

CSCE 5430.003

Sahi Software

Group 11- Project 3 Testing Report

Overview

Sahi is a software testing tool that provides a range of features to support testing.

General Features:

- Sahi has Simple & Powerful APIs to easily identify elements on DOM, perform mouse, keyboard, and touch actions seamlessly.
- Sahi embeds automatic waits and eliminates wait for statements and can navigate inconsistent page loads.
- Sahi Object Spy and Recorder to identify elements across devices and software. It can also work on applications that generate dynamic IDs for elements.
- Sahi implements Business friendly frameworks and uses inbuilt Business Driven Test Automation to let business analysts and non-technical professionals contribute towards test cases.
- Sahi employs automatic logging while automating applications that contains the complete information about the run. This helps testers identify the exact line where script is failing. All logs are stored in database.
- Sahi can run thousands of scripts that are packed into a suite on a single machine in parallel mode or can be distributed across machines.

Some of the notable features include:

- Record and Playback: Sahi Pro allows testers to record their interactions with the application and play them back to create automation scripts.
- Multi-Browser Support: Sahi Pro supports multiple browsers, including Chrome, Firefox, Safari, and Internet Explorer.
- Parallel Execution: Sahi Pro allows tests to be run in parallel across multiple machines, reducing the time required for testing.
- Script less Automation: Sahi Pro supports script less automation, allowing testers to create automation scripts without any programming knowledge.

Roles and Input Combinations

- Sahi is suitable for use by a range of stakeholders involved in software testing, including developers, testers, and QA engineers.
- Sahi uses control + hover combination to identify elements in DOM. Uses Sahi Script to store the user events and is an extension of JavaScript, capable of interacting with browser efficiently and perform reads on file system, access database and can call Java etc.,
- Sahi Pro offers a variety of ways to identify web elements on a page, including through traditional methods such as ID, name, and CSS selectors, as well as advanced

techniques like relation APIs (such as leftOf, under ,near, in, and rightOf) that allow testers to identify elements with respect to each other.

- Sahi is Browser and Operating System independent in most cases. Javascript is injected into webpages using proxy. In 95% of the test cases, this will work. Sahi falls back to native events, when Javascript events may not work. These require focus on test window and prevents parallel testing.
- Sahi can be used to test Web, Mobile, Desktop and SAP applications. It is used in automating test cases.

Sahi Economics

Sahi started as Open-Source project in 2005 with focus on automation of emerging web 2.0 technologies. Sahi evolved into Sahi Pro that handles automation on modern web browsers, mobiles, and desktop applications.

Sahi Pro has 4 pricing plans available for customers:

- Sahi Pro for Web
 - Works across browsers and operating systems
- Sahi Pro for Desktop
 - Works for windows desktop applications (WPF, .NET, etc.,)
 - Java based applications.
- Sahi Pro for Mobile
 - Mobile native & hybrid applications in iOS and Android
- Sahi Pro for SAP
 - SAP GUI for windows

All plans support Database, File system, REST/Web services. Each plan has user / concurrent licenses available. Sahi Pro is used by over 1000 companies globally, including Fortune 500 companies, government organizations, and startups. Some of the notable users of Sahi Pro include Accenture, Cognizant, IBM, Infosys, and Wipro. Sahi Pro offers 30 days free trial. Sahi reviews and testimonials are strong according to the official website. Product selling point is the Record and playback tool functionality which works across all major browsers regardless of their versions.

History

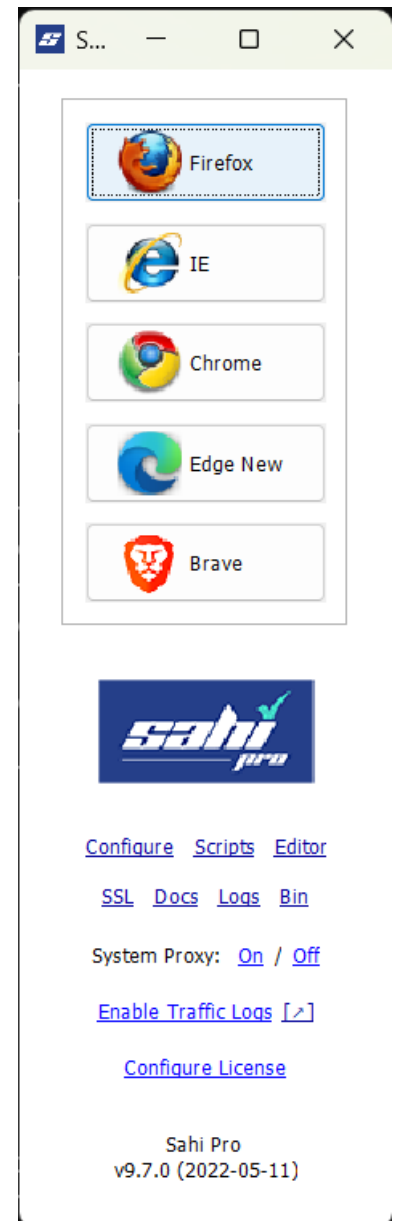
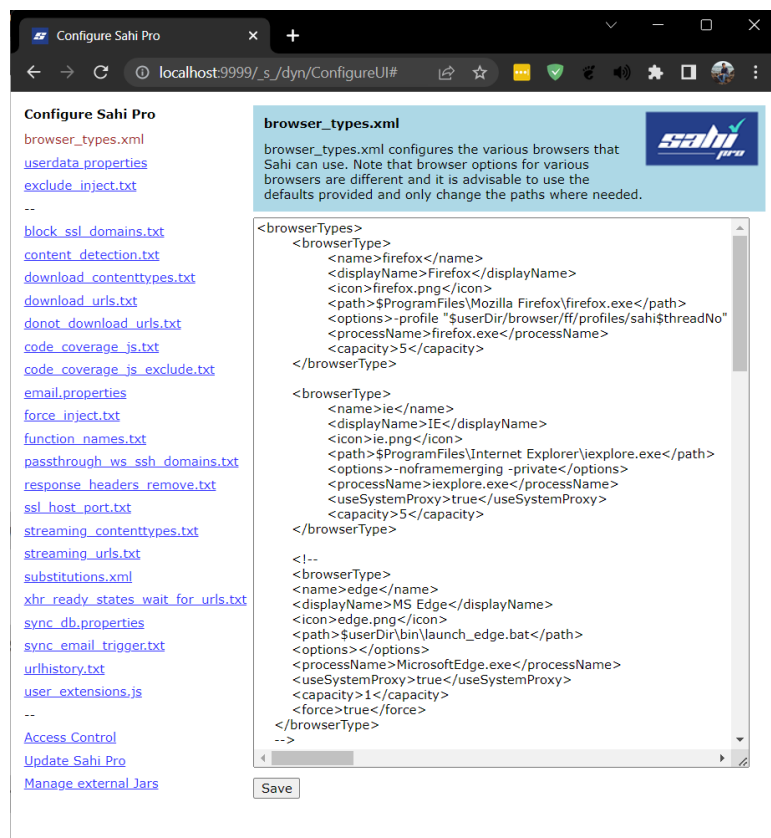
Sahi is an open-source test automation tool for web applications. It was initially created in 2005 with a specific focus on the automation of emerging web 2.0 technologies aimed at testers. Over the years, the tool has undergone significant development, and Sahi Pro is now an enterprise-grade test automation platform used by over 1000 companies worldwide.

Testing Interface

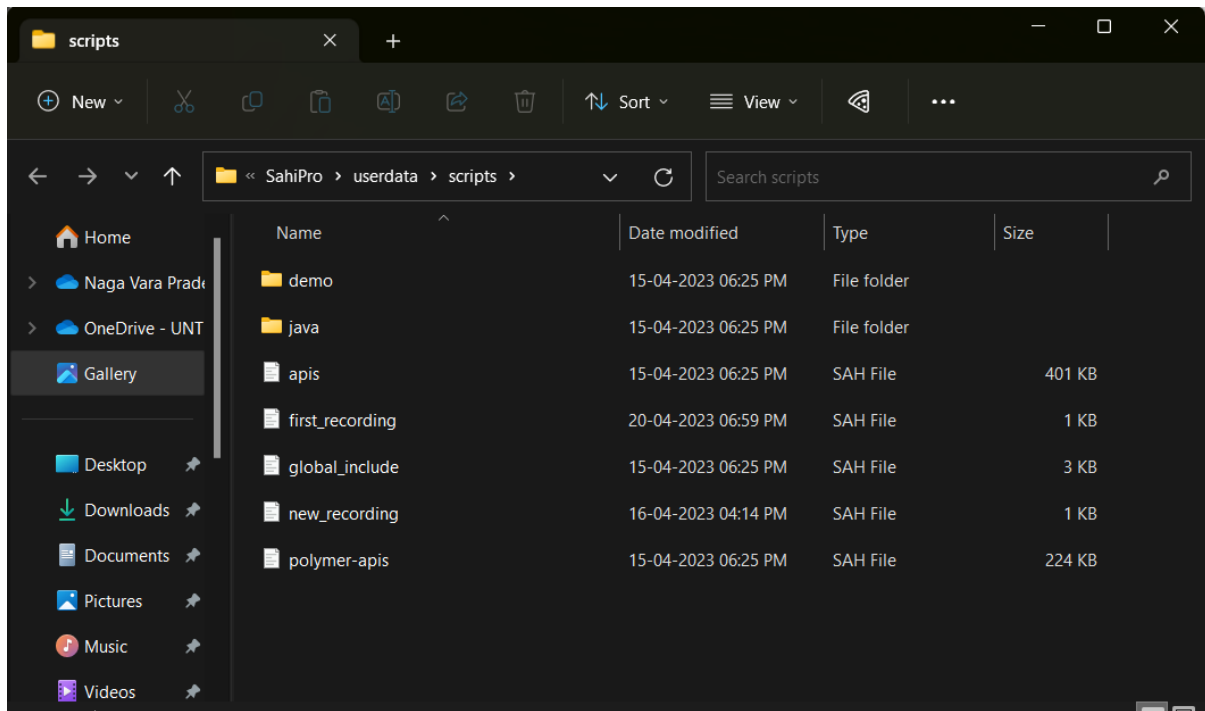
The Sahi Pro user interface is designed to be simple and intuitive. The testing interface consists of several components, including the Sahi Dashboard, Test Suite Editor, Test Runner, and Test Logs.

Sahi Dashboard: Sahi Dashboard consists of available browsers on the system and include links to configure, Scripts, script editor, SSL, Docs, Logs, Bin

Configure: Configure window contains Sahi Pro configurations like browser types, Access Control, URL history. Each configuration details are present at the top of the input window. User can click save to save the changed configuration.



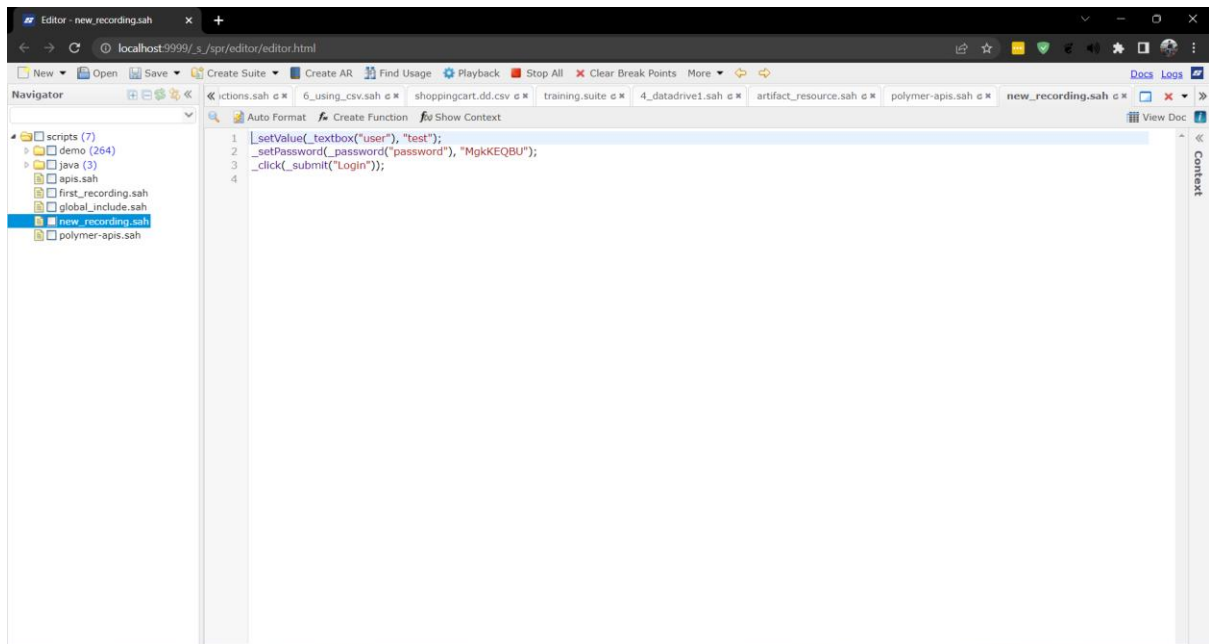
Scripts: clicking on scripts will launch file explorer on windows to show the recorded scripts stored in .sah format.



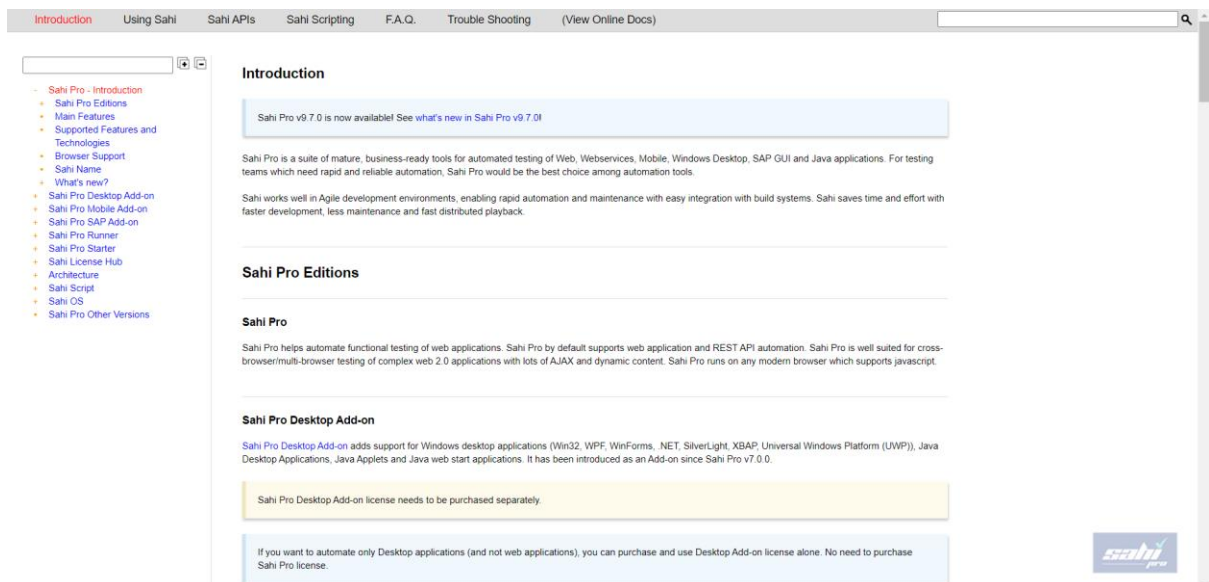
User can review the recorded steps using a text editor like Visual Studio Code

```
first_recording.sah x
1  _setValue(_textbox("user"), "test");
2  _setPassword(_password("password"), "MgkKEQBU");
3  _click(_submit("Login"));
4  _setValue(_textbox("q"), "1");
5  _setValue(_textbox("q[1]"), "12");
6  _click(_row("Python Cookbook7Rs. 350"));
7  _setValue(_textbox("q[2]"), "12");
8  _click(_div("All available books TitleIn stockCostAdd quantity to cart Core Java5Rs. 300 Ruby for Rails12Rs. 200 Python Cookbook7Rs. 350 | |"));
9  _click(_button("Add"));
10 _click(_div("All available books TitleIn stockCostAdd quantity to cart Core Java5Rs. 300 Ruby for Rails12Rs. 200 Python Cookbook7Rs. 350 | |"));
11 _click(_button("Logout"));
12 _navigateTo("http://google.com");
13 _navigateTo("http://google.com");
14
```

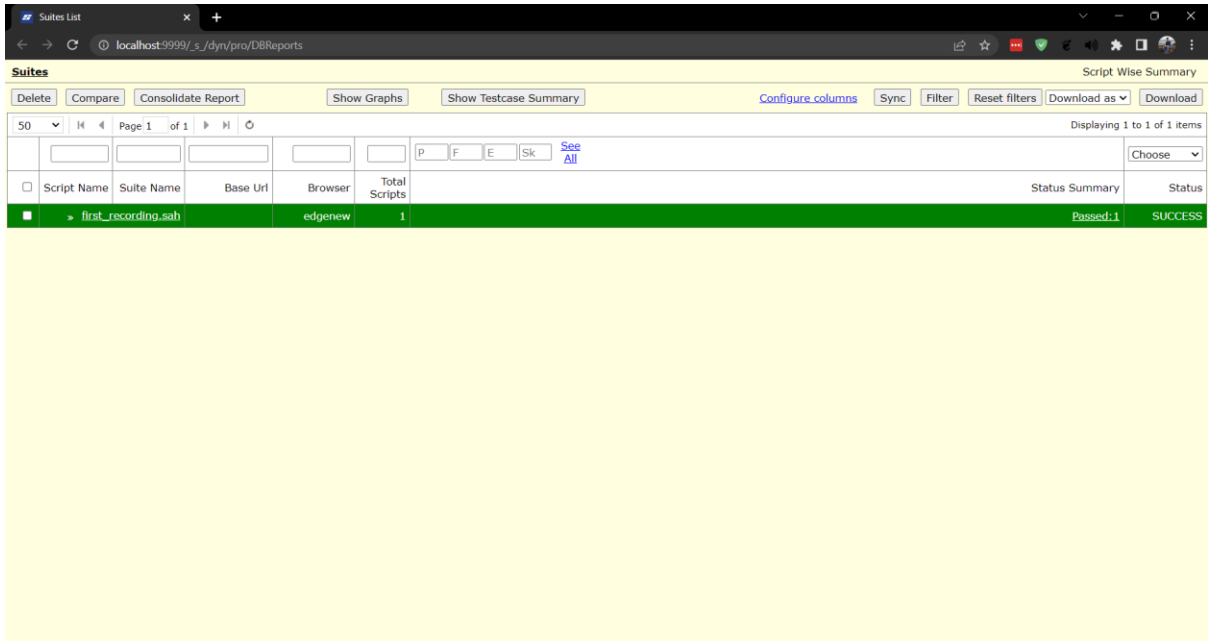
Script Editor: Clicking on Editor opens web browser and localhost:9999 is opened. Script editor contains file navigator, edit window, action buttons at the top to work on Sahi Scripts. Users can segregate script suite, playback written script, and debug sah script.



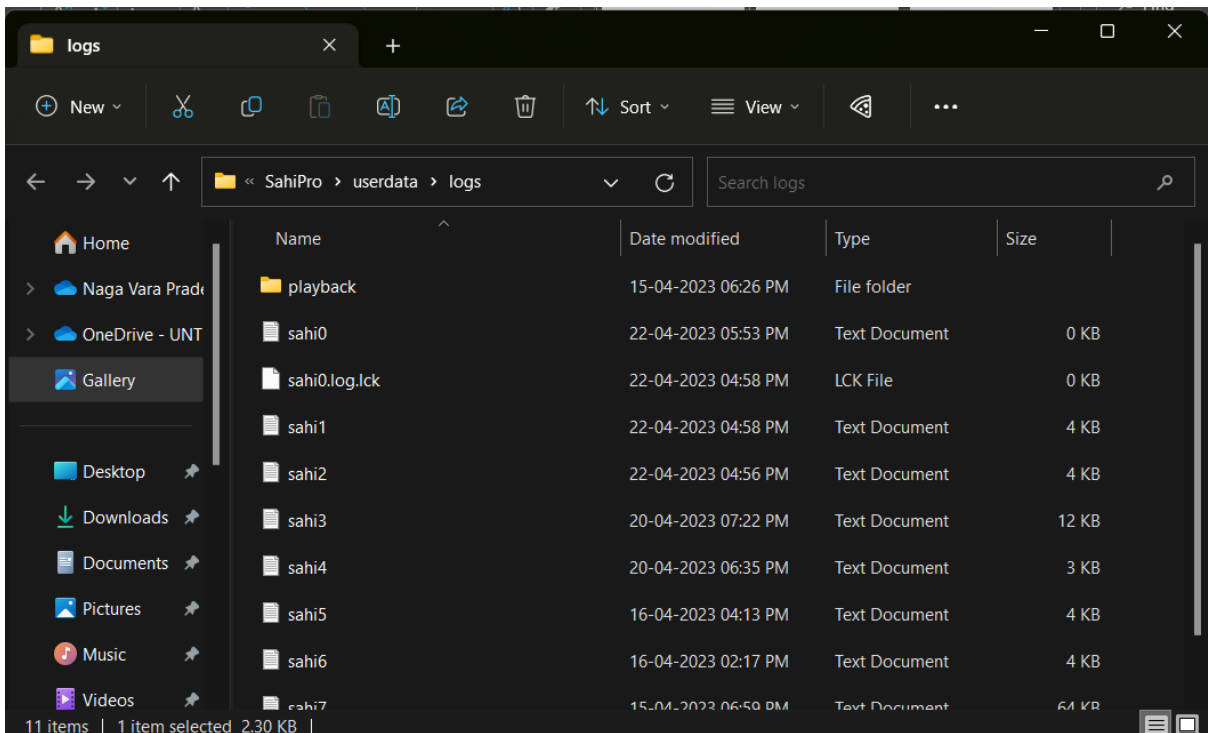
Docs opens Sahi Documentation.



Logs list all the runs completed and shows each action run time in milliseconds.

