10:45:32

Result: Success

Status	Action	Comment
Completed	Testing has started	Initializing the test
Passed	Testing completed	The test is completed - all steps are completed successfully

Browser Hat 1.4.0.0



Autotest Report

Description: Test #2 checks the authorization on the site
File: ExampleTest2.cs
Date: 22.07.2024 10:45:42

Result: Failure

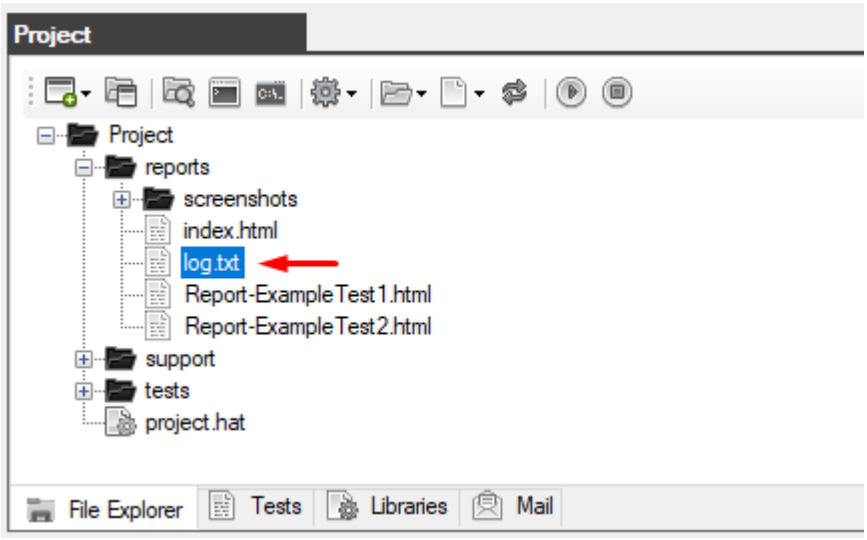
Status	Action	Comment
Completed	Testing has started	Initializing the test
Failed	AssertEqualsAsync(Authorization was successful, Invalid login or password)	The expected and actual value do not match
Failed	Testing completed	Test completed - the test steps were executed unsuccessfully
Screenshot	File: image-22-07-2024-10-45-48.jpeg	

Browser Hat 1.4.0.0

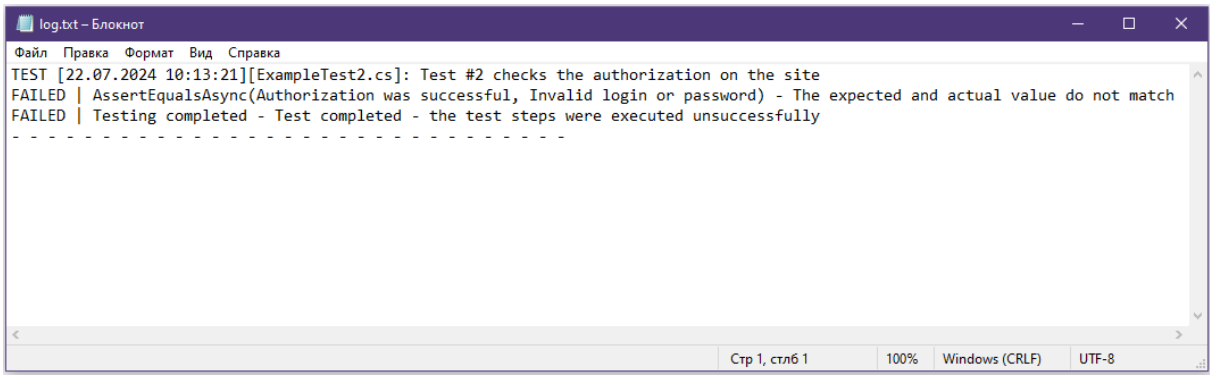
Log file

Log file (log.txt)

If the autotest fails, not only a report is created, but also an entry is made in the log file log.txt



The file records the date and time of the failure, the file name and description of the autotest, as well as the reasons for the failure of the autotest.

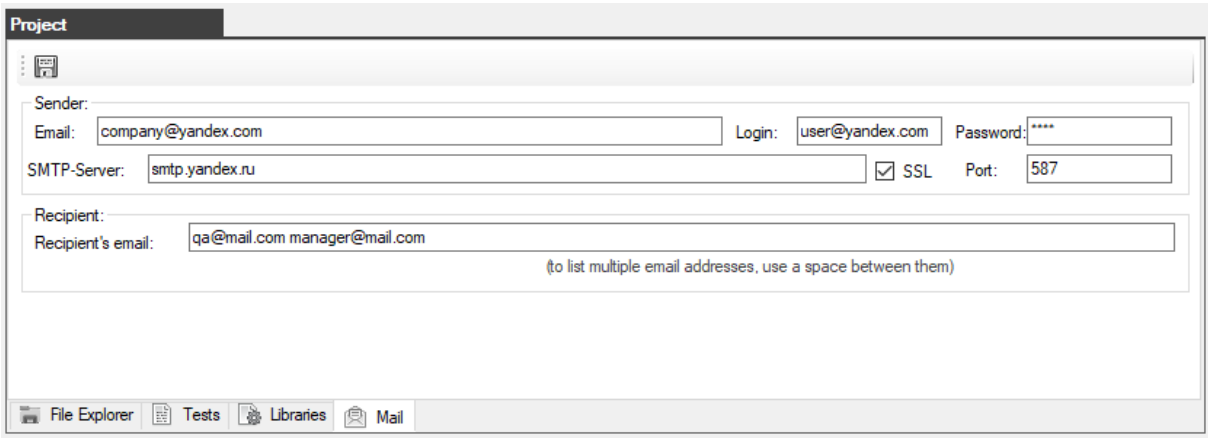


Mail

Send a report on the failure of the autotest to the mail

Send a report on the failure of the autotest to the mail

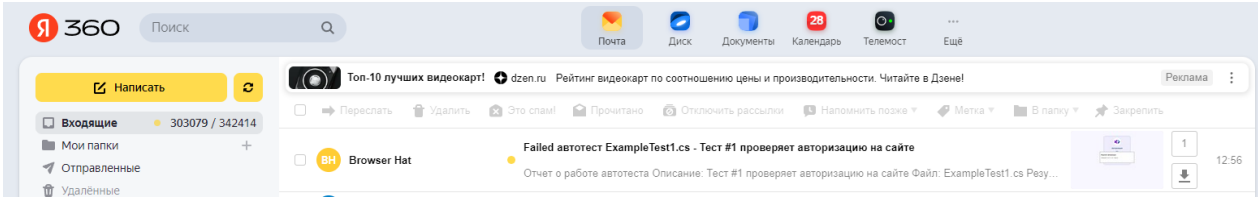
The connection to the mail server is configured in the "Project" window.
This example uses a Yandex mailbox.



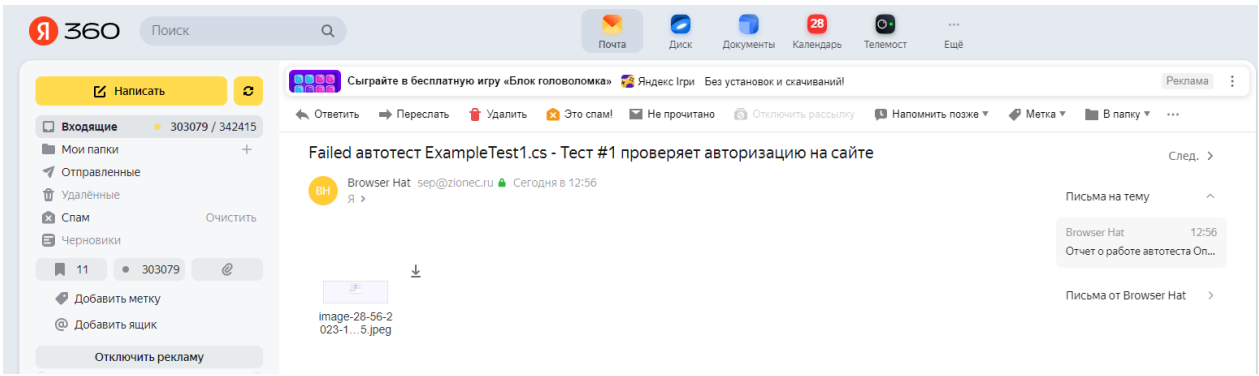
To use the function of sending a report to the mail, you need to enable this option in the autotest using the method BrowserEnableSendMailAsync

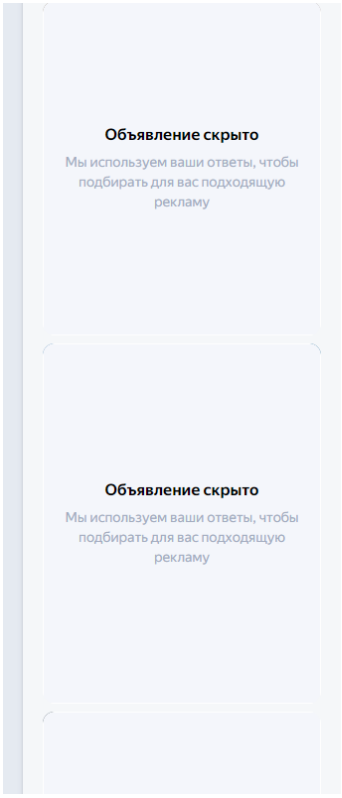
```
        public async Task setUp()
        {
            await tester.BrowserEnableSendMailAsync(true, false); // only in case of failure
            await tester.BrowserEnableSendMailAsync(false, true); // only if successful
            await tester.BrowserEnableSendMailAsync(); // in both cases
        }
```

In case of failure, a report of the following type will be sent to the listed email addresses



The email will include a screenshot at the time of the error and a report with the steps of execution, as well as user messages.





Отчет о работе автотеста

Описание: Тест #1 проверяет авторизацию на сайте

Файл: ExampleTest1.cs

Результат: **Провально**

Статус	Действие	Комментарий
Выполнено	Сообщение	Запущен автотест из файла: ExampleTest1.cs
Выполнено	BrowserFullScreenAsync()	Размер браузера изменён
Выполнено	BrowserEnableSendMailAsync("True", "True")	Включена опция отправки отчета на почту
Выполнено	Тестирование началось	Выполнена инициализация теста
Успешно	GoToUrlAsync('https://somovstudio.github.io/test.html', 5)	Страница загружена
Успешно	WaitVisibleElementByIdAsync('login', 15)	Ожидание элемента - завершено (элемент отображается)
Успешно	SetValueInElementByIdAsync('login', 'admin')	Значение введено в элемент
Успешно	WaitAsync(2)	Ожидание 2 секунд - завершено
Успешно	SetValueInElementByIdAsync('pass', '0001')	Значение введено в элемент
Успешно	WaitAsync(2)	Ожидание 2 секунд - завершено
Успешно	ClickElementByIdAsync('buttonLogin')	Элемент нажат
Успешно	WaitVisibleElementByIdAsync('result', 5)	Ожидание элемента - завершено (элемент отображается)
Успешно	GetValueFromElementByIdAsync('textarea')	Получено значение из элемента
Провально	AssertEqualsAsync(Вы успешно авторизованы, Неверный логин или пароль)	Ожидаемое и актуальное значение не совпадают
Провально	Тестирование завершено	Тест завершен - шаги теста выполнены неуспешно
Скриншот	Файл: image-28-56-2023-12-56-35.jpeg	

You can send an email at any time, you need to use the function SendMsgToMailAsync

```
        public async Task tearDown()
        {
            if(tester.GetTestResult() == Tester.PASSED)
            {
                await tester.SendMsgToMailAsync("Text message", "The test was
completed successfully");
            }
        }
```

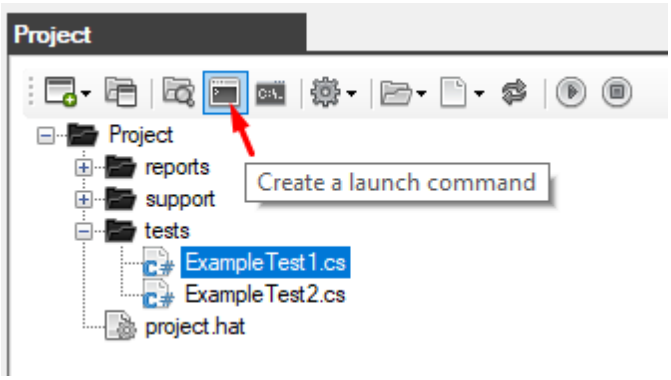

Console launch mode (Jenkins)

Running the autotest from the command line

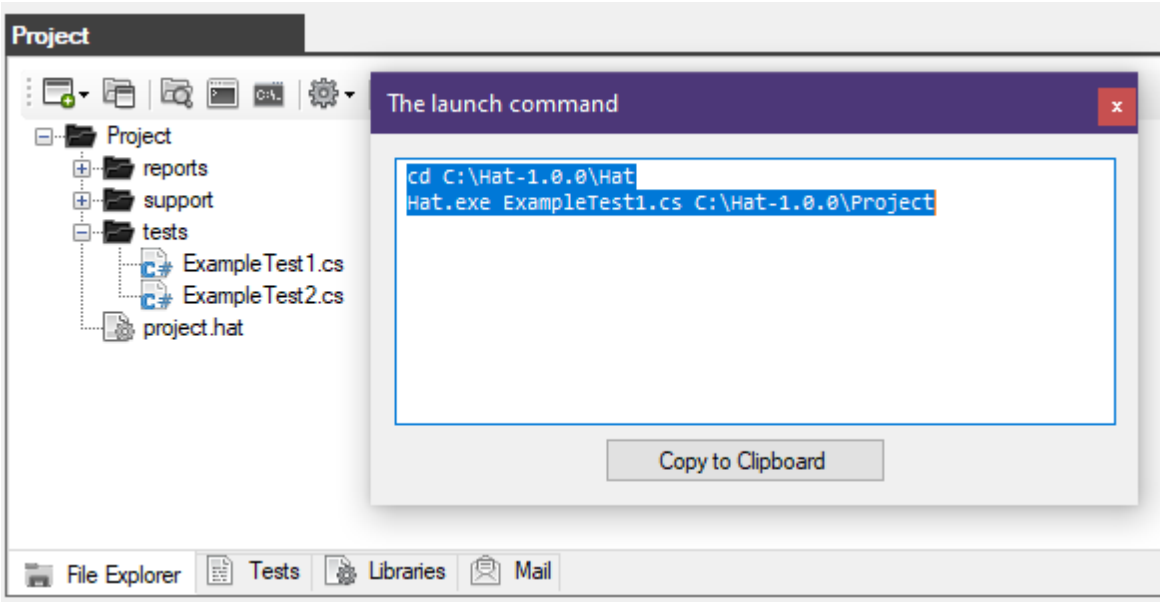
Running the autotest from the command line

To run autotests in popular continuous integration systems, you need to use the Windows command line.

In the "Project" window (on the "Explorer" tab), select the autotest file that you want to run. Then click on the "Create a launch command" button on the toolbar



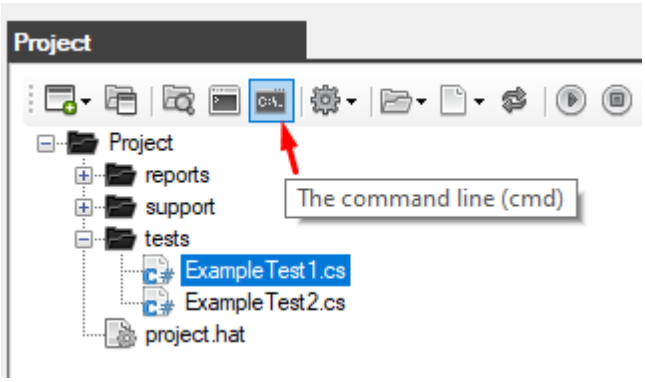
As a result, you will see a window with ready-made commands.



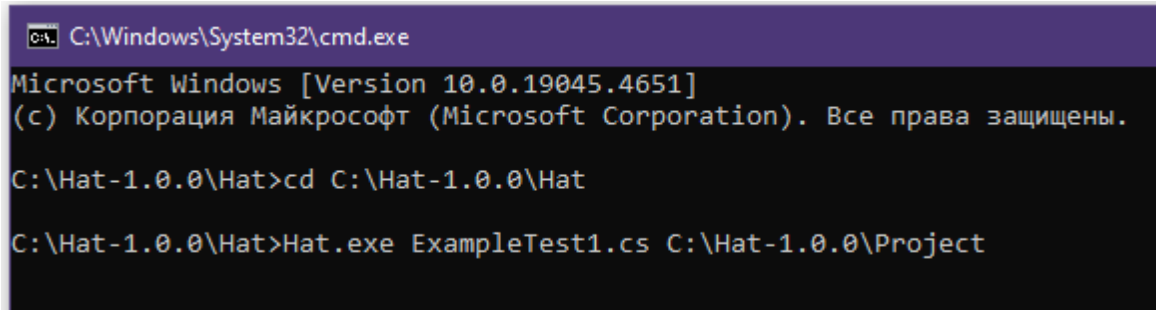
- the first command: **`cd C:\Hat-1.0.0\Hat`** - this is a transition to the folder with the application Hat.exe
- the second command: **`Hat.exe ExampleTest1.cs C:\Hat-1.0.0\Project`** - runs the autotest

The syntax of the startup command:
Hat.exe [**Autotest_file_name**] [**Path_To_The_Project_Folder**]

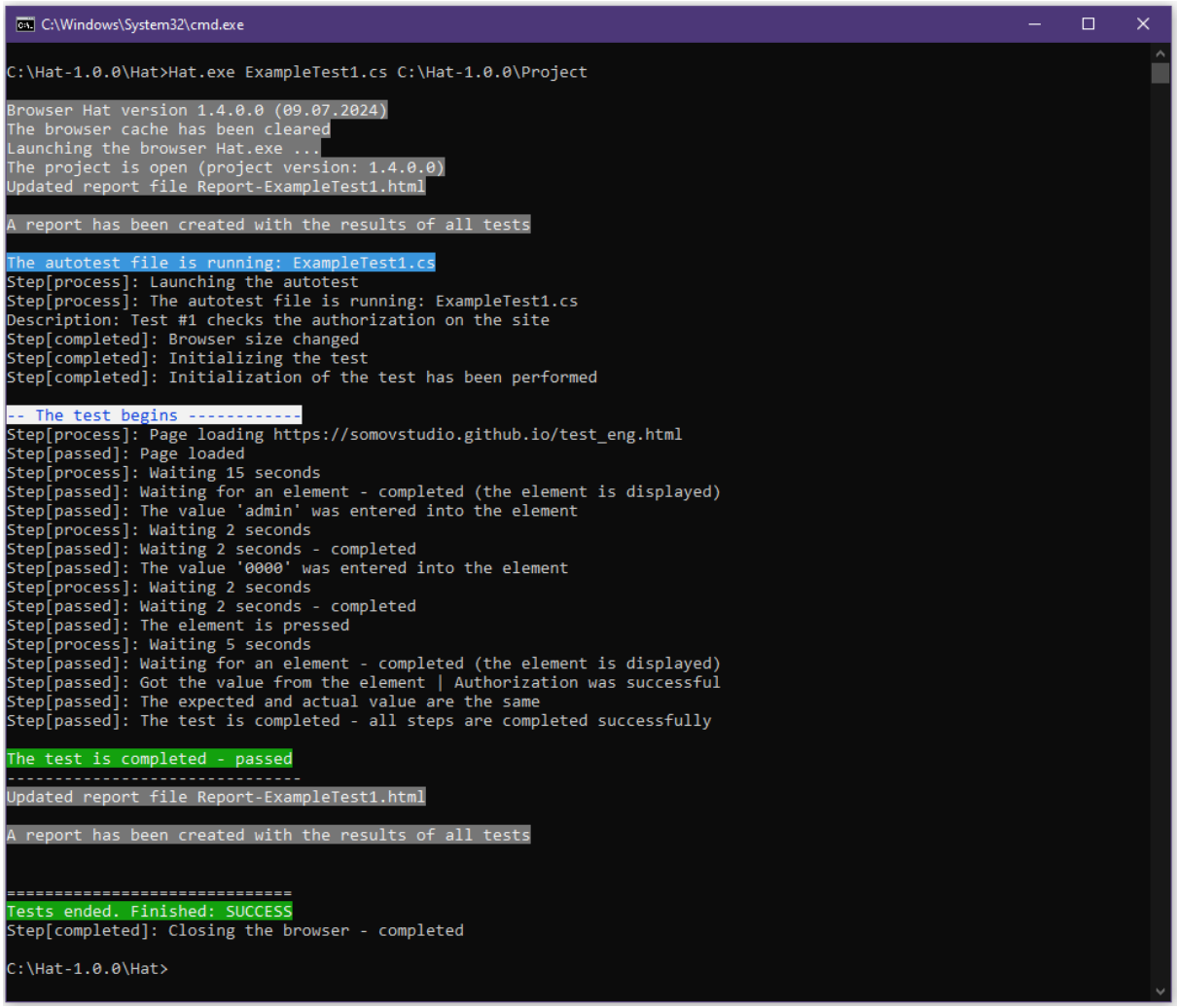
Open the command prompt



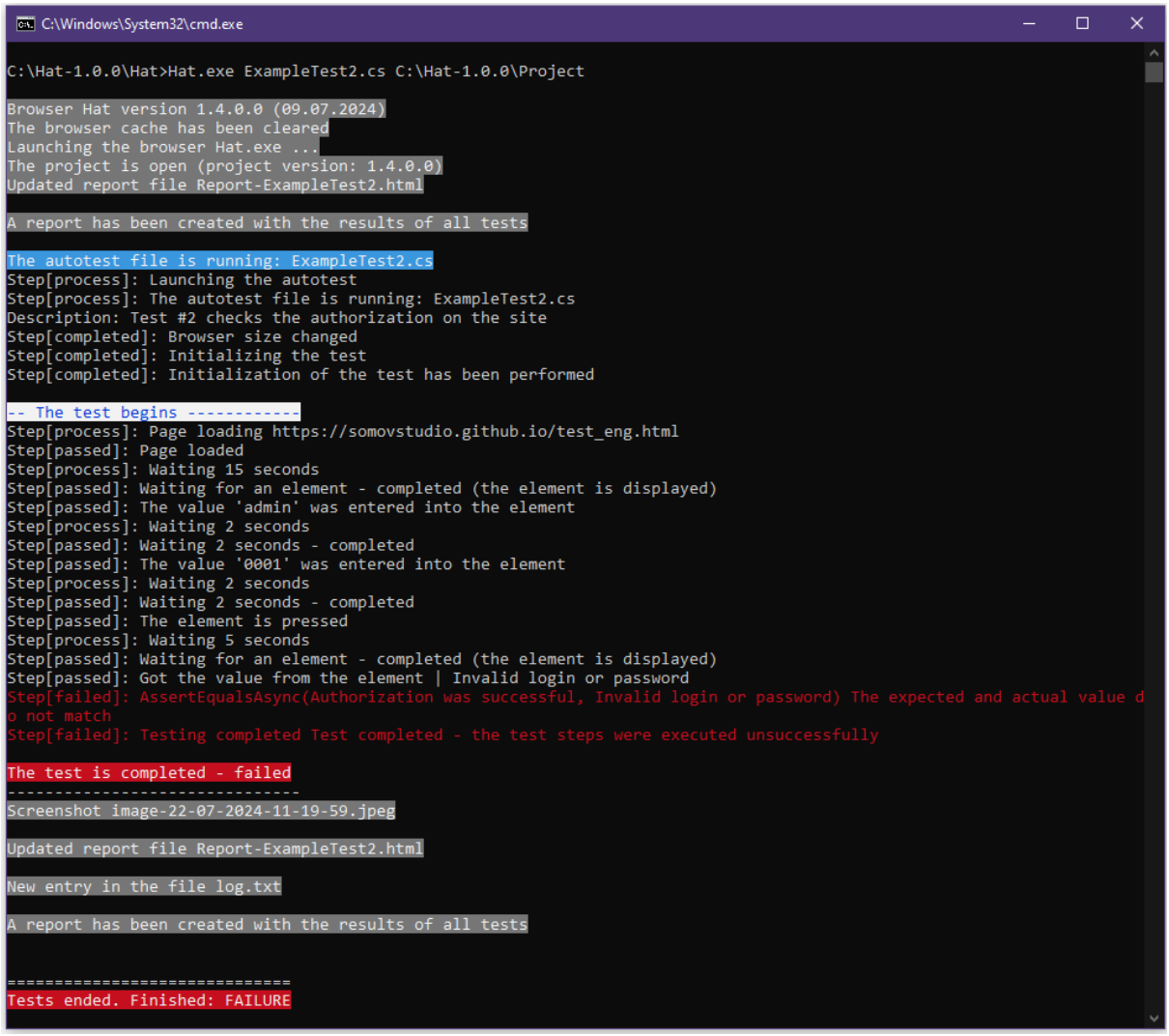
Use the received commands



The autotest will be launched, and all the execution steps will be reflected in the console



In case of an error, the step will show the method and its properties where the error occurred.



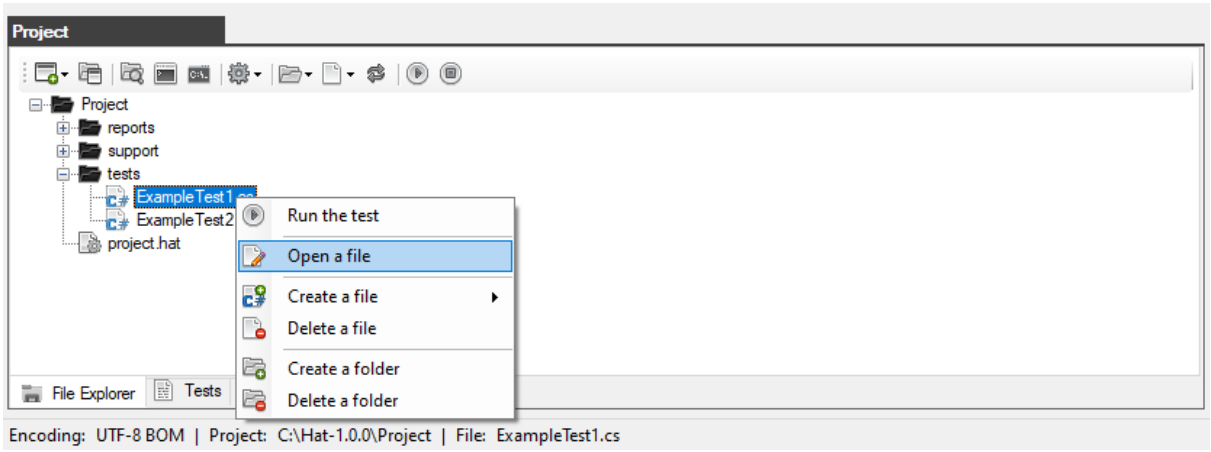
At the end, the result of the entire autotest is shown.

A text code editor

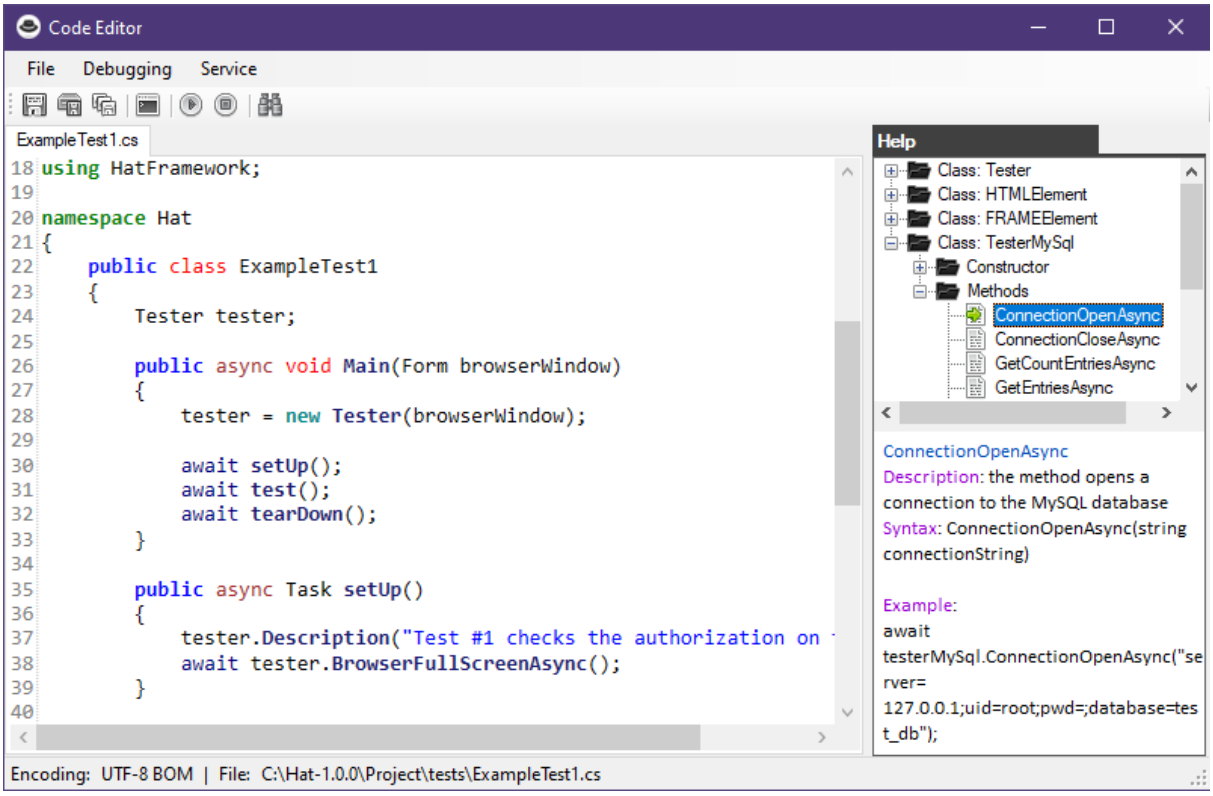
Code Editor (C#)

Code Editor (C#)

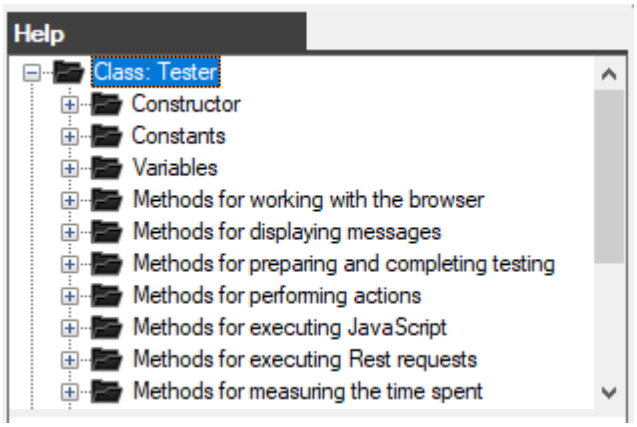
Autotests are edited in a special editor.
To open a file with the *.cs extension in the editor, you need to double-click on the desired file in the Project window (on the Explorer tab) or open the context menu and select "Open file".

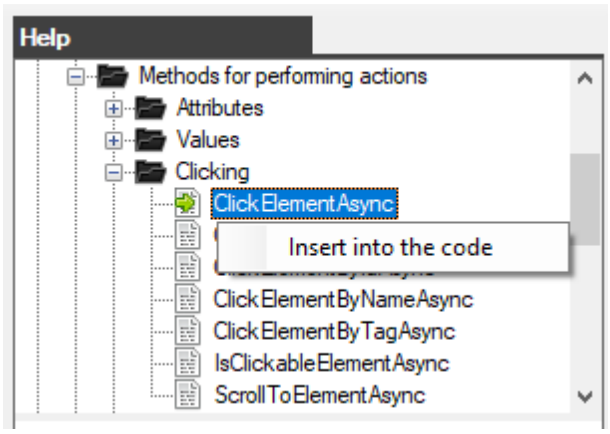


The following editor window will open in front of you.



An editor with keyword highlighting, as well as reference information about the built-in methods of the Net Framework framework.
You do not need to manually type the names of the methods, it will be enough to click on it twice (or through the context menu item) so that the method is automatically inserted into the code.