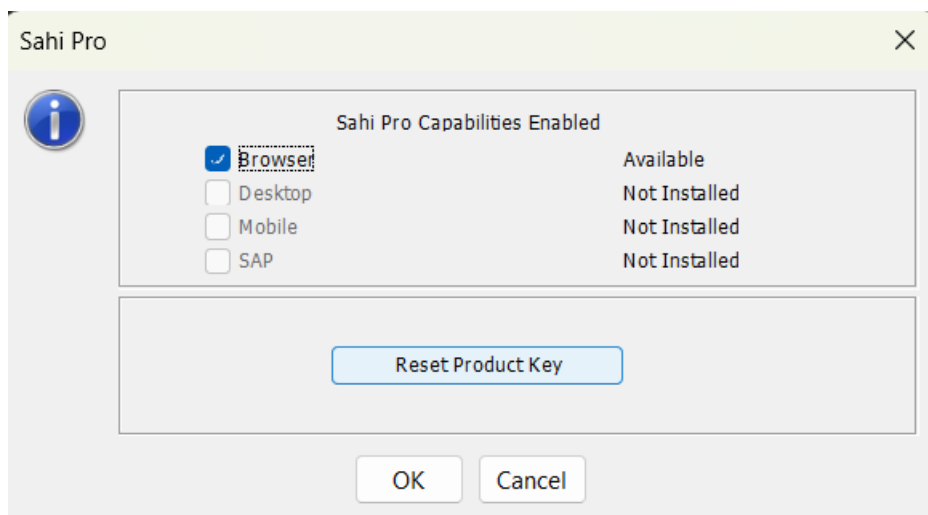


Enabling Traffic logs lists all the sites visited during the automation process.

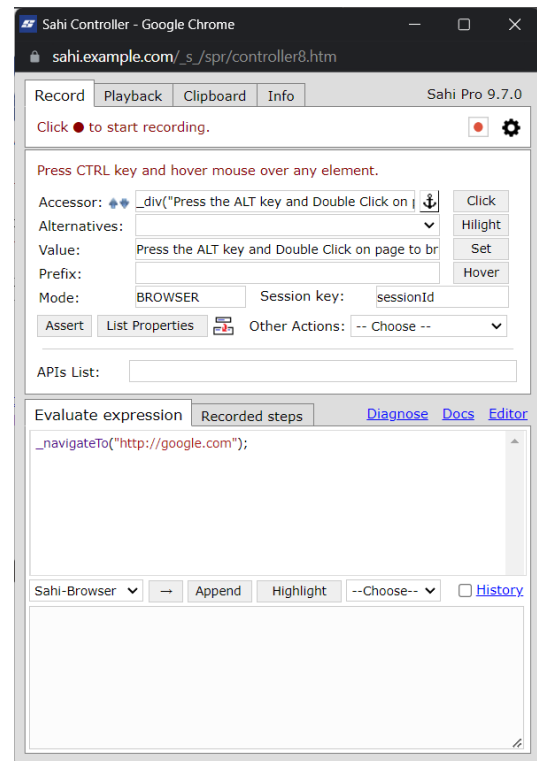
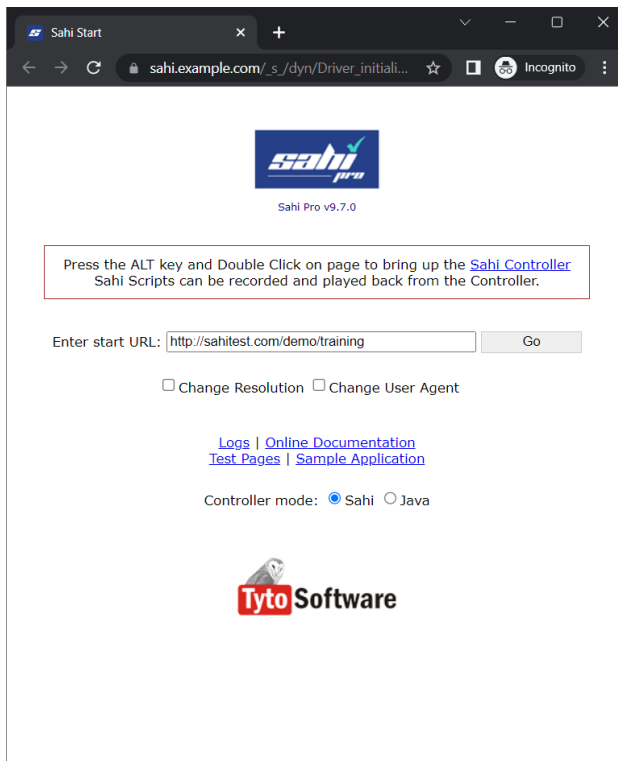


```
File Edit View
Apr 22, 2023 4:56:30 PM net.sf.sahi.config.Configuration reinitialize
INFO: Sahi properties file = C:\SahiPro\config\sahi.properties
Apr 22, 2023 4:56:31 PM net.sf.sahi.config.Configuration reinitialize
INFO: Sahi user properties file = C:\SahiPro\userdata\config\userdata.properties
Apr 22, 2023 4:56:31 PM net.sf.sahi.config.Configuration substituteBrowserProxyHostPort
INFO: Created C:\SahiPro\userdata\browser\vf\profiles\sahi\prefs.js
Apr 22, 2023 4:56:33 PM in.co.sahi.datastore.ReportDB <init>
INFO: Using Database:
driverName=org.h2.Driver
jdbcUrl=jdbc:h2:c:/SahiPro/userdata/database/db0;DB_CLOSE_DELAY=-1;IGNORECASE=TRUE
userName=sa
Apr 22, 2023 4:56:33 PM in.co.sahi.datastore.SahiDB createTables
INFO: Check tables version
Apr 22, 2023 4:56:34 PM in.co.sahi.datastore.SahiDB createTables
INFO: versionFromDB=2022-05-11 10:59:15
Apr 22, 2023 4:56:34 PM in.co.sahi.datastore.SahiDB createTables
INFO: version=2022-05-11 10:59:15
Apr 22, 2023 4:56:34 PM in.co.sahi.datastore.SahiDB createTables
INFO: At current version. No changes were done.
Apr 22, 2023 4:56:34 PM in.co.sahi.datastore.SahiDB createTables
INFO: {result: [{"SAHKEY", "SAHVALUE"}, {"dbVersion", "138"}, {"versionNo", "2022-05-11 10:59:15"}]}
Apr 22, 2023 4:56:34 PM in.co.sahi.datastore.SahiDB createTables
INFO: Finished preparing report tables: 238ms
Apr 22, 2023 4:56:34 PM in.co.sahi.datastore.MetadataDB <init>
INFO: Using Database:
driverName=org.h2.Driver
jdbcUrl=jdbc:h2:c:/SahiPro/userdata/database/metadata_db;DB_CLOSE_DELAY=-1;IGNORECASE=TRUE
userName=sa
Apr 22, 2023 4:56:34 PM in.co.sahi.datastore.SahiDB createTables
INFO: Check tables version
Apr 22, 2023 4:56:34 PM in.co.sahi.datastore.SahiDB createTables
INFO: versionFromDB=2022-05-11 10:59:15
Apr 22, 2023 4:56:34 PM in.co.sahi.datastore.SahiDB createTables
INFO: version=2022-05-11 10:59:15
Apr 22, 2023 4:56:34 PM in.co.sahi.datastore.SahiDB createTables
INFO: At current version. No changes were done.
Apr 22, 2023 4:56:34 PM in.co.sahi.datastore.SahiDB createTables
INFO: {result: [{"SAHKEY", "SAHVALUE"}, {"dbVersion", "12"}, {"versionNo", "2022-05-11 10:59:15"}]}
Apr 22, 2023 4:56:34 PM in.co.sahi.datastore.SahiDB createTables
INFO: Finished preparing metadata tables: 64ms
Ln 1, Col 1 100% Windows (CRLF) UTF-8
```

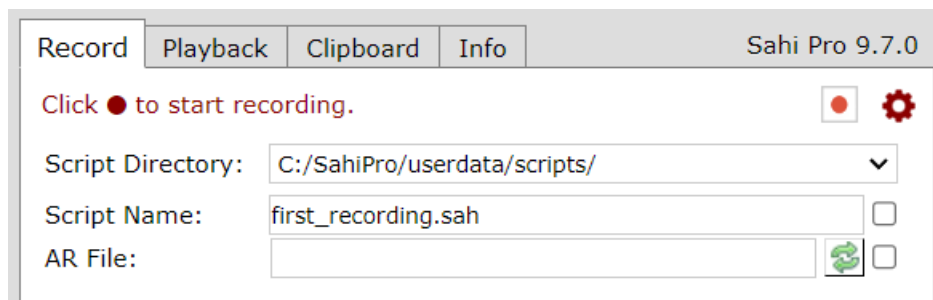
Users can use Configure License button to add license and reset product keys.



Clicking browser name will open the Sahi Start window in the selected browser. User can now open Sahi Controller to start recording the automation steps using Alt key + Double click on page.

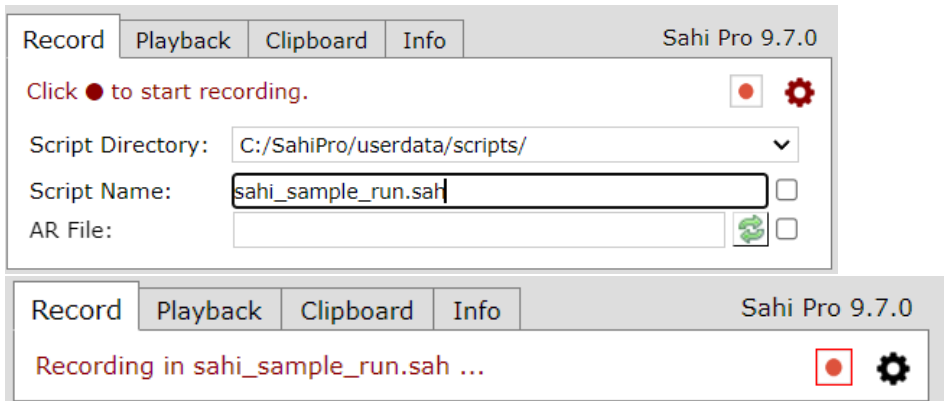
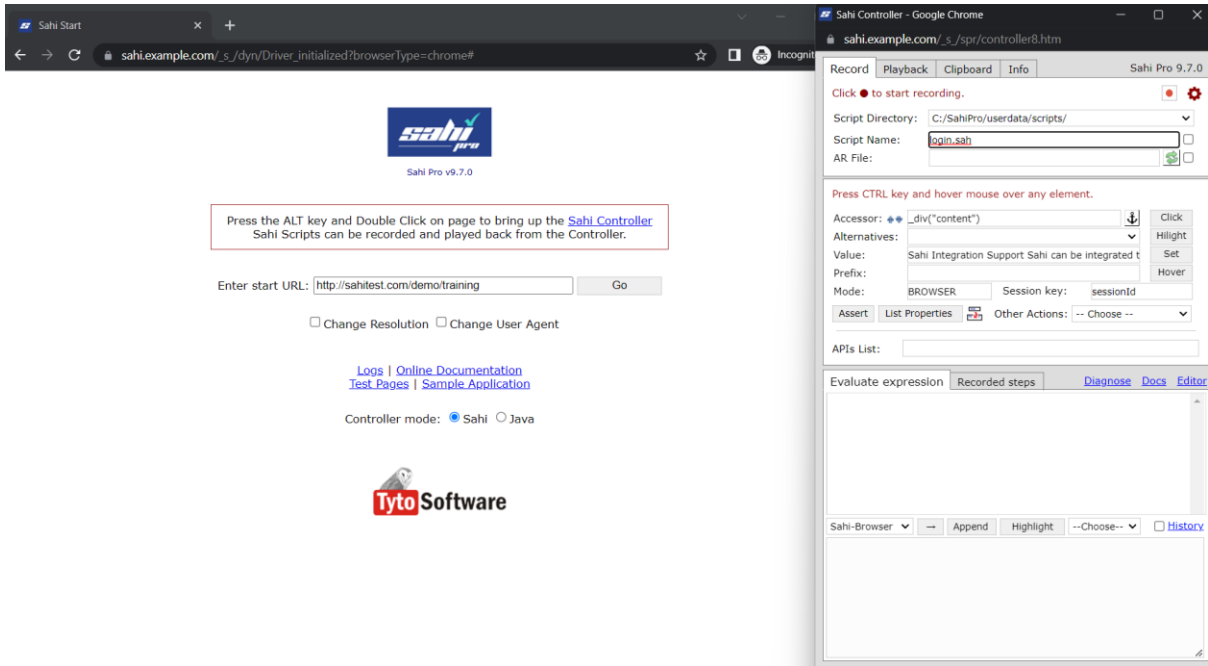


Sahi Controller can be used in two modes, 1) Sahi and 2) Java. In Sahi mode Controller contains Record, Playback, Clipboard, and Info tabs used to automate. User can click on red button at top to start automation. Recorded steps are show in Evaluate expression window. User can again click on red button to stop the recording. Clicking the settings icon shows the Script Directory, Script Name, AR file to save the recorded steps to a .sah file.



Automation using Sahi:

Let us start by recording and playing back a simple script. Sahi Pro is not just a record and playback tool, recording is a steppingstone to creating automation scripts. User can start application by clicking the application shortcut on desktop. Let's record our first script. Enter a script name to save the recording. And click record button.



- Enter a start URL: <http://sahitest.com/demo/training> to open Sahi provided demo application to test.
- Click on Go button to start recording steps in application.
- User can see the navigateTo expression being recorded onto script.
- When user types in username, the event is recorded.
- All the steps are autosaved to the file.
- User now clicks on login button, to record the login action.

Enter start URL:

Evaluate expression

Recorded steps

[Diagnose](#) [Docs](#) [Editor](#)

```
_navigateTo("http://sahitest.com/demo/training");
```

Sahi Training Site

Username

Password

Not a user? [Register](#)

Use test/secret to login

Press CTRL key and hover mouse over any element.

Accessor:

Alternatives:

Value:

Prefix:

Mode: Session key:

APIs List:

Evaluate expression [Recorded steps](#) [Diagnose](#) [Docs](#) [Ed](#)

```
_setValue(_textbox("user"), "test");
```

Sahi Training Site

Username

Password

Not a user? [Register](#)

Use test/secret to login

Press CTRL key and hover mouse over any element.

Accessor:

Alternatives:

Value:

Prefix:

Mode: Session key:

APIs List:

Evaluate expression [Recorded steps](#) [Diagnos](#)

```
_navigateTo("http://sahitest.com/demo/training");
_setValue(_textbox("user"), "test");
_setPassword(_password("password"), "MgkKEQBU");
```

All available books

Title	In stock	Cost	Add quantity to cart
Core Java	5	Rs. 300	<input type="text" value="0"/>
Ruby for Rails	12	Rs. 200	<input type="text" value="0"/>
Python Cookbook	7	Rs. 350	<input type="text" value="0"/>

My Cart

Title	Quantity	Unit Cost	Total Cost
Grand Total: <input type="text" value="0"/>			

Recording in sahi_sample_run.sah ...

Press CTRL key and hover mouse over any element.

Accessor:

Alternatives:

Value:

Prefix:

Mode: Session key:

APIs List:

Evaluate expression [Recorded steps](#) [Diagnose](#) [Docs](#) [Editor](#)

```
_click(_submit("Login"));
```

Sahi-Browser History

Now we add books quantity to cart.

All available books

Title	In stock	Cost	Add quantity to cart
Core Java	5	Rs. 300	<input type="text" value="1"/>
Ruby for Rails	12	Rs. 200	<input type="text" value="2"/>
Python Cookbook	7	Rs. 350	<input type="text" value="2"/>

| |

All the actions are recorded in Recorded steps view.

```

click(_submit("Login"));
setValue(_textbox("q"), "1");
setValue(_textbox("q[1]"), "2");
setValue(_textbox("q[2]"), "2");
click(_button("Add"));

```

After Add button is clicked, books are added to the Cart.

My Cart

Title	Quantity	Unit Cost	Total Cost
Core Java	1	Rs.300	Rs.300
Ruby for Rails	2	Rs.200	Rs.400
Python Cookbook	2	Rs.350	Rs.700

Grand Total:

In this test case we verify that the Grand Total is correct, for the books quantity we added to cart. We identify the total element and create an assertion for it.

To do that we need to follow the step shown in the controller.

Press Control key and hover mouse over total element.

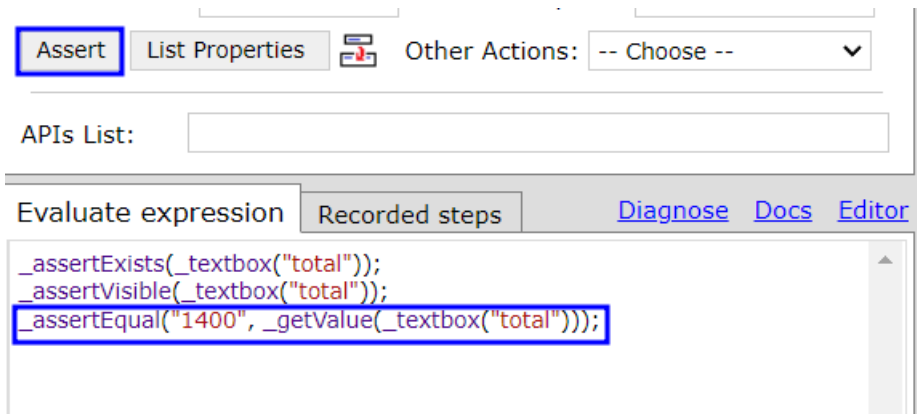
Press CTRL key and hover mouse over any element.

Press CTRL key and hover mouse over any element.

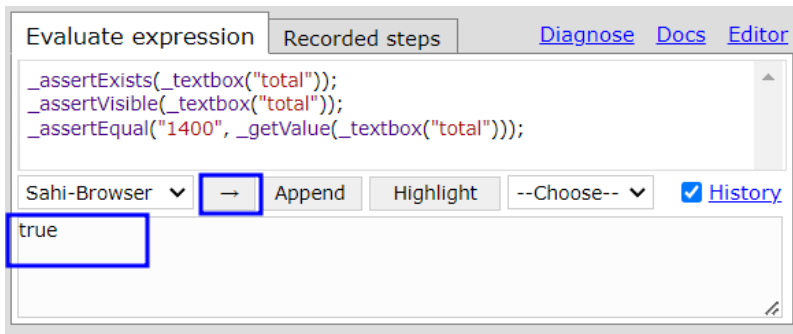
Accessor:	<input total\")"="" type="text" value="_textbox(\"/>	<input type="button" value="Click"/>
Alternatives:	<input total\")"="" type="text" value="_textbox(\"/>	<input type="button" value="Highlight"/>
Value:	<input type="text" value="1400"/>	<input type="button" value="Set"/>
Prefix:	<input type="text"/>	<input type="button" value="Hover"/>
Mode:	<input type="text" value="BROWSER"/>	Session key: <input type="text" value="sessionId"/>
<input type="button" value="Assert"/>	<input type="button" value="List Properties"/>	Other Actions: <input type="text" value="-- Choose --"/>

APIs List:

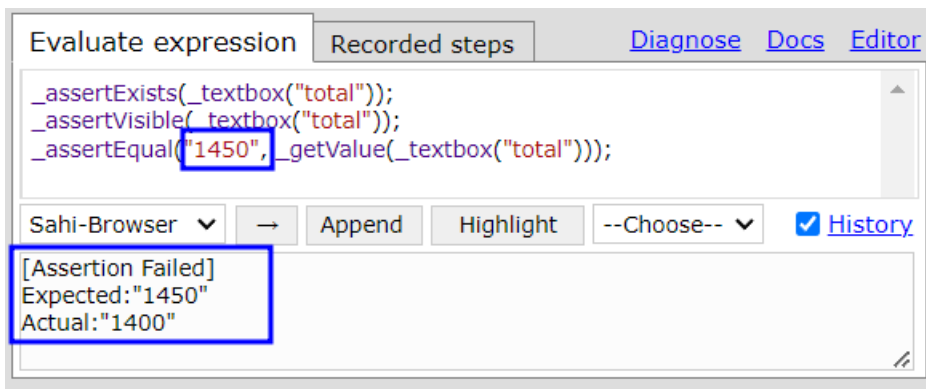
Users can now click on the Assert button to get different assert suggestions that can be done on the selected element.



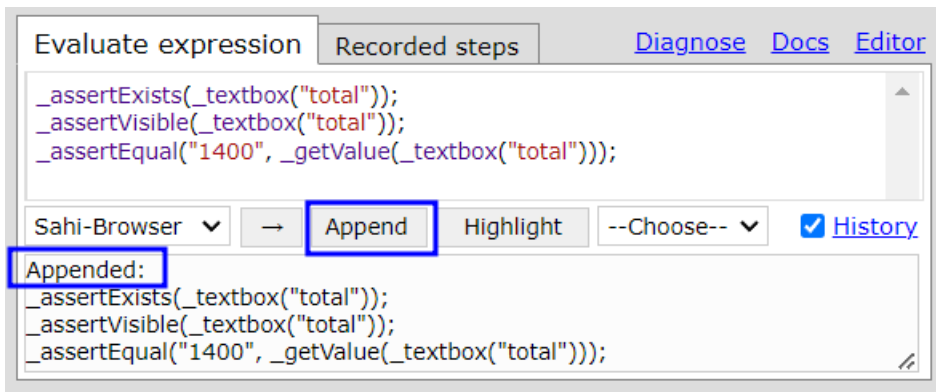
Generated assert statements check if the element is existing in DOM, visible to user and verifies if it is equal to 1400 value. User can verify the assertions by clicking the arrow button below the Evaluate expression tab.



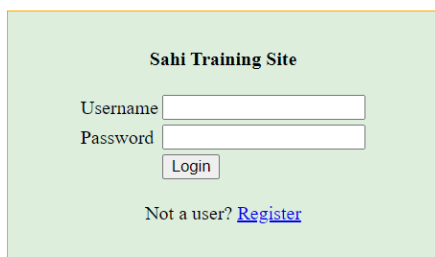
Changing 1400 value to 1450 and then evaluating the asserts will result in test failure.