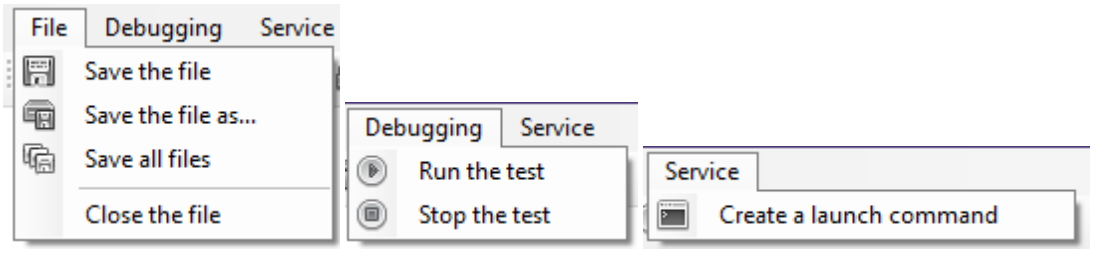




There are buttons on the toolbar:

- Save File - saves changes to the current file
- Save file as - allows you to save the script to a new file
- Save all files - saves changes to all open files
- Form a startup command - opens a window with an already formed autotest startup command for the Windows command line (cmd)
- Run the test - runs the selected autotest
- Stop the test - stops the running autotest
- Search - executes the search box

The main menu



File:

- Save File - saves changes to the current file
- Save file as - allows you to save the script to a new file

- Save all files - saves changes to all open files
- Close file - closes the file in the current tab

Debugging:

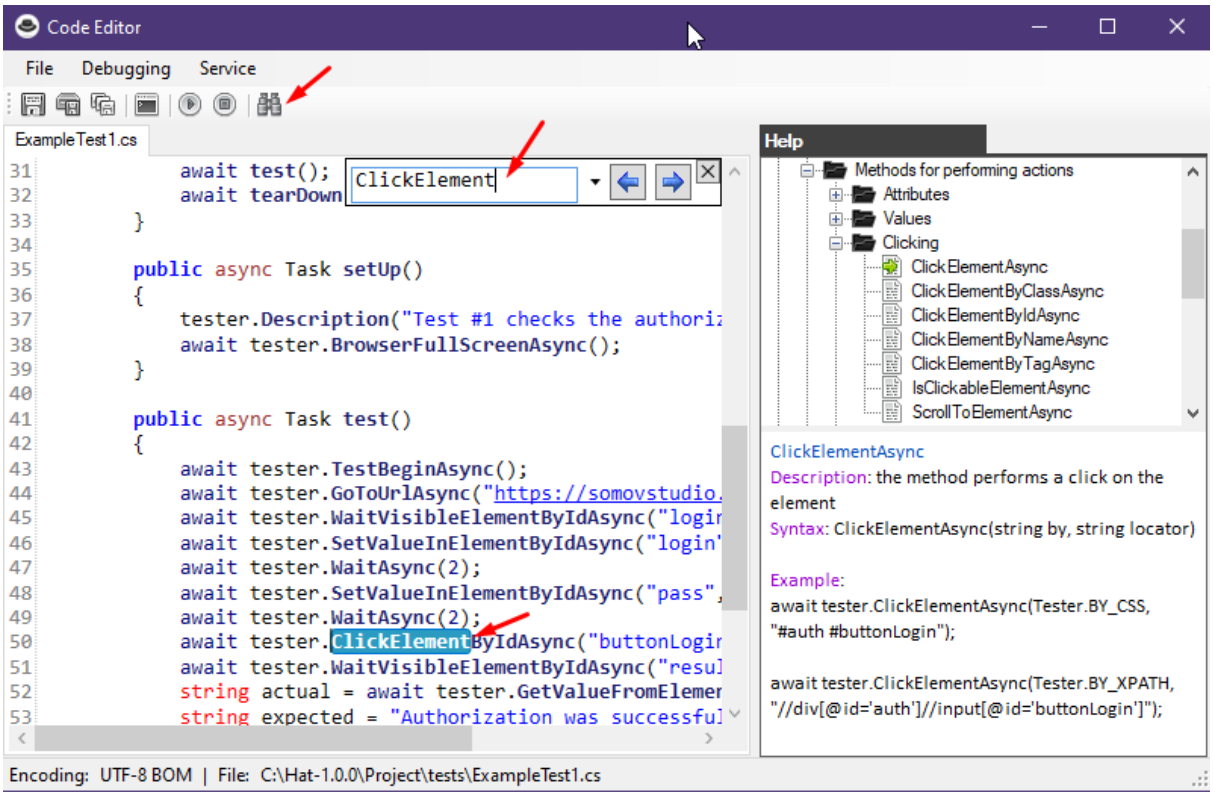
- Run the test - runs the selected autotest
- Stop the test - stops the running autotest

Service:

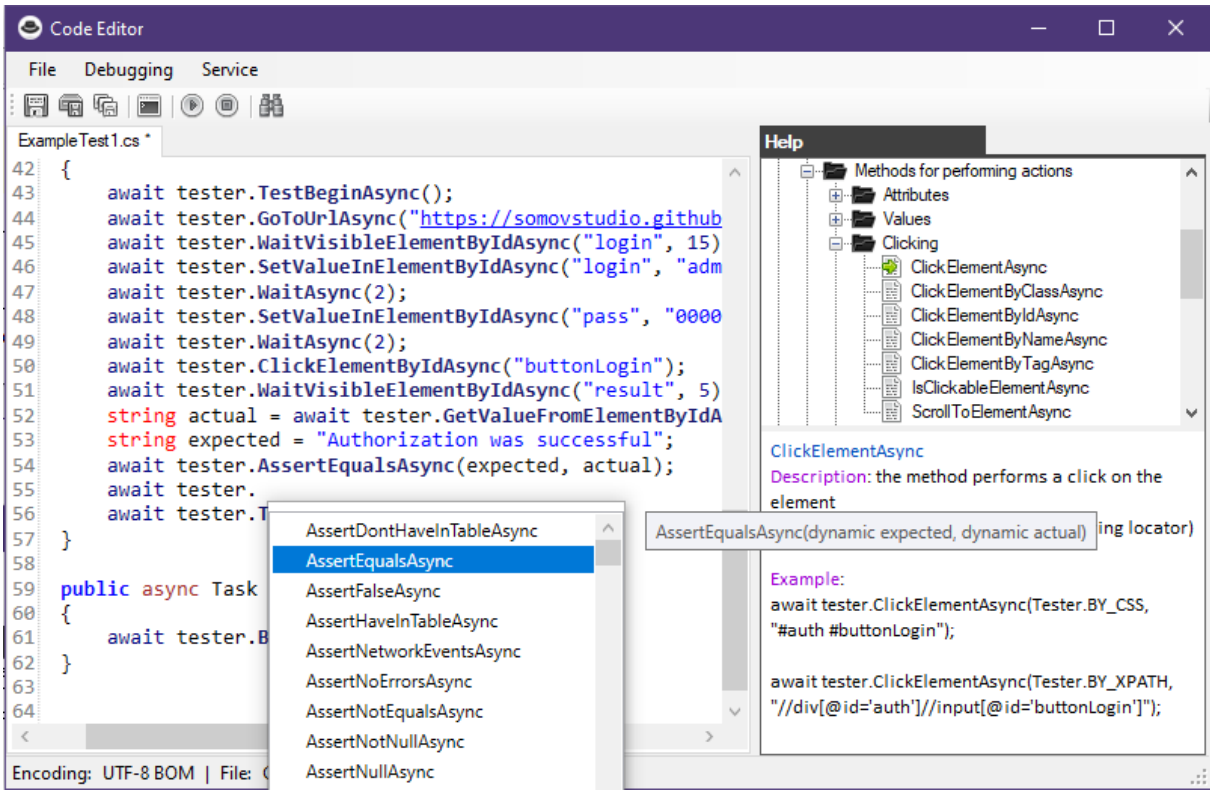
- Form a startup command - opens a window with an already formed autotest startup command for the Windows command line (cmd)

Performing a search

To open the search window, click on the search button or press CTRL+F.



The editor allows you to export a list of built-in methods to simplify writing code.

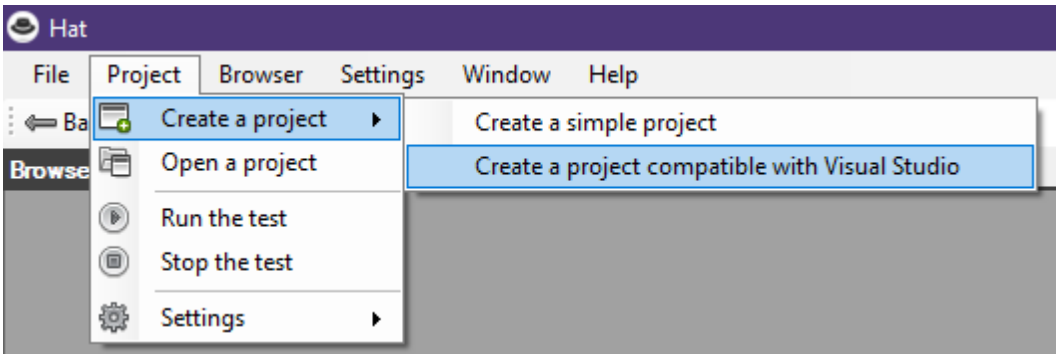


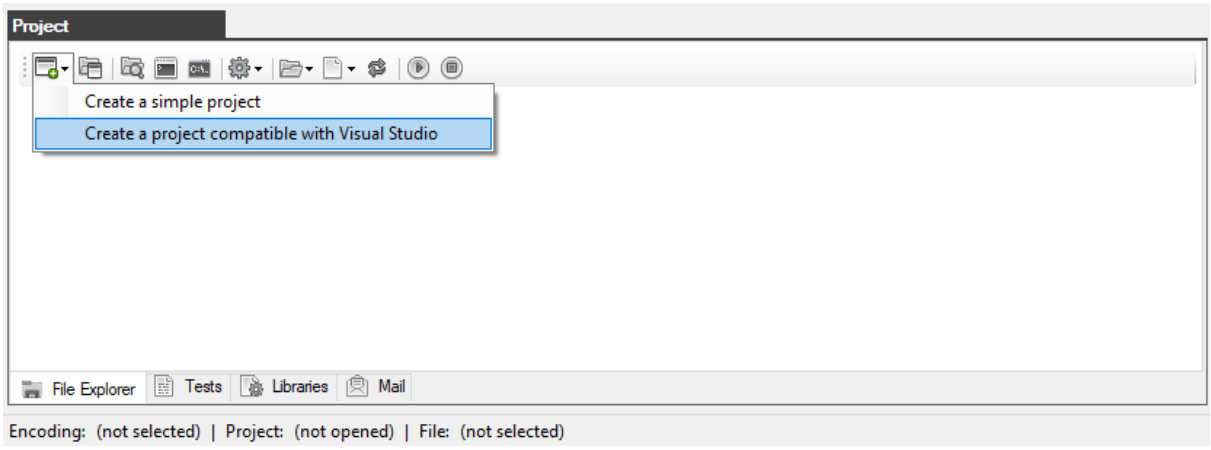
Using Visual Studio to edit code

Using Visual Studio to edit code

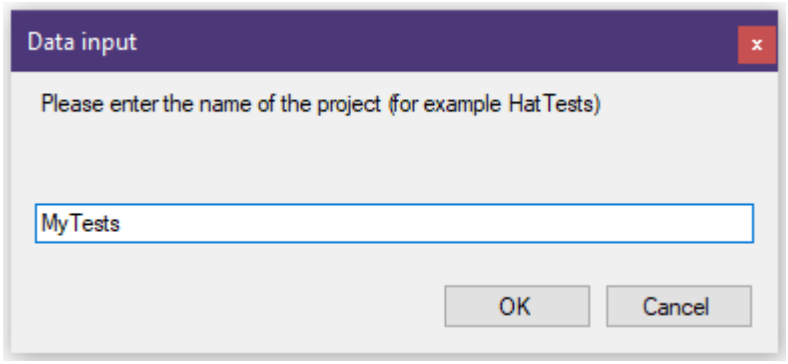
In order to use the autotest project as a Visual Studio editor, it must be created as follows:

1. In the Project window, on the Explorer tab or in the main Create Project menu, click the "Create a project compatible with Visual Studio" button

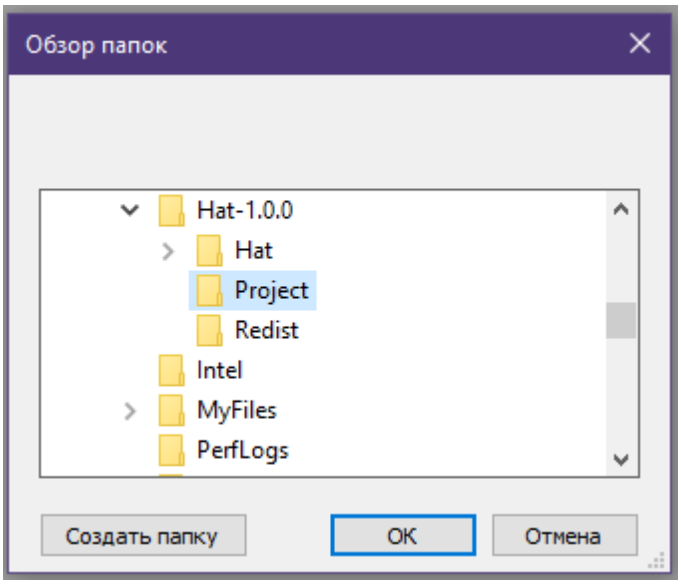




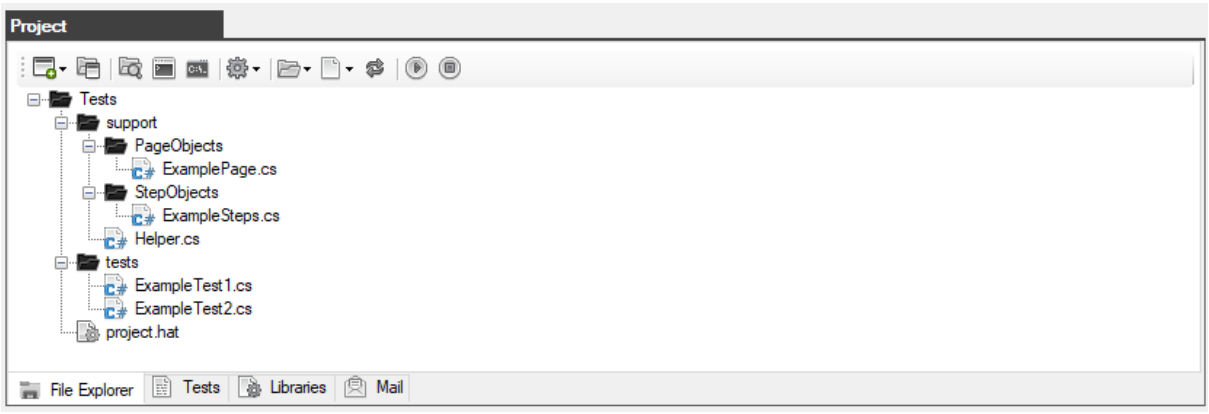
2. A pop-up will open to enter the project name.
Prerequisite the project name must be in Latin letters and without spaces.



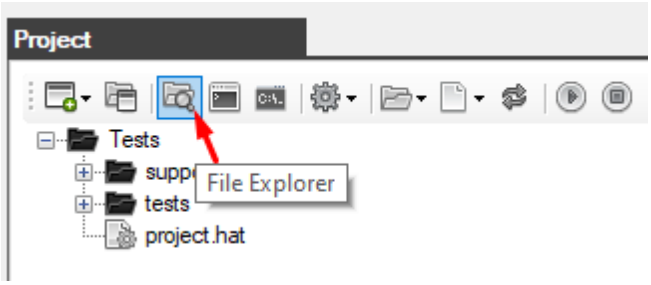
3. Create a folder that will contain the project with any name and anywhere on the disk.
In this case, the Project folder has been created at C:\Hat-1.0.0\
Select the project folder and click OK



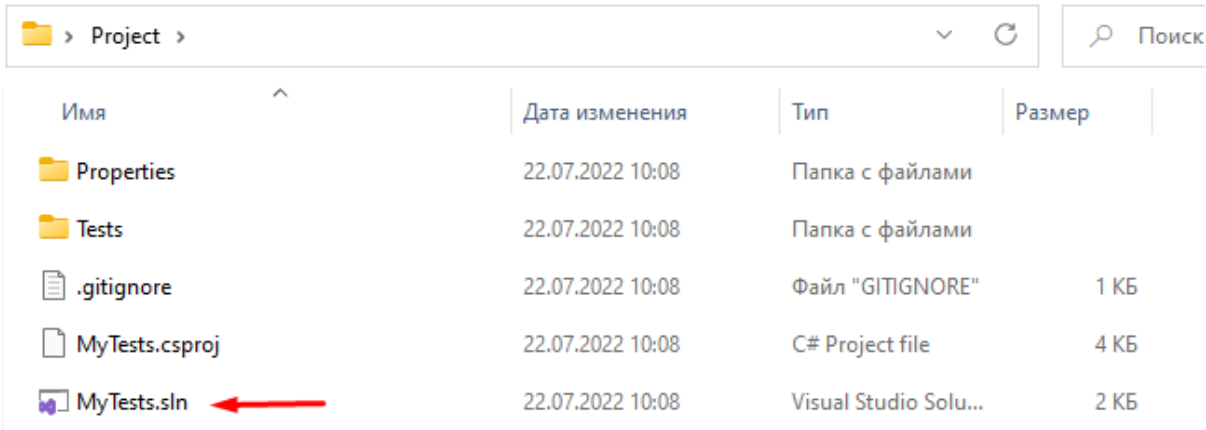
4. In the Project window, the contents of the project will be displayed on the Explorer tab.



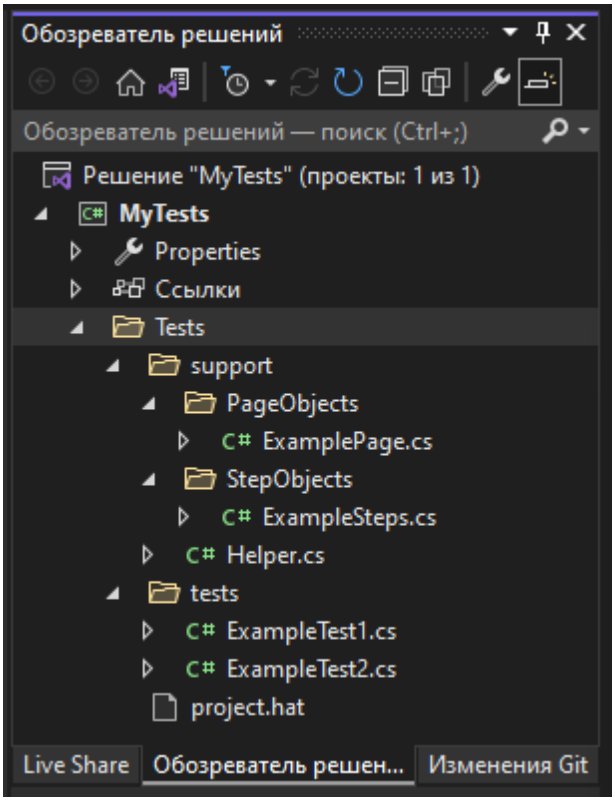
5. This project supports work in Visual Studio (it is assumed that you already have Visual Studio installed)
To open a project in Visual Studio, click on the Explorer button in the Project window



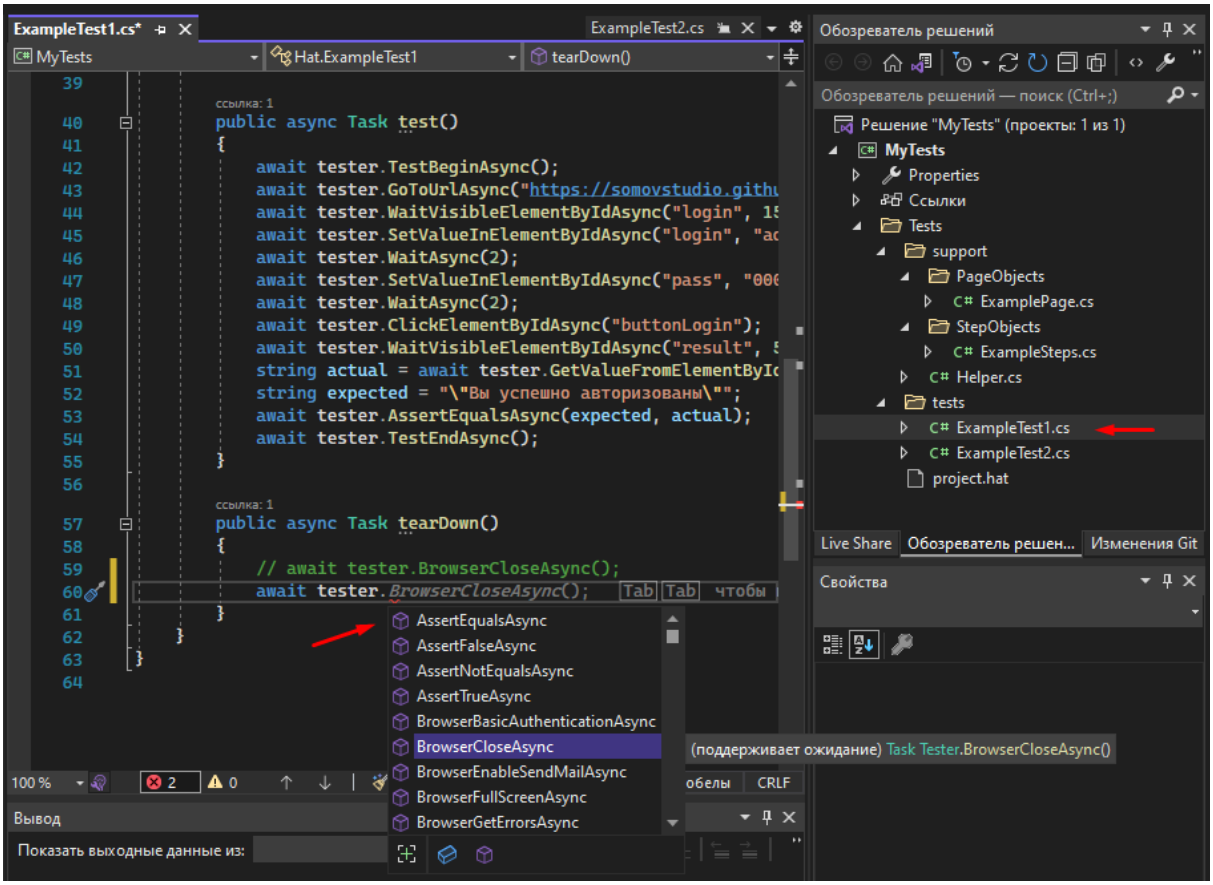
Then double-click on the file with the *.sln extension (in this case, MyTests.sln)



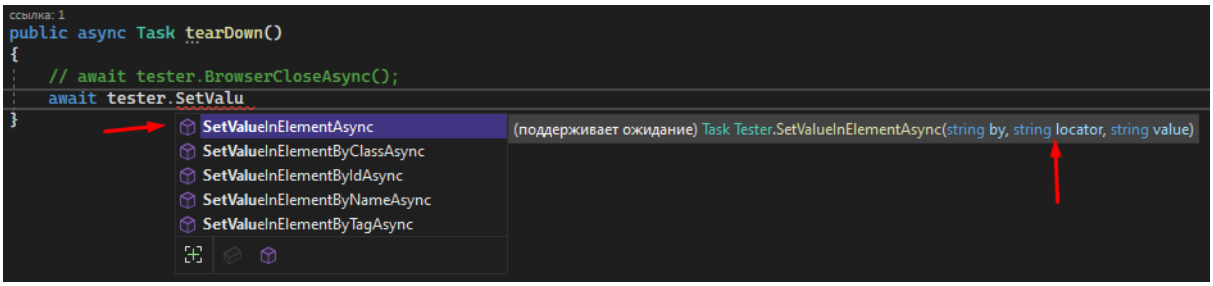
When Visual Studio is loaded, our project will immediately be opened and available for work in the Solution Explorer window



Double-click on the ExampleTest1.cs file to open it in the code editor.
In line 59, a line is commented out with a call to the BrowserCloseAsync() method that closes the browser.
This is necessary so that the browser does not automatically close after the completion of the autotest.



It is very convenient to use Visual Studio because the editor, when entering, shows a drop-down list with all available methods, as well as shows the syntax of the selected method and the variables required by it.



The project has been successfully created and is ready to work with Visual Studio.

The structure of the autotest

Simple project

Simple project

Autotests are described in a programming language C#.

The structure of the autotest:

```
using System;
using
System
.Collections.Generic;
using
System.ComponentModel;
using
System.Windows.Forms;
using System.Threading;
using
System.Threading.Tasks;
using System.IO;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using
System
.Text
.RegularExpressions;
using System.Net;
using System.Net.Http;
using
System.Net.Http.Headers;
using System.Reflection;
using Newtonsoft.Json;
using HatFramework;
```

At the beginning there is a list of plug-in libraries.

HatFramework - this is an embedded framework in the browser.

```
namespace Hat
{
    public class
ExampleTest1
    {

    }
}
```

Namespace and class.
The class name must be the same as the file name.

```
Tester tester;

public async void
Main(Form browserWindow)
{
        tester = new
Tester(browserWindow);

        await setUp();
        await test();
        await
tearDown();
}
```

A global variable is declared whose type is Tester

The main function with which the autotest starts is described.
This function initializes a global variable whose type is Tester.

Next comes the call of asynchronous functions.

```
public async Task
setUp()
{

}

public async Task test()
{
        await
```

Asynchronous functions are described that perform:

- setUp - function before the test
- task - test function
- tearDown - function after the test

(the names of the functions can be any, the meaning itself and their sequence are important)

```
tester.TestBeginAsync();

        await
tester.TestEndAsync();
}

public async Task
tearDown()
{

}
```

Note: the test should always start with the TestBeginAsync() method and end with the TestEndAsync() method

The demo autotest is complete:

Файл: ExampleTest1.cs

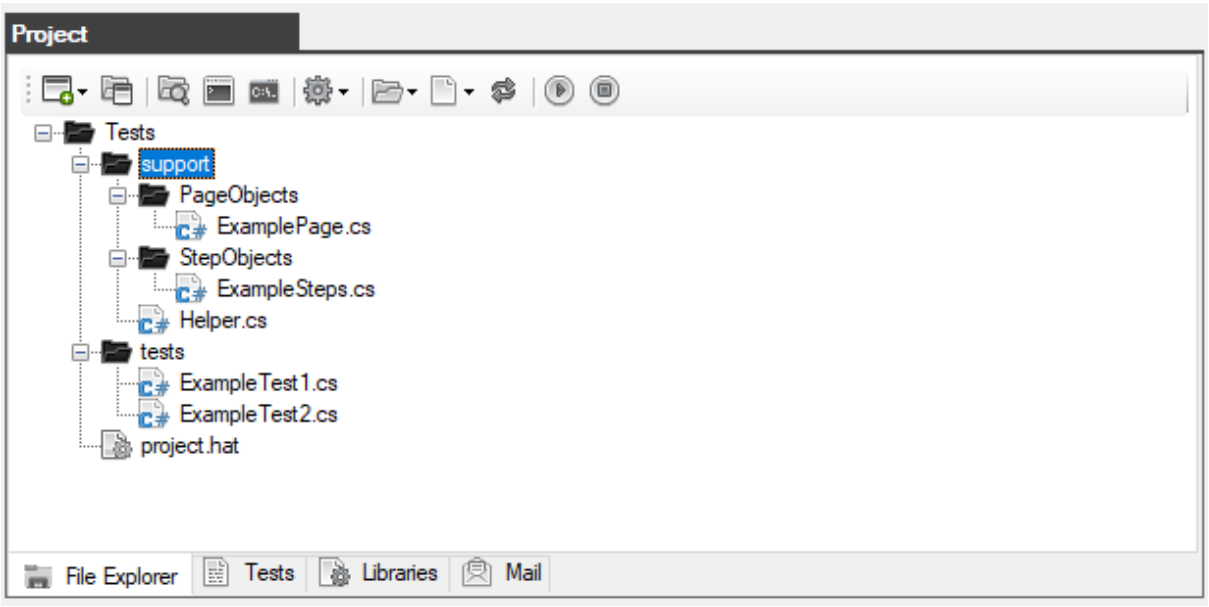
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Windows.Forms;
using System.Threading;
using System.Threading.Tasks;
using System.IO;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Reflection;
using Newtonsoft.Json;
using HatFramework;

namespace Hat
{
    public class ExampleTest1
    {
        Tester tester;

        public async void Main(Form browserWindow)
        {
            tester = new Tester(browserWindow);

            await setUp();
            await test();
            await tearDown();
        }

        public async Task setUp()
        {
            await tester.BrowserFullScreenAsync();
        }

        public async Task test()
        {
            await tester.TestBeginAsync();
```

```
            await
tester.GoToUrlAsync("https://somovstudio.github.io/test_eng.html", 5);
            await tester.WaitVisibleElementByIdAsync("login", 15);
            await tester.SetValueInElementByIdAsync("login", "admin");
            await tester.WaitAsync(2);
            await tester.SetValueInElementByIdAsync("pass", "0000");
            await tester.WaitAsync(2);
            await tester.ClickElementByIdAsync("buttonLogin");
            await tester.WaitVisibleElementByIdAsync("result", 5);
            string actual = await
tester.GetValueFromElementByIdAsync("textarea");
            string expected = "Authorization was successful";
            await tester.AssertEqualsAsync(expected, actual);
            await tester.TestEndAsync();
        }

        public async Task tearDown()
        {
            await tester.BrowserCloseAsync();
        }
    }
}
```

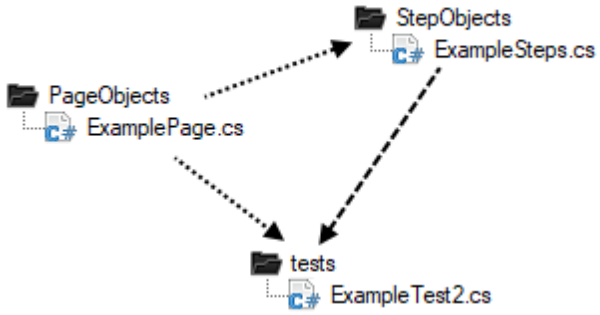

The PageObjects and StepObjects pattern

The PageObjects and StepObjects pattern

Based on the principles of object-oriented programming, you can easily organize the autotest structure using the PageObjects and StepObjects patterns.



The principle of class interaction:



- PageObjects (file ExamplePage.cs) - this is a static class that describes locators and methods that return entire objects to the page.
- StepObjects (file ExampleSteps.cs) - this is a class that inherits all the basic methods from the Tester base class (HatFramework library) and contains its own methods that perform actions with web elements.
- Autotest (file ExampleTest2.cs) - this is a class that directly performs testing using the PageObjects and StepObjects patterns.
- (in this case, the global variable tester has the type ExampleSteps, which inherits the base type Tester)

Demo autotest:

file ExamplePage.cs

```
using System;
using HatFramework;

namespace Hat
{
    public static class ExamplePage
    {
        public static string URL =
@"https://somovstudio.github.io/test_eng.html";
        public static string InputLogin = "login";
        public static string InputPass = "pass";
        public static string ButtonLogin = "buttonLogin";
        public static string Result = "result";
        public static string Textarea = "textarea";
    }
}
```

file ExampleSteps.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Windows.Forms;
using System.Threading;
using System.Threading.Tasks;
using System.IO;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
```

```
using System.Text.RegularExpressions;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Reflection;
using Newtonsoft.Json;
using HatFramework;

namespace Hat
{
    public class ExampleSteps : Tester
    {
        public ExampleSteps(Form browserWindow): base(browserWindow) {}

        public async Task FillForm()
        {
            await this.WaitVisibleElementByIdAsync(ExamplePage.InputLogin,
15);
            await this.SetValueInElementByIdAsync(ExamplePage.InputLogin,
"admin");
            await this.WaitAsync(2);
            await this.SetValueInElementByIdAsync(ExamplePage.InputPass,
"0000");
            await this.WaitAsync(2);
        }
    }
}
```

file ExampleTest2.cs

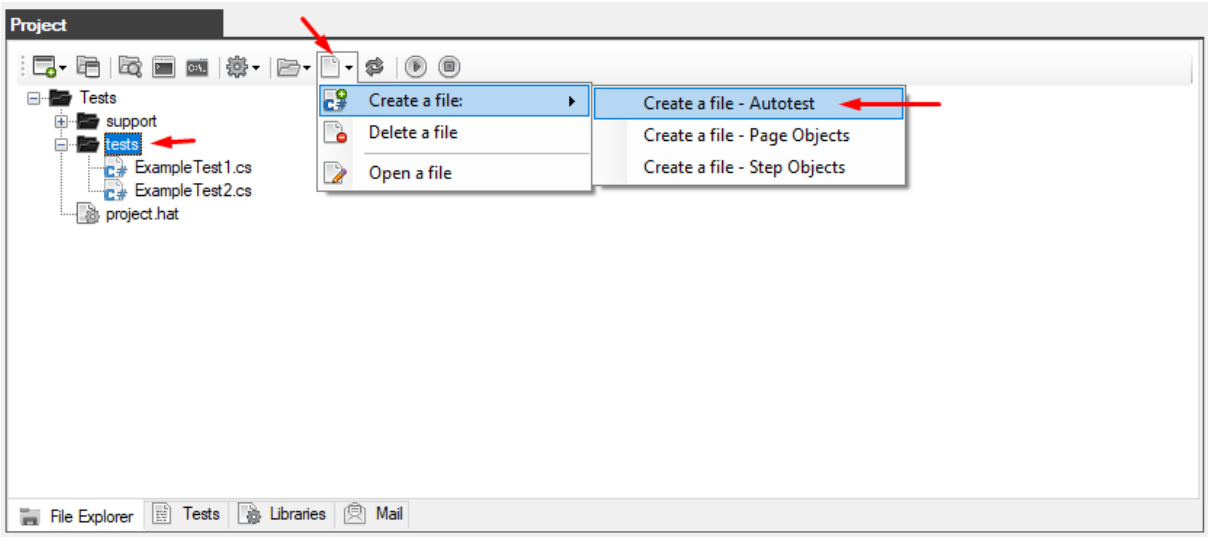
```
using System;
using System.IO;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Forms;
using HatFramework;

namespace Hat
{
    public class ExampleTest2
    {
        ExampleSteps tester;

        public async void Main(Form browserWindow)
        {
            tester = new ExampleSteps(browserWindow);

            await setUp();
            await test();
            await tearDown();
        }
```

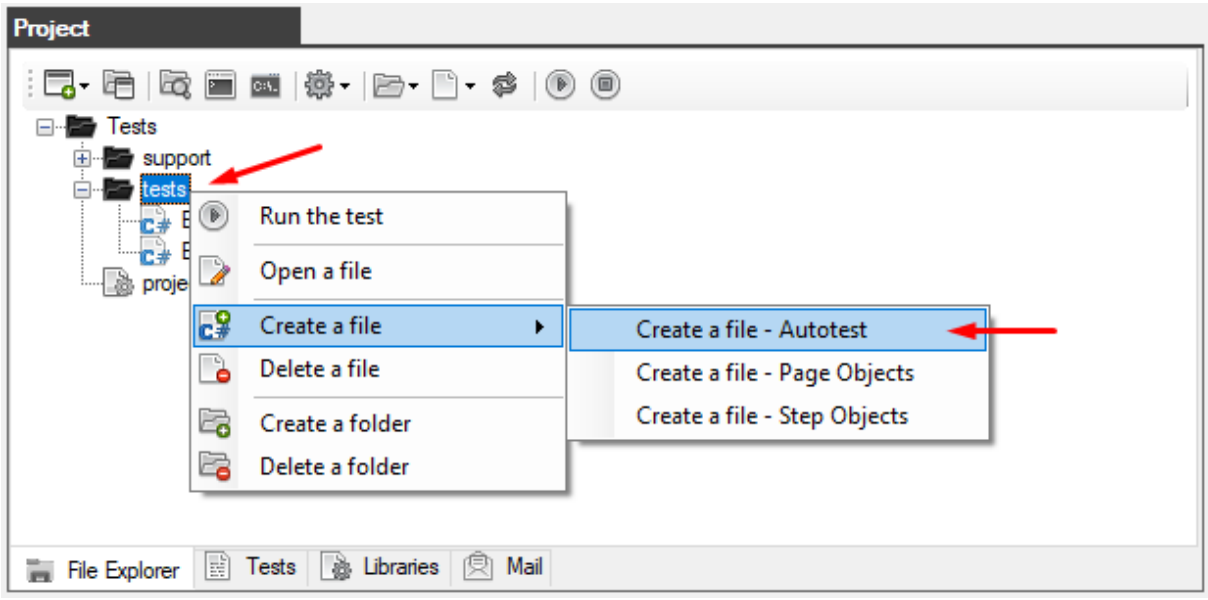
```
        public async Task setUp()
        {
            await tester.BrowserFullScreenAsync();
        }

        public async Task test()
        {
            await tester.TestBeginAsync();
            await tester.GoToUrlAsync(ExamplePage.URL, 5);
            await tester.FillForm();
            await tester.ClickElementByIdAsync(ExamplePage.ButtonLogin);
            await tester.WaitVisibleElementByIdAsync(ExamplePage.Result, 5);
            string actual = await
tester.GetValueFromElementByIdAsync(ExamplePage.Textarea);
            string expected = "Authorization was successful";
            await tester.AssertEqualsAsync(expected, actual);
            await tester.TestEndAsync();
        }

        public async Task tearDown()
        {
            await tester.BrowserCloseAsync();
        }
    }
}
```


Creating a new autotest and patterns

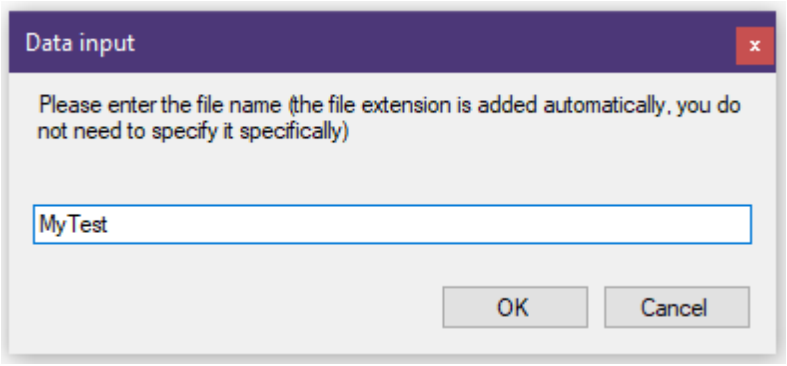
Creating a new autotest

To create a new autotest, you need to select the folder in which you want to create the file in the "Project" window (on the "Explorer" tab), then click on the "Files" button to open the menu and select "Create Autotest file"

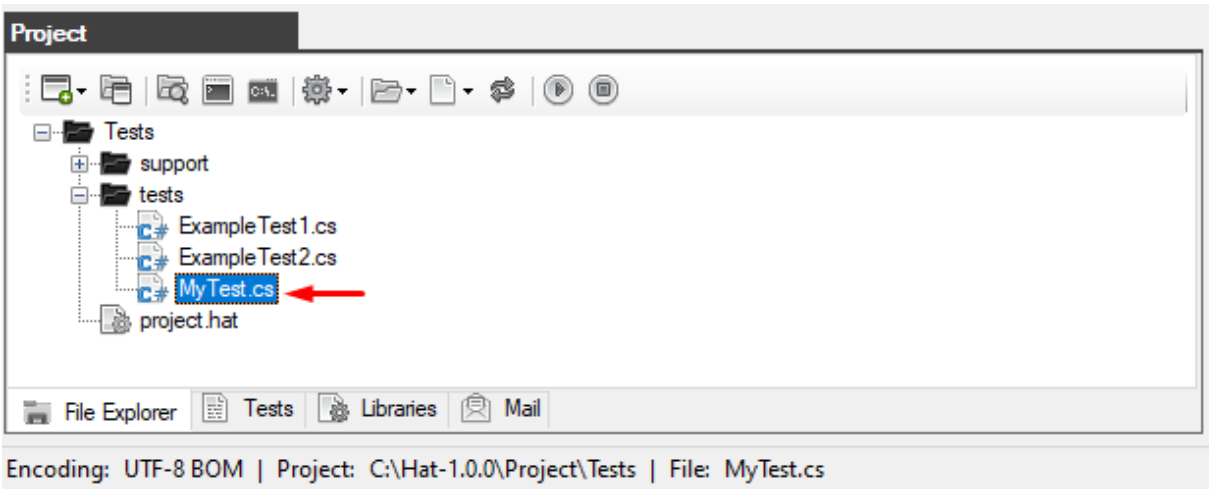




You will be prompted to enter a name for the new file (you do not need to specify the extension)



Click OK and a new autotest file will be created. It will immediately appear in the Project window (on the Explorer tab).



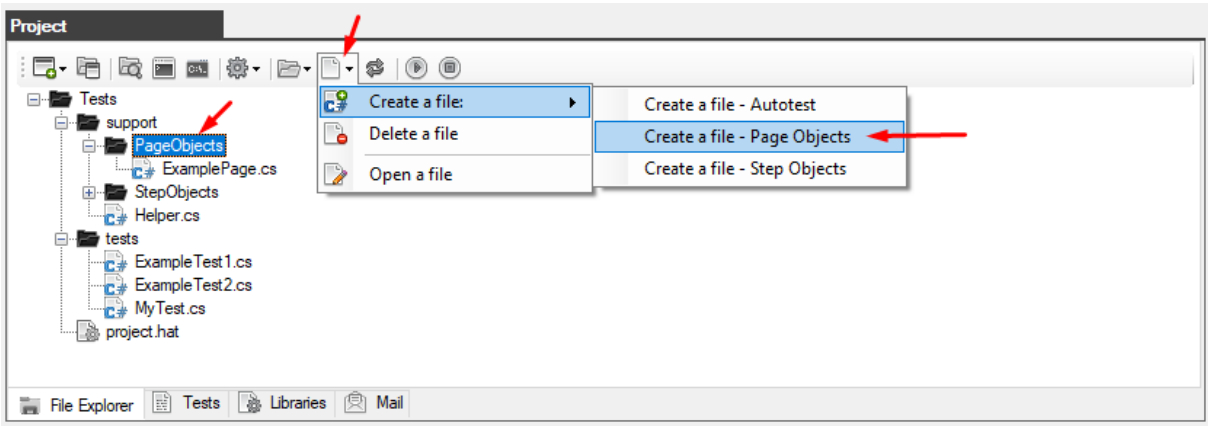
The file is not empty, it will immediately describe all the necessary structure.

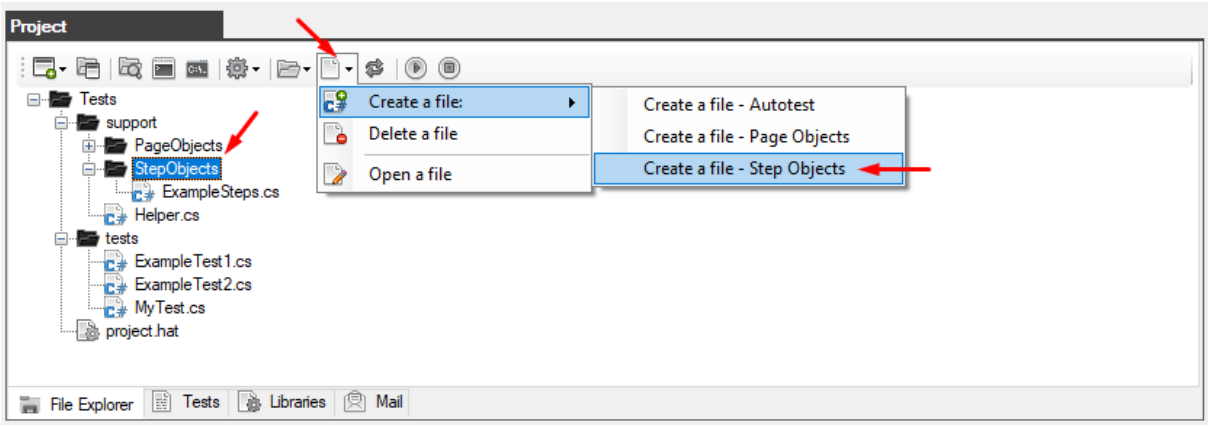
```
using System;
```

```
using System.Collections.Generic;
using System.ComponentModel;
using System.Windows.Forms;
using System.Threading;
using System.Threading.Tasks;
using System.IO
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Reflection;
using Newtonsoft.Json;
using HatFramework;

namespace Hat
{
    public class MyTest
    {
        Tester tester;

        public async void Main(Form browserWindow)
        {
            tester = new Tester(browserWindow);
            await setUp();
            await test();
            await tearDown();
        }

        public async Task setUp()
        {

        }

        public async Task test()
        {

        }

        public async Task tearDown()
        {

        }
    }
}
```

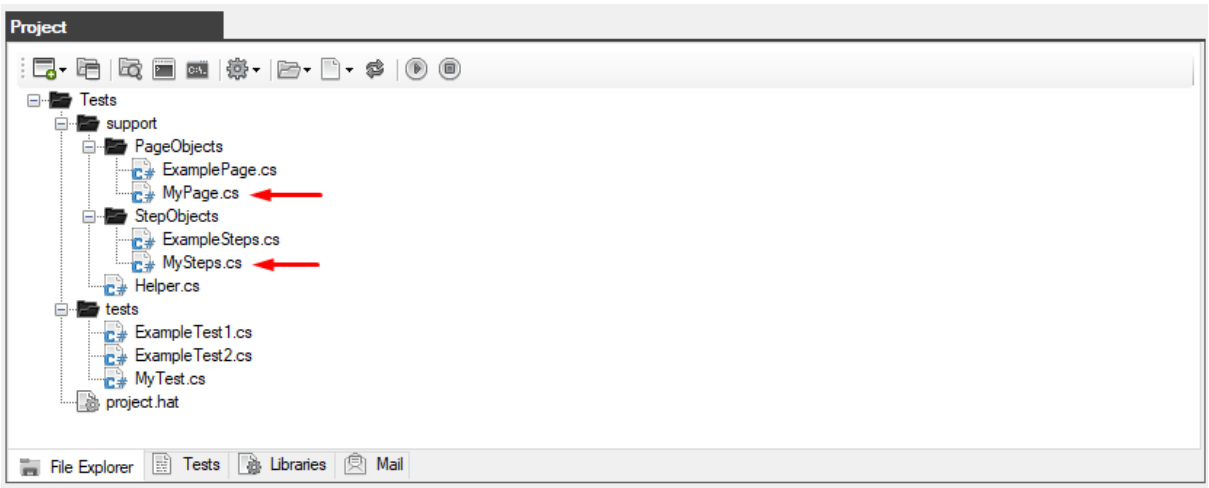
Creating patterns

Special templates are also provided for creating the PageObjects and StepObjects pattern files.





As a result, the following pattern files with a basic description will be created



file: MyPage.cs

```
using System;
using HatFramework;

namespace Hat
{
    public static class MyPage
    {
        public static string URL =
```

file: MySteps.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Windows.Forms;
using System.Threading;
using System.Threading.Tasks;
using System.IO;
using System.Data;
```

```
"https://test.com/";
        public static string
ButtonLogin = "buttonLogin";
    }
}
```

```
using System.Drawing;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Reflection;
using Newtonsoft.Json;
using HatFramework;

namespace Hat
{
    public class MySteps : Tester
    {
        public MySteps(Form
browserWindow): base(browserWindow) {}

        public async Task Test()
        {
            await
this.AssertTrueAsync(true);
        }
    }
}
```


HatFramework

Class: Tester

Constructor

Tester

Tester

Description: the main class of autotests

Syntax: Tester(Form browserForm)

Example:

```
Tester tester;
public async void Main(Form browserWindow)
{
        tester = new Tester(browserWindow);
}
```

Constants

BY_CSS

BY_CSS
Description: the constant indicates the type of locator being entered
Syntax: BY_CSS = "BY_CSS"

Example:
Tester.BY_CSS

BY_XPATH

BY_XPATH
Description: the constant indicates the type of locator being entered
Syntax: BY_XPATH = "BY_XPATH"

Example:
Tester.BY_XPATH

IMAGE_STATUS_PROCESS

IMAGE_STATUS_PROCESS
Description: the index of the image that indicates the status in the process
Syntax: IMAGE_STATUS_PROCESS = 0

Example:
Tester.IMAGE_STATUS_PROCESS

IMAGE_STATUS_PASSED

IMAGE_STATUS_PASSED
Description: the index of the image that indicates the status of success
Syntax: IMAGE_STATUS_PASSED = 1

Example:
Tester.IMAGE_STATUS_PASSED

IMAGE_STATUS_FAILED

IMAGE_STATUS_FAILED
Description: the index of the image that indicates the failed status
Syntax: IMAGE_STATUS_FAILED = 2

Example:
Tester.IMAGE_STATUS_FAILED

IMAGE_STATUS_MESSAGE

IMAGE_STATUS_MESSAGE
Description: the index of the image that indicates the status of the message
Syntax: IMAGE_STATUS_MESSAGE = 3

Example:
Tester.IMAGE_STATUS_MESSAGE

IMAGE_STATUS_WARNING

IMAGE_STATUS_WARNING
Description: the index of the image that indicates the warning status
Syntax: IMAGE_STATUS_WARNING = 4

Example:
Tester.IMAGE_STATUS_WARNING

IMAGE_STATUS_DEBUG

IMAGE_STATUS_DEBUG
Description: the index of the image that indicates the debug status
Syntax: IMAGE_STATUS_DEBUG = 5

Example:
Tester.IMAGE_STATUS_DEBUG

COMPLETED

COMPLETED
Description: the constant indicates the completed status
Syntax: COMPLETED = "COMPLETED"

Example:
Tester.COMPLETED

FAILED

FAILED
Description: the constant indicates the failed status
Syntax: FAILED = "FAILED"

Example:
Tester.FAILED

PASSED

PASSED
Description: the constant indicates the success status
Syntax: PASSED = "PASSED"

Example:
Tester.PASSED

PROCESS

PROCESS
Description: the constant indicates the status in progress
Syntax: PROCESS = "PROCESS"

Example:
Tester.PROCESS

STOPPED

STOPPED

Description: the constant indicates the stopped status
Syntax: STOPPED = "STOPPED"

Example:
Tester.STOPPED

WARNING

WARNING

Description: the constant indicates the warning status
Syntax: WARNING = "WARNING"

Example:
Tester.WARNING

DEBUG

DEBUG

Description: the constant indicates the debug status
Syntax: DEBUG = "DEBUG"

Example:
Tester.DEBUG

DEFAULT

DEFAULT

Description: the constant indicates the encoding type for the file
Syntax: DEFAULT = "DEFAULT"

Example:
```
string text = await tester.FileReadAsync(Tester.DEFAULT, "C:\\Hat\\file.txt");
await tester.FileWriteAsync(text, Tester.DEFAULT, "C:\\Hat\\file_copy.txt");
```


UTF8

UTF8

Description: the constant indicates the encoding type for the file
Syntax: UTF8 = "UTF-8"

Example:
string text = await tester.FileReadAsync(Tester.UTF8, "C:\\Hat\\file.txt");
await tester.FileWriteAsync(text, Tester.UTF8, "C:\\Hat\\file_copy.txt");

UTF8BOM

UTF8BOM

Description: the constant indicates the encoding type for the file
Syntax: UTF8BOM = "UTF-8 BOM"

Example:
string text = await tester.FileReadAsync(Tester.UTF8BOM, "C:\\Hat\\file.txt");
await tester.FileWriteAsync(text, Tester.UTF8BOM, "C:\\Hat\\file_copy.txt");

WINDOWS1251

WINDOWS1251

Description: the constant indicates the encoding type for the file
Syntax: WINDOWS1251 = "WINDOWS-1251"

Example:
string text = await tester.FileReadAsync(Tester.WINDOWS1251, "C:\\Hat\\file.txt");
await tester.FileWriteAsync(text, Tester.WINDOWS1251, "C:\\Hat\\file_copy.txt");

Variables

BrowserView

BrowserView

Description: the variable refers to an object representing the browser area
Syntax: WebView2 BrowserView

Example:
tester.BrowserView.Refresh();
tester.BrowserView.Reload();
tester.BrowserView.Source = new Uri(url);
tester.BrowserView.Update();

BrowserWindow

BrowserWindow
Description: the variable refers to the application window
Syntax: Form BrowserWindow

Example:

```
Tester tester;
public async void Main(Form browserWindow)
{
	tester = new Tester(browserWindow);
	tester.BrowserWindow.Text = "Browser Hat";
	tester.BrowserWindow.Width = 800;
	tester.BrowserWindow.Height = 600;
	tester.BrowserWindow.Close();
}
```


Locators

Locator

Locator
Description: an additional class for creating locators
Syntax:

```
public class Locator
{
	public string name { get; set; }
	public string description { get; set; }
	public string type { get; set; }
	public string value { get; set; }
}
```

Example:

```
Locator locator = new Locator();
locator.name = "inputLogin";
locator.description = "This is the login field";
locator.type = Tester.BY_XPATH;
locator.value = "//input[@id='login']";
```

AddLocator

AddLocator
Description: the method creates a locator and adds it to the shared locator storage.
Syntax: AddLocator(string name, string type, string value, string description)
Return value: no return value

Example:
tester.AddLocator("inputLogin", Tester.BY_XPATH, "//input[@id='login']", "This is the login field");

GetLocators

GetLocators
Description: the method returns the repositories of locators
Syntax: GetLocators()
Return value: Dictionary<string, Locator>

Example:
Dictionary<string, Locator> locators = tester.GetLocators();

GetLocator

GetLocator
Description: the method returns the locator from the locator storage as an object whose class is Locator
Syntax: GetLocator(string name)
Return value: object (Locator)

Example:
await tester.WaitVisibleElementAsync(tester.GetLocator("inputLogin"), 25);

GetLocatorValue

GetLocatorValue
Description: the method returns the locator value from the locator storage
Syntax: GetLocatorValue(string name)
Return value: string

Example:
string value = tester.GetLocatorValue("inputLogin");

GetCountLocators

GetCountLocators
Description: the method returns the number of locators in the locator storage
Syntax: GetCountLocators()
Return value: int

Example:
int count = tester.GetCountLocators();

ClearLocators

ClearLocators
Description: the method clears the locator storage
Syntax: ClearLocators()
Return value: no return value

Example:
tester.ClearLocators();

RemoveLocator

RemoveLocator
Description: the method removes the locator from the locator storage and returns a logical result true or false.
Syntax: RemoveLocator(string name)
Return value: bool (true or false)

Example:
bool result = tester.RemoveLocator("inputLogin");

Methods for working with the browser

BrowserBasicAuthenticationAsync

BrowserBasicAuthenticationAsync
Description: the browser's navigation method performs the forward action with the specified wait in seconds
Syntax: BrowserBasicAuthenticationAsync(string user, string pass)

Return value: no return value

Example:
Tester tester = new Tester(browserForm);
await tester.BrowserBasicAuthenticationAsync("user", "pass");
await tester.TestBeginAsync();
await tester.GoToUrlAsync("http://test.ru/basic_auth.html", 5);
await tester.TestEndAsync();

BrowserClearNetworkAsync

BrowserClearNetworkAsync
Description: the method cleans network events in the browser
Syntax: BrowserClearNetworkAsync()
Return value: no return value

Example:
string events = await tester.BrowserClearNetworkAsync();
tester.ConsoleMsg(events);

BrowserCloseAsync

BrowserCloseAsync
Description: the method closes the browser window
Syntax: BrowserCloseAsync()
Return value: no return value

Example:
await tester.BrowserCloseAsync();

BrowserEnableSendMailAsync

BrowserEnableSendMailAsync
Description: the method includes the option to send a report to the mail in case of failure of the autotest
Syntax: BrowserEnableSendMailAsync(bool byFailure = true, bool bySuccess = true)
Return value: no return value

Example:
Tester tester = new Tester(browserForm);
await tester.BrowserEnableSendMailAsync(true, false); // only in case of failure
await tester.BrowserEnableSendMailAsync(false, true); // only if successful

```
await tester.BrowserEnableSendMailAsync(); // in both cases
await tester.TestBeginAsync();
...
await tester.TestEndAsync();
```


BrowserFullScreenAsync

BrowserFullScreenAsync
Description: the method sets the full-screen size of the browser
Syntax: BrowserFullScreenAsync()
Return value: no return value

Example:

```
await tester.BrowserFullScreenAsync();
```


BrowserScreenshotAsync

BrowserScreenshotAsync
Description: the method takes a screenshot of the browser screen
Syntax: BrowserScreenshotAsync(string filename)
Return value: string

Example:

```
string screenshot = await tester.BrowserScreenshotAsync(null);
screenshot = await tester.BrowserScreenshotAsync("");

screenshot = await tester.BrowserScreenshotAsync("screenshot.jpg");
screenshot = await tester.BrowserScreenshotAsync("C:\\Users\\User\\Desktop\\MyTests\
\reports\\screenshots\\test_screenshot.jpg");

tester.ConsoleMsg(screenshot);
```


BrowserSizeAsync

BrowserSizeAsync
Description: the method sets the browser size
Syntax: BrowserSizeAsync(int width, int height)
Return value: no return value

Example:

```
await tester.BrowserSizeAsync(800, 600);
```

BrowserGetUserAgentAsync

BrowserGetUserAgentAsync
Description: the method returns the lowercase value of the browser's User-Agent
Syntax: BrowserGetUserAgentAsync()
Return value: string

Example:
```
string ua = await tester.BrowserGetUserAgentAsync();
```


BrowserSetUserAgentAsync

BrowserSetUserAgentAsync
Description: the method sets the User-Agent value for the browser
Syntax: BrowserSetUserAgent(string value)
Return value: no return value

Example:
```
await tester.TestBeginAsync();
await tester.BrowserSetUserAgent("my user-agent");
await tester.TestEndAsync();
```


BrowserGetErrorsAsync

BrowserGetErrorsAsync
Description: the method returns a list of browser errors and warnings
Syntax: BrowserGetErrorsAsync()
Return value: List

Example:
```
List<string> errors = await tester.BrowserGetErrorsAsync();
foreach (string error in errors)
{
        tester.ConsoleMsg(error);
}
```


BrowserGetNetworkAsync

BrowserGetNetworkAsync
Description: the method returns all current messages from the network in json format
Syntax: BrowserGetNetworkAsync()
Return value: string

Example:
string events = await tester.BrowserGetNetworkAsync();
tester.ConsoleMsg(events);

BrowserGoBackAsync

BrowserGoBackAsync

Description: the browser's navigation method performs the action backwards with the specified wait in seconds
The abortLoadAfterTime flag automatically completes page loading after the timer is completed
Syntax: BrowserGoBackAsync(int sec, bool abortLoadAfterTime = false)
Return value: no return value

Example:
```
Tester tester = new Tester(browserForm);
await tester.TestBeginAsync();
await tester.GoToUrlAsync("https://www.yahoo.com/", 5);
string currentUrl = await tester.GetUrlAsync();
await tester.AssertEqualsAsync("https://www.yahoo.com/", currentUrl);

await tester.GoToUrlAsync("https://yandex.ru/", 5);
currentUrl = await tester.GetUrlAsync();
await tester.AssertEqualsAsync("https://yandex.ru/", currentUrl);

await tester.BrowserGoBackAsync(10);
currentUrl = await tester.GetUrlAsync();
await tester.AssertEqualsAsync("https://www.yahoo.com/", currentUrl);

await tester.BrowserGoForwardAsync(10);
currentUrl = await tester.GetUrlAsync();
await tester.AssertEqualsAsync("https://yandex.ru/", currentUrl);
await tester.TestEndAsync();
```


BrowserGoForwardAsync

BrowserGoForwardAsync

Description: the browser's navigation method performs the forward action with the specified wait in seconds
The abortLoadAfterTime flag automatically completes page loading after the timer is completed
Syntax: BrowserGoForwardAsync(int sec, bool abortLoadAfterTime = false)
Return value: no return value

Example:
```
Tester tester = new Tester(browserForm);
await tester.TestBeginAsync();
await tester.GoToUrlAsync("https://www.yahoo.com/", 5);
string currentUrl = await tester.GetUrlAsync();
await tester.AssertEqualsAsync("https://www.yahoo.com/", currentUrl);

await tester.GoToUrlAsync("https://yandex.ru/", 5);
currentUrl = await tester.GetUrlAsync();
await tester.AssertEqualsAsync("https://yandex.ru/", currentUrl);

await tester.BrowserGoBackAsync(10);
currentUrl = await tester.GetUrlAsync();
await tester.AssertEqualsAsync("https://www.yahoo.com/", currentUrl);

await tester.BrowserGoForwardAsync(10);
currentUrl = await tester.GetUrlAsync();
await tester.AssertEqualsAsync("https://yandex.ru/", currentUrl);
await tester.TestEndAsync();
```


BrowserPageReloadAsync

BrowserPageReloadAsync
Description: the method reloads the open page in the browser
The abortLoadAfterTime flag automatically completes page loading after the timer is completed
Syntax: BrowserPageReloadAsync(int sec, bool abortLoadAfterTime = false)
Return value: no return value

Example:
```
await tester.BrowserPageReloadAsync(25);
```


Methods for displaying messages

ConsoleMsg

ConsoleMsg
Description: the method displays a message in the browser console
Syntax:

ConsoleMsg(string message)
ConsoleMsg(string messageRus, string messageEng)
Return value: no return value

Example:
tester.ConsoleMsg("Текст на Русском языке", "The text is in English");

ConsoleMsgError

ConsoleMsgError

Description: the method outputs an error message to the system console and the browser console
Syntax: ConsoleMsgError(string message)
Return value: no return value

Example:

```
try
{

}
catch (Exception ex)
{
        tester.ConsoleMsgError(ex.ToString());
}
```


ClearMessage

ClearMessage

Description: the method clears the messages in the output table of the test execution process
Syntax: ClearMessage()
Return value: no return value

Example:
tester.ClearMessage();

Description

Description

Description: the method sets the description of the test, which is then displayed in the report and letter
Syntax: Description(string text)
Return value: no return value

Example:
```
public async Task setUp()
{
        tester.Description("The test checks the authorization on the site");
}
```


DisableDebugInReport

DisableDebugInReport

Description: this method disables the output of SendMessageDebug debug messages to the report
Syntax: DisableDebugInReport()
Return value: no return value

Example:
```
tester.DisableDebugInReport();
```


SendMessage

SendMessage

Description: the method displays messages in the test execution process table
When executed, action is output to the system console if status is FAILED, WARNING, "", null, otherwise only status and comment are output (without action).

Syntax: SendMessage(string action, string status, string comment)

Example:
```
tester.SendMessage("the text of the actions", Tester.PROCESS, "the text of the comment");

/* Examples of console output */
// action comment
tester.SendMessage("actions", null, "comment");
tester.SendMessage("actions", "", "comment");

// Step[progress]: comment
tester.SendMessage("actions", Tester.PROCESS, "comment");
// Step[passed]: comment
tester.SendMessage("actions", Tester.PASSED, "comment");
// Step[completed]: comment
tester.SendMessage("actions", Tester.COMPLETED, "comment");
// Step[stopped]: comment
tester.SendMessage("actions", Tester.STOPPED, "comment");
```

```
// Step[failed]: действие комментарий
tester.SendMessage("actions", Tester.FAILED, "comment");
// Step[warning]: действие комментарий
tester.SendMessage("actions", Tester.WARNING, "comment");
```


SendMessageDebug

SendMessageDebug

Description: the method outputs a debugging message that can be disabled for output to a report and an email
When executed, action is output to the system console if status is FAILED, WARNING, "", null, otherwise only status and comment are output (without action).

Syntax: SendMessageDebug(string actionRus, string actionEng, string status, string commentRus, string commentEng, int image)

Example:

```
tester.SendMessageDebug("действия", "action", Tester.PROCESS, "комментарий", "comment",
 Tester.IMAGE_STATUS_PROCESS);
```


Methods for preparing and completing testing

TestBeginAsync

TestBeginAsync

Description: the method of preparing for the test (each test must necessarily begin with this method)
Syntax: TestBeginAsync()
Return value: no return value

Example:

```
await tester.TestBeginAsync();
```


TestEndAsync

TestEndAsync

Description: the test completion method (each test must necessarily end with this method), it is this method that saves the report and sends it by mail if such an option is configured and

enabled.
Syntax: TestEndAsync()
Return value: no return value

Example:
await tester.TestEndAsync();

TestStopAsync

TestStopAsync
Description: the method forcibly stops the testing process
Syntax: TestStopAsync()
Return value: no return value

Example:
await tester.TestStopAsync();

GetTestResult

GetTestResult
Description: the method returns the status of the test being performed
Syntax: GetTestResult()
Return value: string (the value of which is checked by constants: Tester.PASSED, Tester.FAILED, Tester.PROCESS)

Example:
```
Tester tester = new Tester(browserForm);
await tester.TestBeginAsync();
...
if(tester.GetTestResult() == Tester.PROCESS) { }
...
await tester.TestEndAsync();
if(tester.GetTestResult() == Tester.FAILED) { }
if(tester.GetTestResult() == Tester.PASSED) { }
```


DefineTestStop

DefineTestStop
Description: the method checks the status of the process (stopped or not)
Syntax: DefineTestStop()
Return value: bool (true or false)

Example:

```
if (tester.DefineTestStop() == true) return; // so the test is stopped
```


Methods for performing actions

Attributes

GetAttributeFromElementAsync

GetAttributeFromElementAsync

Description: the method returns a lowercase value from the specified attribute in the selected element

Syntax:

GetAttributeFromElementAsync(string by, string locator, string attribute)

GetAttributeFromElementAsync(Locator locator, string attribute)

Return value: string

Example:

```
string value = await tester.GetAttributeFromElementAsync(Tester.BY_CSS, "input", "name");
string value = await tester.GetAttributeFromElementAsync(Tester.BY_XPATH, "//input", "name");

string value = await tester.GetAttributeFromElementAsync(tester.GetLocator("locatorName"), "name");
```

GetAttributeFromElementByClassAsync

GetAttributeFromElementByClassAsync
Description: the method returns a lowercase value from the specified attribute in the selected element
Syntax: GetAttributeFromElementByClassAsync(string _class, int index, string attribute)
Return value: string

Example:
```
string value = await tester.GetAttributeFromElementByClassAsync("my-element", 0, "href");
```

GetAttributeFromElementByIdAsync

GetAttributeFromElementByIdAsync
Description: the method returns a lowercase value from the specified attribute in the selected element
Syntax: GetAttributeFromElementByIdAsync(string id, string attribute)
Return value: string

Example:
string value = await tester.GetAttributeFromElementByIdAsync("MyElement", "href");

GetAttributeFromElementByNameAsync

GetAttributeFromElementByNameAsync
Description: the method returns a lowercase value from the specified attribute in the selected element
Syntax: GetAttributeFromElementByNameAsync(string name, int index, string attribute)
Return value: string

Example:
string value = await tester.GetAttributeFromElementByNameAsync("MyElement", 0, "href");

GetAttributeFromElementByTagAsync

GetAttributeFromElementByTagAsync
Description: the method returns a lowercase value from the specified attribute in the selected element
Syntax: GetAttributeFromElementByTagAsync(string tag, int index, string attribute)
Return value: string

Example:
string value = await tester.GetAttributeFromElementByTagAsync("a", 0, "href");

GetAttributeFromElementsAsync

GetAttributeFromElementsAsync
Description: the method returns a list of values of the specified attribute from a set of elements
Syntax:
GetAttributeFromElementsAsync(string by, string locator, string attribute)
GetAttributeFromElementsAsync(Locator locator, string attribute)
Return value: List

Example:

```
List<string> values = await tester.GetAttributeFromElementsAsync(Tester.BY_CSS, "input",
"name");
if(values != null)
{
	foreach (string attr in values)
		tester.ConsoleMsg(attr);
}


List<string> values = await tester.GetAttributeFromElementsAsync(Tester.BY_XPATH, "//input",
"name");
if(values != null)
{
	foreach (string attr in values)
		tester.ConsoleMsg(attr);
}

List<string> values = await
tester.GetAttributeFromElementsAsync(tester.GetLocator("locatorName"), "name");
if(values != null)
{
	foreach (string attr in values)
		tester.ConsoleMsg(attr);
}
```

GetAttributeFromElementsByClassAsync

GetAttributeFromElementsByClassAsync
Description: the method returns a list of values of the specified attribute from a set of elements
Syntax: GetAttributeFromElementsByClassAsync(string _class, string attribute)
Return value: List

Example:

```
List<string> values = await tester.GetAttributeFromElementsByClassAsync("text-field", "name");
foreach (string value in values)
{
	tester.ConsoleMsg(value);
}
```

GetAttributeFromElementsByNameAsync

GetAttributeFromElementsByNameAsync
Description: the method returns a list of values of the specified attribute from a set of elements
Syntax: GetAttributeFromElementsByNameAsync(string name, string attribute)
Return value: List

Example:

```
List<string> values = await tester.GetAttributeFromElementsByNameAsync("link", "href");
foreach (string value in values)
{
	tester.ConsoleMsg(value);
}
```

GetAttributeFromElementsByTagAsync

GetAttributeFromElementsByTagAsync
Description: the method returns a list of values of the specified attribute from a set of elements
Syntax: GetAttributeFromElementsByTagAsync(string tag, string attribute)
Return value: List

Example:

```
List<string> values = await tester.GetAttributeFromElementsByTagAsync("a", "href");
foreach (string value in values)
{
	tester.ConsoleMsg(value);
}
```

SetAttributeInElementAsync

SetAttributeInElementAsync
Description: the method inserts an attribute with a value into the specified element
Syntax:
SetAttributeInElementAsync(string by, string locator, string attribute, string value)
SetAttributeInElementAsync(Locator locator, string attribute, string value)
Return value: no return value

Example:

```
await tester.SetAttributeInElementAsync(Tester.BY_CSS, "#auth > h2", "name", "test");
await tester.SetAttributeInElementAsync(Tester.BY_XPATH, "//div[@id='auth']//h2", "name",
"test");

await tester.SetAttributeInElementAsync(tester.GetLocator("locatorName"), "name", "test");
```

SetAttributeInElementByClassAsync

SetAttributeInElementByClassAsync
Description: the method inserts an attribute with a value into the specified element
Syntax: SetAttributeInElementByClassAsync(string _class, int index, string attribute, string value)
Return value: no return value

Example:
await tester.SetAttributeInElementByClassAsync("my-element", 0, "value", "значение");

SetAttributeInElementByIdAsync

SetAttributeInElementByIdAsync
Description: the method inserts an attribute with a value into the specified element
Syntax: SetAttributeInElementByIdAsync(string id, string attribute, string value)
Return value: no return value

Example:
await tester.SetAttributeInElementByIdAsync("MyElement", "value", "value");

SetAttributeInElementByNameAsync

SetAttributeInElementByNameAsync
Description: the method inserts an attribute with a value into the specified element
Syntax: SetAttributeInElementByNameAsync(string name, int index, string attribute, string value)
Return value: no return value

Example:
```
await tester.SetAttributeInElementByNameAsync("MyElement", 0, "value",  "value");
```

SetAttributeInElementByTagAsync

SetAttributeInElementByTagAsync
Description: the method inserts an attribute with a value into the specified element
Syntax: SetAttributeInElementByTagAsync(string tag, int index, string attribute, string value)
Return value: no return value

Example:
await tester.SetAttributeInElementByTagAsync("input", 0, "value", "value");

SetAttributeInElementsAsync

SetAttributeInElementsAsync

Description: the method inserts an attribute with the specified value into a set of elements and returns a list as a result

Syntax:

SetAttributeInElementsAsync(string by, string locator, string attribute, string value)

SetAttributeInElementsAsync(Locator locator, string attribute, string value)

Return value: List

Example:

```
List<string> values = await tester.SetAttributeInElementsAsync(Tester.BY_CSS, "input", "class", "test-class");
foreach (string value in values)
{
	tester.ConsoleMsg(value);
}

List<string> values = await tester.SetAttributeInElementsAsync(Tester.BY_XPATH, "//input", "class",  "test-class");
foreach (string value in values)
{
	tester.ConsoleMsg(value);
}

List<string> values = await tester.SetAttributeInElementsAsync(tester.GetLocator("locatorName"), "class", "test-class");
foreach (string value in values)
{
	tester.ConsoleMsg(value);
}
```

SetAttributeInElementsByClassAsync

SetAttributeInElementsByClassAsync
Description: the method inserts an attribute with the specified value into a set of elements and returns a list as a result
Syntax: SetAttributeInElementsByClassAsync(string _class, string attribute, string value)
Return value: List

Example:

```
List<string> values = await tester.SetAttributeInElementsByClassAsync("text-field", "value", "test");
foreach (string value in values)
{
	tester.ConsoleMsg(value);
}
```

SetAttributeInElementsByNameAsync

SetAttributeInElementsByNameAsync
Description: the method inserts an attribute with the specified value into a set of elements and returns a list as a result
Syntax: SetAttributeInElementsByNameAsync(string name, string attribute, string value)
Return value: List

Example:

```
List<string> values = await tester.SetAttributeInElementsByNameAsync("link", "href",
"www.test.ru");
foreach (string value in values)
{
        tester.ConsoleMsg(value);
}
```

SetAttributeInElementsByTagAsync

SetAttributeInElementsByTagAsync
Description: the method inserts an attribute with the specified value into a set of elements and returns a list as a result
Syntax: SetAttributeInElementsByTagAsync(string tag, string attribute, string value)
Return value: List

Example:

```
List<string> values = await tester.SetAttributeInElementsByTagAsync("a", "href",
"www.test.ru");
foreach (string value in values)
{
        tester.ConsoleMsg(value);
}
```


Values

GetValueFromElementAsync

GetValueFromElementAsync
Description: the method returns the value from the specified element
Syntax:
GetValueFromElementAsync(string by, string locator)
GetValueFromElementAsync(Locator locator)
Return value: string

Example:

```
string value = await tester.GetValueFromElementAsync(Tester.BY_CSS, "input[id='login']");
string value = await tester.GetValueFromElementAsync(Tester.BY_XPATH,
"//input[@id='login']");

string value = await tester.GetValueFromElementAsync(tester.GetLocator("locatorName");
```

GetValueFromElementByClassAsync

GetValueFromElementByClassAsync
Description: the method returns the value from the specified element
Syntax: GetValueFromElementByClassAsync(string _class, int index)
Return value: string

Example:
string value = await tester.GetValueFromElementByClassAsync("my-element", 0);

GetValueFromElementByIdAsync

GetValueFromElementByIdAsync
Description: the method returns the value from the specified element
Syntax: GetValueFromElementByIdAsync(string id)
Return value: string

Example:
string value = await tester.GetValueFromElementByIdAsync("MyElement");

GetValueFromElementByNameAsync

GetValueFromElementByNameAsync
Description: the method returns the value from the specified element
Syntax: GetValueFromElementByNameAsync(string name, int index)
Return value: строка (string)

Example:
```
string value = await tester.GetValueFromElementByNameAsync("MyElement", 0);
```

GetValueFromElementByTagAsync

GetValueFromElementByTagAsync
Description: the method returns the value from the specified element
Syntax: GetValueFromElementByTagAsync(string tag, int index)
Return value: string

Example:
```
string value = await tester.GetValueFromElementByTagAsync("input", 0);
```

SetValueInElementAsync

SetValueInElementAsync
Description: the method inserts value into the specified element
Syntax:
SetValueInElementAsync(string by, string locator, string value)
SetValueInElementAsync(Locator locator, string value)
Return value: no return value

Example:

```
await tester.SetValueInElementAsync(Tester.BY_CSS, "input[id='login']", "admin");
await tester.SetValueInElementAsync(Tester.BY_CSS, "input[id='pass']", "0000");

await tester.SetValueInElementAsync(Tester.BY_XPATH, "//input[@id='login']", "admin");
await tester.SetValueInElementAsync(Tester.BY_XPATH, "//input[@id='pass']", "0000");

await tester.SetValueInElementAsync(tester.GetLocator("locatorName"), "admin");
await tester.SetValueInElementAsync(tester.GetLocator("locatorName"), "0000");
```

SetValueInElementByClassAsync

SetValueInElementByClassAsync
Description: the method inserts value into the specified element
Syntax: SetValueInElementByClassAsync(string _class, int index, string value)
Return value: no return value

Example:
```
await tester.SetValueInElementByClassAsync("my-element", 0, "value");
```

SetValueInElementByIdAsync

SetValueInElementByIdAsync
Description: the method inserts value into the specified element
Syntax: SetValueInElementByIdAsync(string id, string value)
Return value: no return value

Example:
await tester.SetValueInElementByIdAsync("MyElement", "value");

SetValueInElementByNameAsync

SetValueInElementByNameAsync
Description: the method inserts value into the specified element
Syntax: SetValueInElementByNameAsync(string name, int index, string value)
Return value: no return value

Example:
await tester.SetValueInElementByNameAsync("MyElement", 0, "value");

SetValueInElementByTagAsync

SetValueInElementByTagAsync
Description: the method inserts value into the specified element
Syntax: SetValueInElementByTagAsync(string tag, int index, string value)
Return value: no return value

Example:
await tester.SetValueInElementByTagAsync("input", 0, "value");

Clicking

ClickElementAsync

ClickElementAsync
Description: the method performs a click on the element
Syntax:
ClickElementAsync(string by, string locator)
ClickElementAsync(Locator locator)
Return value: no return value

Example:

```
await tester.ClickElementAsync(Tester.BY_CSS, "#auth #buttonLogin");
await tester.ClickElementAsync(Tester.BY_XPATH,
"//div[@id='auth']//input[@id='buttonLogin']");

await tester.ClickElementAsync(tester.GetLocator("locatorName"));
```

ClickElementByClassAsync

ClickElementByClassAsync
Description: the method performs a click on the element
Syntax: ClickElementByClassAsync(string _class, int index);
Return value: no return value

Example:
await tester.ClickElementByClassAsync("my-element", 0)

ClickElementByIdAsync

ClickElementByIdAsync
Description: the method performs a click on the element
Syntax: ClickElementByIdAsync(string id)
Return value: no return value

Example:
await tester.ClickElementByIdAsync("MyElement");

ClickElementByNameAsync

ClickElementByNameAsync
Description: the method performs a click on the element
Syntax: ClickElementByNameAsync(string name, int index)
Return value: no return value

Example:
```
await tester.ClickElementByNameAsync("MyElement", 0);
```

ClickElementByTagAsync

ClickElementByTagAsync
Description: the method performs a click on the element
Syntax: ClickElementByTagAsync(string tag, int index)
Return value: no return value

Example:
await tester.ClickElementByTagAsync("a", 0);

IsClickableElementAsync

IsClickableElementAsync

Description: the method determines the clickability of the element and returns true or false
Syntax:
IsClickableElementAsync(string by, string locator)
IsClickableElementAsync(Locator locator)
Return value: bool (true or false)

Пример:

```
Tester tester = new Tester(browserForm);
await tester.TestBeginAsync();
await tester.GoToUrlAsync("https://somovstudio.github.io/test_eng.html", 5);
bool clickable = await tester.IsClickableElementAsync(Tester.BY_XPATH,
"//*[@id='buttonLogin']");
await tester.AssertTrueAsync(clickable);
await tester.TestEndAsync();

bool clickable = await tester.IsClickableElementAsync(tester.GetLocator("locatorName"));
await tester.AssertTrueAsync(clickable);
```

ScrollToElementAsync

ScrollToElementAsync
Description: the method scrolls to the specified element (the behaviorSmooth parameter determines the smoothness of scrolling)
Syntax:
ScrollToElementAsync(string by, string locator, bool behaviorSmooth = false)
ScrollToElementAsync(Locator locator, bool behaviorSmooth = false)
Return value: no return value

Example:

```
await tester.ScrollToElementAsync(Tester.BY_CSS, "body > footer", true);
await tester.ScrollToElementAsync(Tester.BY_XPATH, "/html/body/footer", true);

await tester.ScrollToElementAsync(tester.GetLocator("locatorName"), true);
```


Objects

GetElementAsync

GetElementAsync

Description: the method returns an element in the form of an object whose class HTMLElement
Syntax:
GetElementAsync(string by, string locator)
GetElementAsync(Locator locator)
Return value: object HTMLElement

Example:

```
HTMLElement element = await tester.GetElementAsync(tester.GetLocator("locatorName"));
await element.ClickAsync();

HTMLElement element = await tester.GetElementAsync(Tester.BY_CSS, "#auth #buttonLogin");
await element.ClickAsync();

HTMLElement element = await tester.GetElementAsync(Tester.BY_XPATH,
"//div[@id='auth']//input[@id='buttonLogin']");
await element.ClickAsync();

HTMLElement element = await tester.GetElementAsync(Tester.BY_XPATH, "//*[@id='MyFile']");
tester.ConsoleMsg("ID: " + element.Id);

HTMLElement element = await tester.GetElementAsync(Tester.BY_XPATH, "//h1");
string text = await element.GetTextAsync();
tester.ConsoleMsg(text);
await element.SetTextAsync("text");

HTMLElement element = await tester.GetElementAsync(Tester.BY_XPATH,
"//*[@id='MyInput']");
await element.SetValueAsync("text");
string value = await element.GetValueAsync();
tester.ConsoleMsg(value);

HTMLElement element = await tester.GetElementAsync(Tester.BY_XPATH, "//h1");
await element.SetAttributeAsync("class", "my-class");
string attrClass = await element.GetAttributeAsync("class");
tester.ConsoleMsg(attrClass);

HTMLElement element = await tester.GetElementAsync(Tester.BY_XPATH, "//h1");
string html = await element.GetHtmlAsync();
tester.ConsoleMsg(html);

HTMLElement element = await tester.GetElementAsync(Tester.BY_XPATH, "//h1");
await element.SetHtmlAsync("<div>text</div>");

HTMLElement element = await tester.GetElementAsync(Tester.BY_XPATH, "/html/body/footer");
await element.ScrollToAsync();
```

```
HTMLElement element = await tester.GetElementAsync(Tester.BY_XPATH, "//h1");
await element.WaitVisibleAsync(2);
await element.WaitNotVisibleAsync(2);
```

GetFrameAsync

GetFrameAsync
Description: the method returns an element in the form of an object whose class
FRAMEElement
Syntax: GetFrameAsync(int index)
Return value: object FRAMEElement

Example:

```
Tester tester = new Tester(browserForm);
await tester.TestBeginAsync();
await tester.GoToUrlAsync("https://somovstudio.github.io/test2.html", 5);

FRAMEElement frame = await tester.GetFrameAsync(0);
tester.ConsoleMsg("Index: " + frame.Index);
tester.ConsoleMsg("Name: " + frame.Name);

string name = await frame.GetAttributeFromElementAsync(Tester.BY_XPATH,
"//input[@id='login']", "name");

List<string> values = await frame.GetAttributeFromElementsAsync(Tester.BY_XPATH, "//input",
"name");
if (values != null)
{
	foreach (string attr in values)
		tester.ConsoleMsg(attr);
}

await frame.SetAttributeInElementAsync(Tester.BY_XPATH, "//input[@id='buttonLogin']",
"name", "NameButtonLogin");

await frame.SetAttributeInElementsAsync(Tester.BY_XPATH, "//input", "name", "test");

await frame.SetValueInElementAsync(Tester.BY_XPATH, "//input[@id='login']", "Тестировщик");
string value = await frame.GetValueFromElementAsync(Tester.BY_XPATH,
"//input[@id='login']");

await frame.ClickElementAsync(Tester.BY_XPATH, "//*[@id='buttonLogin']");

bool result = await frame.IsClickableElementAsync(Tester.BY_XPATH, "//*[@id='buttonLogin']");

await frame.ScrollToElementAsync(Tester.BY_XPATH, "//*[@id='buttonLogin']", true);
```

```
int result = await frame.GetCountElementsAsync(Tester.BY_XPATH, "//input");

string html = await frame.GetHtmlFromElementAsync(Tester.BY_XPATH,
"//*[@id='buttonLogin']");

await frame.SetHtmlInElementAsync(Tester.BY_XPATH, "//*[@id='auth']/h2", "<h2>Тестовый
заголовок</h2>");

await frame.WaitNotVisibleElementAsync(Tester.BY_XPATH, "//*[@id='result']", 5);
await frame.ClickElementAsync(Tester.BY_XPATH, "//*[@id='buttonLogin']");
await frame.WaitVisibleElementAsync(Tester.BY_XPATH, "//*[@id='result']", 5);

bool result = await frame.FindElementAsync(Tester.BY_XPATH, "//*[@id='result']", 5);

bool result = await frame.FindVisibleElementAsync(Tester.BY_XPATH, "//*[@id='result']", 5);

string title = await frame.GetTitleAsync();

string url = await frame.GetUrlAsync();

await frame.SetTextInElementAsync(Tester.BY_XPATH, "//*[@id='auth']/h2", "Это тест");
string text = await frame.GetTextFromElementAsync(Tester.BY_XPATH, "//*[@id='auth']/h2");

await frame.SelectOptionAsync(Tester.BY_XPATH, "//*[@id='MySelect']",
FRAMEElement.BY_INDEX, "2");
await frame.SelectOptionAsync(Tester.BY_XPATH, "//*[@id='MySelect']",
FRAMEElement.BY_VALUE, "Mobile");
await frame.SelectOptionAsync(Tester.BY_XPATH, "//*[@id='MySelect']",
FRAMEElement.BY_TEXT, "Other");

string index = await frame.GetOptionAsync(Tester.BY_XPATH, "//*[@id='MySelect']",
FRAMEElement.BY_INDEX);
string value = await frame.GetOptionAsync(Tester.BY_XPATH, "//*[@id='MySelect']",
FRAMEElement.BY_VALUE);
string text = await frame.GetOptionAsync(Tester.BY_XPATH, "//*[@id='MySelect']",
FRAMEElement.BY_TEXT);
```

GetCountElementsAsync

GetCountElementsAsync
Description: the method returns the number of found items
Syntax:
GetCountElementsAsync(string by, string locator)
GetCountElementsAsync(Locator locator)
Return value: int

Example:

```
int count = await tester.GetCountElementsAsync(Tester.BY_CSS, "input");
int count = await tester.GetCountElementsAsync(Tester.BY_XPATH, "//input");

int count = await tester.GetCountElementsAsync(tester.GetLocator("locatorName"));
```

GetCountElementsByClassAsync

GetCountElementsByClassAsync
Description: the method returns the number of found items
Syntax: GetCountElementsByClassAsync(string _class)
Return value: int

Example:
int count = await tester.GetCountElementsByClassAsync("my-element");

GetCountElementsByNameAsync

GetCountElementsByNameAsync
Description: the method returns the number of found items
Syntax: GetCountElementsByNameAsync(string name)
Return value: int

Example:
```
int count = await tester.GetCountElementsByNameAsync("MyElement");
```

GetCountElementsByTagAsync

GetCountElementsByTagAsync
Description: the method returns the number of found items
Syntax: GetCountElementsByTagAsync(string tag)
Return value: int

Example:
```
int count = await tester.GetCountElementsByTagAsync("h2");
```

GetHtmlFromElementAsync

GetHtmlFromElementAsync

Description: the method returns the html representation of the object in a string expression
Syntax:
GetHtmlFromElementAsync(string by, string locator)
GetHtmlFromElementAsync(Locator locator)
Return value: string

Example:

```
string html = await tester.GetHtmlFromElementAsync(Tester.BY_CSS, "#auth > h2");
string html = await tester.GetHtmlFromElementAsync(Tester.BY_XPATH,
"//div[@id='auth']//h2");

string html = await tester.GetHtmlFromElementAsync(tester.GetLocator("locatorName"));
```

GetHtmlFromElementByClassAsync

GetHtmlFromElementByClassAsync
Description: the method returns the html representation of the object in a string expression
Syntax: GetHtmlFromElementByClassAsync(string _class, int index)
Return value: string

Example:
string html = await tester.GetHtmlFromElementByClassAsync("text-field", 0);

GetHtmlFromElementByIdAsync

GetHtmlFromElementByIdAsync
Description: the method returns the html representation of the object in a string expression
Syntax: GetHtmlFromElementByIdAsync(string id)
Return value: string

Example:
string html = await tester.GetHtmlFromElementByIdAsync("login");

GetHtmlFromElementByNameAsync

GetHtmlFromElementByNameAsync
Description: the method returns the html representation of the object in a string expression
Syntax: GetHtmlFromElementByNameAsync(string name, int index)
Return value: string

Example:
string html = await tester.GetHtmlFromElementByNameAsync("field", 0);

GetHtmlFromElementByTagAsync

GetHtmlFromElementByTagAsync
Description: the method returns the html representation of the object in a string expression
Syntax: GetHtmlFromElementByTagAsync(string tag, int index)
Return value: string

Example:
string html = await tester.GetHtmlFromElementByTagAsync("h1", 0);

IsVisibleElementAsync

IsVisibleElementAsync

Description: the method determines the visibility of the element and returns the value true or false

Syntax:

IsVisibleElementAsync(string by, string locator)

IsVisibleElementAsync(Locator locator)

Return value: bool (true or false)

Example:

```
bool result = await tester.IsVisibleElementAsync(Tester.BY_XPATH, "//*[@id='login']");
bool result = await tester.IsVisibleElementAsync(Tester.BY_CSS, "#login");
await tester.AssertTrueAsync(result);

bool result = await tester.IsVisibleElementAsync(tester.GetLocator("locatorName"));
await tester.AssertTrueAsync(result);
```

MakeElementVisibleAsync

MakeElementVisibleAsync

Description: the method makes the element visible by adding styles
Syntax:
MakeElementVisibleAsync(string by, string locator, string visibility = "visible", int opacity = 1, int index = 1000)
MakeElementVisibleAsync(Locator locator, string visibility = "visible", int opacity = 1, int index = 1000)

Example:

```
await tester.MakeElementVisibleAsync(Tester.BY_XPATH, "//*[@id='button']");
await tester.MakeElementVisibleAsync(Tester.BY_BY_CSS, "//*[@id='button']");

await tester.MakeElementVisibleAsync(tester.GetLocator("locatorName"));
```

SetHtmlInElementAsync

SetHtmlInElementAsync
Description: the method inserts the html representation of the object into the specified element
Syntax:
SetHtmlInElementAsync(string by, string locator, string html)
SetHtmlInElementAsync(Locator locator, string html)
Return value: no return value

Example:

```
await tester.SetHtmlInElementAsync(Tester.BY_CSS, "#auth > h2", "<div>text</div>");
await tester.SetHtmlInElementAsync(Tester.BY_XPATH, "//div[@id='auth']//h2",
"<div>text</div>");

await tester.SetHtmlInElementAsync(tester.GetLocator("locatorName"), "<div>text</div>");
```

SetHtmlInElementByClassAsync

SetHtmlInElementByClassAsync
Description: the method inserts the html representation of the object into the specified element
Syntax: SetHtmlInElementByClassAsync(string _class, int index, string html)
Return value: no return value

Example:
await tester.SetHtmlInElementByClassAsync("text-field", 0, "<h1>text</h1>");

SetHtmlInElementByIdAsync

SetHtmlInElementByIdAsync
Description: the method inserts the html representation of the object into the specified element
Syntax: SetHtmlInElementByIdAsync(string id, string html)
Return value: no return value

Example:
await tester.SetHtmlInElementByIdAsync("auth", "<h1>text</h1>");

SetHtmlInElementByNameAsync

SetHtmlInElementByNameAsync
Description: the method inserts the html representation of the object into the specified element
Syntax: SetHtmlInElementByNameAsync(string name, int index, string html)
Return value: no return value

Example:
```
await tester.SetHtmlInElementByNameAsync("block", 0,  "<h1>text</h1>");
```

SetHtmlInElementByTagAsync

SetHtmlInElementByTagAsync
Description: the method inserts the html representation of the object into the specified element
Syntax: SetHtmlInElementByTagAsync(string tag, int index, string html)
Return value: no return value

Example:
await tester.SetHtmlInElementByTagAsync("div", 0, "<h1>text</h1>");

Waiting

WaitAsync

WaitAsync
Description: the method temporarily stops the test execution for the specified number of seconds
Syntax: WaitAsync(int sec)
Return value: no return value

Example:
await tester.WaitAsync(5);

WaitElementInDomAsync

WaitElementInDomAsync

Description: the method temporarily stops the test execution for the specified number of seconds and waits for the requested element to appear in the DOM
Syntax:
WaitElementInDomAsync(string by, string locator, int sec)
WaitElementInDomAsync(Locator locator, int sec)
Return value: no return value

Example:

```
await tester.WaitElementInDomAsync(Tester.BY_XPATH, "//div[@id='result']", 5);
await tester.WaitElementInDomAsync(Tester.BY_CSS, "#result", 5);

await tester.WaitElementInDomAsync(tester.GetLocator("locatorName"), 5);
```

WaitElementNotDomAsync

WaitElementNotDomAsync

Description: the method temporarily stops the test execution for the specified number of seconds and waits for the requested element to cease to be present in the DOM
Syntax:
WaitElementNotDomAsync(string by, string locator, int sec)
WaitElementNotDomAsync(Locator locator, int sec)
Return value: no return value

Example:

```
await tester.WaitElementNotDomAsync(Tester.BY_XPATH, "//div[@id='element']", 5);
await tester.WaitElementNotDomAsync(Tester.BY_CSS, "#element", 5);

await tester.WaitElementNotDomAsync(tester.GetLocator("locatorName"), 5);
```

WaitNotVisibleElementAsync

WaitNotVisibleElementAsync

Description: the method temporarily stops the test execution for the specified number of seconds and waits for the requested element to stop being displayed

Syntax:

WaitNotVisibleElementAsync(string by, string locator, int sec)

WaitNotVisibleElementAsync(Locator locator, int sec)

Return value: no return value

Example:

```
await tester.WaitNotVisibleElementAsync(Tester.BY_CSS, "div[id='result']", 2);
await tester.WaitNotVisibleElementAsync(Tester.BY_XPATH, "//div[@id='result']", 2);

await tester.WaitNotVisibleElementAsync(tester.GetLocator("locatorName"), 2);
```

WaitNotVisibleElementByClassAsync

WaitNotVisibleElementByClassAsync
Description: the method temporarily stops the test execution for the specified number of seconds and waits for the requested element to stop being displayed
Syntax: WaitNotVisibleElementByClassAsync(string _class, int index, int sec)
Return value: no return value

Example:
await tester.WaitNotVisibleElementByClassAsync("my-element", 0, 25);

WaitNotVisibleElementByIdAsync

WaitNotVisibleElementByIdAsync
Description: the method temporarily stops the test execution for the specified number of seconds and waits for the requested element to stop being displayed
Syntax: WaitNotVisibleElementByIdAsync(string id, int sec)
Return value: no return value

Example:
await tester.WaitNotVisibleElementByIdAsync("MyElement", 25);

WaitNotVisibleElementByNameAsync

WaitNotVisibleElementByNameAsync
Description: the method temporarily stops the test execution for the specified number of seconds and waits for the requested element to stop being displayed
Syntax: WaitNotVisibleElementByNameAsync(string name, int index, int sec)
Return value: no return value

Example:
await tester.WaitNotVisibleElementByNameAsync("MyElement", 0, 25);

WaitNotVisibleElementByTagAsync

WaitNotVisibleElementByTagAsync
Description: the method temporarily stops the test execution for the specified number of seconds and waits for the requested element to stop being displayed
Syntax: WaitNotVisibleElementByTagAsync(string tag, int index, int sec)
Return value: no return value

Example:
await tester.WaitNotVisibleElementByTagAsync("h1", 0, 25);

WaitVisibleElementAsync

WaitVisibleElementAsync

Description: the method temporarily stops the test execution for the specified number of seconds and waits for the requested element to be displayed

Syntax:

WaitVisibleElementAsync(string by, string locator, int sec)

WaitVisibleElementAsync(Locator locator, int sec)

Return value: no return value

Example:

```
await tester.WaitVisibleElementAsync(Tester.BY_CSS, "div[id='result']", 2);
await tester.WaitVisibleElementAsync(Tester.BY_XPATH, "//div[@id='result']", 2);

await tester.WaitVisibleElementAsync(tester.GetLocator("locatorName"), 2);
```

WaitVisibleElementByClassAsync

WaitVisibleElementByClassAsync
Description: the method temporarily stops the test execution for the specified number of seconds and waits for the requested element to be displayed
Syntax: WaitVisibleElementByClassAsync(string _class, int index, int sec)
Return value: no return value

Example:
await tester.WaitVisibleElementByClassAsync("my-element", 0, 25);

WaitVisibleElementByIdAsync

WaitVisibleElementByIdAsync
Description: the method temporarily stops the test execution for the specified number of seconds and waits for the requested element to be displayed
Syntax: WaitVisibleElementByIdAsync(string id, int sec)
Return value: no return value

Example:
await tester.WaitVisibleElementByIdAsync("MyElement", 25);

WaitVisibleElementByNameAsync

WaitVisibleElementByNameAsync
Description: the method temporarily stops the test execution for the specified number of seconds and waits for the requested element to be displayed
Syntax: WaitVisibleElementByNameAsync(string name, int index, int sec)
Return value: no return value

Example:
await tester.WaitVisibleElementByNameAsync("MyElement", 25);

WaitVisibleElementByTagAsync

WaitVisibleElementByTagAsync
Description: the method temporarily stops the test execution for the specified number of seconds and waits for the requested element to be displayed
Syntax: WaitVisibleElementByTagAsync(string tag, int index, int sec)
Return value: no return value

Example:
await tester.WaitVisibleElementByTagAsync("h1", 25);

Search

FindElementAsync

FindElementAsync

Description: the method searches for an element in the DOM with a wait in seconds and returns a logical result true or false

Syntax:

FindElementAsync(string by, string locator, int sec)

FindElementAsync(Locator locator, int sec)

Return value: bool (true or false)

Example:

```
bool result = await tester.FindElementAsync(Tester.BY_CSS, "div[id='result']", 2);
bool result = await tester.FindElementAsync(Tester.BY_XPATH, "//div[@id='result']", 2);

bool result = await tester.FindElementAsync(tester.GetLocator("locatorName"), 2);
```

FindElementByClassAsync

FindElementByClassAsync
Description: the method searches for an element in the DOM with a wait in seconds and returns a logical result true or false
Syntax: FindElementByClassAsync(string _class, int index, int sec)
Return value: bool (true or false)

Example:
bool result = await tester.FindElementByClassAsync("my-element", 0, 5);

FindElementByIdAsync

FindElementByIdAsync
Description: the method searches for an element in the DOM with a wait in seconds and returns a logical result true or false
Syntax: FindElementByIdAsync(string id, int sec)
Return value: bool (true or false)

Example:

```
bool result = await tester.FindElementByIdAsync("MyElement", 5);
```

FindElementByNameAsync

FindElementByNameAsync

Description: the method searches for an element in the DOM with a wait in seconds and returns a logical result true or false

Syntax: FindElementByNameAsync(string name, int index, int sec)

Return value: bool (true or false)

Example:

```
bool result = await tester.FindElementByNameAsync("MyElement", 0, 5);
```

FindElementByTagAsync

FindElementByTagAsync
Description: the method searches for an element in the DOM with a wait in seconds and returns a logical result true or false
Syntax: FindElementByTagAsync(string tag, int index, int sec)
Return value: bool (true or false)

Example:
bool result = await tester.FindElementByTagAsync("h1", 0, 5);

FindVisibleElementAsync

FindVisibleElementAsync

Description: the method searches for a visually displayed element with a wait in seconds and returns a logical result true or false

Syntax:

FindVisibleElementAsync(string by, string locator, int sec)

FindVisibleElementAsync(Locator locator, int sec)

Return value: bool (true or false)

Example:

```
bool result = await tester.FindVisibleElementAsync(Tester.BY_CSS, "#auth #buttonLogin", 2);
bool result = await tester.FindVisibleElementAsync(Tester.BY_XPATH,
"//div[@id='auth']//input[@id='buttonLogin']", 2);

bool result = await tester.FindVisibleElementAsync(tester.GetLocator("locatorName"), 2);
```

FindVisibleElementByClassAsync

FindVisibleElementByClassAsync
Description: the method searches for a visually displayed element with a wait in seconds and returns a logical result true or false
Syntax: FindVisibleElementByClassAsync(string _class, int index, int sec)
Return value: bool (true or false)

Example:
bool result = await tester.FindVisibleElementByClassAsync("my-element", 0, 5);

FindVisibleElementByIdAsync

FindVisibleElementByIdAsync
Description: the method searches for a visually displayed element with a wait in seconds and returns a logical result true or false
Syntax: FindVisibleElementByIdAsync(string id, int sec)
Return value: bool (true or false)

Example:
bool result = await tester.FindVisibleElementByIdAsync("MyElement", 5);

FindVisibleElementByNameAsync

FindVisibleElementByNameAsync
Description: the method searches for a visually displayed element with a wait in seconds and returns a logical result true or false
Syntax: FindVisibleElementByNameAsync(string name, int index, int sec)
Return value: bool (true or false)

Example:
bool result = await tester.FindVisibleElementByNameAsync("MyElement", 0, 5);

FindVisibleElementByTagAsync

FindVisibleElementByTagAsync
Description: the method searches for a visually displayed element with a wait in seconds and returns a logical result true or false
Syntax: FindVisibleElementByTagAsync(string tag, int index, int sec)
Return value: bool (true or false)

Example:
```
bool result = await tester.FindVisibleElementByTagAsync("h1", 0, 5);
```


Styles

GetStyleFromElementAsync

GetStyleFromElementAsync
Description: the method returns the style of the element from the specified property
Syntax:
GetStyleFromElementAsync(string by, string locator, string property)
GetStyleFromElementAsync(Locator locator, string property)
Return value: string

Example:
string style = await tester.GetStyleFromElementAsync(Tester.BY_CSS, "#auth", "padding");
string style = await tester.GetStyleFromElementAsync(Tester.BY_XPATH, "//div[@id='auth']", "position");

string style = await tester.GetStyleFromElementAsync(tester.GetLocator("locatorName"), "padding");

GetStyleFromElementByClassAsync

GetStyleFromElementByClassAsync
Description: the method returns the style of the element from the specified property
Syntax: GetStyleFromElementByClassAsync(string _class, int index, string property)
Return value: string

Example:
string style = await tester.GetStyleFromElementByClassAsync("text-field", 0, "border");

GetStyleFromElementByIdAsync

GetStyleFromElementByIdAsync
Description: the method returns the style of the element from the specified property
Syntax: GetStyleFromElementByIdAsync(string id, string property)
Return value: string

Example:
string style = await tester.GetStyleFromElementByIdAsync("buttonLogin", "background-color");

GetStyleFromElementByNameAsync

GetStyleFromElementByNameAsync
Description: the method returns the style of the element from the specified property
Syntax: GetStyleFromElementByNameAsync(string name, int index, string property)
Return value: string

Example:
string style = await tester.GetStyleFromElementByNameAsync("pass", 0, "height");

GetStyleFromElementByTagAsync

GetStyleFromElementByTagAsync
Description: the method returns the style of the element from the specified property
Syntax: GetStyleFromElementByTagAsync(string tag, int index, string property)
Return value: string

Example:
string style = await tester.GetStyleFromElementByTagAsync("h2", 0, "width");

SetStyleInElementAsync

SetStyleInElementAsync
Description: the method sets the style of the element
Syntax:
SetStyleInElementAsync(string by, string locator, string cssText)
SetStyleInElementAsync(Locator locator, string cssText)
Return value: no return value

Example:
```
await tester.SetStyleInElementAsync(Tester.BY_XPATH, "//div[@id='auth']", "width: 250px; color: white; background-color: #000000;");
await tester.SetStyleInElementAsync(Tester.BY_CSS, "#auth", "width: 250px; color: white; background-color: #000000;");

await tester.SetStyleInElementAsync(tester.GetLocator("locatorName"), "width: 250px; color: white; background-color: #000000;");
```

SetStyleInElementByClassAsync

SetStyleInElementByClassAsync
Description: the method sets the style of the element
Syntax: SetStyleInElementByClassAsync(string _class, int index, string cssText)
Return value: no return value

Example:
await tester.SetStyleInElementByClassAsync("text-field", 0, "background-color: #123456;");

SetStyleInElementByIdAsync

SetStyleInElementByIdAsync
Description: the method sets the style of the element
Syntax: SetStyleInElementByIdAsync(string id, string cssText)
Return value: no return value

Example:
await tester.SetStyleInElementByIdAsync("buttonLogin", "background-color: #123456;");

SetStyleInElementByNameAsync

SetStyleInElementByNameAsync
Description: the method sets the style of the element
Syntax: SetStyleInElementByNameAsync(string name, int index, string cssText)
Return value: no return value

Example:
await tester.SetStyleInElementByNameAsync("pass", 0, "background-color: #123456;");

SetStyleInElementByTagAsync

SetStyleInElementByTagAsync
Description: the method sets the style of the element
Syntax: SetStyleInElementByTagAsync(string tag, int index, string cssText)
Return value: no return value

Example:
await tester.SetStyleInElementByTagAsync("h2", 0, "background-color: #123456;");

Page

GetCookiesAsync

GetCookiesAsync
Description: the method returns a list of cookies (document.cookie)..
Syntax: GetCookiesAsync()

Example:

```
List<string> cookies = await tester.GetCookiesAsync();
if (cookies.Count() > 0)
{
	foreach (string cookie in cookies)
	{
		tester.ConsoleMsg("COOKIE: " + cookie);
	}
}
```

GetListRedirectUrlAsync

GetListRedirectUrlAsync
Description: the method returns a list of redirects that occurred when the page was loaded
Syntax: GetListRedirectUrlAsync()
Return value: List

Example:

```
Tester tester = new Tester(browserForm);
await tester.TestBeginAsync();
await tester.GoToUrlAsync("https://yandex.ru/", 5);

List<string> redirects = await tester.GetListRedirectUrlAsync();
foreach (string url in redirects)
{
        tester.ConsoleMsg(url);
}
```

GetTitleAsync

GetTitleAsync
Description: the method returns the page title
Syntax: GetTitleAsync()
Return value: string

Example:
string text = await tester.GetTitleAsync();

GetUrlAsync

GetUrlAsync
Description: the method returns the current URL
Syntax: GetUrlAsync()
Return value: string

Example:
```
string text = await tester.GetUrlAsync();
```

GetUrlResponseAsync

GetUrlResponseAsync
Description: the method returns the HTTP response of the specified URL
Syntax: GetUrlResponseAsync(string url)
Return value: int

Example:

```
int response = await tester.GetUrlResponseAsync("https://somovstudio.github.io/test.html");
await tester.AssertEqualsAsync(200, response);
```

GoToUrlAsync

GoToUrlAsync

Description: the method loads the website at the specified URL with the specified wait in seconds
The abortLoadAfterTime flag automatically completes page loading after the timer is completed
Syntax: GoToUrlAsync(string url, int sec, bool abortLoadAfterTime = false)
Return value: no return value

Example:
await tester.GoToUrlAsync(@"https://www.google.com/", 25);

GoToUrlBaseAuthAsync

GoToUrlBaseAuthAsync
Description: the method loads the website at the specified URL during basic authorization with the specified wait in seconds
The abortLoadAfterTime flag automatically completes page loading after the timer is completed
Syntax: GoToUrlBaseAuthAsync(string url, string login, string pass, int sec, bool abortLoadAfterTime = false)
Return value: no return value

Example:
await tester.GoToUrlBaseAuthAsync("https://dev.site.com", "login", "pass", 25);

LoadPageAsync

LoadPageAsync

Description: the method loads a web page at the specified URL with the specified wait time in seconds. The abort Load After Time flag forcibly terminates page loading after the timer ends. Errors when loading the page are ignored and do not cause the autotest to crash.
Syntax: LoadPageAsync(string url, int sec, bool abortLoadAfterTime = false)

Example:
await tester.LoadPageAsync (@"https://www.google.com/", 25, true);

Text

GetTextFromElementAsync

GetTextFromElementAsync
Description: the method returns the text from the specified element
Syntax:
GetTextFromElementAsync(string by, string locator)
GetTextFromElementAsync(Locator locator)
Return value: string

Example:

```
string text = await tester.GetTextFromElementAsync(Tester.BY_CSS, "#auth > h2");
string text = await tester.GetTextFromElementAsync(Tester.BY_XPATH, "//div[@id='auth']//h2");

string text = await tester.GetTextFromElementAsync(tester.GetLocator("locatorName"));
```

GetTextFromElementByClassAsync

GetTextFromElementByClassAsync
Description: the method returns the text from the specified element
Syntax: GetTextFromElementByClassAsync(string _class, int index)
Return value: string

Example:
```
string text = await tester.GetTextFromElementByClassAsync("my-element", 0);
```

GetTextFromElementByIdAsync

GetTextFromElementByIdAsync
Description: the method returns the text from the specified element
Syntax: GetTextFromElementByIdAsync(string id)
Return value: string

Example:
string text = await tester.GetTextFromElementByIdAsync("MyElement");

GetTextFromElementByNameAsync

GetTextFromElementByNameAsync
Description: the method returns the text from the specified element
Syntax: GetTextFromElementByNameAsync(string name, int index)
Return value: string

Example:
string text = await tester.GetTextFromElementByNameAsync("MyElement", 0);

GetTextFromElementByTagAsync

GetTextFromElementByTagAsync
Description: the method returns the text from the specified element
Syntax: GetTextFromElementByTagAsync(string tag, int index)
Return value: string

Example:
string text = await tester.GetTextFromElementByTagAsync("h1", 0);

SetTextInElementAsync

SetTextInElementAsync
Description: the method inserts text into the specified element
Syntax:
SetTextInElementAsync(string by, string locator, string text)
SetTextInElementAsync(Locator locator, string text)
Return value: no return value

Example:

```
await tester.SetTextInElementAsync(Tester.BY_CSS, "#auth > h2", "text");
await tester.SetTextInElementAsync(Tester.BY_XPATH, "//div[@id='auth']//h2", "text");

await tester.SetTextInElementAsync(tester.GetLocator("locatorName"), "text");
```

SetTextInElementByClassAsync

SetTextInElementByClassAsync
Description: the method inserts text into the specified element
Syntax: SetTextInElementByClassAsync(string _class, int index, string text)
Return value: no return value

Example:
await tester.SetTextInElementByClassAsync("my-element", 0, "текст");

SetTextInElementByIdAsync

SetTextInElementByIdAsync
Description: the method inserts text into the specified element
Syntax: SetTextInElementByIdAsync(string id, string text)
Return value: no return value

Example:
await tester.SetTextInElementByIdAsync("MyElement", "текст");

SetTextInElementByNameAsync

SetTextInElementByNameAsync
Description: the method inserts text into the specified element
Syntax: SetTextInElementByNameAsync(string name, int index, string text)
Return value: no return value

Example:
await tester.SetTextInElementByNameAsync("MyElement", 0, "текст");

SetTextInElementByTagAsync

SetTextInElementByTagAsync
Description: the method inserts text into the specified element
Syntax: SetTextInElementByTagAsync(string tag, int index, string text)
Return value: no return value

Example:
```
await tester.SetTextInElementByTagAsync("h1", 0, "текст");
```


Methods for executing JavaScript

ExecuteJavaScriptAsync

ExecuteJavaScriptAsync
Description: the method executes JavaScript code and returns a string with the result of execution
Syntax: ExecuteJavaScriptAsync(string script)
Return value: string

Example:
```
string script = "(function(){ var element = document.getElementById('MyElement'); return element.innerText; }());";
string result = await tester.ExecuteJavaScriptAsync(script);
```


Methods for executing Rest requests

RestGetAsync

RestGetAsync
Description: the method executes a Get Rest request and gets the result in json format
Syntax: RestGetAsync(string url, TimeSpan timeout, string charset = "UTF-8")
Return value: string

Example:
```
string result = await tester.RestGetAsync("https://jsonplaceholder.typicode.com/posts/1/", TimeSpan.FromDays(1), "UTF-8");
tester.ConsoleMsg(result);
```

This method uses a standard approach:

```
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;

Uri uri = new Uri(url);
HttpClient client = new HttpClient();
client.Timeout = TimeSpan.FromDays(1);
client.BaseAddress = uri;
client.DefaultRequestHeaders.Clear();
client.DefaultRequestHeaders.Accept.Clear();
client.DefaultRequestHeaders.Accept.Add(new
MediaTypeWithQualityHeaderValue("application/json"));
client.DefaultRequestHeaders.Add("charset", "UTF-8");
client.DefaultRequestHeaders.Add("User-Agent", userAgent);
HttpResponseMessage response = await client.GetAsync(url);
if (response.IsSuccessStatusCode)
{
        return await response.Content.ReadAsStringAsync();
}
```


RestGetBasicAuthAsync

RestGetBasicAuthAsync

Description: the method executes a Get Rest request and gets the result in json format

Syntax: RestGetBasicAuthAsync(string login, string pass, string url, string charset = "UTF-8")

Return value: string

Example:

```
string result = await tester.RestGetBasicAuthAsync("admin", "0000",
"https://jsonplaceholder.typicode.com/posts/1/", TimeSpan timeout, "UTF-8");
tester.ConsoleMsg(result);
```

This method uses a standard approach:

```
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;

byte[] authToken = Encoding.ASCII.GetBytes($"{login}:{pass}");
Uri uri = new Uri(url);
HttpClient client = new HttpClient();
client.Timeout = TimeSpan.FromDays(1);
client.BaseAddress = uri;
```

```
client.DefaultRequestHeaders.Clear();
client.DefaultRequestHeaders.Accept.Clear();
client.DefaultRequestHeaders.Accept.Add(new
MediaTypeWithQualityHeaderValue("application/json"));
client.DefaultRequestHeaders.Add("charset", "UTF-8");
client.DefaultRequestHeaders.Add("User-Agent", userAgent);
client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Basic",
Convert.ToBase64String(authToken));
HttpResponseMessage response = await client.GetAsync(url);
if (response.IsSuccessStatusCode)
{
        return await response.Content.ReadAsStringAsync();
}
```


RestGetStatusCodeAsync

RestGetStatusCodeAsync
Description: the method executes a Get Rest request and as a result receives a status code
Syntax: RestGetStatusCodeAsync(string url)
Return value: int

Example:

```
int statusCode = await tester.RestGetAsync("https://jsonplaceholder.typicode.com");
tester.ConsoleMsg(statusCode.ToString());
```


RestPostAsync

RestPostAsync
Description: the method executes a Post Rest request with sending data in json format and receives the result in the same way in json format
Syntax: RestPostAsync(string url, string json, TimeSpan timeout, string charset = "UTF-8")
Return value: string

Example:

```
string result = await tester.RestPostAsync("https://jsonplaceholder.typicode.com/posts/1/",
"{}", TimeSpan.FromDays(1), "UTF-8");
tester.ConsoleMsg(result);
```

This method uses a standard approach:

```
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
```

```
Uri uri = new Uri(url);
HttpClient client = new HttpClient();
client.Timeout = TimeSpan.FromDays(1);
client.DefaultRequestHeaders.Add("charset", "UTF-8");
client.DefaultRequestHeaders.Add("User-Agent", userAgent);
HttpContent content = new StringContent("{}", Encoding.UTF8, "application/json");
HttpResponseMessage response = await client.PostAsync(uri, content);
if (response.IsSuccessStatusCode)
{
        return await response.Content.ReadAsStringAsync();
}
```


Methods for measuring the time spent

TimerStart

TimerStart
Description: the method starts the countdown and returns the value DateTime
Syntax: TimerStart()
Return value: DateTime

Example:

```
Tester tester = new Tester(browserForm);
DateTime start = await tester.TimerStart();
await tester.TestBeginAsync();
...
await tester.TestEndAsync();
TimeSpan result = await tester.TimerStop(start);
tester.ConsoleMsg("Time " + result.TotalSeconds);
```


TimerStop

TimerStop
Description: the method completes the countdown and returns the TimeSpan value with the result (for example, 7,132157, which means 7 seconds and 132157 milliseconds)
Syntax: TimerStop(DateTime start)
Return value: TimeSpan

Example:

```
Tester tester = new Tester(browserForm);
DateTime start = await tester.TimerStart();
```

```
await tester.TestBeginAsync();
...
await tester.TestEndAsync();
TimeSpan result = await tester.TimerStop(start);
tester.ConsoleMsg("Time " + result.TotalSeconds);
```


Methods for sending email and message

SendMsgToMailAsync

SendMsgToMailAsync
Description: the method sends an email
Syntax: SendMsgToMailAsync(string subject, string body, string filename = "")
Return value: no return value

Example:

```
Tester tester = new Tester(browserForm);
await tester.TestBeginAsync();
...
await tester.TestEndAsync();
if(tester.GetTestResult() == Tester.PASSED)
{
        await tester.SendMsgToMailAsync("Text message", "The test was completed successfully");
}
```


SendMsgToTelegramAsync

SendMsgToTelegramAsync
Description: the method sends a message to Telegram. To do this, you need to create a bot, add it to the chat and transfer the bot token (botToken), chat ID (chatId), message text (text) and, if necessary, specify the encoding (charset) to the function
Syntax: SendMsgToTelegramAsync(string botToken, string chatId, string text, string charset = "UTF-8", int timeHourFrom = 0, int timeHourBefore = 0)
Return value: no return value

Example:

```
Tester tester = new Tester(browserForm);
await tester.TestBeginAsync();
...
```

```
await tester.TestEndAsync();
if(tester.GetTestResult() == Tester.FAILED)
{
        // The message will be sent only if the current sending time falls within the range from 9:00 to 21:00
        // if you specify 0, 0, in this case the message will be sent without taking into account the current time
        await tester.SendMsgToTelegramAsync("0000000001:ABCDabcd123ABCDabcd123ABCDabcd123ZX", "-123456789", "Text message", "UTF-8", 9, 21);
}
```


Methods for checking the result

AssertEqualsAsync

AssertEqualsAsync
Description: the method performs a check between the actual and expected values, in case of a mismatch, the check will be considered a failure
Syntax: AssertEqualsAsync(dynamic expected, dynamic actual)
Return value: bool (true or false)

Example:

```
string expected = "xyz";
string actual = "xyz";
bool result = await tester.AssertEqualsAsync(expected, actual);
```


AssertNotEqualsAsync

AssertNotEqualsAsync
Description: the method performs a check between the actual and expected values, in case of a match, the check will be considered a failure
Syntax: AssertNotEqualsAsync(dynamic expected, dynamic actual)
Return value: bool (true or false)

Example:

```
string expected = "abc";
string actual = "xyz";
bool result = await tester.AssertNotEqualsAsync(expected, actual);
```

AssertTrueAsync

AssertTrueAsync

Description: the method checks the value of true or false, and if the value is false, the check will be considered a failure
Syntax: AssertTrueAsync(bool condition)
Return value: bool (true or false)

Example:
```
bool flag = true;
bool result = await tester.AssertTrueAsync(flag);
```


AssertFalseAsync

AssertFalseAsync

Description: the method checks the value of true or false, and if the value is true, the check will be considered a failure
Syntax: AssertFalseAsync(bool condition)
Return value: bool (true or false)

Example:
```
bool flag = false;
bool result = await tester.AssertFalseAsync(flag);
```


AssertNotNullAsync

AssertNotNullAsync

Description: the method checks a value that should not be null, and if the value is null, the check will be considered a failure
Syntax: AssertNotNullAsync(dynamic obj)
Return value: bool (true or false)

Example:
```
string text = "текст";
await tester.AssertNotNullAsync(text);
```


AssertNullAsync

AssertNullAsync

Description: the method checks the value that should be null, and if the value is not null, the check will be considered a failure
Syntax: AssertNullAsync(dynamic obj)

Return value: bool (true or false)

Example:
```
string text = null;
await tester.AssertNullAsync(text);
```


AssertNoErrorsAsync

AssertNoErrorsAsync
Description: the method checks for the absence of errors on the page and if errors are present, the check will be considered a failure
Syntax: AssertNoErrorsAsync(bool showListErrors = false, string[] listIgnored = null)
Return value: bool (true or false)

Example:
```
Tester tester = new Tester(browserForm);
...
await tester.TestBeginAsync();
await tester.GoToUrlAsync("https://somovstudio.github.io/test_error.html", 25);
await tester.AssertNoErrorsAsync(true, new string[1] { "stats.g.doubleclick.net" });
await tester.TestEndAsync();
```


AssertNetworkEventsAsync

AssertNetworkEventsAsync
Description: the method checks the presence (presence = true) or absence (presence = false) specified events (events), this method is good to use when checking the events Google Analytics and Yandex Metrika
Syntax: AssertNetworkEventsAsync(bool presence, string[] events)
Return value: bool (true or false)

Example:
```
await tester.AssertNetworkEventsAsync(true, new string[] {
        "ec=zayavka", "ea=b2c_new_main", "el=some_shpd_nm", "zayavka_shpd"
});
```


Methods for working with files

FileDownloadAsync

FileDownloadAsync

Description: the method downloads the file at the specified URL
Syntax: FileDownloadAsync(string fileURL, string filename, int waitingSec = 60)
Return value: no return value

Example:
```
await tester.FileDownloadAsync("https://somovstudio.github.io/img/logo.png", "C:\\download\\logo.png", 60);
```


FileGetHashMD5Async

FileGetHashMD5Async
Description: the method determines the MD5 Hash code of the specified file
Syntax: FileGetHashMD5Async(string filename)
Return value: string

Example:
```
string hash = await tester.FileGetHashMD5Async("C:\\download\\logo.png");
```


FileReadAsync

FileReadAsync
Description: the method reads the specified file
Syntax: FileReadAsync(string encoding, string filename)
Return value: string

Example:
```
string text = await tester.FileReadAsync(Tester.UTF8, "C:\\Hat\\file.txt");
```


FileWriteAsync

FileWriteAsync
Description: the method writes text to the specified file
Syntax: FileWriteAsync(string content, string encoding, string filename)
Return value: no return value

Example:
```
string text = "my text";
await tester.FileWriteAsync(text, Tester.UTF8, "C:\\Hat\\file.txt");
```


Methods for different tasks

CreateHashMD5FromTextAsync

CreateHashMD5FromTextAsync
Description: the method creates a HashDM5 code from the specified text
Syntax: CreateHashMD5FromTextAsync(string text)
Return value: string

Example:
```
string text = "Hello World";
string hash = await tester.CreateHashMD5FromTextAsync(text);
tester.ConsoleMsg(hash);
```


Class: HTMLElement

Constructor

HTMLElement

HTMLElement
Description: auxiliary class, an html element object
Syntax: HTMLElement(Tester tester, string by, string locator)

An example of getting an object:
```
HTMLElement element = await tester.GetElementAsync(Tester.BY_CSS, "#auth #buttonLogin");
await element.ClickAsync();

HTMLElement element = await tester.GetElementAsync(Tester.BY_XPATH,
"//div[@id='auth']//input[@id='buttonLogin']");
await element.ClickAsync();
```


Constants

BY_INDEX

BY_INDEX
Description: the constant indicates the type of value being processed

Syntax: BY_INDEX = "BY_INDEX"

Example:
HTMLElement.BY_INDEX

BY_TEXT

BY_TEXT
Description: the constant indicates the type of value being processed
Syntax: BY_TEXT = "BY_TEXT"

Example:
HTMLElement.BY_TEXT

BY_VALUE

BY_VALUE
Description: the constant indicates the type of value being processed
Syntax: BY_VALUE = "BY_VALUE"

Example:
HTMLElement.BY_VALUE

Variables

Id

Id
Description: the variable returns or receives the ID data
Syntax: string Id { get; set; }

Example:
```
HTMLElement element = await tester.GetElementAsync(Tester.BY_XPATH,
"//*[@id='MyInput']");
tester.ConsoleMsg("ID: " + element.Id);
```


Name

Name
Description: the variable returns or receives the Name data

Syntax: string Name { get; set; }

Example:
HTMLElement element = await tester.GetElementAsync(Tester.BY_XPATH,
"//*[@id='MyInput']");
tester.ConsoleMsg("NAME: " + element.Name);

Class

Class

Description: the variable returns or receives Class data
Syntax: string Class { get; set; }

Example:
HTMLElement element = await tester.GetElementAsync(Tester.BY_XPATH,
"//*[@id='MyInput']");
tester.ConsoleMsg("CLASS: " + element.Class);

Type

Type

Description: the variable returns or receives Type data
Syntax: string Type { get; set; }

Example:
HTMLElement element = await tester.GetElementAsync(Tester.BY_XPATH,
"//*[@id='MyInput']");
tester.ConsoleMsg("TYPE: " + element.Type);

Methods

ClickAsync

ClickAsync

Description: the method performs a click on the element
Syntax: ClickAsync();
Return value: no return value

Example:
await element.ClickAsync();

ClickMouseAsync

ClickMouseAsync
Description: the method performs clicking on an element by emulating a mouse
Syntax: ClickMouseAsync()
Return value: no return value

Example:
await element.ClickMouseAsync();

GetAttributeAsync

GetAttributeAsync
Description: the method returns the value of the specified attribute from the element
Syntax: GetAttributeAsync(string name);
Return value: string

Example:
string attrClass = await element.GetAttributeAsync("class");

GetHtmlAsync

GetHtmlAsync
Description: the method returns the html representation of the element in a string expression
Syntax: GetHtmlAsync();
Return value: string

Example:
string html = await element.GetHtmlAsync();

GetLocatorAsync

GetLocatorAsync
Description: the method returns the element's locator
Syntax: GetLocatorAsync()
Return value: string

Example:
string locator = await element.GetLocatorAsync();
tester.ConsoleMsg(locator);

GetOptionAsync

GetOptionAsync

Description: the method returns the index, text, or value of the selected option of the selected option from the select element
Syntax: GetOptionAsync(string by)
Return value: string

Example:

```
Tester tester = new Tester(browserForm);
await tester.TestBeginAsync();
await tester.GoToUrlAsync("https://somovstudio.github.io/test2.html", 5);
HTMLElement element = await tester.GetElementAsync(Tester.BY_XPATH,
"//*[@id='MySelect']");
await element.SelectOptionAsync(HTMLElement.BY_INDEX, "2");
await element.SelectOptionAsync(HTMLElement.BY_VALUE, "Mobile");
await element.SelectOptionAsync(HTMLElement.BY_TEXT, "Other");
string index = await element.GetOptionAsync(HTMLElement.BY_INDEX);
string text = await element.GetOptionAsync(HTMLElement.BY_TEXT);
string value = await element.GetOptionAsync(HTMLElement.BY_VALUE);
...
await tester.TestEndAsync();
```


GetStyleAsync

GetStyleAsync

Description: the method returns the style of the element from the specified property
Syntax: GetStyleAsync(string property)
Return value: string

Example:

```
string style = await element.GetStyleAsync("width");
```


GetTextAsync

GetTextAsync

Description: the method returns the text from the element
Syntax: GetTextAsync();
Return value: string

Example:

```
string text = await element.GetTextAsync();
```


GetValueAsync

GetValueAsync
Description: the method returns the value from the element
Syntax: GetValueAsync();
Return value: string

Example:

```
string value = await element.GetValueAsync();
```


IsClickableAsync

IsClickableAsync
Description: the method determines the clickability of the element and returns true or false
Syntax: IsClickableAsync()
Return value: bool (true or false)

Example:

```
Tester tester = new Tester(browserForm);
await tester.TestBeginAsync();
await tester.GoToUrlAsync("https://somovstudio.github.io/test2.html", 5);
HTMLElement element = await tester.GetElementAsync(Tester.BY_XPATH, "//button");
bool clickable = await element.IsClickableAsync();
await tester.AssertTrueAsync(clickable);
await tester.TestEndAsync();
```


MakeVisibleAsync

MakeVisibleAsync
Description: the method makes the element visible by adding styles
Syntax: MakeVisibleAsync(string by, string locator, string visibility = "visible", int opacity = 1, int index = 1000)

Example:

```
await tester.MakeVisibleAsync(Tester.BY_XPATH, "//*[@id='button']");

await tester.MakeVisibleAsync(Tester.BY_BY_CSS, "//*[@id='button']");
```

ScrollToAsync

ScrollToAsync

Description: the method scrolls to the element (the behavior Smooth parameter determines the smoothness of scrolling)
Syntax: ScrollToAsync(bool behaviorSmooth = false);
Return value: no return value

Example:

```
await element.ScrollToAsync();
await element.ScrollToAsync(true);
await element.ScrollToAsync(false);
```


SelectOptionAsync

SelectOptionAsync

Description: the method selects an option from the select element by the specified index, text, or value.
Syntax: SelectOptionAsync(string by, string value)
Return value: no return value

Example:

```
Tester tester = new Tester(browserForm);
await tester.TestBeginAsync();
await tester.GoToUrlAsync("https://somovstudio.github.io/test2.html", 5);
HTMLElement element = await tester.GetElementAsync(Tester.BY_XPATH,
"//*[@id='MySelect']");
await element.SelectOptionAsync(HTMLElement.BY_INDEX, "2");
await element.SelectOptionAsync(HTMLElement.BY_VALUE, "Mobile");
await element.SelectOptionAsync(HTMLElement.BY_TEXT, "Other");
string index = await element.GetOptionAsync(HTMLElement.BY_INDEX);
string text = await element.GetOptionAsync(HTMLElement.BY_TEXT);
string value = await element.GetOptionAsync(HTMLElement.BY_VALUE);
...
await tester.TestEndAsync();
```


SetAttributeAsync

SetAttributeAsync

Description: the method inserts the value of the specified attribute into the element
Syntax: SetAttributeAsync(string name, string value);
Return value: no return value

Example:
await element.SetAttributeAsync("class", "my-class");

SetHtmlAsync

SetHtmlAsync
Description: the method inserts html into the element
Syntax: SetHtmlAsync(string html);
Return value: no return value

Example:
await element.SetHtmlAsync(" Это тест ");

SetStyleAsync

SetStyleAsync
Description: the method sets the style of the element
Syntax: SetStyleAsync(string cssText)
Return value: no return value

Example:
await element.SetStyleAsync("width: 250px; background-color: #000000;");

SetTextAsync

SetTextAsync
Description: the method inserts text into the element
Syntax: SetTextAsync(string text);
Return value: no return value

Example:
await element.SetTextAsync("this is a test text");

SetValueAsync

SetValueAsync
Description: the method inserts a value into an element
Syntax: SetValueAsync(string value);
Return value: no return value

Example:
await element.SetValueAsync("this is a text value");

WaitNotVisibleAsync

WaitNotVisible
Description: the method waits for the specified number of seconds until the element is no longer displayed
Syntax: WaitNotVisibleAsync(int sec)
Return value: no return value

Example:
await element.WaitNotVisibleAsync(2);

WaitVisibleAsync

WaitVisibleAsync
Description: the method waits for the element to be displayed for the specified number of seconds
Syntax: WaitVisibleAsync(int sec)
Return value: no return value

Example:
await element.WaitVisibleAsync(2);

Class: FRAMEElement

Constructor

FRAMEElement

FRAMEElement
Description: an auxiliary class for working with the frame of the specified index
Syntax: FRAMEElement(Tester tester, int index)

An example of getting an object:
FRAMEElement frame = await tester.GetFrameAsync(0);
tester.ConsoleMsg("Index: " + frame.Index);
tester.ConsoleMsg("Name: " + frame.Name);

Constants

BY_INDEX

BY_INDEX
Description: the constant indicates the type of value being processed
Syntax: BY_INDEX = "BY_INDEX"

Example:
FRAMEElement.BY_INDEX

BY_TEXT

BY_TEXT
Description: the constant indicates the type of value being processed
Syntax: BY_TEXT = "BY_TEXT"

Example:
FRAMEElement.BY_TEXT

BY_VALUE

BY_VALUE
Description: the constant indicates the type of value being processed
Syntax: BY_VALUE = "BY_VALUE"

Example:
FRAMEElement.BY_VALUE

Variables

Name

Name
Description: the variable returns or gets the name of the frame
Syntax: string Name { get; set; }

Example:

```
FRAMEElement frame = await tester.GetFrameAsync(0);
tester.ConsoleMsg("Name: " + frame.Name);
```

Index

Index
Description: the variable returns or receives the index of the frame
Syntax: int Index { get; set; }

Example:
FRAMEElement frame = await tester.GetFrameAsync(0);
tester.ConsoleMsg("Index: " + frame.Index);

Index

Index
Description: the variable returns or receives the index of the frame
Syntax: int Index { get; set; }

Example:
FRAMEElement frame = await tester.GetFrameAsync(0);
tester.ConsoleMsg("Index: " + frame.Index);

Methods

ClickElementAsync

ClickElementAsync
Description: the method performs a click on the element
Syntax: ClickElementAsync(string by, string locator)
Return value: no return value

Example:
FRAMEElement frame = await tester.GetFrameAsync(0);
await frame.ClickElementAsync(Tester.BY_CSS, "#auth #buttonLogin");

await frame.ClickElementAsync(Tester.BY_XPATH,
"//div[@id='auth']//input[@id='buttonLogin']");

FindElementAsync

FindElementAsync

Description: the method searches for an element in the DOM with a wait in seconds and returns a logical result true or false
Syntax: FindElementAsync(string by, string locator, int sec)
Return value: bool (true or false)

Example:
FRAMEElement frame = await tester.GetFrameAsync(0);
bool result = await frame.FindElementAsync(Tester.BY_CSS, "div[id='result']", 2);

bool result = await frame.FindElementAsync(Tester.BY_XPATH, "//div[@id='result']", 2);

FindVisibleElementAsync

FindVisibleElementAsync

Description: the method searches for a visually displayed element with a wait in seconds and returns a logical result true or false
Syntax: FindVisibleElementAsync(string by, string locator, int sec)
Return value: bool (true or false)

Example:
FRAMEElement frame = await tester.GetFrameAsync(0);
bool result = await frame.FindVisibleElementAsync(Tester.BY_CSS, "#auth #buttonLogin", 2);

bool result = await frame.FindVisibleElementAsync(Tester.BY_XPATH, "//div[@id='auth']//input[@id='buttonLogin']", 2);

GetAttributeFromElementAsync

GetAttributeFromElementAsync

Description: the method returns a lowercase value from the specified attribute in the selected element
Syntax: GetAttributeFromElementAsync(string by, string locator, string attribute)
Return value: string

Example:
FRAMEElement frame = await tester.GetFrameAsync(0);
string value = await frame.GetAttributeFromElementAsync(Tester.BY_CSS, "input", "name");

string value = await frame.GetAttributeFromElementAsync(Tester.BY_XPATH, "//input", "name");

GetAttributeFromElementsAsync

GetAttributeFromElementsAsync
Description: the method returns a list of values of the specified attribute from a set of elements
Syntax: GetAttributeFromElementsAsync(string by, string locator, string attribute)
Return value: List

Example:

```
FRAMEElement frame = await tester.GetFrameAsync(0);
List<string> values = await frame.GetAttributeFromElementsAsync(Tester.BY_CSS, "input",
"name");
if(values != null)
{
	foreach (string attr in values)
		tester.ConsoleMsg(attr);
}


List<string> values = await frame.GetAttributeFromElementsAsync(Tester.BY_XPATH, "//input",
"name");
if(values != null)
{
	foreach (string attr in values)
		tester.ConsoleMsg(attr);
}
```


GetCountElementsAsync

GetCountElementsAsync
Description: the method returns the number of found items
Syntax: GetCountElementsAsync(string by, string locator)
Return value: int

Example:

```
FRAMEElement frame = await tester.GetFrameAsync(0);
int count = await frame.GetCountElementsAsync(Tester.BY_CSS, "input");

int count = await frame.GetCountElementsAsync(Tester.BY_XPATH, "//input");
```


GetHtmlFromElementAsync

GetHtmlFromElementAsync
Description: the method returns the html representation of the object in a string expression

Syntax: GetHtmlFromElementAsync(string by, string locator)
Return value: string

Example:
FRAMEElement frame = await tester.GetFrameAsync(0);
string html = await frame.GetHtmlFromElementAsync(Tester.BY_CSS, "#auth > h2");

string html = await frame.GetHtmlFromElementAsync(Tester.BY_XPATH,
"//div[@id='auth']//h2");

GetOptionAsync

GetOptionAsync

Description: the method returns the index, text, or value of the selected option of the selected option from the select element
Syntax: GetOptionAsync(string by, string locator, string type)
Return value: string

Example:
FRAMEElement frame = await tester.GetFrameAsync(0);

string index = await frame.GetOptionAsync(Tester.BY_CSS, "#MySelect",
FRAMEElement.BY_INDEX);
string value = await frame.GetOptionAsync(Tester.BY_CSS, "#MySelect",
FRAMEElement.BY_VALUE);
string text = await frame.GetOptionAsync(Tester.BY_CSS, "#MySelect",
FRAMEElement.BY_TEXT);

string index = await frame.GetOptionAsync(Tester.BY_XPATH, "//*[@id='MySelect']",
FRAMEElement.BY_INDEX);
string value = await frame.GetOptionAsync(Tester.BY_XPATH, "//*[@id='MySelect']",
FRAMEElement.BY_VALUE);
string text = await frame.GetOptionAsync(Tester.BY_XPATH, "//*[@id='MySelect']",
FRAMEElement.BY_TEXT);

GetStyleFromElementAsync

GetStyleFromElementAsync

Description: the method returns the style of the element from the specified property
Syntax: GetStyleFromElementAsync(string by, string locator, string property)
Return value: string

Example:

```
FRAMEElement frame = await tester.GetFrameAsync(0);
string style = await frame.GetStyleFromElementAsync(Tester.BY_CSS, "#auth > h2", "width");

string style = await frame.GetStyleFromElementAsync(Tester.BY_XPATH, "//div[@id='auth']",
"width");
```


GetTextFromElementAsync

GetTextFromElementAsync
Description: the method returns the text from the specified element
Syntax: GetTextFromElementAsync(string by, string locator)
Return value: string

Example:

```
FRAMEElement frame = await tester.GetFrameAsync(0);
string text = await frame.GetTextFromElementAsync(Tester.BY_CSS, "#auth > h2");

string text = await frame.GetTextFromElementAsync(Tester.BY_XPATH, "//div[@id='auth']//h2");
```


GetTitleAsync

GetTitleAsync
Description: the method returns the page title
Syntax: GetTitleAsync()
Return value: string

Example:

```
FRAMEElement frame = await tester.GetFrameAsync(0);
string title = await frame.GetTitleAsync();
```


GetUrlAsync

GetUrlAsync
Description: the method returns the current URL
Syntax: GetUrlAsync()
Return value: string

Example:

```
FRAMEElement frame = await tester.GetFrameAsync(0);
string url = await frame.GetUrlAsync();
```

GetValueFromElementAsync

GetValueFromElementAsync
Description: the method returns the value from the specified element
Syntax: GetValueFromElementAsync(string by, string locator)
Return value: string

Example:
```
FRAMEElement frame = await tester.GetFrameAsync(0);
string value = await frame.GetValueFromElementAsync(Tester.BY_CSS, "input[id='login']");

string value = await frame.GetValueFromElementAsync(Tester.BY_XPATH,
"//input[@id='login']");
```


IsClickableElementAsync

IsClickableElementAsync
Description: the method determines the clickability of the element and returns true or false
Syntax: IsClickableElementAsync(string by, string locator)
Return value: bool (true or false)

Example:
```
FRAMEElement frame = await tester.GetFrameAsync(0);
bool clickable = await frame.IsClickableElementAsync(Tester.BY_CSS, "#buttonLogin");
await tester.AssertTrueAsync(clickable);

bool clickable = await frame.IsClickableElementAsync(Tester.BY_XPATH,
"//*[@id='buttonLogin']");
await frame.AssertTrueAsync(clickable);
```


IsVisibleElementAsync

IsVisibleElementAsync
Description: the method determines the visibility of the element and returns the value true or false
Syntax: IsVisibleElementAsync(string by, string locator)
Return value: bool (true or false)

Example:
```
FRAMEElement frame = await tester.GetFrameAsync(0);
```

```
bool result = await frame.IsVisibleElementAsync(Tester.BY_XPATH, "//*[@id='login']");

bool result = await frame.IsVisibleElementAsync(Tester.BY_CSS, "#login");

await frame.AssertTrueAsync(result);
```


MakeVisibleAsync

MakeVisibleAsync

Description: the method makes the element visible by adding styles
Syntax: MakeVisibleAsync(string by, string locator, string visibility = "visible", int opacity = 1, int index = 1000)

Example:

```
await tester.MakeVisibleAsync(Tester.BY_XPATH, "//*[@id='button']");

await tester.MakeVisibleAsync(Tester.BY_BY_CSS, "//*[@id='button']");
```


ScrollToElementAsync

ScrollToElementAsync

Description: the method scrolls to the specified element (the behavior Smooth parameter determines the smoothness of scrolling)
Syntax: ScrollToElementAsync(string by, string locator, bool behaviorSmooth = false)
Return value: no return value

Example:

```
FRAMEElement frame = await tester.GetFrameAsync(0);
await frame.ScrollToElementAsync(Tester.BY_CSS, "body > footer", true);

await frame.ScrollToElementAsync(Tester.BY_XPATH, "/html/body/footer", true);
```


SelectOptionAsync

SelectOptionAsync

Description: the method selects an option from the select element by the specified index, text, or value
Syntax: SelectOptionAsync(string by, string locator, string type, string value)
Return value: no return value

Example:

```
FRAMEElement frame = await tester.GetFrameAsync(0);
```

```
await frame.SelectOptionAsync(Tester.BY_CSS, "#MySelect", FRAMEElement.BY_INDEX, "2");
await frame.SelectOptionAsync(Tester.BY_CSS, "#MySelect", FRAMEElement.BY_VALUE,
"Mobile");
await frame.SelectOptionAsync(Tester.BY_CSS, "#MySelect", FRAMEElement.BY_TEXT, "Other");

await frame.SelectOptionAsync(Tester.BY_XPATH, "//*[@id='MySelect']",
FRAMEElement.BY_INDEX, "2");
await frame.SelectOptionAsync(Tester.BY_XPATH, "//*[@id='MySelect']",
FRAMEElement.BY_VALUE, "Mobile");
await frame.SelectOptionAsync(Tester.BY_XPATH, "//*[@id='MySelect']",
FRAMEElement.BY_TEXT, "Other");
```


SetAttributeInElementAsync

SetAttributeInElementAsync
Description: the method inserts an attribute with a value into the specified element
Syntax: SetAttributeInElementAsync(string by, string locator, string attribute, string value)
Return value: no return value

Example:

```
FRAMEElement frame = await tester.GetFrameAsync(0);
await frame.SetAttributeInElementAsync(Tester.BY_CSS, "#auth > h2", "name", "test");

await frame.SetAttributeInElementAsync(Tester.BY_XPATH, "//div[@id='auth']//h2", "name",
"test");
```


SetAttributeInElementsAsync

SetAttributeInElementsAsync
Description: the method inserts an attribute with the specified value into a set of elements and returns a list as a result
Syntax: SetAttributeInElementsAsync(string by, string locator, string attribute, string value)
Return value: List

Example:

```
FRAMEElement frame = await tester.GetFrameAsync(0);

List<string> values = await frame.SetAttributeInElementsAsync(Tester.BY_CSS, "input", "class",
"test-class");
foreach (string value in values)
{
        tester.ConsoleMsg(value);
```

```
}

List<string> values = await frame.SetAttributeInElementsAsync(Tester.BY_XPATH, "//input",
"class",  "test-class");
foreach (string value in values)
{
        tester.ConsoleMsg(value);
}
```


SetHtmlInElementAsync

SetHtmlInElementAsync
Description: the method inserts the html representation of the object into the specified element
Syntax: SetHtmlInElementAsync(string by, string locator, string html)
Return value: no return value

Example:

```
FRAMEElement frame = await tester.GetFrameAsync(0);
await frame.SetHtmlInElementAsync(Tester.BY_CSS, "#auth > h2", "<div>Тестовый
блок</div>");

await frame.SetHtmlInElementAsync(Tester.BY_XPATH, "//div[@id='auth']//h2", "<div>Тестовый
блок</div>");
```


SetStyleInElementAsync

SetStyleInElementAsync
Description:the method sets the style of the element
Syntax: SetStyleInElementAsync(string by, string locator, string cssText)
Return value: no return value

Example:

```
FRAMEElement frame = await tester.GetFrameAsync(0);
await frame.SetStyleInElementAsync(Tester.BY_XPATH, "//div[@id='auth']", "background-color:
#000000;");

await frame.SetStyleInElementAsync(Tester.BY_CSS, "#auth", "background-color: #000000;");
```


SetTextInElementAsync

SetTextInElementAsync

Description: the method inserts text into the specified element
Syntax: SetTextInElementAsync(string by, string locator, string text)
Return value: no return value

Example:
FRAMEElement frame = await tester.GetFrameAsync(0);
await frame.SetTextInElementAsync(Tester.BY_CSS, "#auth > h2", "Тестовый заголовок");

await frame.SetTextInElementAsync(Tester.BY_XPATH, "//div[@id='auth']//h2", "Тестовый заголовок");

SetValueInElementAsync

SetValueInElementAsync

Description: the method inserts a value into the specified element
Syntax: SetValueInElementAsync(string by, string locator, string value)
Return value: no return value

Example:
FRAMEElement frame = await tester.GetFrameAsync(0);
await frame.SetValueInElementAsync(Tester.BY_CSS, "input[id='login']", "admin");
await frame.SetValueInElementAsync(Tester.BY_CSS, "input[id='pass']", "0000");

await frame.SetValueInElementAsync(Tester.BY_XPATH, "//input[@id='login']", "admin");
await frame.SetValueInElementAsync(Tester.BY_XPATH, "//input[@id='pass']", "0000");

WaitNotVisibleElementAsync

WaitNotVisibleElementAsync

Description: the method temporarily stops the test execution for the specified number of seconds and waits for the requested element to stop being displayed
Syntax: WaitNotVisibleElementAsync(string by, string locator, int sec)
Return value: no return value

Example:
FRAMEElement frame = await tester.GetFrameAsync(0);
await frame.WaitNotVisibleElementAsync(Tester.BY_CSS, "div[id='result']", 2);

await frame.WaitNotVisibleElementAsync(Tester.BY_XPATH, "//div[@id='result']", 2);

WaitVisibleElementAsync

WaitVisibleElementAsync
Description: the method temporarily stops the test execution for the specified number of seconds and waits for the requested element to be displayed
Syntax: WaitVisibleElementAsync(string by, string locator, int sec)
Return value: no return value

Example:

```
FRAMEElement frame = await tester.GetFrameAsync(0);
await frame.WaitVisibleElementAsync(Tester.BY_CSS, "div[id='result']", 2);

await frame.WaitVisibleElementAsync(Tester.BY_XPATH, "//div[@id='result']", 2);
```


Plugins

HatPluginMySql

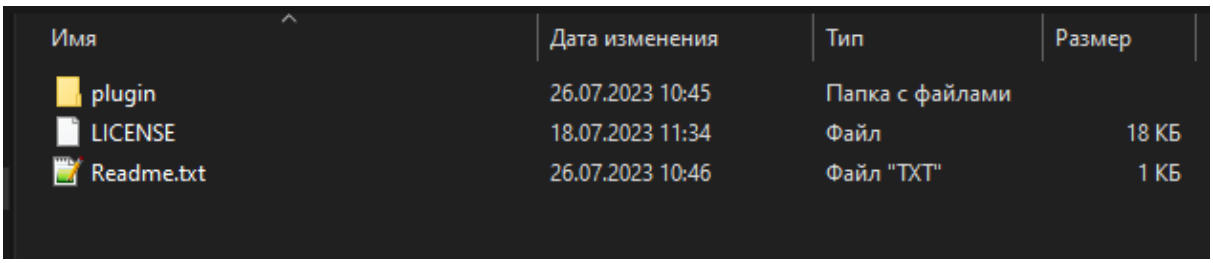
Installing the plugin

Installing the plugin HatPluginMySql

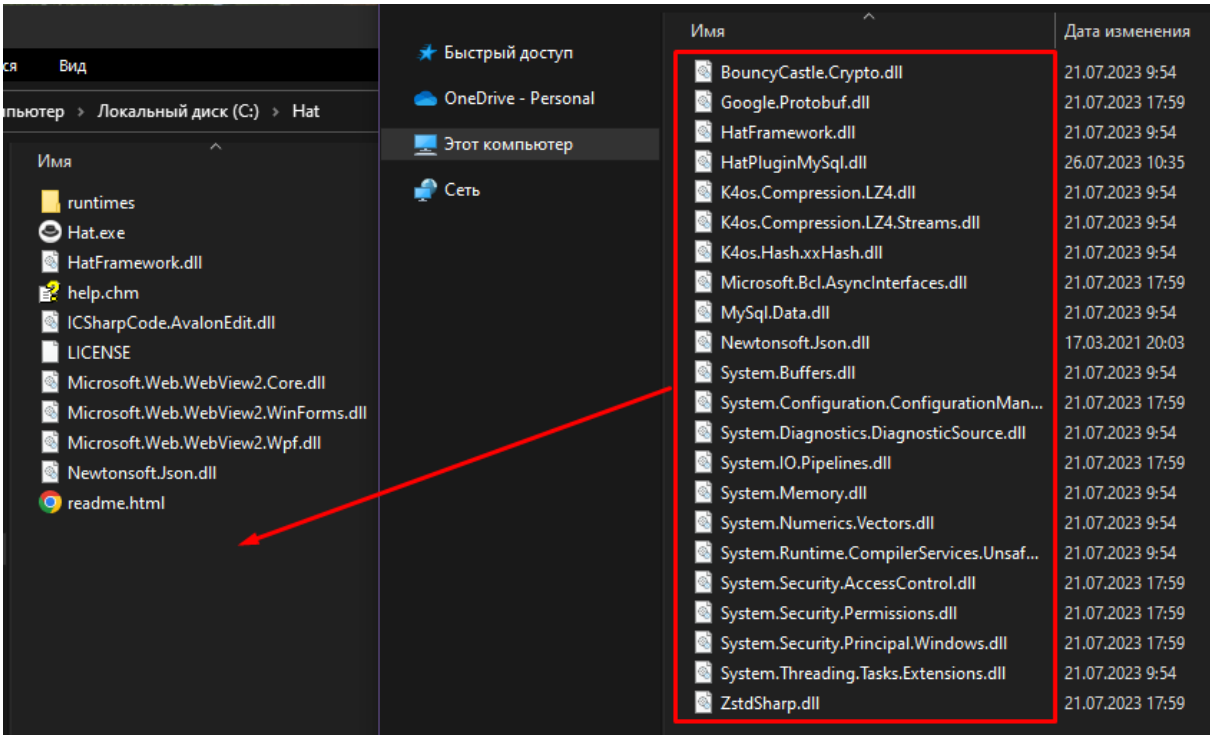
1. Download and install the browser **Hat**
link: https://github.com/SomovStudio/Hat/releases

2. Download the plugin **HatPluginMySql**
link: https://github.com/SomovStudio/HatPluginMySql/releases

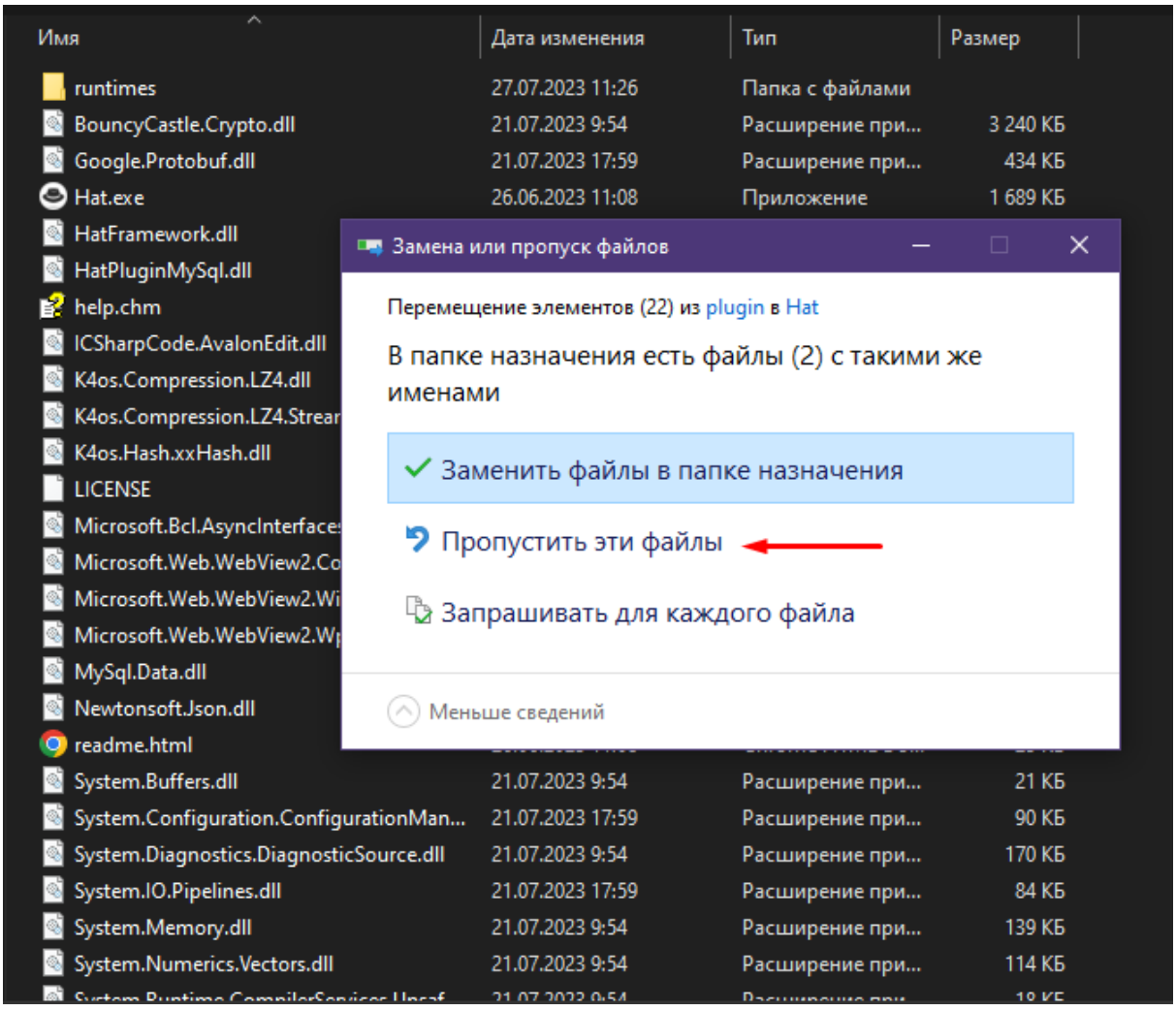
3. Unpack the archive **HatPluginMySql-1.0.0.zip**



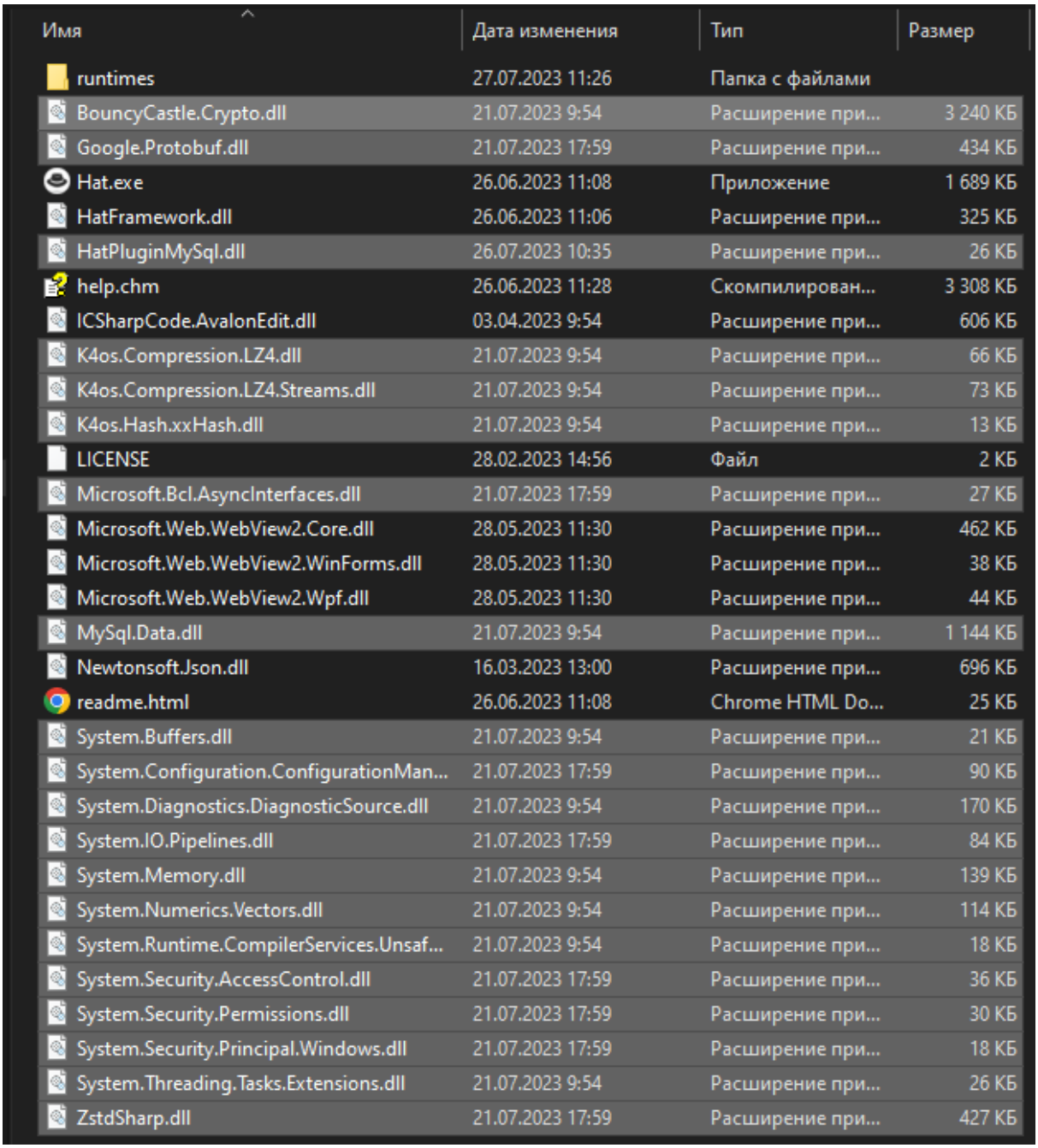
4. Copy the contents of the **plugin** folder to the root of the **Hat** browser folder



When copying, you may be asked about replacing existing files.
In this case, it is recommended to select "Skip these files".



As a result, the following files should be in the root of the folder



Имя	Дата изменения	Тип	Размер
runtimes	27.07.2023 11:26	Папка с файлами	
BouncyCastle.Crypto.dll	21.07.2023 9:54	Расширение при...	3 240 КБ
Google.Protobuf.dll	21.07.2023 17:59	Расширение при...	434 КБ
Hat.exe	26.06.2023 11:08	Приложение	1 689 КБ
HatFramework.dll	26.06.2023 11:06	Расширение при...	325 КБ
HatPluginMySql.dll	26.07.2023 10:35	Расширение при...	26 КБ
help.chm	26.06.2023 11:28	Скомпилирован...	3 308 КБ
ICSharpCode.AvalonEdit.dll	03.04.2023 9:54	Расширение при...	606 КБ
K4os.Compression.LZ4.dll	21.07.2023 9:54	Расширение при...	66 КБ
K4os.Compression.LZ4.Streams.dll	21.07.2023 9:54	Расширение при...	73 КБ
K4os.Hash.xxHash.dll	21.07.2023 9:54	Расширение при...	13 КБ
LICENSE	28.02.2023 14:56	Файл	2 КБ
Microsoft.Bcl.AsyncInterfaces.dll	21.07.2023 17:59	Расширение при...	27 КБ
Microsoft.Web.WebView2.Core.dll	28.05.2023 11:30	Расширение при...	462 КБ
Microsoft.Web.WebView2.WinForms.dll	28.05.2023 11:30	Расширение при...	38 КБ
Microsoft.Web.WebView2.Wpf.dll	28.05.2023 11:30	Расширение при...	44 КБ
MySql.Data.dll	21.07.2023 9:54	Расширение при...	1 144 КБ
Newtonsoft.Json.dll	16.03.2023 13:00	Расширение при...	696 КБ
readme.html	26.06.2023 11:08	Chrome HTML Do...	25 КБ
System.Buffers.dll	21.07.2023 9:54	Расширение при...	21 КБ
System.Configuration.ConfigurationMan...	21.07.2023 17:59	Расширение при...	90 КБ
System.Diagnostics.DiagnosticSource.dll	21.07.2023 9:54	Расширение при...	170 КБ
System.IO.Pipelines.dll	21.07.2023 17:59	Расширение при...	84 КБ
System.Memory.dll	21.07.2023 9:54	Расширение при...	139 КБ
System.Numerics.Vectors.dll	21.07.2023 9:54	Расширение при...	114 КБ
System.Runtime.CompilerServices.Unsaf...	21.07.2023 9:54	Расширение при...	18 КБ
System.Security.AccessControl.dll	21.07.2023 17:59	Расширение при...	36 КБ
System.Security.Permissions.dll	21.07.2023 17:59	Расширение при...	30 КБ
System.Security.Principal.Windows.dll	21.07.2023 17:59	Расширение при...	18 КБ
System.Threading.Tasks.Extensions.dll	21.07.2023 9:54	Расширение при...	26 КБ
ZstdSharp.dll	21.07.2023 17:59	Расширение при...	427 КБ

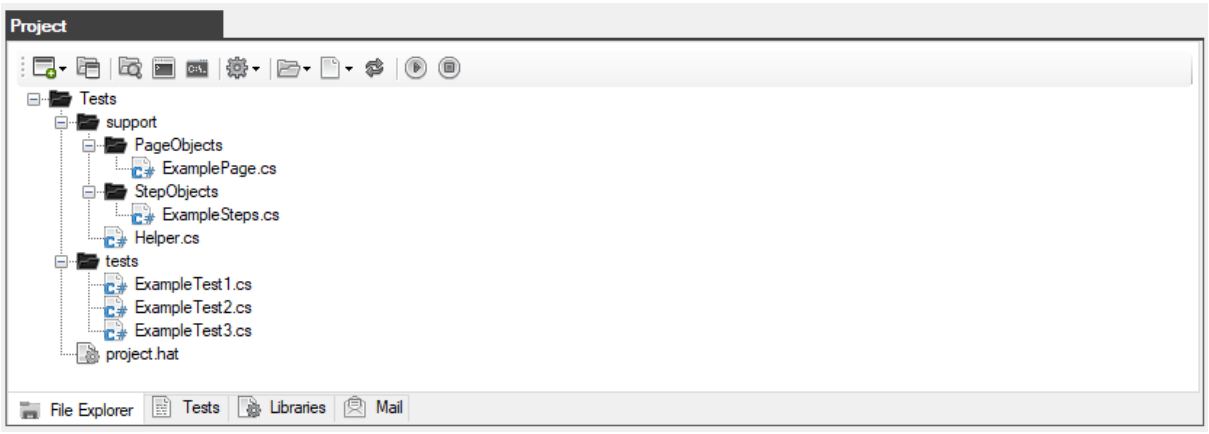
This completes the installation of the plugin!

Connecting the plugin to the project

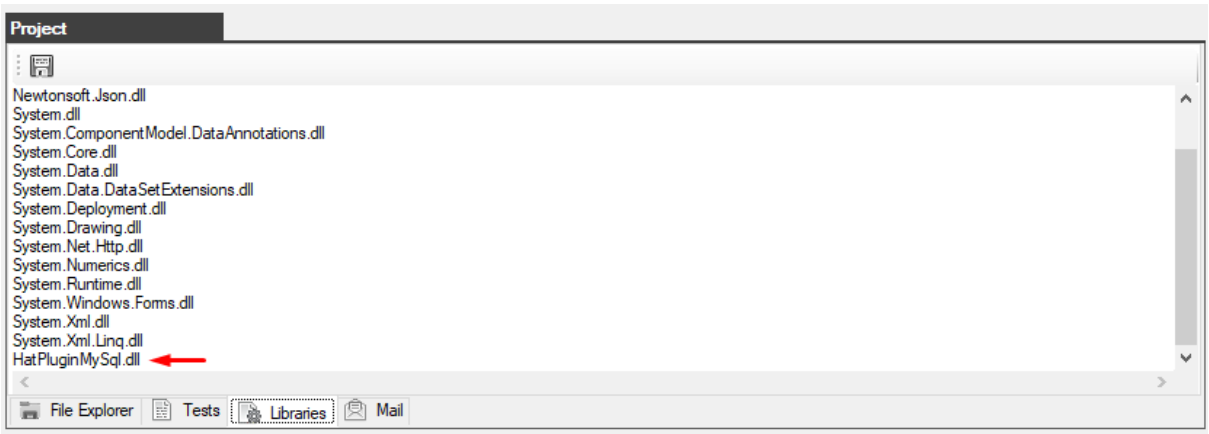
Connecting the HatPluginMySql plugin to the autotest project

1. Launch the browser **Hat**

2. Create a new project or open an existing project



3. Go to the Libraries tab and enter the name of one file at the end of the list **HatPluginMySql.dll**



4. Click the "Save" button.

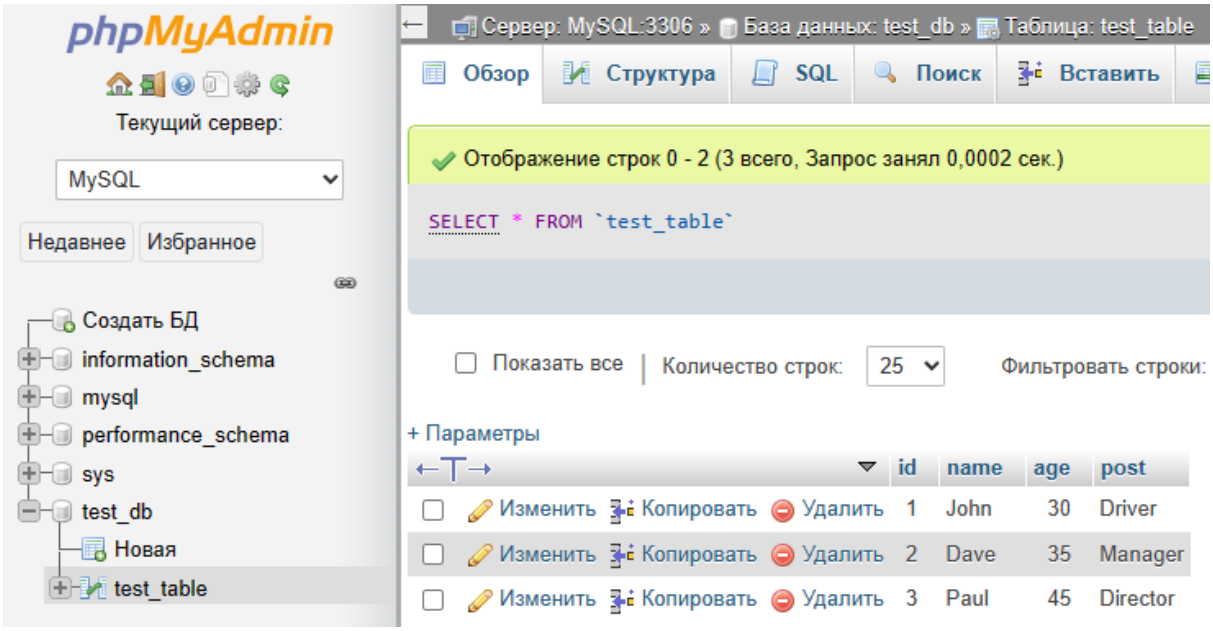
This completes the plug-in connection to the project

An example of an autotest

An example of an autotest

After the plugin has been installed and connected to the project, let's consider a simple autotest.

We have created a simple database in MySQL named test_db
There is one test_table table in this database with the fields: id, name, age, post



Now let's describe a simple autotest that will add, modify and delete data in the database while checking the correctness of the results.

Файл: ExampleTest3.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Windows.Forms;
using System.Threading;
using System.Threading.Tasks;
using System.IO;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Reflection;
using Newtonsoft.Json;
using HatFramework;
using HatPluginMySql;

namespace Hat
{
    public class ExampleTest3
    {
            Tester tester;

        public async void Main(Form browserWindow)
        {
            tester = new Tester(browserWindow);
```

```
            await setUp();
            await test();
            await tearDown();
        }

        public async Task setUp()
        {
            tester.Description("Test #3 checking the
database");
            await tester.BrowserFullScreenAsync();
        }

        public async Task test()
        {
                DataTable dataTable = null;
            List<List<string>> entries = new
List<List<string>>();
                TesterMySql testerMySql = new
TesterMySql(tester);

            await tester.TestBeginAsync();

            await
testerMySql.ConnectionOpenAsync
("server=127.0.0.1;uid=root;pwd=;database=test_db");

            int count = await
testerMySql.GetCountEntriesAsync("SELECT * FROM test_table");
            if (count > 0)
            {
                    entries =
testerMySql.GetEntriesAsync("SELECT * FROM test_table");
                    dataTable =
testerMySql.GetDataTableAsync("SELECT * FROM test_table");
                foreach (DataRow row in dataTable.Rows)
                    foreach (DataColumn col in
dataTable.Columns)
                        tester.ConsoleMsg(row[col].ToString()
);
            }

            await testerMySql.SetEntryAsync("INSERT INTO
test_table VALUES(NULL, 'I am Tester', 100, 'My post QA')");
            bool result = await
testerMySql.FindEntryAsync("test_table", "name", "'I am
Tester'");
            await
testerMySql.AssertHaveInTableAsync("test_table", "name", "'I
am Tester'");
```

```
                entries = await
testerMySql.GetEntriesFromTableAsync("test_table");
            if (entries != null){
                foreach(List<string> entry in entries)
                    foreach(string value in entry)
                        tester.ConsoleMsg(value);
            }

            await testerMySql.EditEntryAsync("UPDATE
test_table SET age = 111 WHERE name = 'I am Tester'");
                result = await
testerMySql.FindEntryAsync("test_table", "age", "111");
            await
testerMySql.AssertHaveInTableAsync("test_table", "age",
"111");

            await testerMySql.RemoveEntryAsync("DELETE FROM
test_table WHERE name = 'I am Tester'");
                result = await
testerMySql.FindEntryAsync("test_table", "name", "'I am
Tester'");
            await
testerMySql.AssertDontHaveInTableAsync("test_table", "name",
"'I am Tester'");

            await testerMySql.ConnectionCloseAsync();

            await tester.TestEndAsync();
        }

        public async Task tearDown()
        {
            //  await  tester.BrowserCloseAsync();
        }
    }
}
```