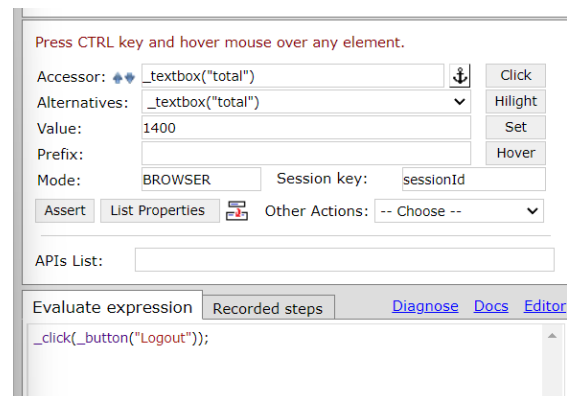
When we are good with the assert statements, clicking on Append button appends the generated asserts to the script.



Actions that we perform on the browser are automatically recorded. Actions that we perform on the Sahi Controller, need to be appended to the script manually.



Use test/secret to login

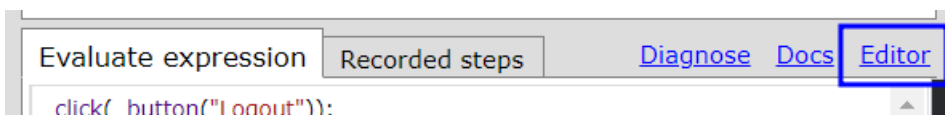


Let's logout, examine the recorded script and run the automation from start.

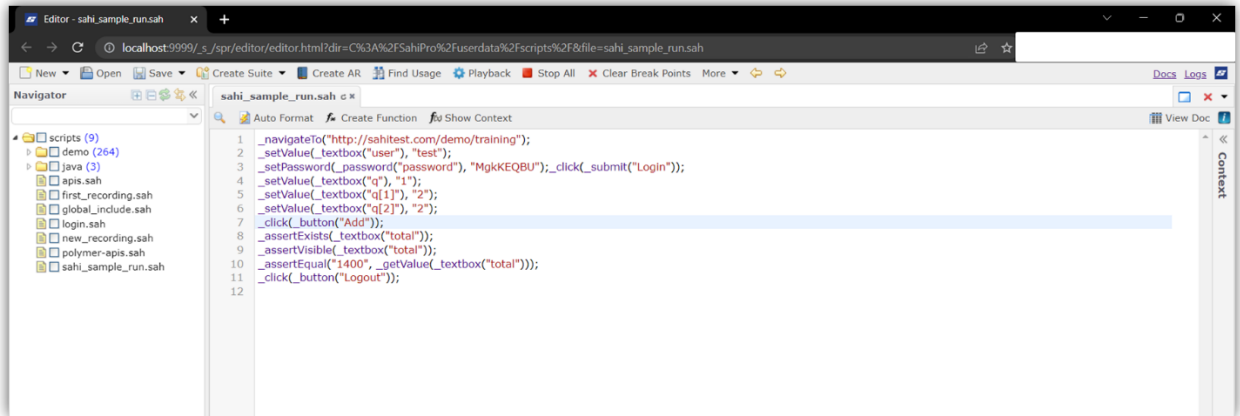
Clicking red button will stop the recording and save all recorded steps to the .sah file provided at the start of recording.



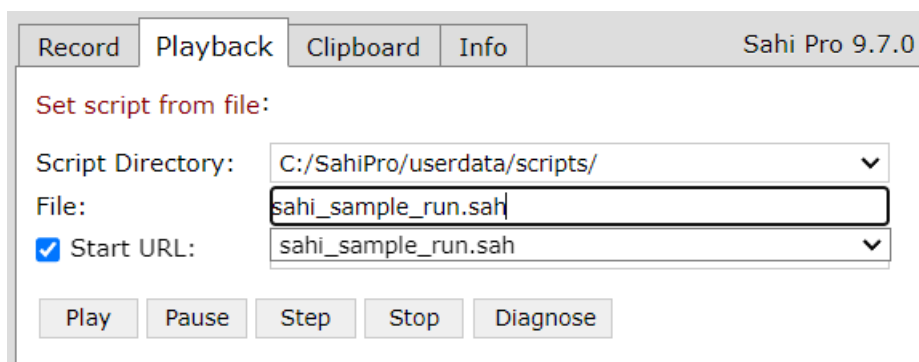
Click on Editor to open Sahi Scripts Editor. This opens browser and reaches localhost:9999 website to open editor hosted locally by Sahi.



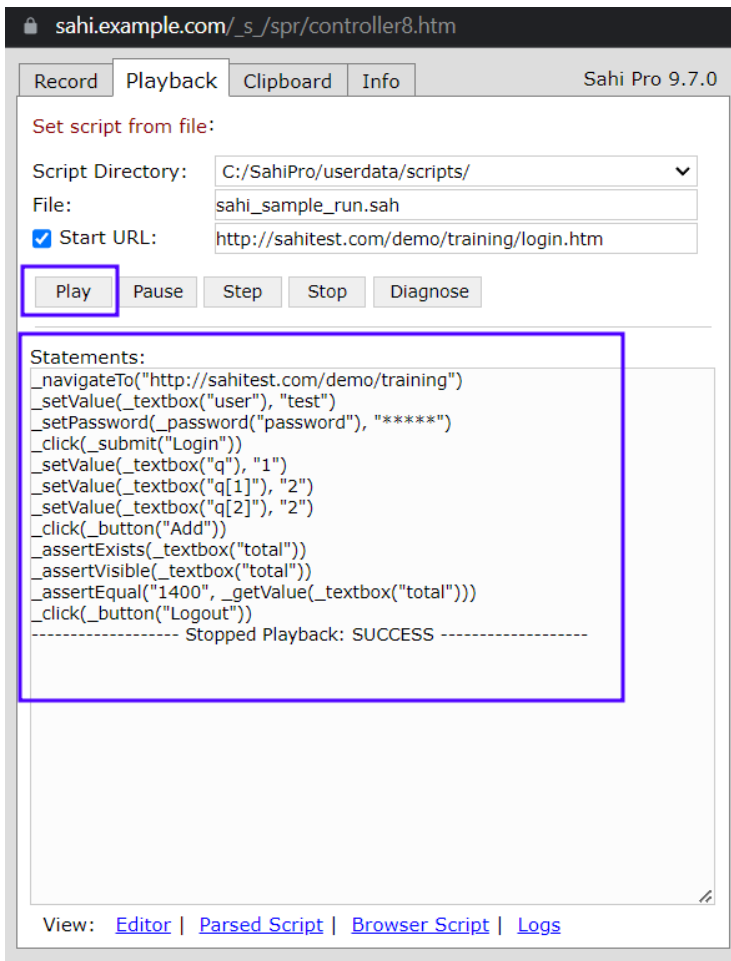
User can see the recorded steps while automating web application. All steps are recorded in JavaScript statements. Main advantage is user does not have to deal with xpaths or CSS selectors to identify an element on DOM. Wait statements are also taken care implicitly by Sahi.



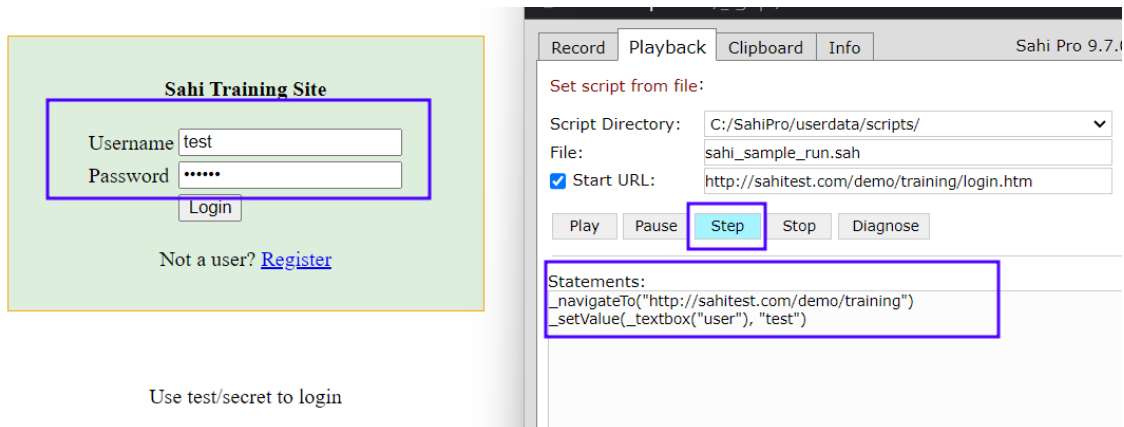
Let's playback the script that we just recorded. Open Sahi Controller, and go to Playback tab.



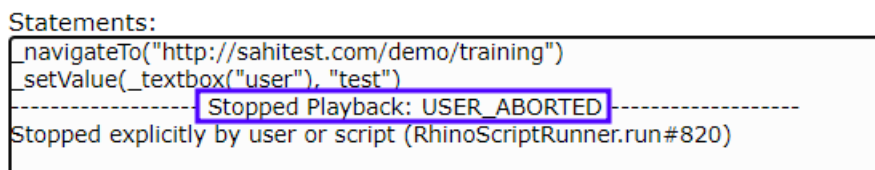
Clicking Play button starts the script and all Statements executed are shown in Statements window below.



User can use Pause button to stop the script execution.



Step button is used to run the script step by step.

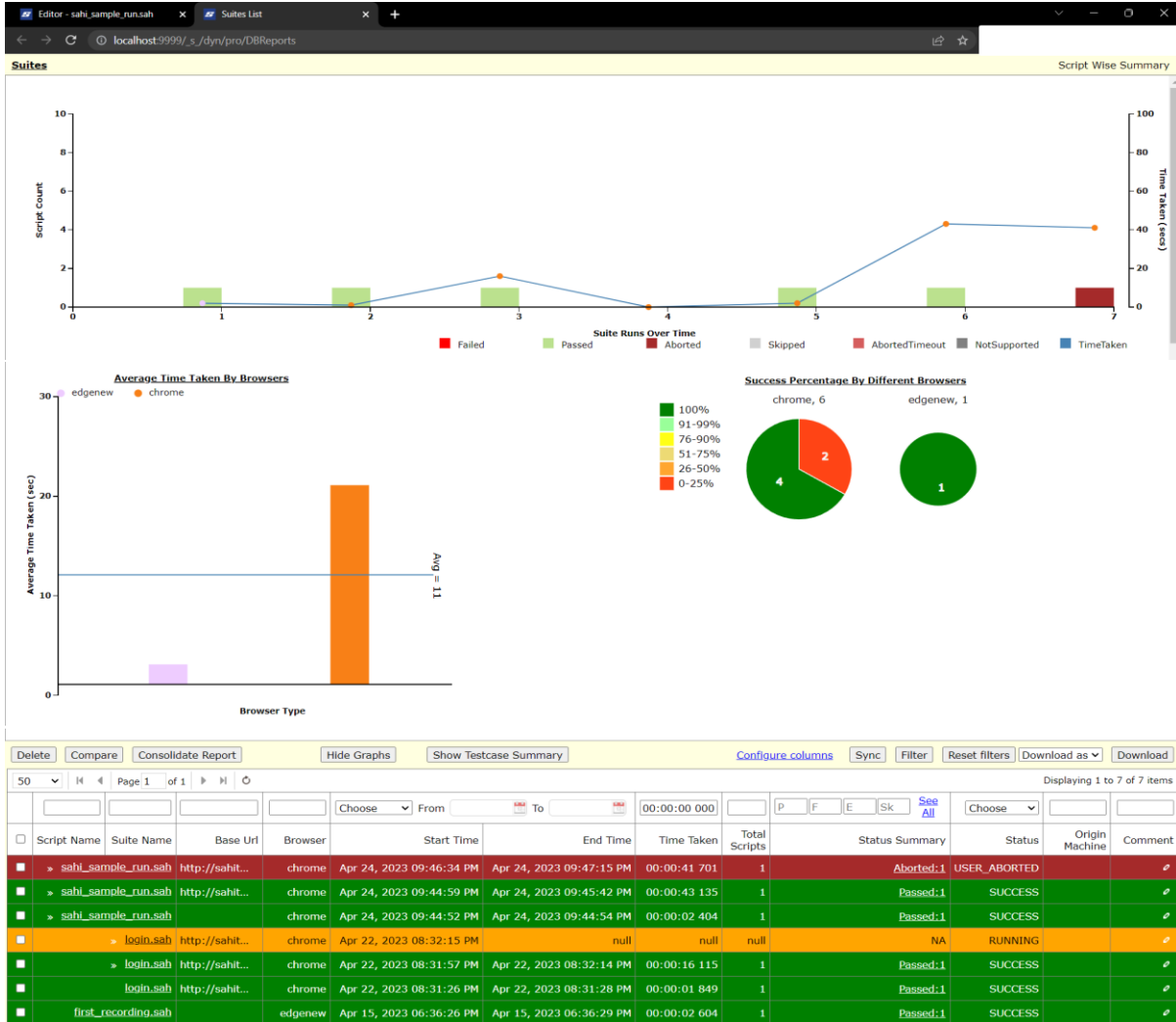


Logging in Sahi

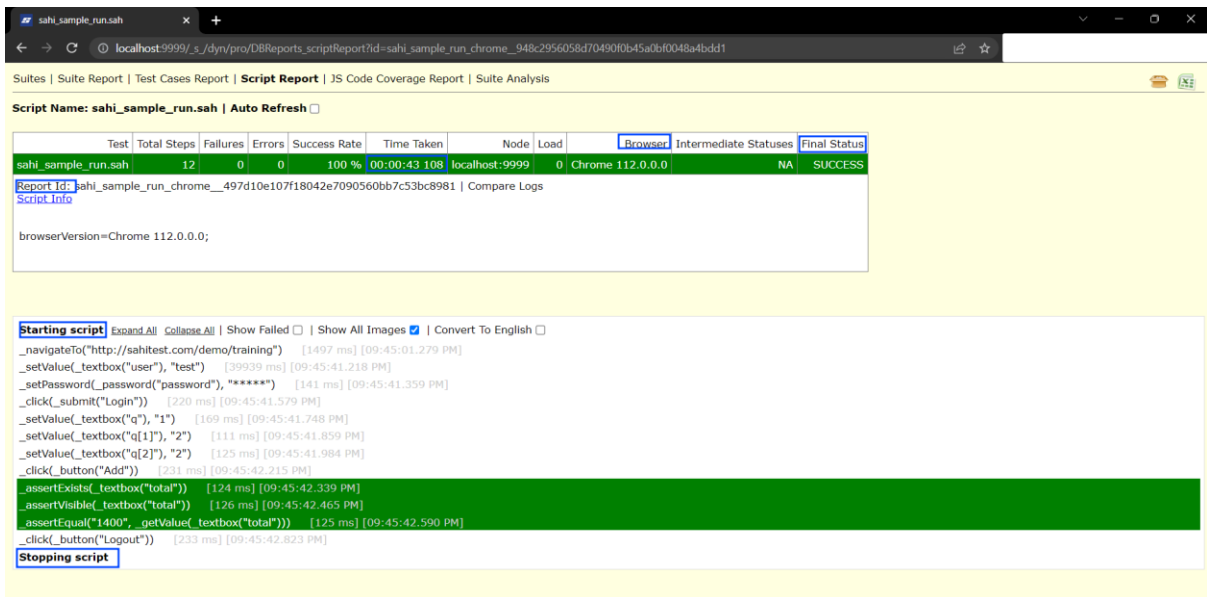
User can go through the logs using Logs link in Controller window.

View: [Editor](#) | [Parsed Script](#) | [Browser Script](#) | [Logs](#)

Logs contains suite runs, Average Time Taken By Browsers graph, Success Percentage By Different Browsers pie chart and all the Playbacks conducted.



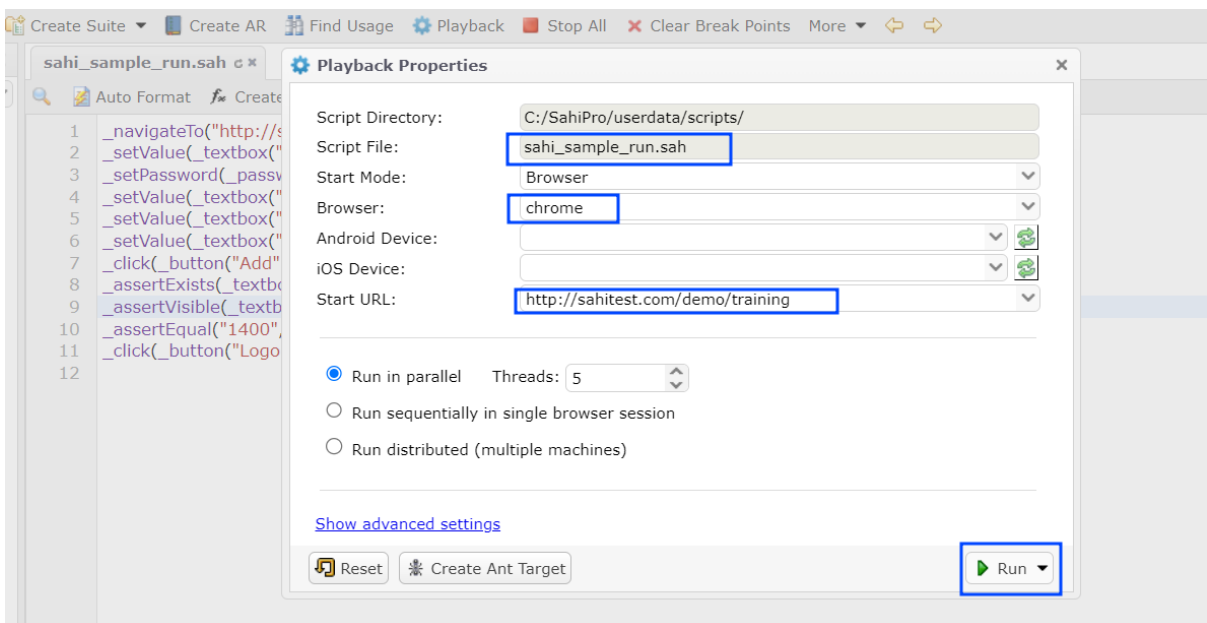
User can go through the Sahi reports to know more about the execution process, the time taken for each action to run.



Let's see a failure test case in the script. Changing quantity of book 3 in the cart will give us a different total. When the grand total is not equal to 1400, Sahi logs an error in the test case.

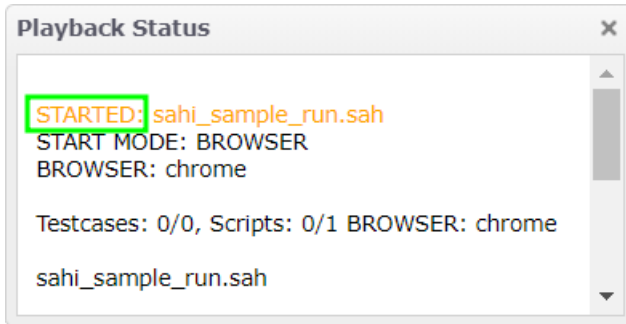
```
_setValue(_textbox("q[2]"), "10");
```

User can playback the script from Sahi Editor without reaching Controller. Select the script file, browser and start URL and click on Run button to start the playback.

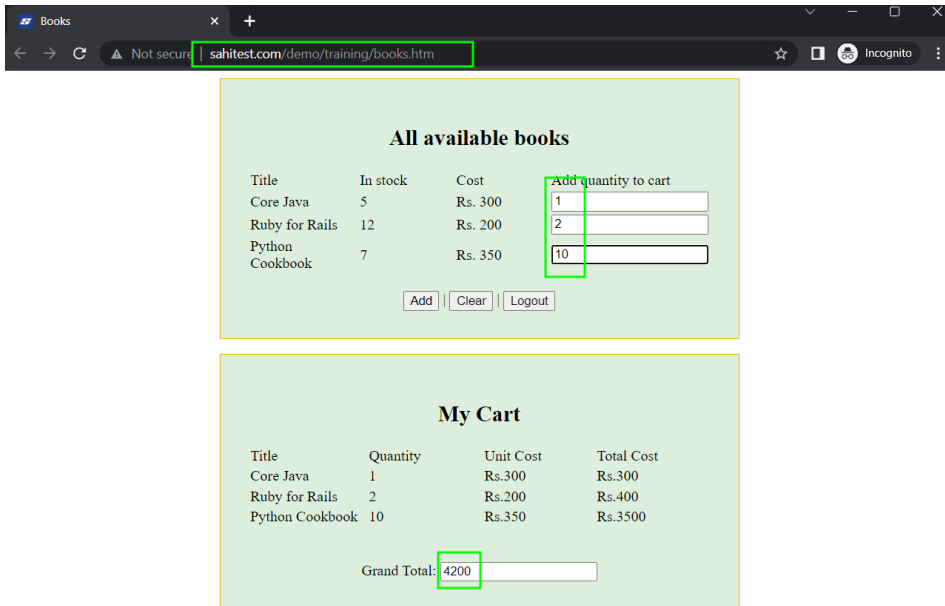


Let's see a failure test case in the script. Changing quantity of book 3 in the cart will give us a different total. When the grand total is not equal to 1400, Sahi logs an error in the test case.

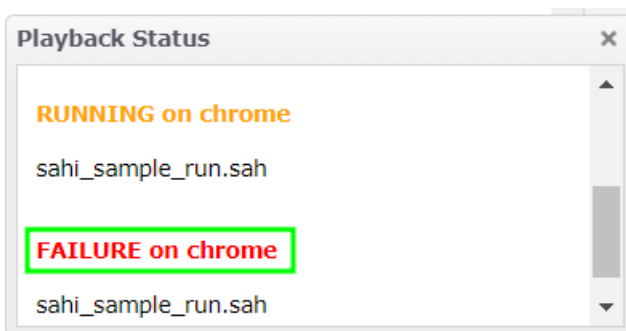
User can playback the script from Sahi Editor without reaching Controller.



Before execution of any step, Sahi waits for any Ajax activity, network activity to subside.



If a step seems to be failing, Sahi waits for 2 seconds and tries to re-execute that step. It will do this 5 times, between the retries, if the system recovers, then Sahi will execute that step, else it'll mark the step as a failure.



Let's go through the logs, we see that script itself is in red.

Script Name	Suite Name	Base Url	Browser	Start Time	End Time	Time Taken	Total Scripts	Status Summary	Status	Origin Machine	Comment
sahi_sample_run.sah		http://sahit...	chrome	Apr 25, 2023 11:57:32 AM	Apr 25, 2023 11:57:59 AM	00:00:26 120	1	Failed:1	FAILURE		
sahi_sample_run.sah		http://sahit...	chrome	Apr 25, 2023 11:18:16 AM	Apr 25, 2023 11:18:26 AM	00:00:10 069	1	Passed:1	SUCCESS		
sahi_sample_run.sah		http://sahit...	chrome	Apr 24, 2023 09:46:34 PM	Apr 24, 2023 09:47:15 PM	00:00:41 701	1	Aborted:1	USER_ABORTED		
sahi_sample_run.sah		http://sahit...	chrome	Apr 24, 2023 09:44:59 PM	Apr 24, 2023 09:45:42 PM	00:00:43 135	1	Passed:1	SUCCESS		

Opening the script, the assert step is marked in red and shows the expected and actual values mismatch. Opening onScriptFailure step shows the screenshot when the assert statement failure happened during test run.

Script Name: sahi_sample_run.sah | Auto Refresh

Test	Total Steps	Failures	Errors	Success Rate	Time Taken	Node	Load	Browser	Intermediate Statuses	Final Status
sahi_sample_run.sah	16	1	0	94 %	00:00:21.318	localhost:9999	-1	Chrome 112.0.0.0	FAILURE	FAILURE

Report Id: sahi_sample_run_chrome__fcc901060688504ac80a54208474d9cc79ee | Compare Logs
[Script Info](#)

Starting script Expand All Collapse All Show Failed Show All Images Convert To English

```

_navigateTo("http://sahitest.com/demo/training") [1038 ms] [11:57:35.151 AM]
_setValue(_textbox("user"), "test") [303 ms] [11:57:35.454 AM]
_setPassword(_password("password"), "*****") [246 ms] [11:57:35.700 AM]
_click(_submit("Login")) [132 ms] [11:57:35.832 AM]
_setValue(_textbox("q"), "1") [690 ms] [11:57:36.522 AM]
_setValue(_textbox("q[1]"), "2") [133 ms] [11:57:36.655 AM]
_setValue(_textbox("q[2]"), "10") [127 ms] [11:57:36.782 AM]
_click(_button("Add")) [229 ms] [11:57:37.011 AM]
_assertExists(_textbox("total")) [124 ms] [11:57:37.135 AM]
_assertVisible(_textbox("total")) [126 ms] [11:57:37.261 AM]
_assertEqual("1400", _getValue(_textbox("total"))) [13767 ms] [11:57:51.028 AM]
[Assertion Failed!
Expected: "1400"
Actual: "4200"
at: (/scripts/sahi_sample_run.sah#n=11)
+] onScriptFailure([+][object])
_click(_button("Logout")) [79 ms] [11:57:55.337 AM]
Stopping script

```

```

[-] onScriptFailure([+][object])
[-] onScriptFailureDefault([+][object])
_lockWindow() [158 ms] [11:57:51.186 AM]
_focusWindow() [3176 ms] [11:57:54.362 AM]
_takePageScreenshot() [833 ms] [11:57:55.195 AM]

```

Library Functions

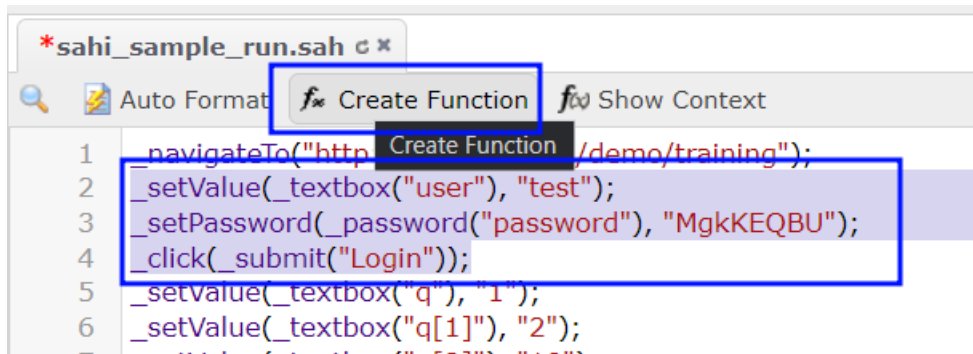
The steps that we see in the Sahi Script are low level instructions to the computer to perform actions on a browser.

The language of the business is much more in human speaking terms like – login to the system, add books quantities to the cart, check the total amount, and logout from the system.

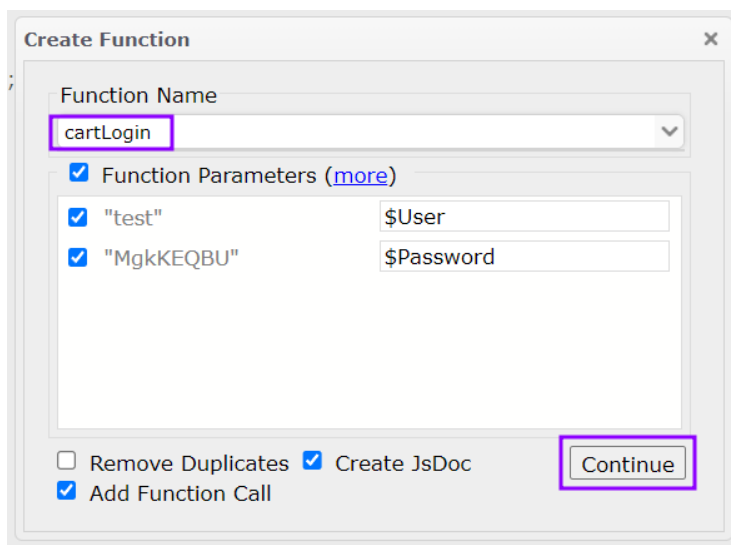
So, we need to create Business Level Abstractions out of the steps that we recorded.

We create Business Level Abstractions using Functions in Sahi.

To create a function, we select steps that pertain to a logical business step and click on Create Function button, provide a function name and click on continue.



```
*sahi_sample_run.sah c x
Auto Format Create Function Show Context
1  _navigateTo("http://demo/training");
2  _setValue(_textbox("user"), "test");
3  _setPassword(_password("password"), "MgkKEQBU");
4  _click(_submit("Login"));
5  _setValue(_textbox("q"), "1");
6  _setValue(_textbox("q[1]"), "2");
```



Create Function

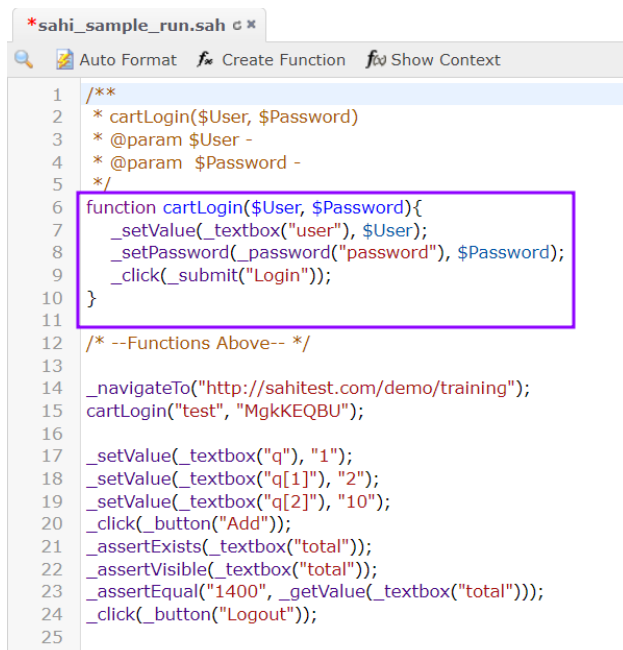
Function Name
cartLogin

Function Parameters (more)

<input checked="" type="checkbox"/> "test"	\$User
<input checked="" type="checkbox"/> "MgkKEQBU"	\$Password

Remove Duplicates Create JsDoc

Add Function Call



```
*sahi_sample_run.sah c x
Auto Format Create Function Show Context
1  /**
2  * cartLogin($User, $Password)
3  * @param $User -
4  * @param $Password -
5  */
6  function cartLogin($User, $Password){
7    _setValue(_textbox("user"), $User);
8    _setPassword(_password("password"), $Password);
9    _click(_submit("Login"));
10 }
11
12 /* --Functions Above-- */
13
14 _navigateTo("http://sahitest.com/demo/training");
15 cartLogin("test", "MgkKEQBU");
16
17 _setValue(_textbox("q"), "1");
18 _setValue(_textbox("q[1]"), "2");
19 _setValue(_textbox("q[2]"), "10");
20 _click(_button("Add"));
21 _assertExists(_textbox("total"));
22 _assertVisible(_textbox("total"));
23 _assertEqual("1400", _getValue(_textbox("total")));
24 _click(_button("Logout"));
25
```

Sahi creates a function and extracts possible parameters from the recorded steps and takes them as arguments to the function.

In the recorded steps place, it will call the created function with the values from the steps.

We do similar procedure for remaining steps, combine the quantity adding steps to a single function.

```
*/  
function cartLogin($User, $Password){  
    _setValue(_textbox("user"), $User);  
    _setPassword(_password("password"), $Password);  
    _click(_submit("Login"));  
}  
  
/* --Functions Above-- */  
  
_navigateTo("http://sahitest.com/demo/training");  
cartLogin("test", "MgkKEQBU");
```

Auto Format **2** Create Function Show Context

```
1 /**  
2  * cartLogin($User, $Password)  
3  * @param $User -  
4  * @param $Password -  
5  */  
6 function cartLogin($User, $Password){  
7     _setValue(_textbox("user"), $User);  
8     _setPassword(_password("password"), $Password);  
9     _click(_submit("Login"));  
10 }  
11  
12 /* --Functions Above-- */  
13  
14 _navigateTo("http://sahitest.com/demo/training");  
15 cartLogin("test", "MgkKEQBU");  
16  
17 1 _setValue(_textbox("q"), "1");  
18 _setValue(_textbox("q[1]"), "2");  
19 _setValue(_textbox("q[2]"), "10");  
20 _click(_button("Add"));  
21 _assertExists(_textbox("total"));  
22 _assertVisible(_textbox("total"));  
23 _assertEqual("1400", _getValue(_textbox("total")));  
24 _click(_button("Logout"));  
25
```

3 Function Name
addItemToCart

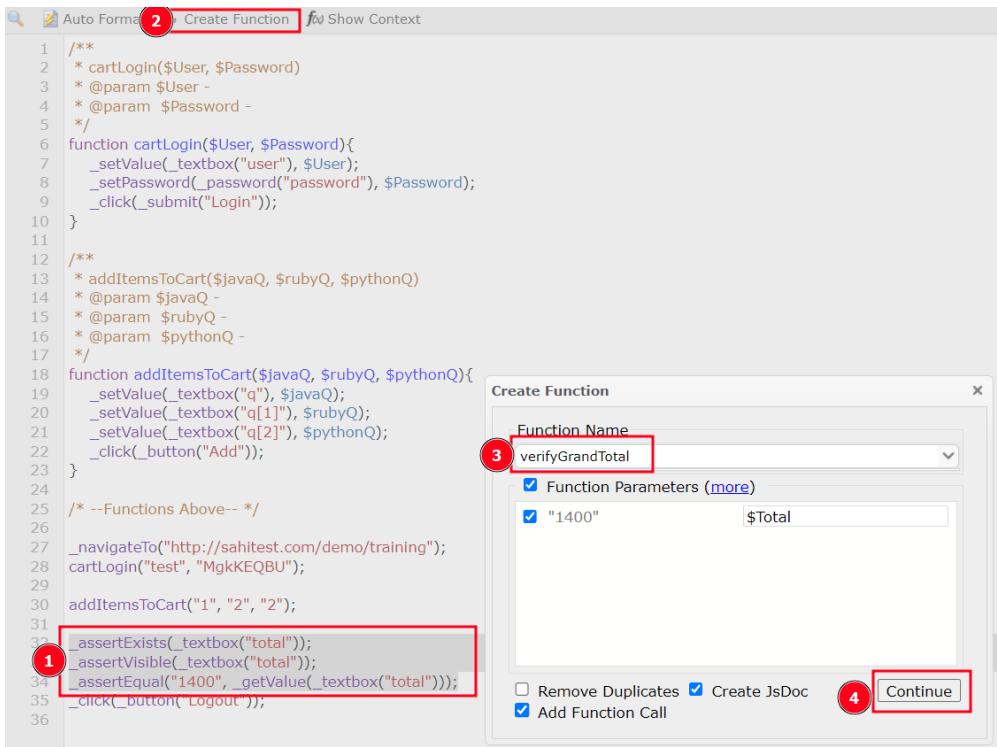
Function Parameters (more)

"1" \$javaQ **4**
 "2" \$rubyQ
 "10" \$pythonQ

Remove Duplicates Create JsDoc
 Add Function Call

5 Continue

```
function addItemToCart($javaQ, $rubyQ, $pythonQ){  
    _setValue(_textbox("q"), $javaQ);  
    _setValue(_textbox("q[1]"), $rubyQ);  
    _setValue(_textbox("q[2]"), $pythonQ);  
    _click(_button("Add"));  
}  
  
/* --Functions Above-- */  
  
_navigateTo("http://sahitest.com/demo/training");  
cartLogin("test", "MgkKEQBU");  
  
addItemsToCart("1", "2", "10");
```



```

/**
 * cartLogout()
 */
function cartLogout(){
  _click(_button("Logout"));
}

/* --Functions Above-- */

_navigateTo("http://sahitest.com/demo/training");
cartLogin("test", "MgkKEQBU");

addItemToCart("1", "2", "2");

verifyGrandTotal("1400");

cartLogout();

```

To reuse the functions generated, we place them in a separate .sah script file.

Press Control and click on the function to open Context panel.



Context

File Path:

Function Name:

Arguments:

\$User

\$Password

Show JsDoc

Accessor Repository (AR) Parameters

File Path:

Key:

Value:

Select the library file from File Path where the function was defined and implemented. Click on Include button to add import statement to the current script.

```
1  _include("cart_library.sah");  
2  
3  _navigateTo("http://sahitest.com/demo/training");  
4  cartLogin("test", "MgkKEQBU");  
5  
6  addItemToCart("1", "2", "2");  
7  
8  verifyGrandTotal("1400");  
9  
10 cartLogout();
```

Playback the script.

Let's look at the logs for the execution completed in Sahi Logs window.

Each function is listed in log and time for execution is listed after each step.

Script Name: sahi_sample_run.sah | Auto Refresh

Test	Total Steps	Failures	Errors	Success Rate	Time Taken	Node	Load	Browser	Intermediate Statuses	Final Status
sahi_sample_run.sah	12	0	0	100 %	00:00:04.127	localhost:9999	-1	Chrome 112.0.0.0	NA	SUCCESS

Report Id: sahi_sample_run_chrome__40fad06a0464204afb0beac0c5984b400674 | Compare Logs
[Script Info](#)

Starting script Expand All Collapse All Show Failed Show All Images Convert To English

```

navigateTo("http://sahitest.com/demo/training") [1426 ms] [04:43:23.408 PM]
[-] cartLogin("test", "MgkKEQBU")
  setValue(_textbox("user"), "test") [304 ms] [04:43:23.712 PM]
  setPassword(_password("password"), "*****") [191 ms] [04:43:23.903 PM]
  click_submit("Login") [157 ms] [04:43:24.060 PM]
[+] addItemsToCart("1", "2", "2")
[+] verifyGrandTotal("1400")
[+] cartLogout()
Stopping script

```

Let's change the script a little to raise an error in grand total assertion, and see the logs.

`verifyGrandTotal("1450");`

Script Name	Suite Name	Base Url	Browser	Start Time	End Time	Time Taken	Scripts	Status Summary	Status	Machine	Comments
sahi_sample_run.sah	http://sahit...	chrome	Apr 25, 2023 04:49:13 PM	Apr 25, 2023 04:49:37 PM	00:00:24.102	1	Failed:1	FAILURE			

Script Name: sahi_sample_run.sah | Auto Refresh

Test	Total Steps	Failures	Errors	Success Rate	Time Taken	Node	Load	Browser	Intermediate Statuses	Final Status
sahi_sample_run.sah	16	1	0	94 %	00:00:16.862	localhost:9999	-1	Chrome 112.0.0.0	FAILURE	FAILURE

Report Id: sahi_sample_run_chrome__c963aa3106b1004f98086e0cb6ffe799e71 | Compare Logs
[Script Info](#)

Starting script Expand All Collapse All Show Failed Show All Images Convert To English

```

navigateTo("http://sahitest.com/demo/training") [1268 ms] [04:49:16.209 PM]
[+] cartLogin("test", "MgkKEQBU")
[+] addItemsToCart("1", "2", "2")
[-] verifyGrandTotal("1450")
[+] cartLogout()
Stopping script

```

Grouping as functions lets us detect the code where the assertion failed. Screenshot is also provided in logs for the failure occurred.

```

[-] verifyGrandTotal("1450")
  assertExists(_textbox("total")) [123 ms] [04:49:18.259 PM]
  assertVisible(_textbox("total")) [115 ms] [04:49:18.374 PM]
  assertEquals("1450", _getValue(_textbox("total"))) [10191 ms] [04:49:28.565 PM]
[Assertion Failed]
Expected: "1450"
Actual: "1400"
at: (/scripts/cart_library.sah&n=30) verifyGrandTotal
at: (/scripts/sahi_sample_run.sah&n=5)

```

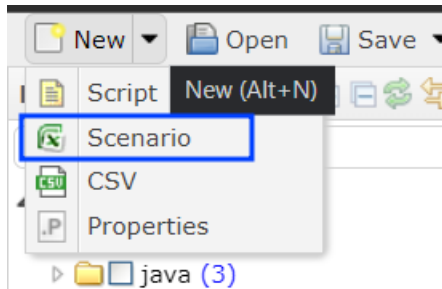
Scenarios in Sahi

Sahi Scripts are written in JavaScript language.

Testers are present where they understand the business functionality of the application, but do not necessarily understand the syntax and semantics of Sahi Script.

To help them participate in the automation process, we have an alternative way of defining our test cases and scenarios. We do that using a scenario file.

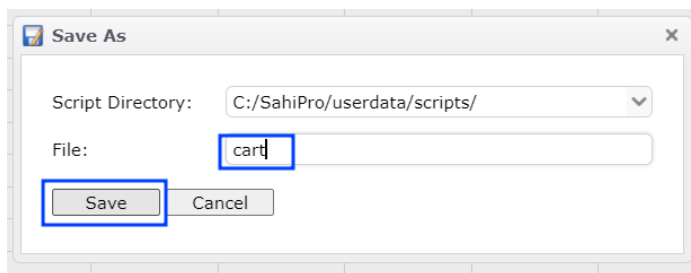
Click on New button and select Scenario button.



Click on Save button and save the file.

sahi_sample_run.sah c * *new 1.s.csv c *

	A	B	C	D	E	F	G	H	I
1	TestCase	Key Word	Argument 1	Argument 2					
2									
3	My First Testcase	[Documentation]	My testcase description. Modify as needed						
4		My Function	my argument	my argument 2					
5									
6									
7									



Give a name in the test case column.

	A	B	C	D	E
	TestCase	Key Word	Argument 1	Argument 2	
	Test Grand Total	[Documentation]	My testcase description. Modify as needed		

Add a description to the test case in Argument 1 column.

A	B	C	D	E
TestCase	Key Word	Argument 1	Argument 2	
Test Grand Total	[Documentation]	Add books to cart and verify total		

User can use different functions available in cart_library file by including in the current file.

When user starts to type the function name, Sahi shows the available functions.

A	B	C	D	E
TestCase	Key Word	Argument 1	Argument 2	
Test Grand Total	[Documentation]	Add books to cart and verify total		
		<div style="border: 1px solid red; padding: 5px;"> cart cart Login cart Logout add Items To ... </div>		

Ctrl+click on the function to know more details about it

Function Details □ ×

File:
 Open Include

Function Name:

Arguments:

\$User

\$Password

Set as field:value OK Cancel

Select Include button to include cart_library.sah in the current scenario file.

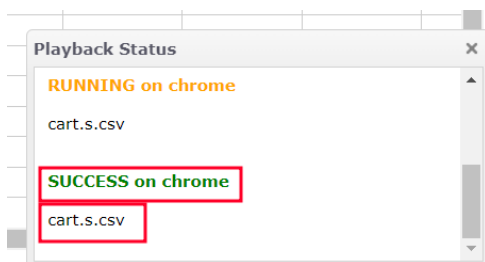
Now user can enter the data that is needed for login to happen i.e., username and password in Argument 1 and Argument 2 columns.

3	Test Grand Total	[Documentation]	Add books to cart and verify total		
4		cartLogin	test	MgkKEQBU	
5					

User can pass the parameters values in Function Details window.

	A	B	C	D	E	F
1	TestCase	Key Word	Argument 1	Argument 2		
2		loadSahi	cart_library.sah			
3						
4	Test Grand Total	[Documentation]	Add books to cart and verify total			
5		cartLogin	test	MgkKEQBU		
6		addItemsToCart	java Q : 1	ruby Q : 2	python Q : 2	
7		verify Grand Total	Total : 1400			
8		cart Logout				
9						

Let's run the script.