Please note that for the autotests to work correctly, you need to connect the library

```
using HatPluginMySql;
```

First, there is a connection to the database

```
TesterMySql testerMySql = new TesterMySql(tester);
await
testerMySql.ConnectionOpenAsync
("server=127.0.0.1;uid=root;pwd=;database=test_db");
```

Data manipulation is performed using different methods

```
await testerMySql.GetCountEntriesAsync("SELECT * FROM
```

```
test_table");
await testerMySql.GetEntriesAsync("SELECT * FROM
test_table");
await testerMySql.GetEntriesFromTableAsync("test_table");
await testerMySql.GetDataTableAsync("SELECT * FROM
test_table");
await testerMySql.SetEntryAsync("INSERT INTO test_table
VALUES(NULL, 'I am Tester', 100, 'My post QA')");
await testerMySql.FindEntryAsync("test_table", "name", "'I
am Tester'");
await testerMySql.EditEntryAsync("UPDATE test_table SET age
= 111 WHERE name = 'I am Tester'");
await testerMySql.RemoveEntryAsync("DELETE FROM test_table
WHERE name = 'I am Tester'");
```

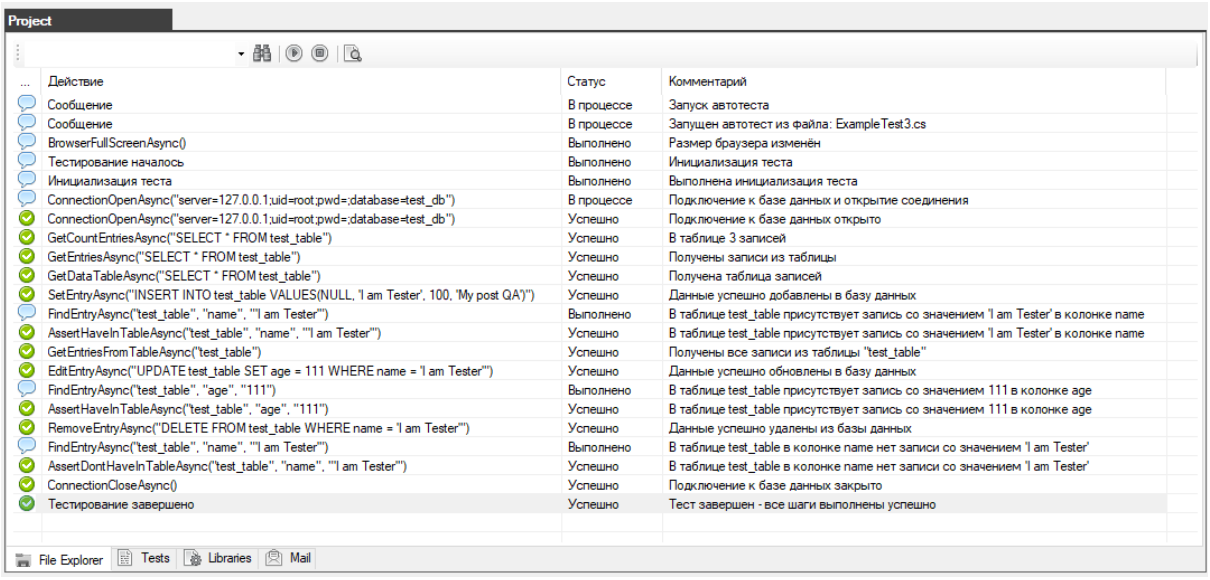
Special methods for checking the presence or absence of data in the database

```
await testerMySql.AssertHaveInTableAsync("test_table",
"name", "'I am Tester'");
await testerMySql.AssertDontHaveInTableAsync("test_table",
"name", "'I am Tester'")
```

At the end, the work with the database is completed by closing the connection to it

```
await testerMySql.ConnectionCloseAsync();
```

Let's run the autotest and look at the progress of the check and the result



All actions were completed, all checks were successful.

The test was completed successfully.

Class: TesterMySql

Constructor

TesterMySql

TesterMySql
Description: the main class of working with a MySQL database
Syntax: TesterMySql(Tester tester)

Example:
Tester tester = new Tester(browserWindow);
TesterMySql testerMySql = new TesterMySql(tester);

Methods

ConnectionOpenAsync

ConnectionOpenAsync
Description: the method opens a connection to the MySQL database
Syntax: ConnectionOpenAsync(string connectionString)
Return value: no return value

Example:
await testerMySql.ConnectionOpenAsync("server=127.0.0.1;uid=root;pwd=;database=test_db");

ConnectionCloseAsync

ConnectionCloseAsync
Description: the method closes the connection to the MySQL database
Syntax: ConnectionCloseAsync()
Return value: no return value

Example:
await testerMySql.ConnectionCloseAsync();

GetCountEntriesAsync

GetCountEntriesAsync
Description: the method returns the number of records in the table after executing the query
Syntax: GetCountEntriesAsync(string sqlQuertSelect)
Return value: int

Example:
int count = await testerMySql.GetCountEntriesAsync("SELECT * FROM test_table");

GetEntriesAsync

GetEntriesAsync
Description: the method returns a list of records from the table after executing the query
Syntax: GetEntriesAsync(string sqlQuertSelect)
Return value: List

Example:

```
List<List<string>> entries = new List<List<string>>();
entries = await testerMySql.GetEntriesAsync("SELECT * FROM test_table");

foreach(List<string> entry in entries)
        foreach(string value in entry)
                tester.ConsoleMsg(value);
```

GetEntriesFromTableAsync

GetEntriesFromTableAsync
Description: the method returns a list of records from the specified database table
Syntax: GetEntriesFromTableAsync(string tableName)
Return value: List

Example:

```
List<List<string>> entries = new List<List<string>>();
entries = await testerMySql.GetEntriesFromTableAsync("test_table");

foreach(List<string> entry in entries)
        foreach(string value in entry)
                tester.ConsoleMsg(value);
```

GetDataTableAsync

GetDataTableAsync
Description: the method returns a table of records from the database table after executing the query
Syntax: GetDataTableAsync(string sqlQuertSelect)
Return value: DataTable

Example:

```
DataTable dataTable = null;
dataTable = await testerMySql.GetDataTableAsync("SELECT * FROM test_table");

foreach (DataRow row in dataTable.Rows)
                foreach (DataColumn col in dataTable.Columns)
                        tester.ConsoleMsg(row[col].ToString());
```

SetEntryAsync

SetEntryAsync
Description: the method executes a query that adds data to the database table and returns the record number
Syntax: SetEntryAsync(string sqlQuertInsert)
Return value: int

Example:
int result = await testerMySql.SetEntryAsync("INSERT INTO test_table VALUES(NULL, 'I am Tester', 100, 'My post QA')");

EditEntryAsync

EditEntryAsync
Description: the method executes a query that modifies the data in the database table and returns the record number
Syntax: EditEntryAsync(string sqlQuertUpdate)
Return value: int

Example:
int result = await testerMySql.EditEntryAsync("UPDATE test_table SET age = 111 WHERE name = 'I am Tester'");

RemoveEntryAsync

RemoveEntryAsync
Description: the method executes a query that deletes the data in the database table and returns the record number
Syntax: RemoveEntryAsync(string sqlQuertDelete)
Return value: int

Example:
int result = await testerMySql.RemoveEntryAsync("DELETE FROM test_table WHERE name = 'I am Tester'");

FindEntryAsync

FindEntryAsync
Description: the method searches for data in the specified database table and returns the boolean value of the search result
Syntax: FindEntryAsync(string tableName, string columnName, string value)
Return value: bool (true or false)

Example:
bool result = await testerMySql.FindEntryAsync("test_table", "name", "'I am Tester'");

AssertHaveInTableAsync

AssertHaveInTableAsync

Description: the method checks the data in the specified database table and returns the boolean value of the search result, in case of a negative result, the check will be considered a failure
Syntax: AssertHaveInTableAsync(string tableName, string columnName, string value)
Return value: bool (true or false)

Example:

```
bool result = await testerMySql.AssertHaveInTableAsync("test_table", "name", "'I am Tester'");
```

AssertDontHaveInTableAsync

AssertDontHaveInTableAsync
Description: the method checks the data in the specified database table and returns the boolean value of the search result, in case of a positive result, the check will be considered a failure
Syntax: AssertDontHaveInTableAsync(string tableName, string columnName, string value)
Return value: bool (true or false)

Example:
bool result = await testerMySql.AssertDontHaveInTableAsync("test_table", "name", "'I am Tester'");

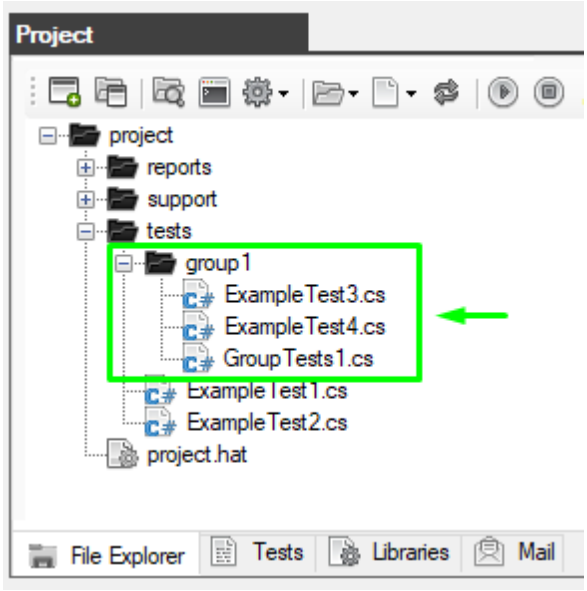
Practical examples

Autotest Group

Autotest Group

You can organize a group of autotests and run them.

To do this, create a folder in which the autotests of one group will be stored (for example, the folder group 1)



Next, you need to describe several tests and put them in the group1 folder.

Attention: for tests, the Main function must be declared as a Task

the first file: ExampleTest3.cs

File: ExampleTest3.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Windows.Forms;
using System.Threading;
using System.Threading.Tasks;
using System.IO;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Reflection;
using Newtonsoft.Json;
using HatFramework;

namespace Hat
{
    // Auxiliary class for JSON data
    public class TestJson
    {
        public int userId { get; set; }
        public int id { get; set; }
        public string title { get; set; }
        public string body { get; set; }
    }

    public class ExampleTest3
    {
        // The main variable for the autotest operation
        Tester tester;

        // The main input function is declared as a Task. This is important!
(the autotest starts working with this function)
        public async Task Main(Form browserWindow)
        {
            // Initialization of the main variable and additional functions
            tester = new Tester(browserWindow);

            await setUp();
            await test();
            await tearDown();
        }

        // The function of getting started with the autotest
        public async Task setUp()
        {
            await tester.BrowserFullScreenAsync(); // Sets the browser
resolution to full screen
        }

        // The function of performing autotest actions
        public async Task test()
        {
            await tester.TestBeginAsync();
```

```
            await
tester.GoToUrlAsync("https://jsonplaceholder.typicode.com", 5);
            string result = await
tester.RestGetAsync(@"https://jsonplaceholder.typicode.com/posts/1/",
TimeSpan.FromDays(1), "UTF-8");
            TestJson dataJson =
JsonConvert.DeserializeObject<TestJson>(result);
            tester.AssertEqualsAsync("1", dataJson.userId.ToString());
            await tester.TestEndAsync();
        }

        // Autotest shutdown function
        public async Task tearDown()
        {
            // await tester.BrowserCloseAsync(); in this case, we don't need
the browser to close
        }
    }
}
```

the second file: ExampleTest4.cs

File: ExampleTest4.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Windows.Forms;
using System.Threading;
using System.Threading.Tasks;
using System.IO;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Reflection;
using Newtonsoft.Json;
using HatFramework;

namespace Hat
{
    public class ExampleTest4
    {
        // The main variable for the autotest operation
        Tester tester;

        // The main input function is declared as a Task. This is important!
(the autotest starts working with this function)
        public async Task Main(Form browserWindow)
        {
            // Initialization of the main variable and additional functions
            tester = new Tester(browserWindow);

            await setUp();
            await test();
```

```
            await tearDown();
        }

        // The function of getting started with the autotest
        public async Task setUp()
        {
            await tester.BrowserFullScreenAsync(); // Sets the browser
resolution to full screen
        }

        // The function of performing autotest actions
        public async Task test()
        {
            await tester.TestBeginAsync();
            await
tester.GoToUrlAsync("https://somovstudio.github.io/test_eng.html", 5);
            string script = @"(function(){ var element =
document.getElementsByTagName('h2')[0]; return element.outerText; }());";
            string actual = await tester.ExecuteJavaScriptAsync(script);
            string expected = "Authorization";
            await tester.AssertEqualsAsync(expected, actual);
            await tester.TestEndAsync();
        }

        // Autotest shutdown function
        public async Task tearDown()
        {
            // await tester.BrowserCloseAsync(); in this case, we don't need
the browser to close
        }
    }
}
```

We have created two autotests "ExampleTest3.cs" and "ExampleTest4.cs" which are located in the "group1" folder.
Now you need to create a combining autotest GroupTests1.cs that will trigger other autotests.

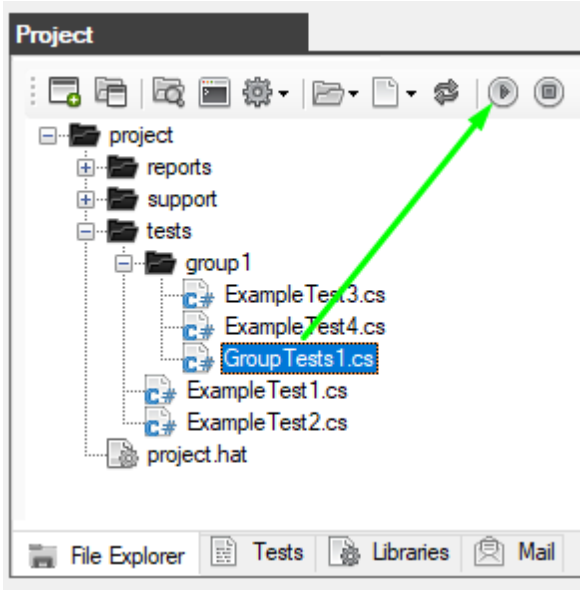
File: GroupTests1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Windows.Forms;
using System.Threading;
using System.Threading.Tasks;
using System.IO;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Reflection;
using Newtonsoft.Json;
using HatFramework;
```

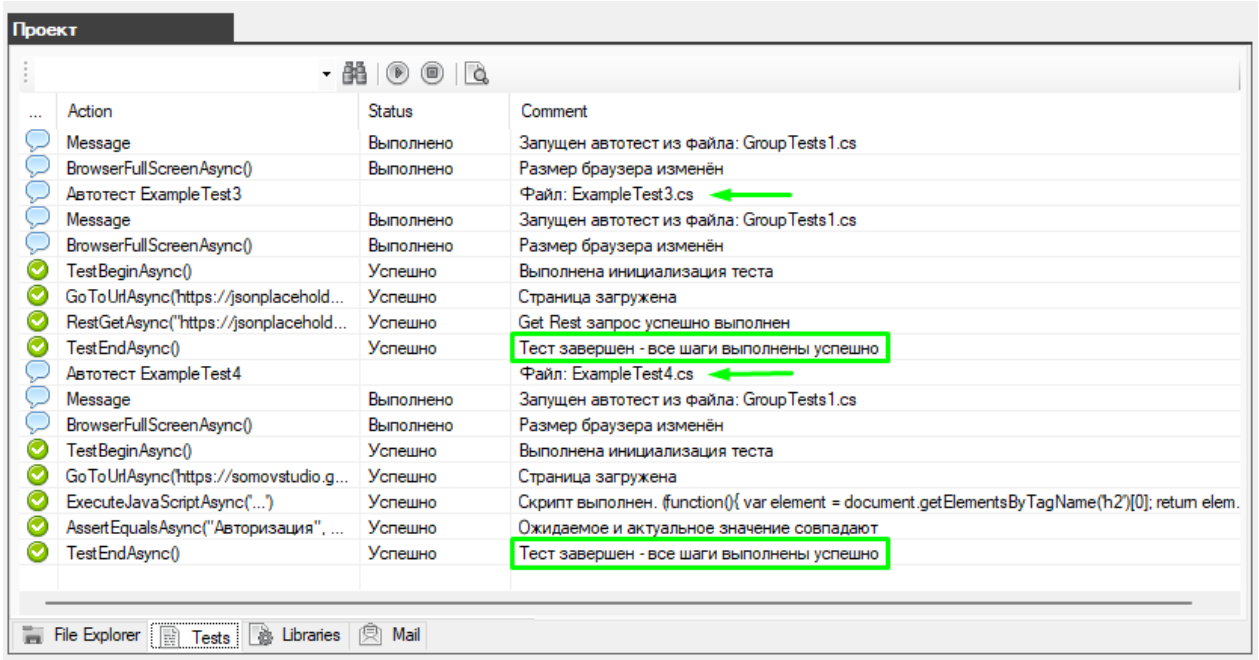
```
namespace Hat
{
    public class GroupTests1
    {
        // The main variable for the autotest operation
        Tester tester;

        // The main input function (the autotest starts with this function)
        public async void Main(Form browserWindow)
        {
            tester = new Tester(browserWindow); // Initializing the main
variable
            await setUp(); // The function of getting started with the
autotest
            await test1(browserWindow); // The function that calls the
autotest from the script ExampleTest3.cs
            await test2(browserWindow); // The function that calls the
autotest from the script ExampleTest4.cs
            await tearDown(); // Autotest shutdown function
        }

        public async Task setUp()
        {
            await tester.BrowserFullScreenAsync();
        }

        // This function runs the autotest from the script ExampleTest3.cs
        public async Task test1(Form browserWindow)
        {
            tester.SendMessage("Автотест ExampleTest3", "", "Файл:
ExampleTest3.cs", Tester.IMAGE_STATUS_MESSAGE);
            ExampleTest3 exampleTest3 = new ExampleTest3(); // An autotest
variable is being created
            await exampleTest3.Main(browserWindow); // The autotest is
started
        }

        // This function runs the autotest from the script ExampleTest4.cs
        public async Task test2(Form browserWindow)
        {
            tester.SendMessage("Автотест ExampleTest4", "", "Файл:
ExampleTest4.cs", Tester.IMAGE_STATUS_MESSAGE);
            ExampleTest4 exampleTest4 = new ExampleTest4(); // An autotest
variable is being created
            await exampleTest4.Main(browserWindow); // The autotest is
started
        }

        // Autotest shutdown function
        public async Task tearDown()
        {
            await tester.BrowserCloseAsync(); // Closes the browser
        }
    }
}
```

As you can see in the ExampleTest3.CS and ExampleTest4.cs autotest will be called sequentially in the test1 and test2 methods.

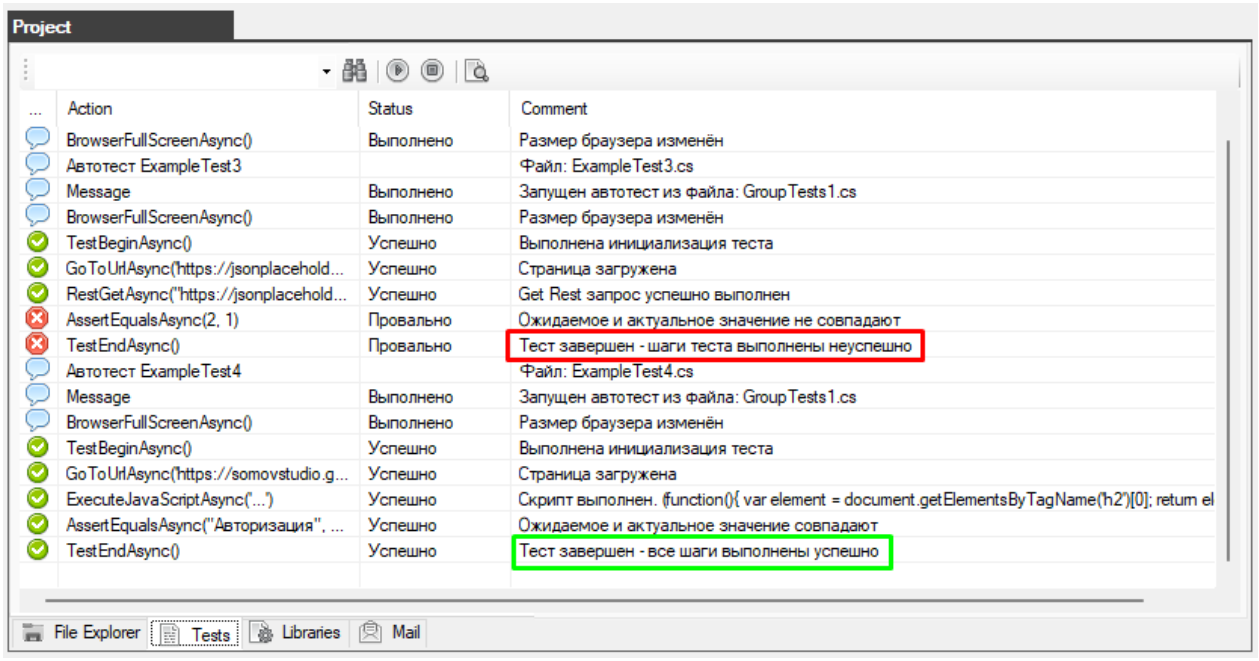
Run the GroupTests1.cs group autotest



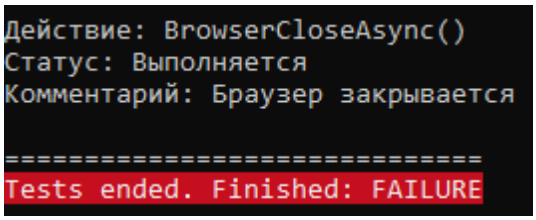
The result on the "Test" tab will reflect the entire progress of the sequential execution of all autotests step by step.



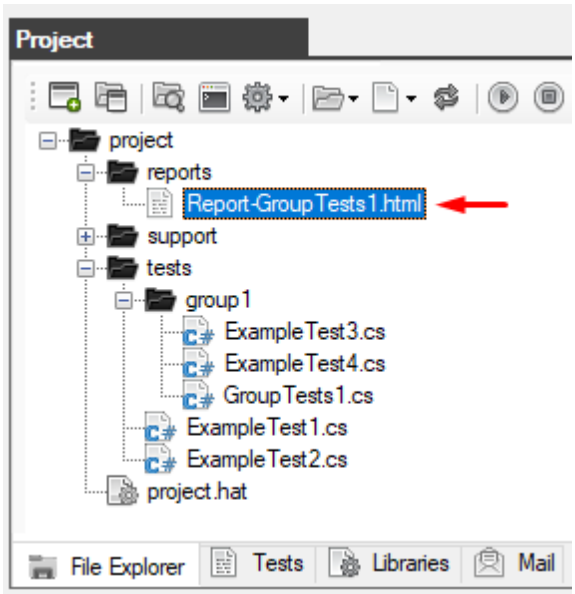
If any of the autotests fails, the check will continue anyway until all the autotests are completed

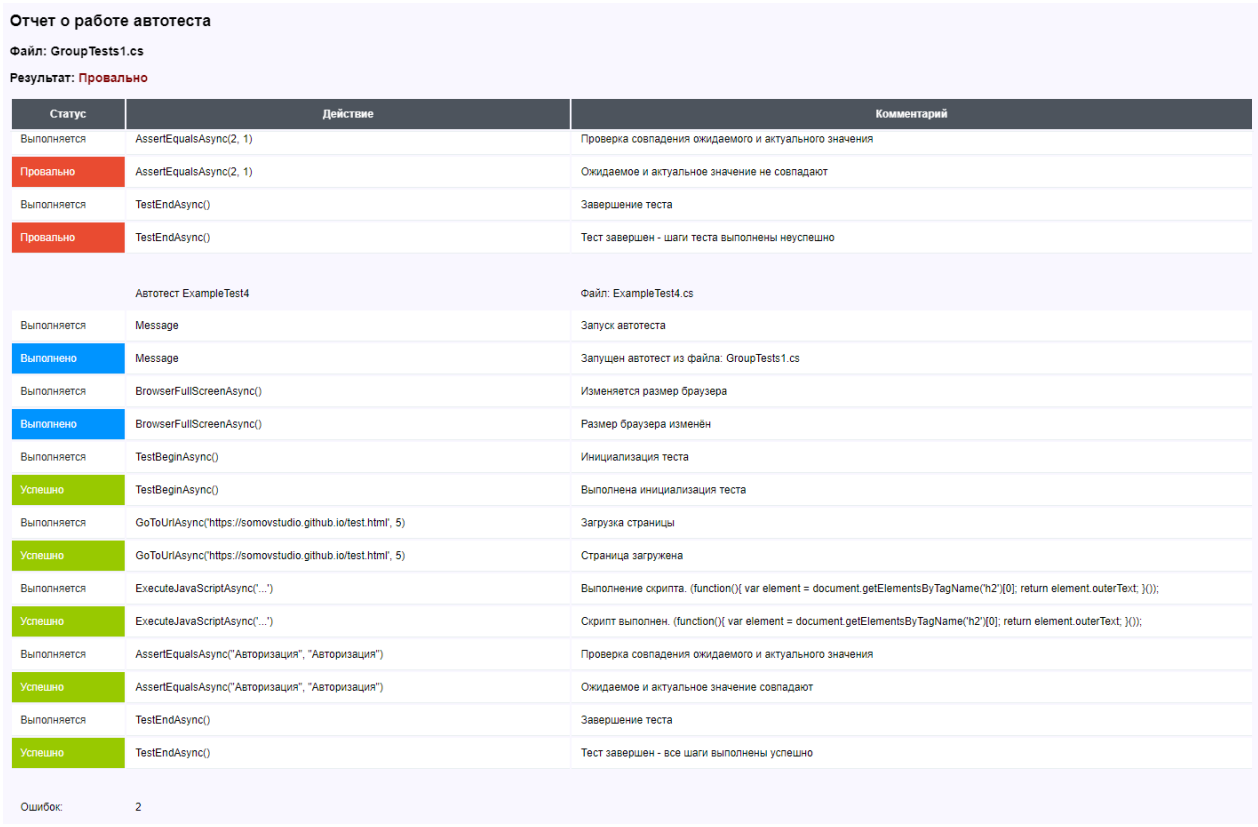


at the same time, the overall result of the GroupTests1.cs group autotest will be considered a failure.



a report on the complete completion of all tests will also be created





Отчет о работе автотеста

Файл: GroupTests1.cs

Результат: Провально

Статус	Действие	Комментарий
Выполняется	AssertEqualsAsync(2, 1)	Проверка совпадения ожидаемого и актуального значения
Провально	AssertEqualsAsync(2, 1)	Ожидаемое и актуальное значение не совпадают
Выполняется	TestEndAsync()	Завершение теста
Провально	TestEndAsync()	Тест завершен - шаги теста выполнены неуспешно
	Автотест ExampleTest4	Файл: ExampleTest4.cs
Выполняется	Message	Запуск автотеста
Выполнено	Message	Запущен автотест из файла: GroupTests1.cs
Выполняется	BrowserFullScreenAsync()	Изменяется размер браузера
Выполнено	BrowserFullScreenAsync()	Размер браузера изменён
Выполняется	TestBeginAsync()	Инициализация теста
Успешно	TestBeginAsync()	Выполнена инициализация теста
Выполняется	GoToUrlAsync('https://somovstudio.github.io/test.html', 5)	Загрузка страницы
Успешно	GoToUrlAsync('https://somovstudio.github.io/test.html', 5)	Страница загружена
Выполняется	ExecuteJavaScriptAsync('...')	Выполнение скрипта. (function(){ var element = document.getElementsByTagName('h2')[0]; return element.outerText; }());
Успешно	ExecuteJavaScriptAsync('...')	Скрипт выполнен. (function(){ var element = document.getElementsByTagName('h2')[0]; return element.outerText; }());
Выполняется	AssertEqualsAsync("Авторизация", "Авторизация")	Проверка совпадения ожидаемого и актуального значения
Успешно	AssertEqualsAsync("Авторизация", "Авторизация")	Ожидаемое и актуальное значение совпадают
Выполняется	TestEndAsync()	Завершение теста
Успешно	TestEndAsync()	Тест завершен - все шаги выполнены успешно
Ошибок:	2	

Executing JavaScript code

Executing JavaScript code

A special ExecuteJavaScriptAsync method has been added to HatFramework to execute JavaScript code.

First, you need to declare a text variable and describe in it the script that you want to execute.

```
string script = @"(function(){
var element = document.getElementsByTagName('h2')[0];
return element.outerText;
}());";
```

Now, using the ExecuteJavaScriptAsync method, we will execute the above script and get the result.