Opening logs

Suites | Suite Report | Test Cases Report | **Script Report** | JS Code Coverage Report | Suite Analysis

Script Name: cart.s.csv | Auto Refresh

Test	Total Steps	Failures	Errors	Success Rate	Time Taken	Node	Load	Browser	Intermediate Statuses	Final Status
cart.s.csv	11	0	0	100 %	00:00:04.171	localhost:9999	-1	Chrome 112.0.0.0	NA	SUCCESS

Report Id: cart_chrome__4125f1310e9f004d79091c4021f4425bc481 | Compare Logs
[Script Info](#)

Total Test Cases	Passed	Failed	Success Rate
1	1	0	100 %

Test Case Id	Description	Time Taken	Intermediate Statuses	Final Status
Test Grand Total	Add books to cart and verify total	00:00:02.748	NA	SUCCESS

Starting script [Expand All](#) [Collapse All](#) | Show Failed | Show All Images | Convert To English

loadSahi	cart_library.sah
Test Grand Total	[Documentation] Add books to cart and verify total
cartLogin	test MgkKEQBU
addItemToCart	java Q:1 ruby Q:2 python Q:2
verify Grand Total	Total:1400
cart Logout	

Stopping script

Clicking the step name shows the function called and the code executed.

cartLogin	test	MgkKEQBU
-----------	------	----------

```
[-] cartLogin("test", "MgkKEQBU")
  _setValue(_textbox("user"), "test") [1553 ms] [05:14:31.951 PM]
  _setPassword(_password("password"), "*****") [239 ms] [05:14:32.190 PM]
  _click(_submit("Login")) [205 ms] [05:14:32.395 PM]
```

Data Driven Suites in Sahi

Let's run our scripts in batches using Sahi Parallel playback.

To execute multiple scripts in a batch, we need to create a suite file.

We create a suite file by first choosing the scripts that we want to execute.

Now click on Create Suite → Data Driven Suite

The screenshot shows the Sahi IDE interface. In the top menu bar, the 'Create Suite' option is highlighted with a red box and a blue circle labeled '3'. Below it, in the 'Suite' dropdown menu, the 'Data Driven Suite' option is highlighted with a red box and a blue circle labeled '4'. In the left-hand 'Navigator' pane, the file 'cart.s.csv' is selected with a red box and a blue circle labeled '1', and the file 'login.sah' is also selected with a red box and a blue circle labeled '2'. The main editor area displays a table representing the Data Driven Suite:

	A	B	C
1	TestCase	Key Word	Argume
2		loadSahi	cart_libr
3			
4	Test Grand Total	[Documentation]	Add boo cart and total
5		cartLogin	test
6		addItemsToCart	java Q :
7		verify Grand Total	Total : 1
8		cart Logout	

The above step creates a suite file. Save the file.

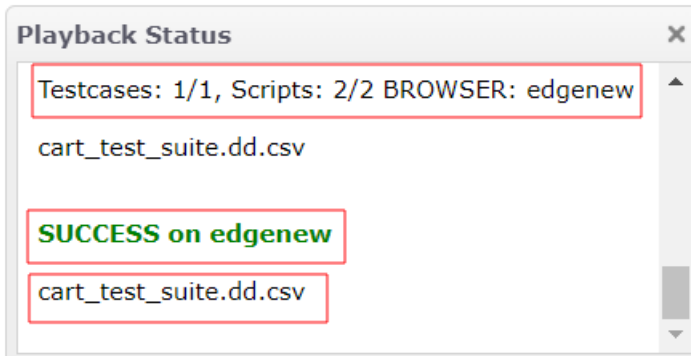
The screenshot shows the Sahi IDE interface after saving the file. The 'Navigator' pane shows a new file 'new 2.dd.csv' created, highlighted with a red box. The main editor area displays the content of this file:

	A	B	C	D	E	F	G	H	I
1	#script	url	tags	depends	prerequisites	timeout_minutes			
2	cart.s.csv								
3	login.sah								
4									
5									
6									
7									
8									
9									
10									
11									
12									

Let's playback the suite using 'Playback' button.



Playback happens parallelly opening 2 edge browsers.



Logs show the graph showing the scripts and their run time. User can see the time taken for running the scripts. Clicking on the test link, Sahi opens detailed report of the execution.

Suite Report | Test Cases Report | JS Code Coverage Report | Suite Analysis

Run Time (Sec)

Scripts

Suite Name: cart_test_suite.dd.csv | Auto Refresh

Browser Type: edgenew | Total scripts: 2
 Start Time: Apr 25, 2023 05:34:34 PM | Scripts passed: 2
 End Time: Apr 25, 2023 05:34:48 PM | Scripts failed: 0
 Time Taken: 00:00:14 050 | Scripts errored: 0
 Scripts skipped: 0
 Scripts aborted: 0
 Scripts aborted timeout: 0
 Scripts not supported: 0

Status: **SUCCESS**

Scripts:

Test	Start Url	Steps: Total/Failed/Errors/%	Testcases: Total/Passed/Failed/%	Start Time	Time Taken	Node	Load	Intermediate Statuses: Healed, Failure, Error, NA	Final Status: Success, Failure, Error, Abort
cart_test_suite.dd.csv									
cart.s.csv	Http://sahite...	11 0 0 100 %	1 1 0 100 %	Apr 25, 2023 05:34:35 PM	00:00:05 771	localhost:9999	-1	NA	SUCCESS
login.sah	Http://sahite...	5 0 0 100 %	0 0 0 0%	Apr 25, 2023 05:34:38 PM	00:00:05 288	localhost:9999	-1	NA	SUCCESS

Data Driven Suites Logs Detailed View

Script Name: cart.s.csv Auto Refresh

Test	Total Steps	Failures	Errors	Success Rate	Time Taken	Node	Load	Browser	Intermediate Statuses	Final Status
cart.s.csv	11	0	0	100 %	00:00:05 771	localhost:9999	-1	Edge 112.0.1722.58	NA	SUCCESS

Report Id: cart_test_suite_edgenew__375218090cedb0414d0953001e9b553145bc | Compare Logs
[Script Info](#)

Total Test Cases	Passed	Failed	Success Rate
1	1	0	100 %

Test Case Id	Description	Time Taken	Intermediate Statuses	Final Status
Test Grand Total	Add books to cart and verify total	00:00:02 606	NA	SUCCESS

Starting script [Expand All](#) [Collapse All](#) | Show Failed | Show All Images | Convert To English

loadSahi	cart_library.sah		
Test Grand Total	[Documentation]	Add books to cart and verify total	
cartLogin	test	MgkKEQBU	
[-] cartLogin("test", "MgkKEQBU")			
<pre> _setValue(_textbox("user"), "test") [3241 ms] [05:34:39.222 PM] _setPassword(_password("password"), "*****") [194 ms] [05:34:39.416 PM] _click(_submit("Login")) [297 ms] [05:34:39.713 PM] </pre>			
addItemToCart	java Q:1	ruby Q:2	python Q:2
verify Grand Total	Total:1400		
cart Logout			

Stopping script

Script Name: login.sah Auto Refresh

Test	Total Steps	Failures	Errors	Success Rate	Time Taken	Node	Load	Browser	Intermediate Statuses	Final Status
login.sah	5	0	0	100 %	00:00:05 288	localhost:9999	-1	Edge 112.0.1722.58	NA	SUCCESS

Report Id: cart_test_suite_edgenew__375218090cedb0414d0953001e9b553145bc | Compare Logs
[Script Info](#)

Starting script [Expand All](#) [Collapse All](#) | Show Failed | Show All Images | Convert To English

```

_navigateTo("http://sahitest.com/demo/training") [3793 ms] [05:34:41.822 PM]
_setValue(_textbox("user"), "test") [383 ms] [05:34:42.205 PM]
_setPassword(_password("password"), "*****") [133 ms] [05:34:42.338 PM]
_click(_submit("Login")) [220 ms] [05:34:42.558 PM]
_click(_button("Logout")) [723 ms] [05:34:43.281 PM]
                    
```

Stopping script

BDTA – Business-Driven Test Automation

Business-Driven Test Automation (BDTA) allows test automation to begin much earlier in the project lifecycle - right at the feature conceptualization stage. The application under test need not be ready to begin BDTA. With BDTA, prior to the readiness of your feature or application, we can:

1. Specify the desired application flow using straightforward language, employing key words for various actions (e.g., login, add books, etc.).

2. Incorporate optional parameters into different steps based on specific testing requirements, allowing for future adjustments or additions when the application is fully developed (e.g., login | username: test | password: |).
3. Implement data-driven testing for scenarios that involve extensive data manipulation, enabling the creation and population of relevant data input files.
4. Organize our tests and scenarios into suites, ensuring a structured and systematic approach to test automation management.
5. Utilize tags to logically separate tests and enhance control over their execution, enabling better test management and organization.
6. Seek verification and validation of our test automation efforts by involving peers and managers to review and provide feedback.
7. Version control your test automation artifacts by checking them into a designated repository, ensuring proper tracking and management of changes over time.

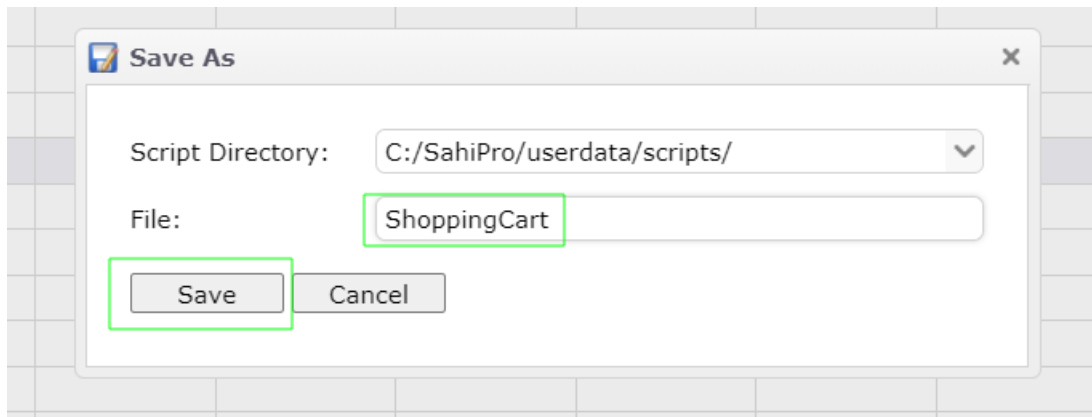
BDTA Implementation using Sahi:

let us imagine that we are going to work on a simple shopping application wherein the user can purchase books and the use cases to add books to their cart and verify the cart total amount now let us see how we can implement it using business driven test automation.

Open Sahi Pro Editor. We first create a new scenario file. Our first use case is login we write that in Key Word column, a login will need a username and a password typically so we can add that data as a parameter name:value in Argument 1 & argument 2 columns. Next step is to add books, we do not yet know what books we will be adding so we can leave parameters empty.

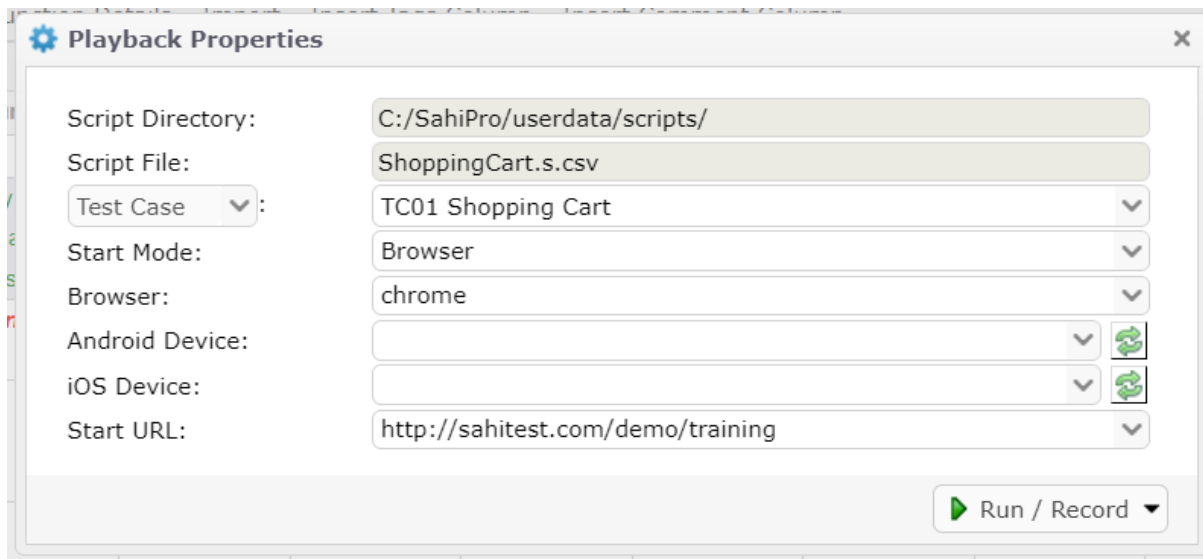
	A	B	C	D	E
	TestCase	Key Word	Argument 1	Argument 2	
		▼			
	TC01 Shopping Cart	▼ [Documentation]	Verify cart total after adding books		
		User Login ▼	username : test	password :	
		Add Programming Books ▼			
		Verify Total Amount ▼			
		User Logout ▼			
		▼			

Save the file.

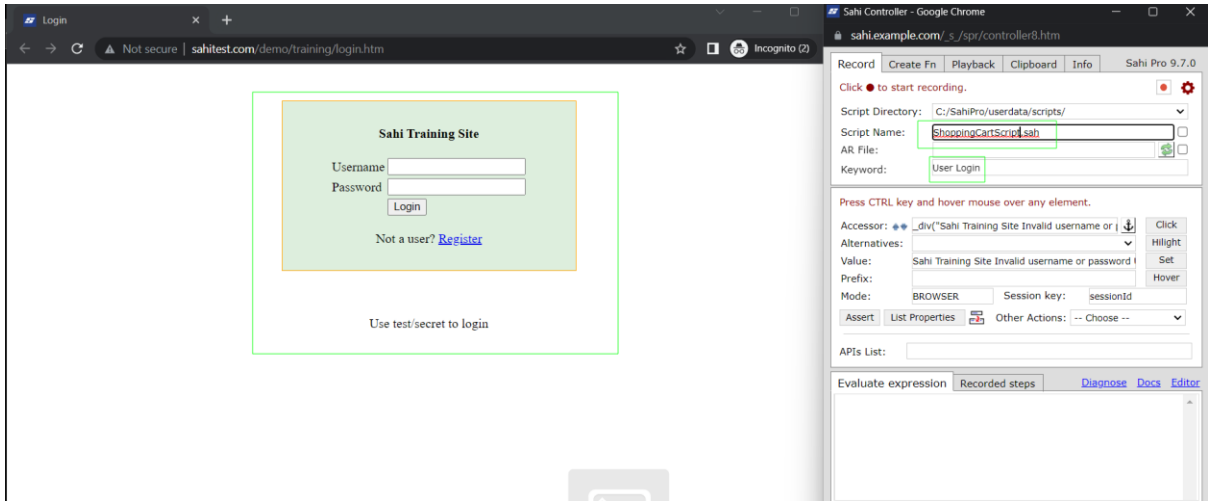


Now we will begin implementing the key words listed in the above file.

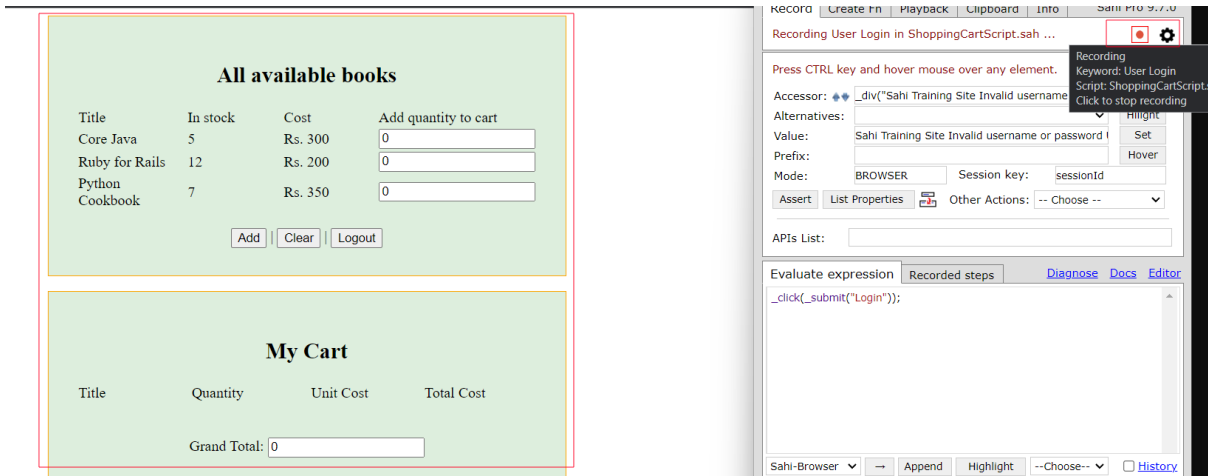
Right click on the test case name and select 'Run/Record'. Enter start URL and click Run/Record.

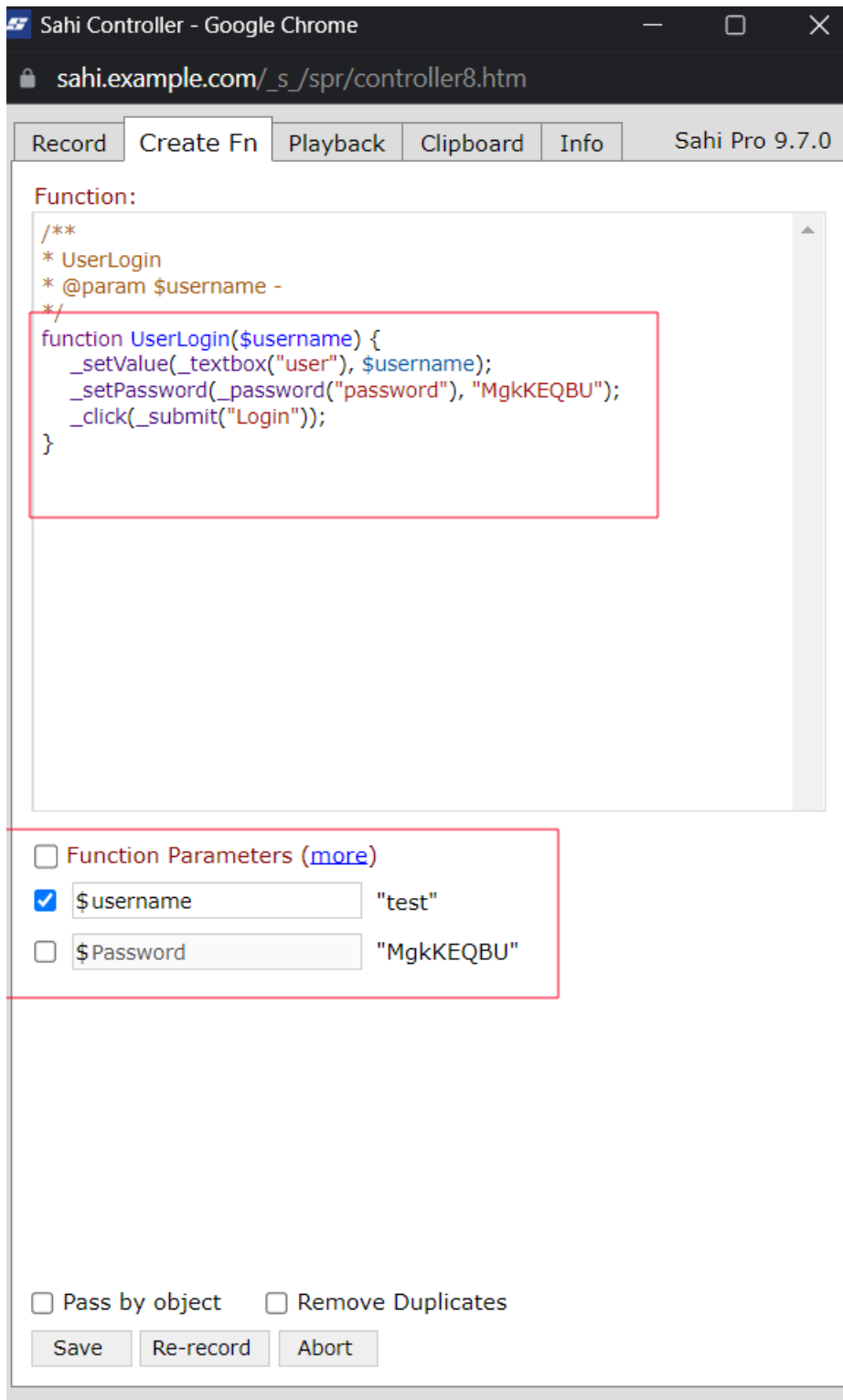


Now about run record this feature runs the selective test case with the intent of recording any missing or non-implemented keywords or functions as you begin it looks in the entire database of our Sahi scripts looks for a function with the keyword name and if not found prompts you to record it by opening the Sahi pro controller.



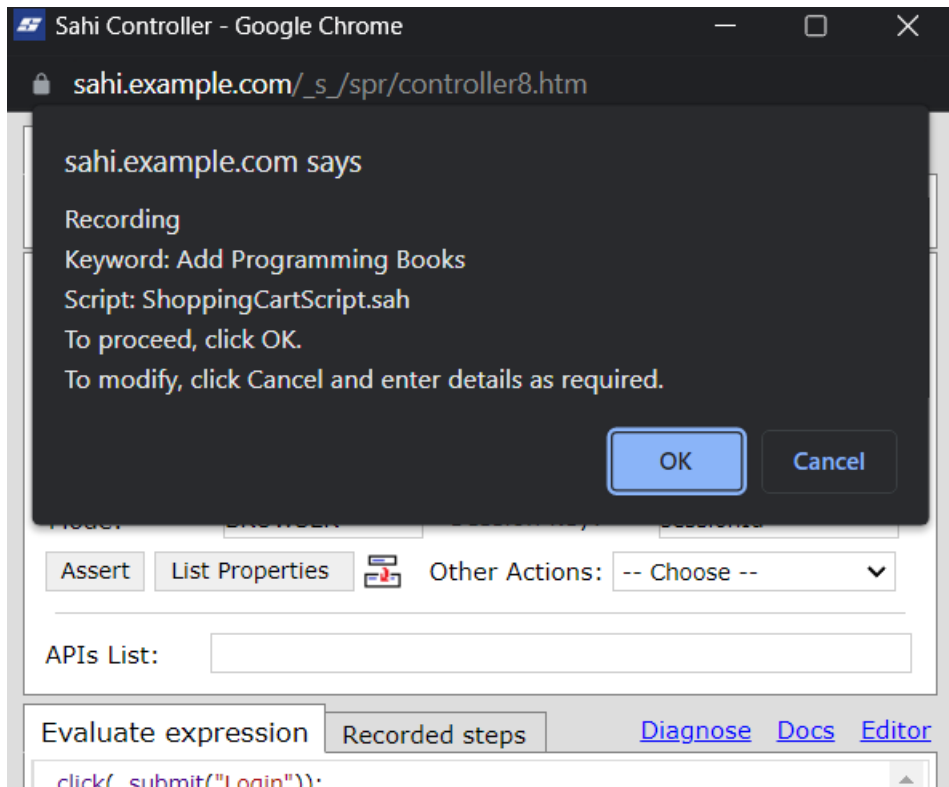
After login is done, select red button to stop recording.



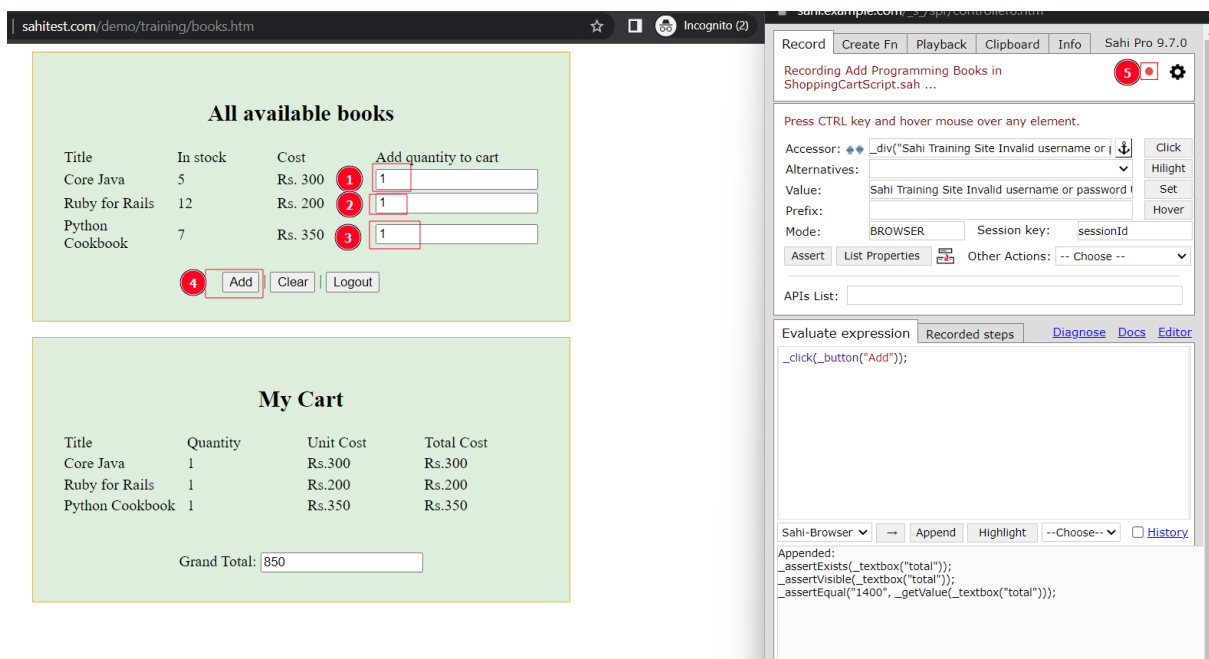


Sahi opens Create Function tab to tell us the parameters used. We can select the additional parameters if the required attribute is not default selected. The default selection is done when the parameter value in Sahi scenario file matches the input value used by the user.

Click on save to save the function.



Now if the next step in the scenario is already implemented, Sahi continues the playback, else it will prompt to record the function.



Now click on stop record button to see the recorded steps in the function.

Record Create Fn Playback Clipboard Info Sahi Pro 9.7.0

Function:

```
/**
 * AddProgrammingBooks
 */
function AddProgrammingBooks() {
  _click(_row("Core Java5Rs. 300"));
  _setValue(_textbox("q"), "1");
  _click(_row("Ruby for Rails12Rs. 200"));
  _setValue(_textbox("q[1]"), "1");
  _click(_row("Python Cookbook7Rs. 350"));
  _setValue(_textbox("q[2]"), "1");
  _click(_div("All available books TitleIn stockCostAdd quantity to cart Cc
  _click(_button("Add"));
}
```

Function Parameters ([more](#))

1 \$Q "1"

2 \$Q1 "1"

3 \$Q2 "1"

Pass by object Remove Duplicates

4



Now Sahi will prompt us to record Verify Total Amount function.

The image shows a screenshot of the Sahi browser interface. At the top, the address bar displays `sahi.example.com/_s_/spr/controller8.htm`. A dark dialog box is overlaid on the page, containing the following text: "sahi.example.com says", "Recording", "Keyword: Verify Total Amount", "Script: ShoppingCartScript.sah", "To proceed, click OK.", and "To modify, click Cancel and enter details as required." The "OK" button in the dialog is highlighted with a red box. Below the dialog, the interface includes a toolbar with buttons for "Assert", "List Properties", and "Other Actions: -- Choose --". An "APIs List:" field is also present. The main area is divided into tabs: "Evaluate expression", "Recorded steps", "Diagnose", "Docs", and "Editor". The "Evaluate expression" tab is active, showing a script: `_click(_button("Add"));`. Below the script editor, there are controls for "Sahi-Browser", "Append", "Highlight", and a dropdown menu. A "History" checkbox is also visible. At the bottom, the "Appended:" section shows a list of assertions: `_assertExists(_textbox("total"));`, `_assertVisible(_textbox("total"));`, and `_assertEqual("1400", _getValue(_textbox("total")));`. A small "c" is located at the bottom right corner of the screenshot.




Hover on total amount element


sahi.example.com/_s_/spi/controller8.htm

Record Create Fn Playback Clipboard Info Sahi Pro 9.7.0

Recording Verify Total Amount in ShoppingCartScript.sah ...  

Press CTRL key and hover mouse over any element.



Accessor:    Click

Alternatives:  Highlight

Value: Set

Prefix: Hover



Mode: Session key:

 Other Actions: 

APIs List:

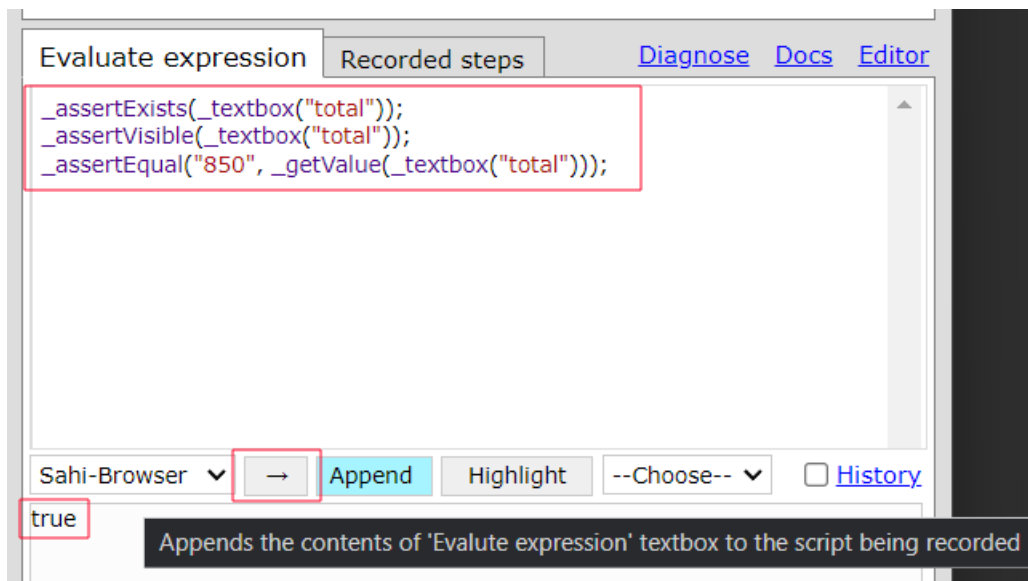
Evaluate expression Recorded steps [Diagnose](#) [Docs](#) [Editor](#)

```
_click(_button("Add"));
```

Sahi-Browser  →  [History](#)

Appended:
`_assertExists(_textbox("total"));`
`_assertVisible(_textbox("total"));`
`_assertEqual("1400", _getValue(_textbox("total")));`

And add assert statements.



Append the assertions using Append button to the current function. Stop the recording.

sahi.example.com/_s_/spr/controller8.htm

Record Create Fn Playback Clipboard Info Sahi Pro 9.7.0

Function:

```
/**
 * VerifyTotalAmount
 */
function VerifyTotalAmount() {
  1 _assertExists(_textbox("total"));
  _assertVisible(_textbox("total"));
  _assertEqual("850", _getValue(_textbox("total")));
}
```

2 Function Parameters ([more](#))

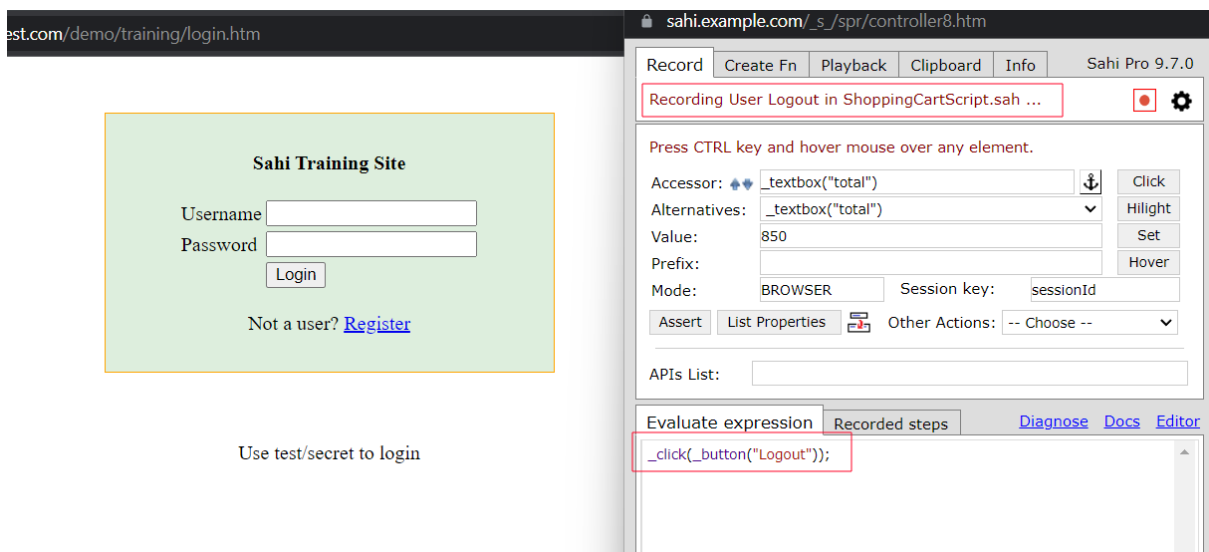
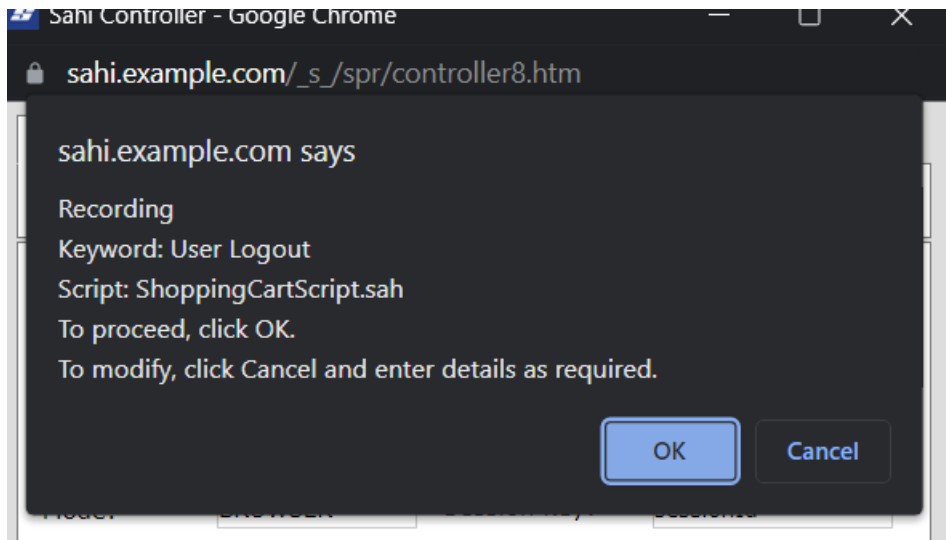
\$Total "850"

Pass by object Remove Duplicates

3 Save Re-record Abort

Add Total to function parameters and save the function.

Sahi will prompt us to record Logout functionality in User Logout function.



Stop the recording and save the function.

This will close the Sahi controller and goes back to editor.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	TestCase	Key Word	Argument 1	Argument 2									
2		loadSahi	ShoppingCartScript.sah										
3													
4	TC01 Shopping Cart	[Documentation]	Verify cart total after adding books										
5		User Login	username : test	Password : M9kKQEQU									
6		Add Programming Books	Q : 1	Q1 : 1	Q2 : 1								
7		Verify Total Amount	Total : 850										
8		User Logout											
9													
10													
11													
12													
13													
14													
15													
16													
17													
18													
19													
20													
21													
22													
23													

Playback Status

Testcases: 0/0, Scripts: 0/1 BROWSER: chrome

ShoppingCart.s.csv

SUCCESS on chrome

ShoppingCart.s.csv

We can ctrl+click on ShoppingCartScript.sah to see the script file and implemented functions.

```

1 /**
2 * UserLogin
3 * @param $username -
4 * @param $password -
5 */
6 function UserLogin($username, $password) {
7     _setValue(_textbox("user"), $username);
8     _setPassword(_password("password"), $password);
9     _click(_submit("Login"));
10 }
11 /**
12 * AddProgrammingBooks
13 * @param $Q -
14 * @param $Q1 -
15 * @param $Q2 -
16 */
17 function AddProgrammingBooks($Q, $Q1, $Q2) {
18     _click(_row("Core Java5Rs. 300"));
19     _setValue(_textbox("q"), $Q);
20     _click(_row("Ruby for Rails12Rs. 200"));
21     _setValue(_textbox("q[1]", $Q1);
22     _click(_row("Python Cookbook7Rs. 350"));
23     _setValue(_textbox("q[2]", $Q2);
24     _click(_div("All available books TitleIn stockCostAdd quantity to cart Core Java5Rs. 300 Ruby for Rails12Rs. 200 Python Cookbook7Rs. 350 | I"));
25     _click(_button("Add"));
26 }
27 /**
28 * VerifyTotalAmount
29 * @param $Total -
30 */
31 function VerifyTotalAmount($Total) {
32     _assertExists(_textbox("total"));
33     _assertVisible(_textbox("total"));
34     _assertEqual($Total, _getValue(_textbox("total")));
35 }
36 /**
37

```

Playback Status

Testcases: 0/0, Scripts: 0/1 BROWSER: chrome

ShoppingCart.s.csv

SUCCESS on chrome

ShoppingCart.s.csv

User can now run the playback using Playback functionality.

Playback Properties

Script Directory: C:/SahiPro/userdata/scripts/
 Script File: ShoppingCart.s.csv
 Test Case: TC01 Shopping Cart
 Start Mode: Browser
 Browser: chrome
 Android Device:
 iOS Device:
 Start URL: http://sahitest.com/demo/training

Run in parallel Threads: 5
 Run sequentially in single browser session
 Run distributed (multiple machines)

[Show advanced settings](#)

Reset Create Ant Target Run

Books Not secure | saहितest.com/demo/training/books.htm Incognito

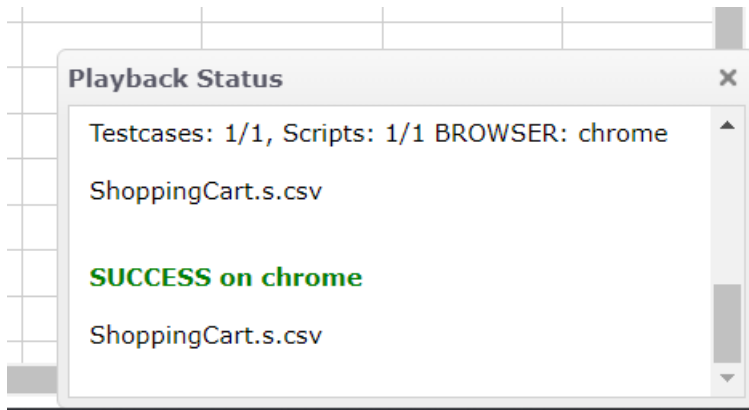
All available books

Title	In stock	Cost	Add quantity to cart
Core Java	5	Rs. 300	<input type="text" value="1"/>
Ruby for Rails	12	Rs. 200	<input type="text" value="1"/>
Python Cookbook	7	Rs. 350	<input type="text" value="1"/>

My Cart

Title	Quantity	Unit Cost	Total Cost
Core Java	1	Rs.300	Rs.300
Ruby for Rails	1	Rs.200	Rs.200
Python Cookbook	1	Rs.350	Rs.350

Grand Total:



Seeing logs will report all the steps executed.

localhost:9999/_s/dyn/pro/DBReports_scriptReport?id=ShoppingCart_chrome_903e703607cde041110b9810708580777ab0

Suites | Suite Report | Test Cases Report | **Script Report** | JS Code Coverage Report | Suite Analysis

Script Name: ShoppingCart.s.csv | Auto Refresh

Test	Total Steps	Failures	Errors	Success Rate	Time Taken	Node	Load	Browser	Intermediate Statuses	Final Status
ShoppingCart.s.csv	15	0	0	100 %	00:00:04 842	localhost:9999	-1	Chrome 112.0.0.0	NA	SUCCESS

Report Id: ShoppingCart_chrome__a89c7e6609c2904a8a0983d0548b23f204ad | Compare Logs
[Script Info](#)

Total Test Cases	Passed	Failed	Success Rate
1	1	0	100 %

Test Case Id	Description	Time Taken	Intermediate Statuses	Final Status
TC01 Shopping Cart	Verify cart total after adding books	00:00:03 398	NA	SUCCESS

Starting script [Expand All](#) [Collapse All](#) | Show Failed | Show All Images | Convert To English

	loadSahi	ShoppingCartScript.sah
TC01 Shopping Cart	[Documentation]	Verify cart total after adding books
	User Login	username:test Password:MgkKEQBU
	Add Programming Books	Q:1 Q1:1 Q2:1
	Verify Total Amount	Total:850
	User Logout	

Stopping script

User Login	username:test	Password:MgkKEQBU
------------	---------------	-------------------

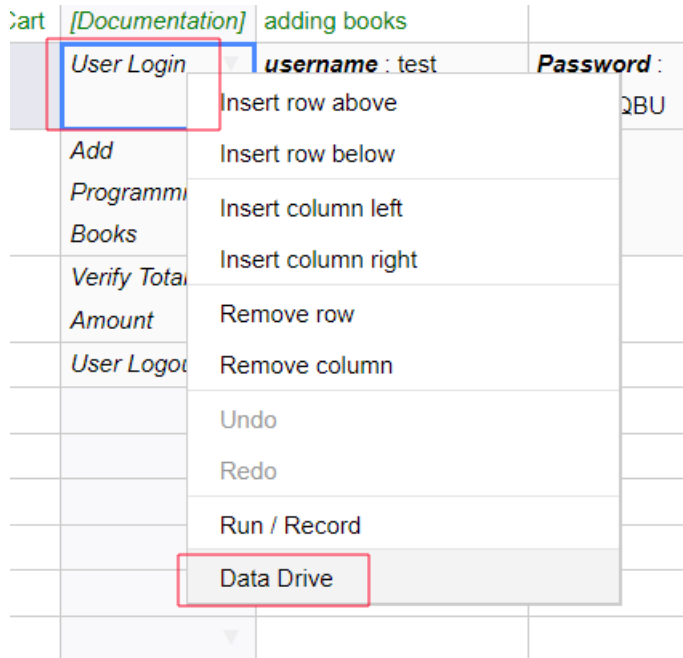
```
[-] UserLogin("test", "MgkKEQBU")
  _setValue(_textbox("user"), "test") [1506 ms] [06:43:49.734 PM]
  _setPassword(_password("password"), "*****") [197 ms] [06:43:49.931 PM]
  _click(_submit("Login")) [145 ms] [06:43:50.076 PM]
```

Data Drive Test Cases in Sahi

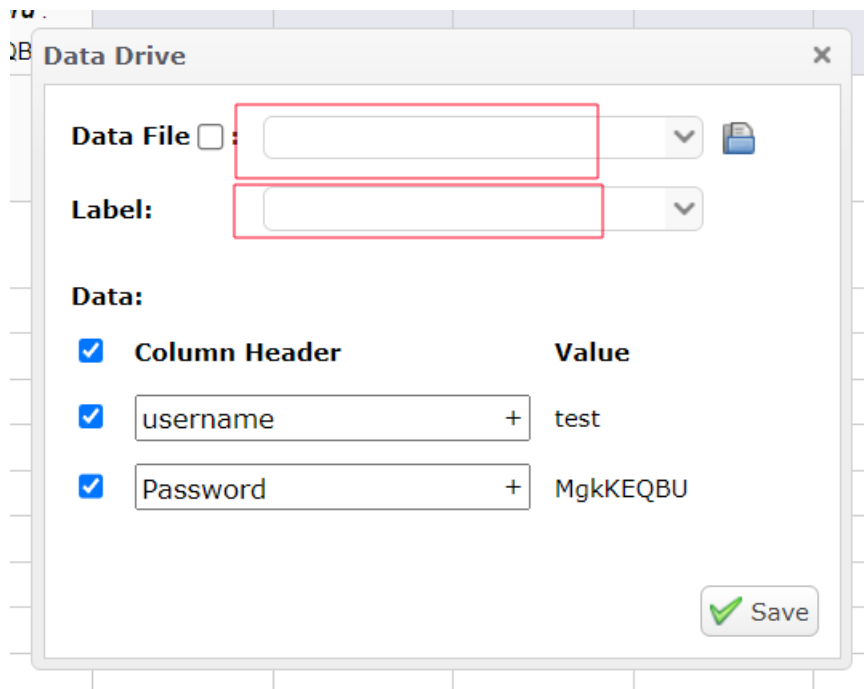
We take the scenario we implemented further by data driving our tests. Testers often need to keep the data separate from their test cases, there could be two reasons, firstly we may need to access the same data for multiple test cases and secondly a test case or some of its steps

may need to be repeated with multiple sets of data. So, we need a framework to enable a test case to be run repeatedly with multiple sets of data. Let's data drive the login scenario using different data.