```
string actual = await tester.ExecuteJavaScriptAsync(script);
```

In this example, you will get the text from the H2 element from the page where this script was called.

Full example:

File: ExampleTest4.cs

```
using System;
```

```
using System.Collections.Generic;
using System.ComponentModel;
using System.Windows.Forms;
using System.Threading;
using System.Threading.Tasks;
using System.IO;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Reflection;
using Newtonsoft.Json;
using HatFramework;

namespace Hat
{
    public class ExampleTest4
    {
        Tester tester; // The main variable for the autotest operation

        // The main input function (the autotest starts with this function)
        public async void Main(Form browserWindow)
        {
            tester = new Tester(browserWindow); // Initializing the main
variable

            await setUp(); // The function of getting started with the
autotest
            await test(); // The function of performing test actions
            await tearDown(); // Autotest shutdown function
        }

        public async Task setUp()
        {
            await tester.BrowserFullScreenAsync(); // Sets the browser
resolution to full screen
        }

        public async Task test()
        {
            await tester.TestBeginAsync(); // The beginning of the execution
of actions
            await
tester.GoToUrlAsync("https://somovstudio.github.io/test_eng.html", 5); //
Loading the page at the specified address

            // Declaring a variable with a JavaScript script description
            string script = @"(function(){
                  var element = document.getElementsByTagName('h2')[0];
                  return element.outerText;
                  }());";

            // Script Execution
            string actual = await tester.ExecuteJavaScriptAsync(script);
```

```
            // Checking the result
            string expected = "Authorization";
            await tester.AssertEqualsAsync(expected, actual); // The
expected and actual data must match

            await tester.TestEndAsync(); // Completing actions
        }

        public async Task tearDown()
        {
            await tester.BrowserCloseAsync(); // Closes the browser
        }
    }
}
```

Examples of JavaScript and jQuery scripts for the method ExecuteJavaScriptAsync :

- focus on the input element and data entry

```
var element = document.evaluate("//input[@id='phone']", document, null,
XPathResult.FIRST_ORDERED_NODE_TYPE,  null).singleNodeValue;
element.focus();
element.value = '9999999999';
element.setAttribute("value",  "9999999999");
```

- focus on the input element and insert data

```
document.getElementById("phone").focus();
document.execCommand("insertHTML",  false,  "9999999999");
```

- entering data into input and calling events

```
var element = document.evaluate("//input[@id='phone']", document, null,
XPathResult.FIRST_ORDERED_NODE_TYPE,  null).singleNodeValue;
element.value = '9999999999';
element.dispatchEvent(new KeyboardEvent('keydown', { bubbles: true }));
element.dispatchEvent(new KeyboardEvent('keypress', { bubbles: true }));
element.dispatchEvent(new KeyboardEvent('keyup', { bubbles: true }));
element.dispatchEvent(new Event('input', { bubbles: true }));
element.dispatchEvent(new Event('change', { bubbles: true }));
```

- highlight the value in the input

```
var element = document.evaluate("//input[@id='phone']", document, null,
XPathResult.FIRST_ORDERED_NODE_TYPE,  null).singleNodeValue;
element.select();
```

- mouse click

```
var clickEvent = new MouseEvent("click", { "view": window, "bubbles": true,
"cancelable": false });
var element = document.evaluate("//input[@id='phone']", document, null,
XPathResult.FIRST_ORDERED_NODE_TYPE,  null).singleNodeValue;
element.dispatchEvent(clickEvent);

var theEvent = document.createEvent("MouseEvent");
theEvent.initMouseEvent("click", true, true, window, 0, 0, 0, 0, 0, false,
```

```
    false, false, false, 0, null);
    element.dispatchEvent(theEvent);
```

- using jQuery to click a button (if the library is connected on the page under test)

```
    string script = @"$(function() {
    console.log('autotest JQUERY');
    $('#btn-send > div.btn > a').click();
    });";
```


Processing Json data using Newtonsoft

Processing Json data using Newtonsoft

The Newtonsoft.Json library has been added to work with Json data
The official page: https://www.newtonsoft.com/json

As an example, we use a Rest request to the address
https://jsonplaceholder.typicode.com/posts/1/
to get test data in Json format:

```
{ "userId": 1, "id": 1, "title": "sunt aut facere repellat provident occaecati
excepturi optio reprehenderit", "body": "quia et suscipit\nsuscipit recusandae
consequuntur expedita et cum\nreprehenderit molestiae ut ut quas
totam\nnostrum rerum est autem sunt rem eveniet architecto" }
```

Next, with the help of Newtonsoft.Json processing of the received result.
First, you need to describe a separate class that lists the fields that are present in the Json

```
    public class TestJson
    {
        public int userId { get; set; }
        public int id { get; set; }
        public string title { get; set; }
        public string body { get; set; }
    }
```

We execute a Rest request and get the Json data

```
    string result = await
tester.RestGetAsync(@"https://jsonplaceholder.typicode.com/posts/1/",
TimeSpan.FromDays(1), "UTF-8");
```

We process the received data and output them

```
    TestJson dataJson = JsonConvert.DeserializeObject<TestJson>(result);
    tester.ConsoleMsg("UserID: " + dataJson.userId.ToString());
```

Full example:

File: ExampleTest3.cs

```
using System;
```

```
using System.Collections.Generic;
using System.ComponentModel;
using System.Windows.Forms;
using System.Threading;
using System.Threading.Tasks;
using System.IO;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Reflection;
using Newtonsoft.Json;
using HatFramework;

namespace Hat
{
    // Auxiliary class for JSON data
    public class TestJson
    {
        public int userId { get; set; }
        public int id { get; set; }
        public string title { get; set; }
        public string body { get; set; }
    }

    public class ExampleTest3
    {
        Tester tester; // The main variable for the autotest operation

        // The main input function (the autotest starts with this function)
        public async void Main(Form browserWindow)
        {
            tester = new Tester(browserWindow); // Initializing the main
variable

            await setUp(); // Функция начала работы с автотестом
            await test(); // Функция выполнения действий теста
            await tearDown(); // Функция завершения работы с автотестом
        }

        public async Task setUp()
        {
            await tester.BrowserFullScreenAsync(); // Sets the browser
resolution to full screen
        }

        public async Task test()
        {
            await tester.TestBeginAsync(); // The beginning of the execution
of actions
            await
tester.GoToUrlAsync("https://jsonplaceholder.typicode.com", 5); // Loading
the page at the specified address

            // Executes a Rest request to the API at the specified URL and
```

```
gets the result in JSON format
            string result = await
tester.RestGetAsync(@"https://jsonplaceholder.typicode.com/posts/1/",
TimeSpan.FromDays(1), "UTF-8");
            tester.ConsoleMsg(result); // Outputs the result to the console

            // Fetches data from the received JSON
            TestJson dataJson =
JsonConvert.DeserializeObject<TestJson>(result);

            // Outputs data to the console
            tester.ConsoleMsg("UserID: " + dataJson.userId.ToString());
            tester.ConsoleMsg("ID: " + dataJson.id.ToString());
            tester.ConsoleMsg("Title: " + dataJson.title.ToString());
            tester.ConsoleMsg("Body: " + dataJson.body.ToString());

            await tester.TestEndAsync(); // Completing actions
        }

        public async Task tearDown()
        {
            await tester.BrowserCloseAsync(); // Closes браузер
        }
    }
}
```


Processing XML data for checking the Sitemap

Processing XML data for checking the Sitemap

The following libraries are used to work with XML

```
using System.Collections;
using System.Xml;
```

Let's describe the ReadSitemapXML auxiliary method for reading data from the site map sitemap.xml
This method can be described in the Steps class, but in order for it to be available in any class, it is better to describe it in the Helper.
The blank of the ReadSitemapXML method is closed when reading a link from a file sitemap.xml and as a result, return the list of read links.

File: Helper.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Windows.Forms;
using System.Threading;
using System.Threading.Tasks;
using System.IO;
using System.Data;
using System.Drawing;
using System.Linq;
```

```
using System.Text;
using System.Text.RegularExpressions;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Reflection;
using Newtonsoft.Json;
using HatFramework;
using System.Collections;
using System.Xml;

namespace Hat
{
    public static class Helper
    {
        /* Method: reading the XML file of the site map */
        public static async Task<List<string>> ReadSitemapXML(Tester tester,
string filename, bool useragent)
        {
            // Declaring a variable whose type is a list (a list for storing
links)
            List<string> list = new List<string>();

            // Declaring a variable that is an error flag (the flag is
disabled by default)
            bool errors = false;

            try
            {
                // Setting up the system connection parameters
     ServicePointManager.Expect100Continue = true;
                ServicePointManager.SecurityProtocol =
SecurityProtocolType.Tls12;
                ServicePointManager.ServerCertificateValidationCallback =
delegate { return true; };

                // We are reading the xml file
                XmlDocument xDoc;
                if (useragent == false)
                {
                    xDoc = new XmlDocument();
                    xDoc.Load(filename);
                }
                else
                {
                    WebClient client = new WebClient();
                    client.Headers["User-Agent"] = await
tester.BrowserGetUserAgentAsync();
                    client.Headers["Accept"] =
"text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8";
                    string data = client.DownloadString(filename);

                    xDoc = new XmlDocument();
                    xDoc.LoadXml(data);
                }

                // Reading the elements from the resulting xml file
                XmlElement xRoot = xDoc.DocumentElement;
                foreach (XmlNode xnode in xRoot)
```

```
                {
                    for (int j = 0; j <= xnode.ChildNodes.Count; j++)
                    {
                        if (xnode.ChildNodes[j].Name == "loc")
                        {
                            // We save the received link to the list
                            list.Add(xnode.ChildNodes[j].InnerText);
                            break;
                        }
                    }
                }
            }
            catch (Exception ex)
            {
                errors = true; // enabling the error flag
            }

            // Checking the error flag (if the flag is set to true, then a
failure has occurred and the test will be aborted
            await tester.AssertFalseAsync(errors);

            return list; // returning the list of links
        }
    }
}
```

Now let's describe an autotest in which we will check the site map

File: Sitemap_Test.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Windows.Forms;
using System.Threading;
using System.Threading.Tasks;
using System.IO;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Reflection;
using Newtonsoft.Json;
using HatFramework;

namespace Hat
{
    public class Sitemap_Test
    {
        // The main variable for the autotest operation
        Tester tester;

        // The main input function (the autotest starts with this function)
        public async void Main(Form browserWindow)
        {
```

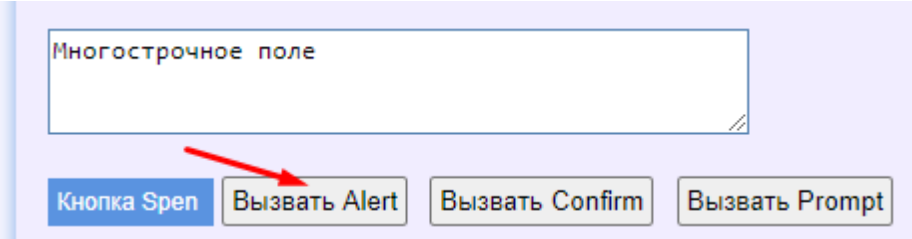
```
            // Initialization of the main variable and additional functions
            tester = new Tester(browserWindow);
            await setUp();
            await test();
            await tearDown();
        }

        // The function of getting started with the autotest
        public async Task setUp()
        {
            await tester.BrowserFullScreenAsync(); // Sets the browser
resolution to full screen
        }

        // The function of performing autotest actions
        public async Task test()
        {
            await tester.TestBeginAsync(); // The beginning of the execution
of actions

            // Loading the page at the specified address
            await
tester.GoToUrlAsync("https://somovstudio.github.io/sitemap.xml", 25);

            // Reading links from a file sitemap.xml
            List<string> list = await Helper.ReadSitemapXML(tester,
"https://somovstudio.github.io/sitemap.xml", false);

            // Processing the list of links
            int status = 0;
            foreach (string url in list)
            {
                // Getting the page status (200 - correct, 404 - no page,
502 - the server is not responding, etc.)
                status = await tester.RestGetStatusCodeAsync(url);
                // Output the result to the console
                tester.ConsoleMsg("Status: " + status.ToString() + " | Page:
" + url);
                // Checking the correctness of the result
                await tester.AssertEqualsAsync(200, status);
            }

            await tester.TestEndAsync(); // Completion of actions
        }

        // Функция завершения работы с автотестом
        public async Task tearDown()
        {
            await tester.BrowserCloseAsync(); // Closes the browser
        }
    }
}
```

Alert, prompt and confirm dialog windows

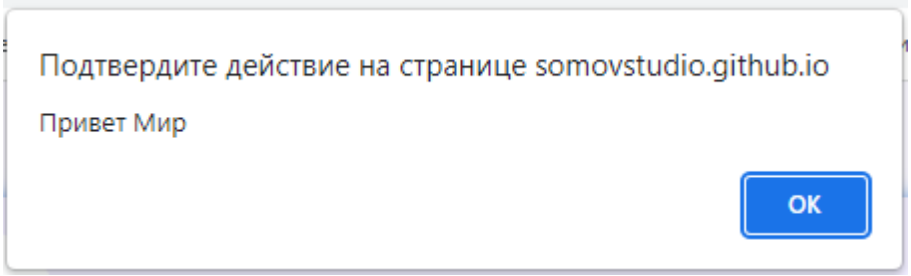
Alert, prompt and confirm dialog windows

The browser has disabled support for the alert, prompt and confirm dialog windows.
So that the code is executed ignoring calls to these dialog windows.

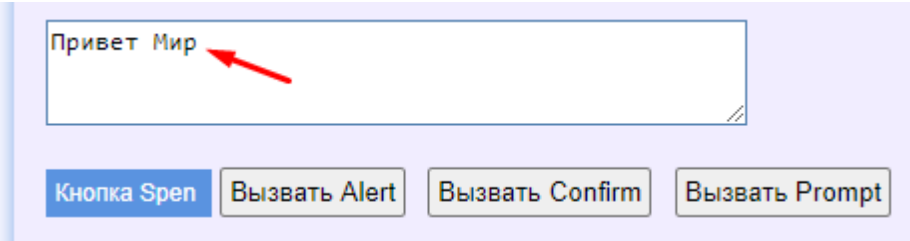
How does it work? Let's take a test page as an example
https://somovstudio.github.io/test2.html
Click on the button "Вызвать Alert"



a dialog window opens

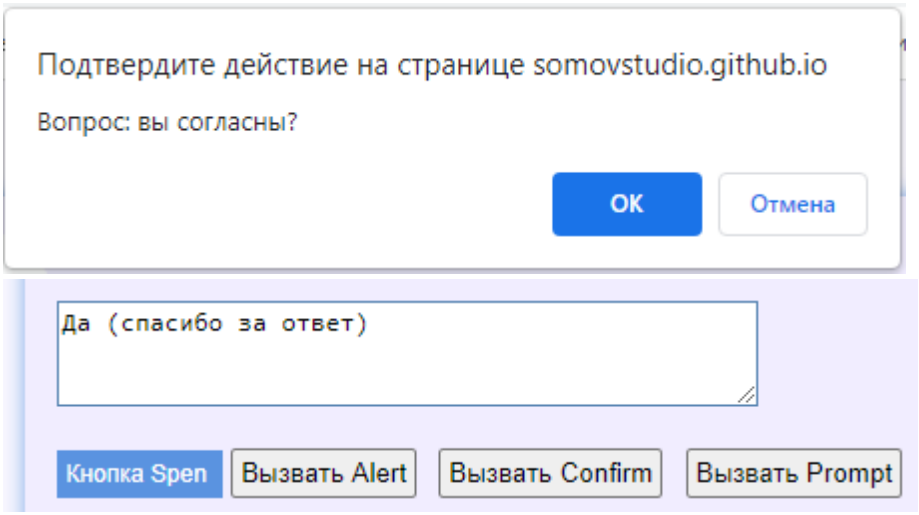


by clicking OK, we will see the message "Привет Мир" in the field

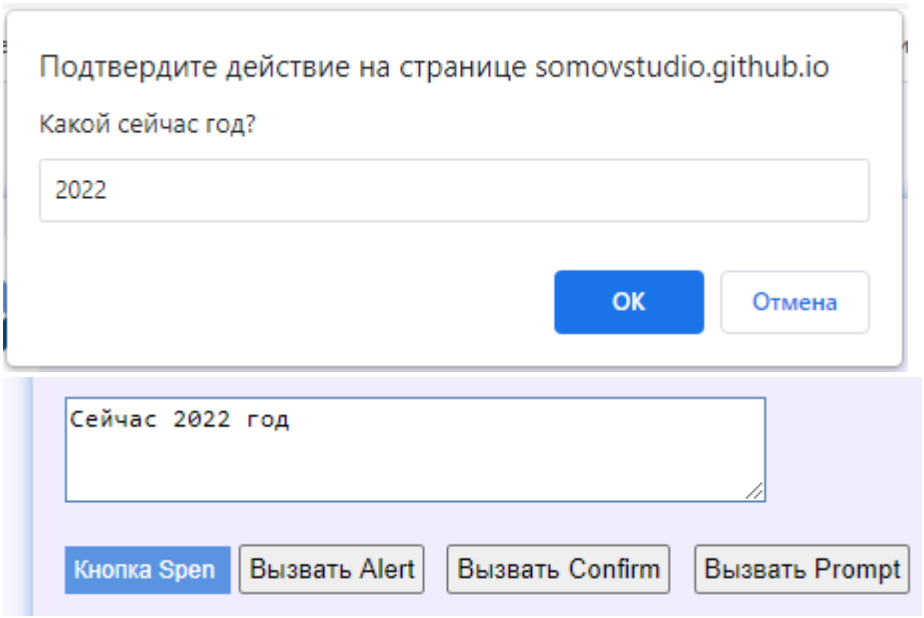


That is, the message "Привет Мир" will be displayed in the field only after clicking OK in the Alert dialog box.

If you press the button "Вызвать Confirm" a question will be asked and the answer will be displayed in the field after clicking on "OK" (or "Cancel")



If you press the button "Вызвать Confirm" a question will be asked and the answer will be displayed in the field after entering the number and clicking OK



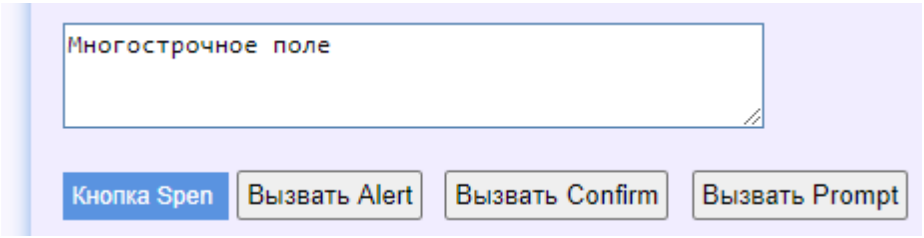
Now let's do the same with autotest hands

File: ExampleTest5.cs

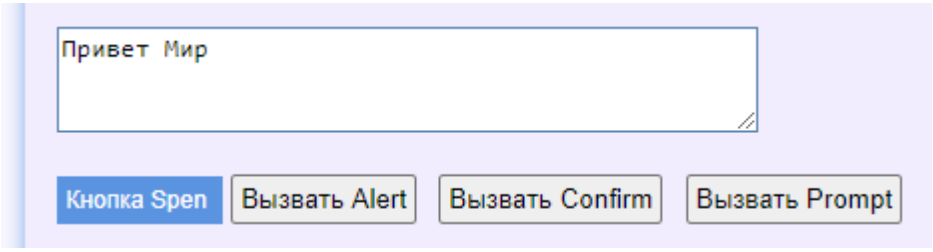
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Windows.Forms;
using System.Threading;
using System.Threading.Tasks;
using System.IO;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Net;
```

```
using System.Net.Http;
using System.Net.Http.Headers;
using System.Reflection;
using Newtonsoft.Json;
using HatFramework;

namespace Hat
{
    public class ExampleTest5
    {
        Tester tester; // The main variable for the autotest operation

        // The main input function (the autotest starts with this function)
        public async void Main(Form browserWindow)
        {
            tester = new Tester(browserWindow); // Initializing the main
variable

            await setUp(); // The function of getting started with the
autotest
            await test(); // The function of performing test actions
            await tearDown(); // Autotest shutdown function
        }

        public async Task setUp()
        {
            await tester.BrowserFullScreenAsync(); // Sets the browser
resolution to full screen
        }

        public async Task test()
        {
            await tester.TestBeginAsync(); // The beginning of the execution
of actions
            await
tester.GoToUrlAsync("https://somovstudio.github.io/test2.html", 5); //
Loading the page at the specified address
            await tester.WaitAsync(2); // Waiting for 2 seconds
            await tester.ClickElementByIdAsync("btnAlert"); // Pressing the
button
            await tester.WaitAsync(2); // Waiting for 2 seconds
            await tester.ClickElementByIdAsync("btnConfirm"); // Pressing
the button
            await tester.WaitAsync(2); // Waiting for 2 seconds
            await tester.ClickElementByIdAsync("btnPrompt"); // Pressing the
button
            await tester.TestEndAsync(); // Completing actions
        }

        public async Task tearDown()
        {
            await tester.BrowserCloseAsync(); // Closes the browser
        }
    }
}
```

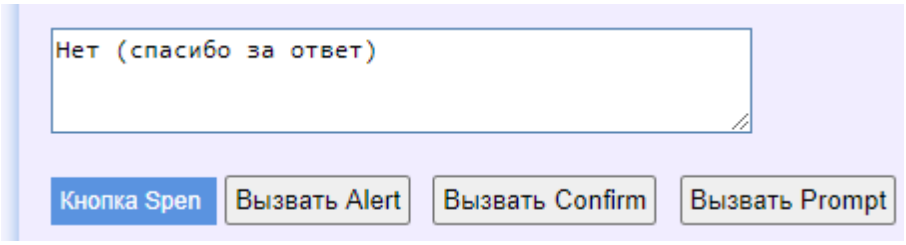
After running the autotest, we will see that the dialog boxes do not open, and the data changes in the field.
The page will load first in its default state



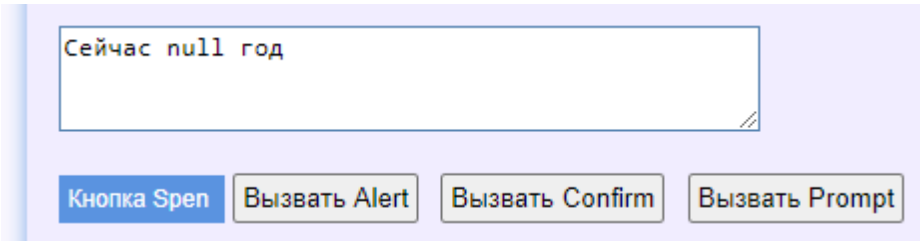
Then, autotest, click the "Call Alert" button, the dialog box will not open, and the message "Hello World" will immediately appear in the field



Then, autotest, click the "Call Confirm" button, the dialog box will not open, and the message "No (thanks for the answer)" will immediately appear in the field



Then, autotest, click the "Call Prompt" button, the dialog box will not open, and the message "Now is a null year" will immediately appear in the field

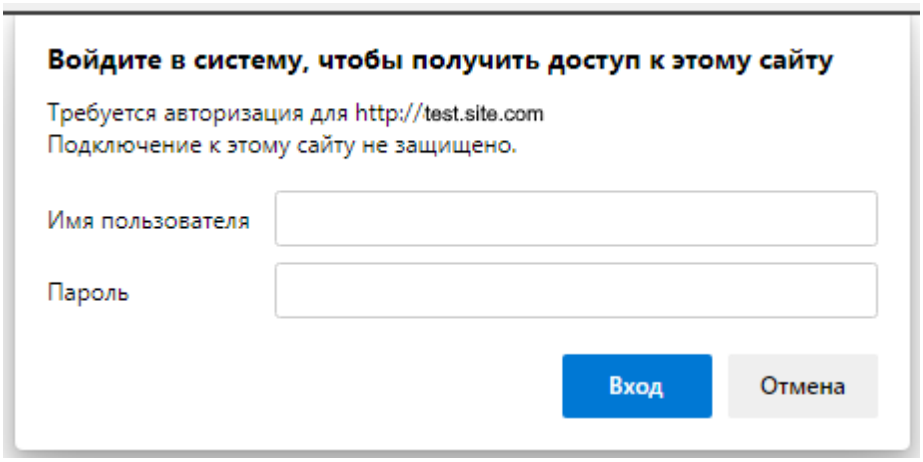


As you can see, the browser ignored the dialog boxes and displayed the default values. Thus, dialog boxes do not interfere with the execution of autotests.

Basic authorization

Basic authorization

There are cases when the page is closed with basic authorization



There are two methods for working with basic authorization:

- BrowserBasicAuthenticationAsync - performs authorization before completing the autotest steps
- GoToUrlBaseAuthAsync - navigates to the page with the username and password specified in the address URL

An example of using the method BrowserBasicAuthenticationAsync

File: ExampleTest6.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Windows.Forms;
using System.Threading;
using System.Threading.Tasks;
using System.IO;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Reflection;
using Newtonsoft.Json;
using HatFramework;

namespace Hat
{
    public class ExampleTest5
    {
        Tester tester; // The main variable for the autotest operation

        // The main input function (the autotest starts with this function)
        public async void Main(Form browserWindow)
        {
            tester = new Tester(browserWindow); // Initializing the main
variable

            await setUp(); // The function of getting started with the
autotest
```

```
            await test(); // The function of performing test actions
            await tearDown(); // Autotest shutdown function
        }

        public async Task setUp()
        {
            await tester.BrowserFullScreenAsync(); // Sets the browser
resolution to full screen
            await tester.BrowserBasicAuthenticationAsync("login",
"pass"); // Performing basic authorization
        }

        public async Task test()
        {
            await tester.TestBeginAsync(); // The beginning of the execution
of actions
            await tester.GoToUrlAsync("https://test.site.com", 5); //
Loading the page at the specified address
            await tester.TestEndAsync(); // Completing actions
        }

        public async Task tearDown()
        {
            await tester.BrowserCloseAsync(); // Closes the browser
        }
    }
}
```

An example of using the method GoToUrlBaseAuthAsync

File: ExampleTest6.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Windows.Forms;
using System.Threading;
using System.Threading.Tasks;
using System.IO;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Reflection;
using Newtonsoft.Json;
using HatFramework;

namespace Hat
{
    public class ExampleTest5
    {
        Tester tester; // The main variable for the autotest operation
        // The main input function (the autotest starts with this function)
        public async void Main(Form browserWindow)
        {
```

```
            tester = new Tester(browserWindow); // Initializing the main
variable

            await setUp(); // The function of getting started with the
autotest
            await test(); // The function of performing test actions
            await tearDown(); // Autotest shutdown function
        }

        public async Task setUp()
        {
            await tester.BrowserFullScreenAsync(); // Sets the browser
resolution to full screen
        }

        public async Task test()
        {
            await tester.TestBeginAsync(); // The beginning of the execution
of actions

            // Performing basic authorization when loading a page
            await tester.GoToUrlBaseAuthAsync("https://test.site.com",
"login", "pass", 5);

            await tester.TestEndAsync(); // Completing actions
        }

        public async Task tearDown()
        {
            await tester.BrowserCloseAsync(); // Closes the browser
        }
    }
}
```


Google analytics and yandex metrika events

Google analytics and yandex metrika events

To get GA and YM events, use special methods.

```
await tester.AssertNetworkEventsAsync(true, new string[] {
    "ec=ga_category", "ea=ga_action", "el=ga_label", "yandex_event"
});
await Helper.ClearBrowserNetworkLogs(tester); // clearing events
```

Another way to intercept the YM event is to use it **?_ym_debug=1** in the address and browser event handling.

File: ExampleTest7.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Windows.Forms;
```

```
using System.Threading;
using System.Threading.Tasks;
using System.IO;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Reflection;
using Newtonsoft.Json;
using HatFramework;

namespace Hat
{
    public class ExampleTest5
    {
        Tester tester; // The main variable for the autotest operation
        List<string> events; // The variable into which all browser events
are collected

        // The main input function (the autotest starts with this function)
        public async void Main(Form browserWindow)
        {
            tester = new Tester(browserWindow); // Initializing the main
variable

            await setUp(); // The function of getting started with the
autotest
            await test(); // The function of performing test actions
            await tearDown(); // Autotest shutdown function
        }

        // A function that listens to browser events
        private void browserConsoleEvents(object sender,
Microsoft.Web.WebView2.Core.CoreWebView2DevToolsProtocolEventReceivedEventAr
gs e)
        {
            events.Add(e.ParameterObjectAsJson);
            tester.ConsoleMsg(e.ParameterObjectAsJson);
        }

        public async Task setUp()
        {
            await tester.BrowserFullScreenAsync(); // Sets the browser
resolution to full screen

            /*
             * Let's turn on listening for browser events
             * A source: https://chromedevtools.github.io/devtools-
protocol/tot/Network/
             */
            tester.BrowserView.CoreWebView2.GetDevToolsProtocolEventReceiver
("Log.entryAdded").DevToolsProtocolEventReceived += browserConsoleEvents;
            await
tester.BrowserView.CoreWebView2.CallDevToolsProtocolMethodAsync
("Log.enable", "{}");
```

```
            tester.BrowserView.CoreWebView2.GetDevToolsProtocolEventReceiver
("Console.messageAdded").DevToolsProtocolEventReceived  +=
browserConsoleEvents;
            await
tester.BrowserView.CoreWebView2.CallDevToolsProtocolMethodAsync
("Console.enable", "{}");
            tester.BrowserView.CoreWebView2.GetDevToolsProtocolEventReceiver
("Runtime.consoleAPICalled").DevToolsProtocolEventReceived  +=
browserConsoleEvents;
            tester.BrowserView.CoreWebView2.GetDevToolsProtocolEventReceiver
("Runtime.exceptionThrown").DevToolsProtocolEventReceived  +=
browserConsoleEvents;
            await
tester.BrowserView.CoreWebView2.CallDevToolsProtocolMethodAsync
("Runtime.enable", "{}");
tester.BrowserView.CoreWebView2.GetDevToolsProtocolEventReceiver
("Network.dataReceived").DevToolsProtocolEventReceived  +=
browserConsoleEvents;
            await
tester.BrowserView.CoreWebView2.CallDevToolsProtocolMethodAsync
("Network.enable", "{}");
            tester.BrowserView.CoreWebView2.GetDevToolsProtocolEventReceiver
("Debugger.scriptParsed").DevToolsProtocolEventReceived  +=
browserConsoleEvents;
            await
tester.BrowserView.CoreWebView2.CallDevToolsProtocolMethodAsync
("Debugger.enable", "{}");
        }

        public async Task test()
        {
            await tester.TestBeginAsync(); // The beginning of the execution
of actions
            await tester.GoToUrlAsync("https://test.site.com/?_ym_debug=1",
5); // Loading a page with YM event debugging enabled

            await tester.SetValueInElementAsync(Tester.BY_XPATH,
"//input[@id='SECOND_NAME']", Helper.TestName);
            await tester.WaitAsync(1);
            await tester.SetValueInElementAsync(Tester.BY_XPATH,
"//input[@id='NAME']", Helper.TestName);
            await tester.WaitAsync(1);
            await tester.SetValueInElementAsync(Tester.BY_XPATH,
"//input[@id='PHONE']", Helper.TestPhone);
            await tester.WaitAsync(1);
            await tester.ClickElementAsync(Tester.BY_XPATH,
"//input[@id='fb_close']");
            await tester.WaitAsync(1);
            await tester.WaitVisibleElementAsync(Tester.BY_XPATH,
"//p[text() = 'Спасибо! Ваша заявка принята.']", 25);
            await tester.WaitAsync(5);

            // Checking the YM event in the browser event list
            tester.SendMessage("Message", Tester.COMPLETED, "Проверка:
Событие YM (Goal id: yandex_event)", Tester.IMAGE_STATUS_MESSAGE);
            bool checkYM = false;
            foreach(string value in events)
            {
                checkYM =value.Contains("Goal id: yandex_event");
```

```
                if (checkYM == true)
                {
                    tester.SendMessage("Message", Tester.COMPLETED,
"Сработало событие YM: " + value, Tester.IMAGE_STATUS_MESSAGE);
                    break;
                }
            }
            await tester.AssertTrueAsync(checkYM); // The result of the
check

            await tester.TestEndAsync(); // Completing actions
        }

        public async Task tearDown()
        {
            await tester.BrowserCloseAsync(); // Closes the browser
        }


    }
}
```

Thus, the autotest listens to all browser events and, after performing the actions, looks through all events in search of new ones
If the YM event is found, the test is considered successful.

Version 1.3 (not current)

Description

Starting with browser version 1.3, the following methods have been changed:

- SendMessage - the method of change
- SendMessageDebug - the method of change
- EditMessage - the method has been deleted
- EditMessageDebug - the method has been deleted
- DefineTestStop -the method of change

SendMessage [changed since version 1.3.0]

SendMessage
Description: the method displays messages in the test execution process table
When executed, action is output to the system console if status is FAILED, WARNING, "", null, otherwise only status and comment are output (without action).

Syntax: SendMessage(string action, string status, string comment)

Example:
int step = tester.SendMessage("the text of the actions", Tester.PROCESS, "the text of the comment");

/* Examples of console output */
// action comment
int step = tester.SendMessage("actions", **null**, "comment");
int step = tester.SendMessage("actions", **""**, "comment");

// Step[**progress**]: comment
int step = tester.SendMessage("actions", **Tester.PROCESS**, "comment");
// Step[**passed**]: comment
int step = tester.SendMessage("actions", **Tester.PASSED**, "comment");
// Step[**completed**]: comment
int step = tester.SendMessage("actions", **Tester.COMPLETED**, "comment");
// Step[**stopped**]: comment
int step = tester.SendMessage("actions", **Tester.STOPPED**, "comment");

// Step[**failed**]: действие комментарий
int step = tester.SendMessage("actions", **Tester.FAILED**, "comment");
// Step[**warning**]: действие комментарий
int step = tester.SendMessage("actions", **Tester.WARNING**, "comment");

SendMessageDebug [changed since version 1.3.0]

SendMessageDebug
Description: the method outputs a debugging message that can be disabled for output to a report and an email
When executed, action is output to the system console if status is FAILED, WARNING, "", null, otherwise only status and comment are output (without action).

Syntax: SendMessageDebug(string actionRus, string actionEng, string status, string commentRus, string commentEng, int image)

Example:
int step = tester.SendMessageDebug("действия", "action", Tester.PROCESS, "комментарий", "comment", Tester.IMAGE_STATUS_PROCESS);

DefineTestStop [changed since version 1.3.0]

DefineTestStop
Description: the method checks the status of the process (stopped or not)
Syntax: DefineTestStop()
Return value: bool (true or false)

Example:

```
int step = tester.SendMessage("the text of the actions", Tester.FAILED, "the text of the comment");
if (tester.DefineTestStop(step) == true) return; // so the test is stopped
```