Right click on the key word and select Data Drive.



We'll get a pop up to enter data file name and label.



Data Drive [X]

Data File : ShoppingCartData [v] [file icon]

Label: Credentials [v]

Data:

Column Header **Value**

username + test

Password + MgkKEQBU

[Save]

Click Save.

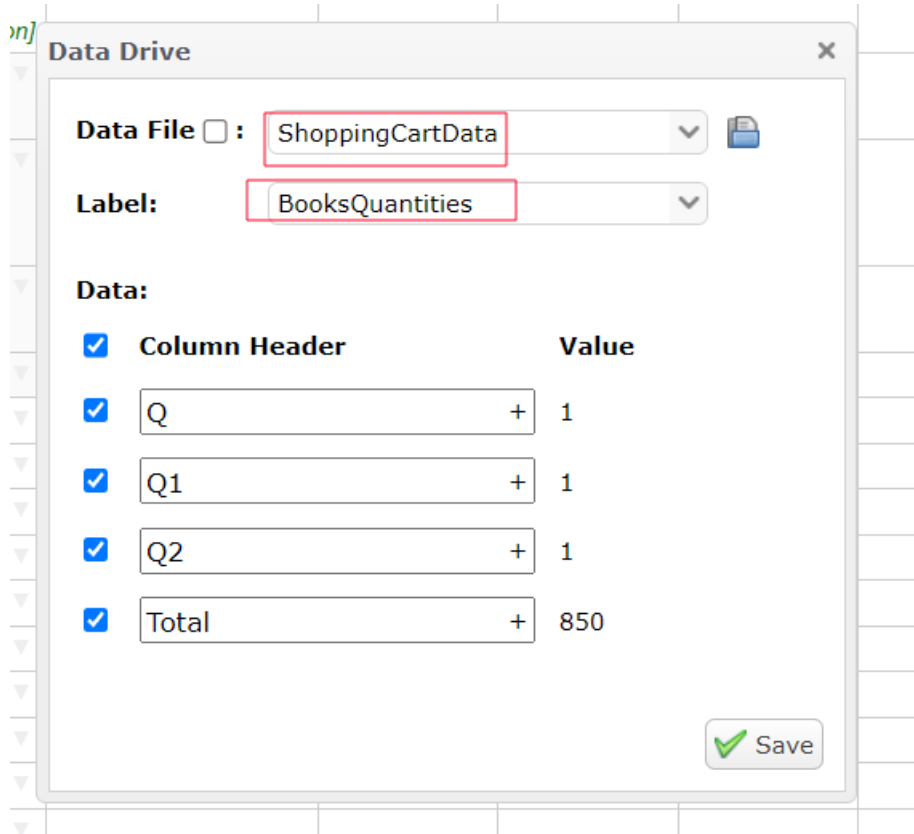
TC01			Verify cart total after adding books		
Shopping Cart	[ShoppingCartData.csv:Credentials]	[Documentation] User Login	username : test	Password : MgkKEQBU	
		Add	Q : 1	Q1 : 1	Q2 : 1
		Programming			

Let's see data drive for multiple rows. Selecting Add Programming Books and Verify Total Amount steps. Right click and select data drive.

Shopping Cart	[ShoppingCartData.csv:Credentials]	[Documentation] adding books			
	[ShoppingCartData.csv:Credentials]	User Login	username : test	Password : MgkKEQBU	
		Add	Q : 1	Q1 : 1	Q2 : 1
		Programming			
		Books			
		Verify Tot			
		Amount			
		User Log			

- Insert row above
- Insert row below
- Insert column left
- Insert column right
- Remove rows
- Remove columns
- Undo
- Redo
- Run / Record
- Data Drive**

[Play]



Click Save button.

Now we can see that both steps are wrapped in [Repeat] [End] loop.

[ShoppingCartData.csv:Credentials]	User Login	username : test	Password : MgkKEQBU	
[ShoppingCartData.csv:BooksQuantities]	[Repeat]			
	Add Programming Books	Q : 1	Q1 : 1	Q2 : 1
	Verify Total Amount	Total : 850		
	[End]			
	User Logout			

ShoppingCartData file can be seen by clicking ctrl+file name in the current file.

ShoppingCart.s.csv c * ShoppingCartData.csv c *

	A	B	C	D	E
1	Credentials	username	Password		
2		test	MgkKEQBU		
3					
4	BooksQuantities	Q	Q1	Q2	Total
5		1	1	1	850
6					
7					

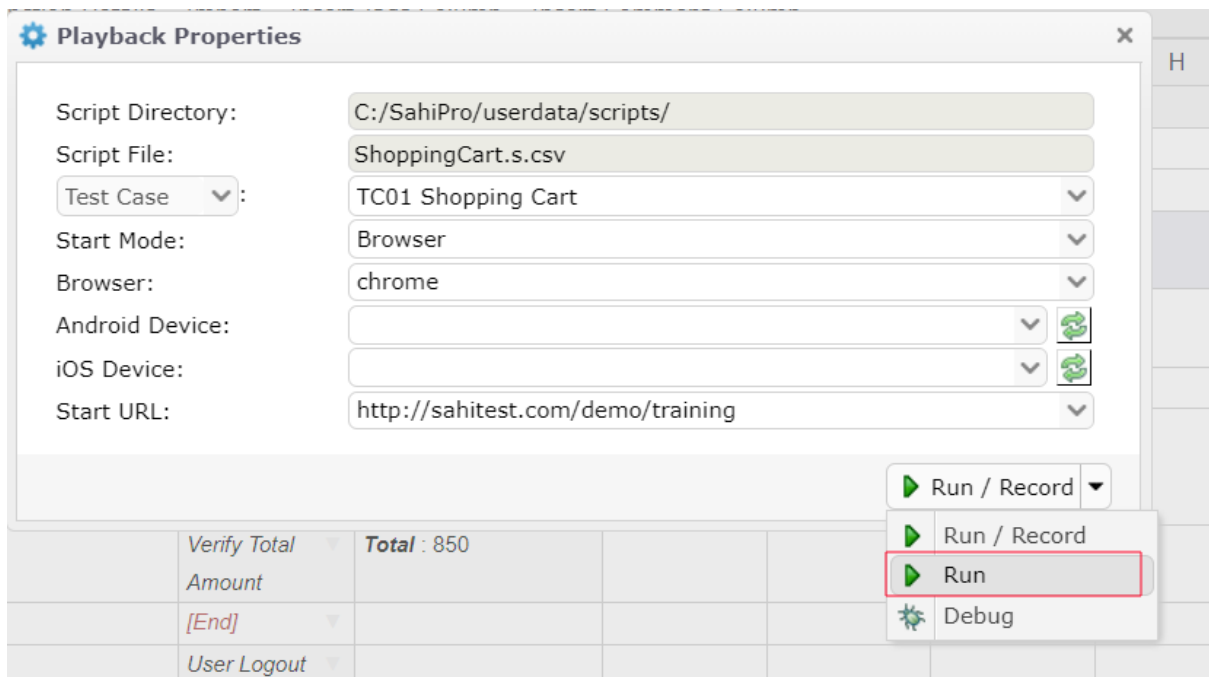
Now let's add some more data to BooksQuantities to run the test multiple times.

	A	B	C	D	E	F
1	Credentials	username	Password			
2		test	MgkKEQBU			
3						
4	BooksQuantities	Q	Q1	Q2	Total	
5		1	1	1	850	
6		10	5	3	5050	
7		0	0	0	0	
8		-2	ab	-5	0	
9						
10						

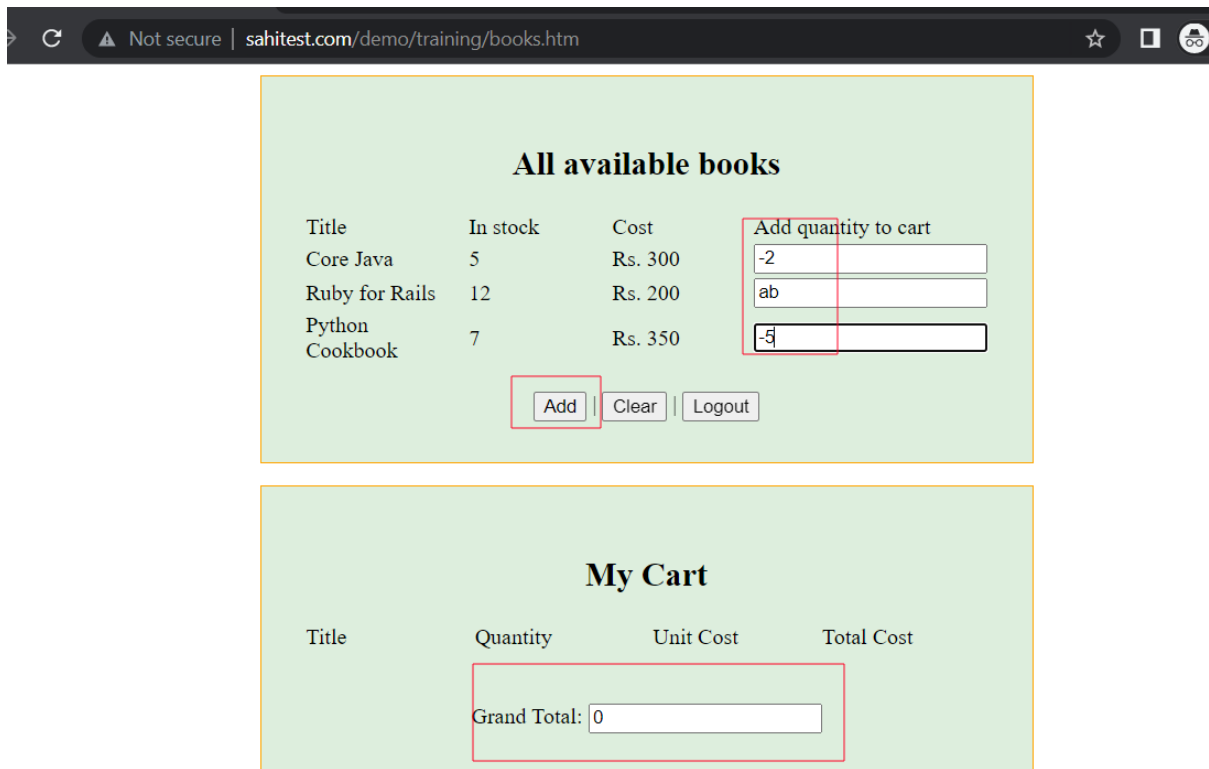
Let's run the test case. Right click on the test case name and select Run/record.

TC01	Insert row above	
Shopp	Insert row below	
	Insert column left	ntial
	Insert column right	Qua
	Remove row	
	Remove column	
	Undo	
	Redo	
	Run / Record	
	Data Drive	

In the Run/Record dialog select Run



We can notice that multiple times books are added according to the data we provided in the data file.



Let's look at the logs:

Suites | Suite Report | Test Cases Report | **Script Report** | JS Code Coverage Report | Suite Analysis

Script Name: ShoppingCart.s.csv | Auto Refresh

Test	Total Steps	Failures	Errors	Success Rate	Time Taken	Node	Load	Browser	Intermediate Statuses	Final Status
ShoppingCart.s.csv	48	0	0	100 %	00:00:11.503	localhost:9999	-1	Chrome 112.0.0.0	NA	SUCCESS

Report Id: ShoppingCart_chrome__4248a3d90f350426c09e72061b17e9ecf1 | Compare Logs
[Script Info](#)

Total Test Cases	Passed	Failed	Success Rate
1	1	0	100 %

Test Case Id	Description	Time Taken	Intermediate Statuses	Final Status
TC01 Shopping Cart	Verify cart total after adding books	00:00:10.062	NA	SUCCESS

Starting script [Expand All](#) | [Collapse All](#) | Show Failed | Show All Images | Convert To English

loadSahi ShoppingCartScript.sah

TC01 Shopping Cart [Documentation] Verify cart total after adding books

[ShoppingCartData.csv:Credentials] User Login username:test Password:MgkKEQBU

username	Password
test	MgkKEQBU

[ShoppingCartData.csv:BooksQuantities] [Repeat]

Q	Q1	Q2	Total
1	1	1	850
10	5	3	5050
0	0	0	0
-2	ab	-5	0

[End]

User Logout

Stopping script

Data driven steps are shown with yellow heading table and data is listed in the table below. Clicking on data reveals the functions called using the data.

oksQuantities [Repeat]

Q	Q1	Q2	Total
1	1	1	850

```

[-] AddProgrammingBooks(1, 1, 1)
  _click_row("Core Java5Rs. 300") [890 ms] [07:57:59.916 PM]
  _setValue(_textbox("q"), 1) [125 ms] [07:58:00.041 PM]
  _click_row("Ruby for Rails12Rs. 200") [242 ms] [07:58:00.283 PM]
  _setValue(_textbox("q[1]"), 1) [143 ms] [07:58:00.426 PM]
  _click_row("Python Cookbook7Rs. 350") [232 ms] [07:58:00.658 PM]
  _setValue(_textbox("q[2]"), 1) [142 ms] [07:58:00.800 PM]
  _click_div("All available books TitleIn stockCostAdd quantity to cart Core Java5Rs. 300 Ruby for Rails12Rs. 200 Python Cookbook7Rs. 350 | (*)") [241 ms] [07:58:01.041 PM]
  _click_button("Add") [253 ms] [07:58:01.294 PM]
[+] VerifyTotalAmount(850)
  
```

Editing and Maintaining Test Scripts

When a change happens in business use case or business scenarios, tester can add new requirements to the existing test case without disrupting other steps. In our above test case, let's say we must clear cart before logging out. User can do this step easily by adding step to the file by right clicking and clicking Insert row above option.

Amount
[End]

Insert row above

Let's add function Clear to clear the cart. User can select run/record option to record the mentioned unimplemented test step.

	Amount	
	[End]	▼
	Clear	▼
	User Logout	▼
		▼

TC01
Shoppi

- Insert row above
- Insert row below
- Insert column left
- Insert column right
- Remove row
- Remove column
- Undo
- Redo
- Run / Record
- Data Drive

entials]

Quantil

0

1

2

Sahi runs the implemented functions. If it encounters any function that is not yet recorded, it'll open Controller to record the function.

The screenshot shows a web browser window with two pages: "All available books" and "My Cart". The "All available books" page has a table with columns: Title, In stock, Cost, and Add quantity to cart. The "My Cart" page has a table with columns: Title, Quantity, Unit Cost, and Total Cost, and a "Grand Total" field.

The Sahi Controller interface is open on the right, showing a recording window with the following details:

- Recording
- Keyword: Clear
- Script: sahi_sample_run.sah
- To proceed, click OK.
- To modify, click Cancel and enter details as required.

The "OK" button is highlighted in red. Below the recording window, there are options for "Assert", "List Properties", and "Other Actions". The "Evaluate expression" tab is selected, showing a list of recorded steps and a script editor with the following code:

```

_assertExists(_textbox("total"));
_assertVisible(_textbox("total"));
_assertEqual("1400", _getValue(_textbox("total")));

```

We click on clear button to add it to steps list, next we verify if the form cleared or not using control + hover on any of the quantity input fields. And then use assert button to add an assertion to the script.

Record Create Fn Playback Clipboard Info Sa

Function:

```
/**
 * Clear
 */
function Clear() {
  _click(_button("Clear"));
  _assertEqual("", _getValue(_textbox("q")));
}
```

Function Parameters ([more](#))

\$Q

Pass by object Remove Duplicates

Sahi goes through the clear step and will logout on cart application and stops.

The screenshot shows the Sahi IDE interface with a test case table and a playback status window. The table has columns A through K. The test case is named 'Shopping Cart' (TC01) and includes steps like 'User Login', 'Add Programming Books', and 'Verify Total Amount'. A 'Clear' button is highlighted in the table. The playback status window shows 'SUCCESS on chrome' for the test case 'ShoppingCart.s.csv'.

1	A	B	C	D	E	F	G	H	I	J	K
1	TestCase	Data	Key Word	Argument 1	Argument 2						
2			loadSahi	sahi_sample_run.sah							
3			loadSahi	ShoppingCartScript.sah							
4											
5	TC01	Shopping Cart	[Documentation]	Verify cart total after adding books							
6		[ShoppingCartData.csv:Credentials]	User Login	username : test	Password : MggKEQBU						
7		[ShoppingCartData.csv:BooksQuantities]	[Repeat]								
8			Add Programming Books	Q : 1	Q1 : 1	Q2 : 1					
9			Verify Total Amount	Total : 850							
10			[End]								
11			Clear								
12			User Logout								

In logs:

The screenshot shows the Sahi logs for the 'Clear' action. The log entry is: `[-] Clear()`. Below it, the log shows the execution of `_click(_button("Clear"))` and `_assertEqual("", _getValue(_textbox("q")))`.

```

[End]
Clear
[-] Clear()
_click(_button("Clear")) [343 ms] [09:38:19.431 PM]
_assertEqual("", _getValue(_textbox("q"))) [123 ms] [09:38:19.554 PM]

```

Sahi Accessor Repository

To enhance script manageability and facilitate centralized editing, storage, and management of accessors, we aim to segregate accessors from script file code. Sahi's accessor repository functionality comes in handy here which allows us to create accessor repositories automatically either during the recording or thereafter using a single click.

Going to our functions library file, click on Create AR button in navigation bar.

The screenshot shows the Sahi IDE interface with the 'Create AR' dialog box open. The dialog box has fields for 'Script Directory', 'Script File', and 'AR File'. The 'AR File' field is highlighted with a red box. The 'Proceed' button is also highlighted with a red box.

```

1
2 /**
3 * UserLogin
4 * @param $username -
5 * @param $Password -
6 */
7 function UserLogin($username, $Password) {
8   _setValue(_textbox("user"), $username);
9   _setPassword(_password("password"), $Password);
10  _click(_submit("Login"));
11 }
12 /**
13 * AddProgrammingBooks
14 * @param $Q -
15 * @param $Q1 -
16 * @param $Q2 -
17 */
18 function AddProgrammingBooks($Q, $Q1, $Q2) {

```

Create AR

Script Directory: C:/SahiPro/userdata/scripts/

Script File: ShoppingCartScript.sah

AR File: ShoppingCartAR .ar.csv

Proceed Cancel

localhost:9999 says

AR file: C:/SahiPro/userdata/scripts/ShoppingCartAR.ar.csv created successfully.

OK

ShoppingCart.s.csv * ShoppingCartScript.sah * **ShoppingCartAR.ar.csv *** sahi_s

	A	B	C	D
1	Key	Value		
2	\$_TEXTBOX_USER	_textbox("user")		
3	\$_PASSWORD_PASSWORD	_password("password")		
4	\$_SUBMIT_LOGIN	_submit("Login")		
5	\$_ROW_COREJAVA5RS300	_row("Core Java5Rs. 300")		
6	\$_TEXTBOX_Q	_textbox("q")		
7	\$_ROW_RUBYFORRAILS12RS200	_row("Ruby for Rails12Rs. 200")		
8	\$_TEXTBOX_Q1	_textbox("q[1]")		
9	\$_ROW_PYTHONCOOKBOOK7RS350	_row("Python Cookbook7Rs. 350")		
10	\$_TEXTBOX_Q2	_textbox("q[2]")		
11	\$_BUTTON_ADD	_button("Add")		
12	\$_TEXTBOX_TOTAL	_textbox("total")		
13	\$_BUTTON_CLEAR	_button("Clear")		
14	\$_BUTTON_LOGOUT	_button("Logout")		

Changes in Implementation of Sahi Script

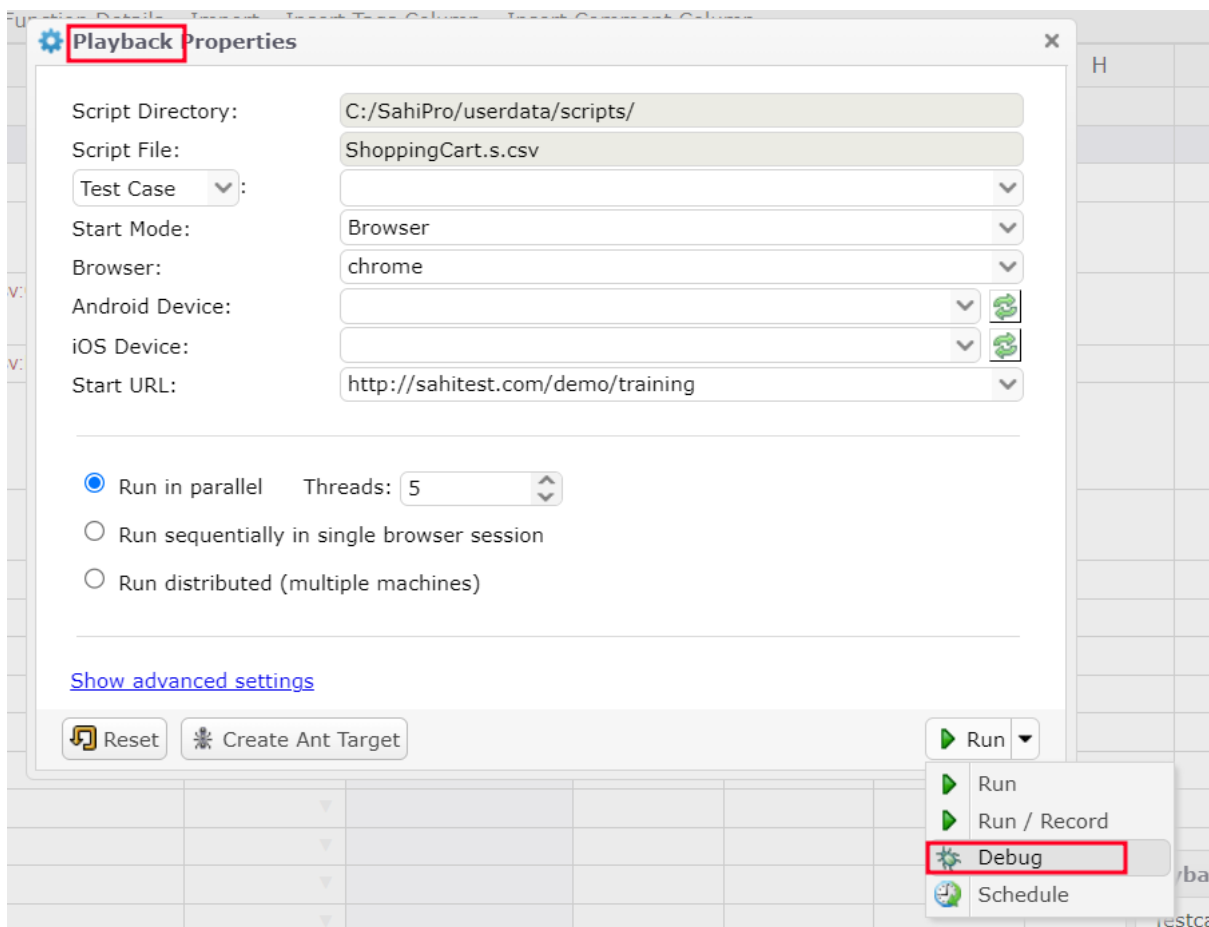
Let's say user want to change the accessor element for a particular element. To update the accessor, first method is to open the ar file and change the accessor(only change if value is known), second method is to put a break point at the line of element in sahi script and run the script in debug mode. During the run, script execution is stopped when break point is encountered. User can then change the element's accessor and the use Save AR button to save the AR to file.

4	\$_SUBMIT_LOGIN	_submit("Login")
5	\$_ROW_COREJAVA5RS300	_row("Core Java5Rs. 300")
6	\$_TEXTBOX_Q	_textbox("q")
7	\$ ROW RUBYFORRAILS12RS200	row("Rubv for

1st method

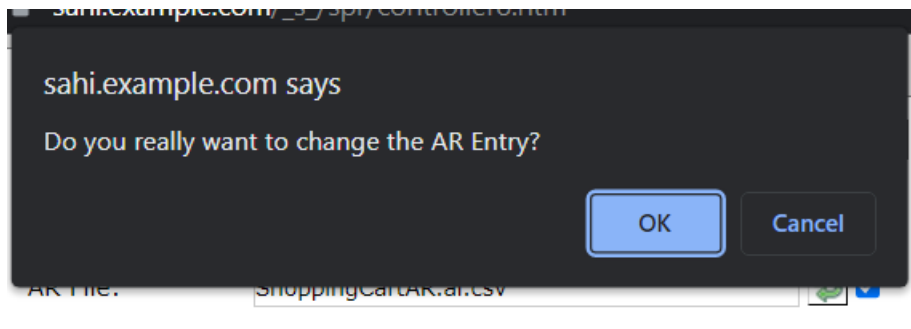
2nd method:

```
19 function AddProgrammingBooks($Q, $Q1, $Q2) {
20   _click($_ROW_COREJAVA5RS300);
21   _setValue($_TEXTBOX_Q, $Q); ← breakpoint
22   _click($_ROW_RUBYFORRAILS12RS200);
23   _setValue($_TEXTBOX_Q1, $Q1);
24   click($ ROW PYTHONCOOKBOOK7RS350):
```



Got to Record tab, and verify AR file path, enable AR repository checkmark, ctrl+hover on the required accessor. It's AR key and accessor will populate.

Select appropriate accessor and click on Save AR button.



Press CTRL key and hover mouse over any element.

AR Key:	<input type="text" value="\$ _TEXTBOX_Q"/>	<input type="button" value="SaveAR"/>
Accessor:	<input type="text" value='_textbox("q", _near(_cell("Core Java")))'/> <input type="button" value="Anchor"/>	<input type="button" value="Click"/>
Alternatives:	<input type="text" value='_textbox("q", _near(_cell("Core Java")))'/> <input type="button" value="Dropdown"/>	<input type="button" value="Highlight"/>
Value:	<input type="text" value="1"/>	<input type="button" value="Set"/>
Prefix:	<input type="text"/>	<input type="button" value="Hover"/>
Mode:	<input type="text" value="BROWSER"/>	Session key: <input type="text" value="sessionId"/>

Click ok and go to Playback tab to stop the run.

	A	B	C
1	Key	Value	
2	\$ _TEXTBOX_USER	_textbox("user")	
3	\$ _PASSWORD_PASSWORD	_password("password")	
4	\$ _SUBMIT_LOGIN	_submit("Login")	
5	\$ _ROW_COREJAVA5RS300	_row("Core Java5Rs. 300")	
6	\$ _CELL_COREJAVA	_cell("Core Java")	
7	\$ _TEXTBOX_Q	_textbox("q", _near(\$ _CELL_COREJAVA))	
8	\$ _ROW_RUBYFORRAILS12RS200	_row("Ruby for Rails12Rs.	

User can also change the key to match the new accessor of the element.

Go to the script file and Ctrl+click on the key, it opens context menu.

```

1  _includeAR("ShoppingCartAR.ar.csv", "Value");
2
3  /**
4  * UserLogin
5  * @param $username -
6  * @param $password -
7  */
8  function UserLogin($username, $password) {
9      _setValue($_TEXTBOX_USER, $username);
10     _setPassword($_PASSWORD_PASSWORD, $password);
11     _click($_SUBMIT_LOGIN);
12 }
13 /**
14 * AddProgrammingBooks
15 * @param $Q -
16 * @param $Q1 -
17 * @param $Q2 -
18 */
19 function AddProgrammingBooks($Q, $Q1, $Q2) {
20     _click($_ROW_COREJAVA5RS300);
21     _setValue($_TEXTBOX_Q, $Q);
22     _click($_ROW_RUBYFORRAILS12RS200);
23     _setValue($_TEXTBOX_Q1, $Q1);
24     _click($_ROW_PYTHONCOOKBOOK7RS350);
25     _setValue($_TEXTBOX_Q2, $Q2);
26     _click($_BUTTON_ADD);
27 }
28 /**
29 * VerifyTotalAmount
30 * @param $Total -
31 */
32 function VerifyTotalAmount($Total) {
33     _assertExists($_TEXTBOX_TOTAL);
34     _assertVisible($_TEXTBOX_TOTAL);
35     _assertEqual($_Total, _getValue($_TEXTBOX_TOTAL));
36 }

```

»
Context

File Path: Open Include

Function Name:

Arguments:

Show JsDoc Insert ↵ Insert Update

Accessor Repository (AR) Parameters

File Path: Open Include

Key: Edit AR Key

Value: Save AR Value

User can see the name of the AR file, key and value. It shows option to edit the key.

Find Usage - Functions and Variables

Find what: Find

Replace with: Replace

Exact Match: Function: Variable:

File Pattern:

(* = any string | eg: *.sah or D:/test.sah,*test.s.csv)

3 matches found in 2 files.

- ShoppingCartAR.ar.csv
- 5: var \$_TEXTBOX_Q = _textbox("q");
- ShoppingCartScript.sah
- 21: _setValue(\$_TEXTBOX_Q, \$Q);
- 42: _assertEqual("", _getValue(\$_TEXTBOX_Q));

»
Context

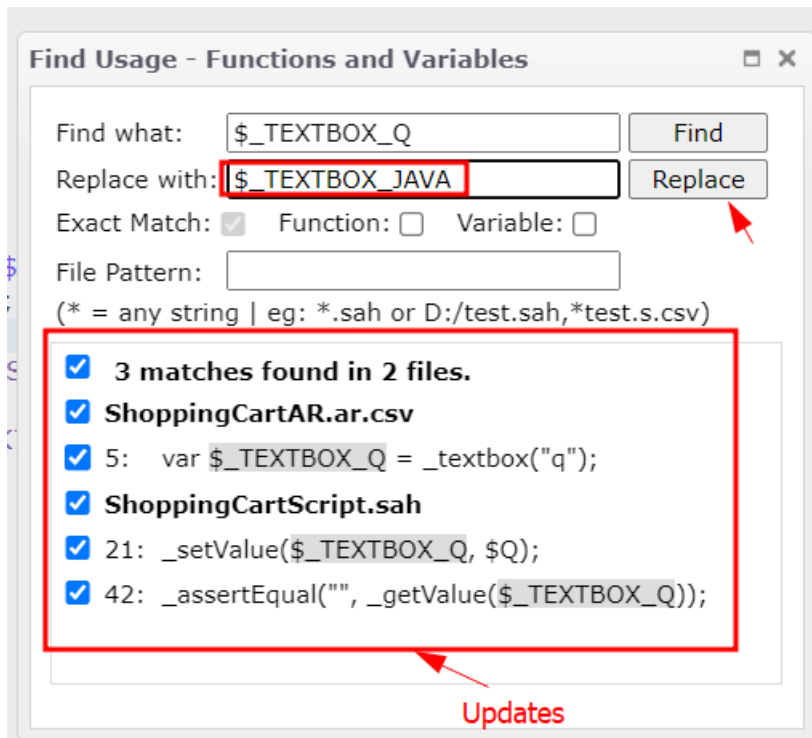
Show JsDoc Insert ↵ Insert Update

Accessor Repository (AR) Parameters

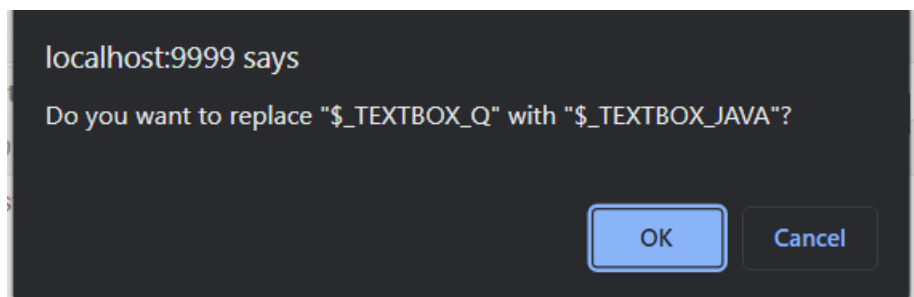
File Path: Open Include

Key: Edit AR Key

Value: Save AR Value



Click on Replace button. And click ok.



Updated key

```
function AddProgrammingBooks($Q, $Q1, $Q2) {
  _click($_ROW_COREJAVA5RS300);
  _setValue($_TEXTBOX_JAVA, $Q);
  _click($_ROW_RUBYFORRAILS12RS200);
  _setValue($_TEXTBOX_Q1, $Q1);
  _click($_ROW_PYTHONCOOKBOOK7RS350);
  _setValue($_TEXTBOX_Q2, $Q2);
  _click($_BUTTON_ADD);
}
/**
```

Updated key in ar file.

<input type="text" value="\$_TEXTBOX_JAVA"/>	<input q\",<br="" type="text" value="_textbox(\"/> _near(\$_CELL_COREJAVA))"/>
<input type="text" value="\$_ROW_RUBYFORRAILS12RS200"/>	<input d...\""="" type="text" value="\"/>

Sahi Testing Methods

Static Testing: Sahi is mainly used for automating applications. It also provides static testing to review source code of web applications.

Dynamic Testing: Dynamic testing is a technique that involves executing the software and observing its behaviour, often with the aim of identifying bugs or other issues.

Sahi Pro offers a wide range of features for dynamic testing of web applications. These include:

Recording and playback of test scripts: Sahi Pro allows users to record their interactions with a web application and play them back as test scripts, making it easy to automate repetitive testing tasks.

Cross-browser testing: Sahi Pro supports testing across a wide range of web browsers, allowing users to ensure that their applications function correctly across multiple platforms.

Data-driven testing: Sahi Pro supports data-driven testing, allowing users to test their application with different input data sets, to ensure that it functions correctly under different conditions.

Test reporting and analysis: Sahi Pro provides detailed reports and analysis of test results, allowing users to identify issues and track progress over time.

Black-box Testing: Black-box testing is a testing technique that focuses on the external behaviour of the software being tested, without looking at the internal workings of the code. In Sahi Pro, black-box testing can be performed by designing and executing test cases that focus on the functionality and user interface of the web application being tested.

Sahi Pro's record-and-playback feature, which allows users to record their interactions with the web application being tested and play them back as test scripts. This makes it easy to design and execute tests that focus on the user interface and functionality of the application.

Testing Levels

In Sahi Pro, unit tests can be written as functions that start with "test", such as "testUserNameMax50Characters" and "testPasswordAlphaNumeric".

For example, the "testPasswordAlphaNumeric" function could perform the following steps:

1. Generate a random password string of a length greater than 6.
2. Generate each character in the password string as a random alphanumeric character.
3. Verify that the operation of setting the password is allowed.

It is recommended to write comprehensive unit test cases and organize them in a single script for efficient testing.

To execute the tests, you can use the "_runUnitTests()" function. Running this function with no parameters will execute all tests. Alternatively, you can pass an array of specific test cases

to run only those tests. For instance, "_runUnitTests(["testUserNameMax50Characters", "testPasswordAlphaNumeric"])" will execute only the "testUserNameMax50Characters" and "testPasswordAlphaNumeric" test cases.

Operational Acceptance Testing can be done by running all the test cases in Sahi Pro using parallel test case execution. This testing verifies that a software application is ready to be deployed and used in a production environment.

Create test scripts: Create test scripts for each test case using Sahi Pro. These test scripts should include all the necessary actions and verifications required to validate the functionality of the application.

Execute the test scripts: Run the test scripts on the production-like environment. Ensure that the test environment is isolated from the actual production environment to avoid any unintended consequences.

Analyse the results: Analyse the test results to identify any defects or issues that need to be addressed before the application is deployed in the production environment.

Iterate and refine: If any defects or issues are identified, iterate, and refine the test cases and scripts until all issues are resolved.

Automatic correction and fix

Autoheal is a feature in Sahi Pro that enables automatic healing and recovery of failing scripts in web automation. It uses data from previous runs to identify alternate solutions when an element is not found, reducing failures and the need for manual intervention. Autoheal is only available for web automation in Sahi Pro.

Testing Types

Installation testing involves verifying that the software application can be installed successfully on different environments and configurations. Here are the steps to perform installation testing using SahiPro.

Compatibility testing involves verifying that the software application works correctly on different hardware, software, and network configurations.

Regression testing is a type of testing that is performed to ensure that changes made to the software application do not negatively impact existing functionality. In Sahi Pro, regression testing can be performed by re-executing existing test cases after making changes to the application and verifying that all existing functionality is still working as intended.

Steps to perform regression testing using Sahi Pro:

Identify the test cases: Identify the test cases that need to be re-executed as part of regression testing. These test cases should cover all the critical functionality of the application.

Create test scripts: Create test scripts for each identified test case using Sahi Pro. These test scripts should include all the necessary actions and verifications required to validate the functionality and performance of the application.

Execute the test scripts: Run the test scripts on the updated application and verify that all existing functionality is still working as intended.

Analyse the results: Analyse the test results to identify any defects or issues that need to be addressed before the updated application is deployed in the production environment.

Iterate and refine: If any defects or issues are identified, iterate, and refine the test cases and scripts until all issues are resolved.

Continuous testing is a software testing approach that involves automated testing throughout the software development lifecycle, from code changes to production deployment. Continuous testing ensures that issues are caught early in the development cycle, reducing the cost and effort of fixing defects later. In Sahi Pro, continuous testing can be achieved using the following steps:

Test automation: Write and automate tests using Sahi Pro for each new feature or change made to the application.

Continuous integration: Integrate automated tests into a continuous integration (CI) pipeline, such as Jenkins, Bamboo, or Travis CI, so that tests are run automatically after each code change.

Test reporting and analysis: Use Sahi Pro's test reporting and analysis features to generate test reports and identify defects and performance issues.

Test environment management: Manage test environments using tools like Docker, which provides a consistent and isolated environment for each test run

Collaborative testing: Collaborate with developers, testers, and other stakeholders to continuously improve the testing process and ensure that tests are up-to-date and relevant.

Destructive testing is a software testing technique that involves intentionally causing the software to fail to validate its robustness and resilience. In Sahi Pro, destructive testing can be performed using a variety of techniques, including:

Fuzz testing: Fuzz testing involves sending random and invalid input to the software to see how it handles unexpected input. This can help to identify potential security vulnerabilities, memory leaks, and other issues that could cause the software to crash or behave unexpectedly.

Stress testing: Stress testing involves putting the software under heavy load to see how it performs under extreme conditions. This can help to identify performance bottlenecks, memory leaks, and other issues that could cause the software to fail or become unresponsive.

Error injection: Error injection involves intentionally introducing errors, faults, or failures into the software to see how it responds. This can help to identify potential issues with error handling, recovery, and fault tolerance.

Boundary testing: Boundary testing involves testing the software at the limits of its input range, such as maximum and minimum values. This can help to identify potential issues with input validation and boundary checking.

Software performance testing is the process of testing software applications to measure their performance in terms of response time, throughput, scalability, and resource utilization under various workloads.

Security testing is the process of identifying vulnerabilities and weaknesses in a web application's security and ensuring that it can withstand malicious attacks.

Here are some tests we can perform security testing in Sahi Pro:

1. **Authentication and Authorization Testing:** Test the web application's authentication and authorization mechanisms by attempting to login with incorrect credentials, attempting to access pages or resources that require authorization, etc.
2. **Injection Testing:** Test the web application for SQL injection, XSS injection, command injection, etc. by injecting malicious input and observing the application's response.
3. **Session Management Testing:** Test the web application's session management mechanisms by attempting to steal another user's session, forcing the application to create a new session, etc.
4. **Cross-site Scripting (XSS) Testing:** Test the web application for XSS vulnerabilities by injecting scripts that can be executed by the browser, such as JavaScript or HTML.
5. **Cross-site Request Forgery (CSRF) Testing:** Test the web application for CSRF vulnerabilities by attempting to make unauthorized requests on behalf of the user, such as changing the user's password or transferring money.
6. **Security Configuration Testing:** Test the web application's security configuration settings, such as encryption, password policies, etc.
7. **Vulnerability Scanning:** Use Sahi Pro's vulnerability scanning feature to scan the web application for common vulnerabilities and weaknesses.

Testing Plugins

Users can run Sahi scripts using Maven. User can create a Maven project in Eclipse and set JDK path for running. Editing pom.xml file to the below code will enable run.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>sahi-examples</groupId>
  <artifactId>sahi-integration-example</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>sahi-integration-example</name>

  <build>
    <plugins>
      <plugin>
        <artifactId>maven-antrun-plugin</artifactId>
        <version>1.7</version>
        <executions>
          <execution>
            <phase>generate-sources</phase>
            <configuration>
```



```

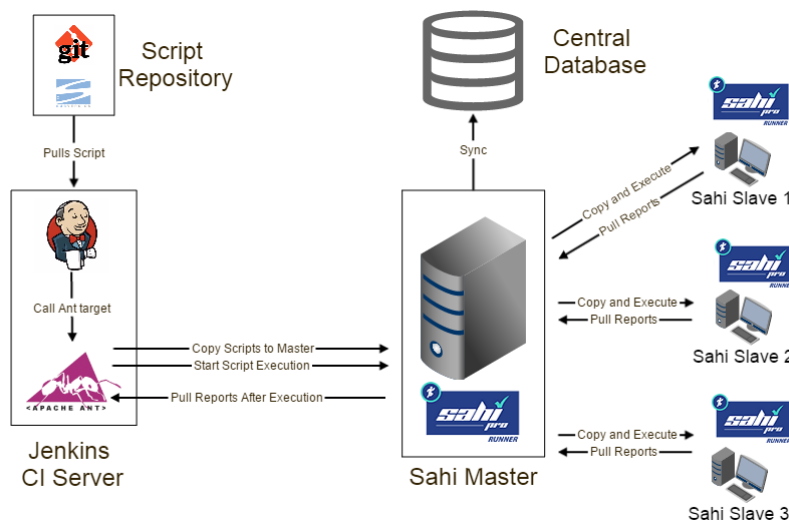
        <target name="target_to_be_executed">
            <ant dir="<SahiInstallationDir>" antfile="build.xml"
/>
        </target>
    </configuration>
    <goals>
        <goal>run</goal>
    </goals>
</execution>
</executions>
</plugin>
</plugins>
</build>

</project>

```

Changing **SahiInstallationDir**, **target name** and **antfile** according to your Configuration will enable Maven to build and run test cases. User can use <https://resources.sahipro.com/docs/using-sahi/playback-via-ant.html> to create antfile required.

Sahi offers integration with **Jenkins Continuous Integration system**. Sahi suites are executed periodically and CI system like Jenkins can trigger the test suites execution when code changes occur and committed to the code base. To playback test scenarios, Sahi Pro requires real browsers. However, many Jenkins/build machines run in server mode without a GUI, which makes it difficult to use Sahi Pro. As a result, end-users must either use headless browsers or execute Sahi scripts remotely to work around this issue.



Sahi integrates with JetBrains - TeamCity CI system. Configuration needs to be set before CI system can trigger test suite execution.

Details on setup are found here: <https://resources.sahipro.com/docs/using-sahi/integrate-with-teamcity.html>.

Sahi Integration Support

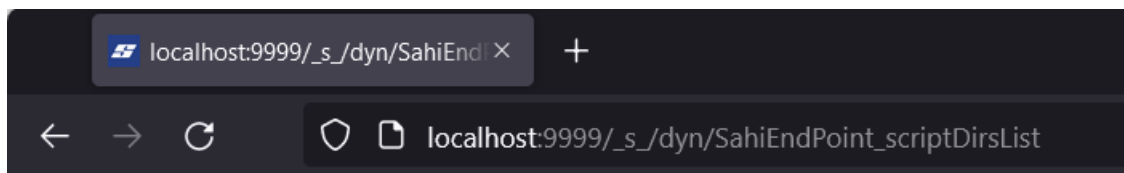
Sahi can be integrated with external tools using predefined URL calls. This document provides a step-by-step guide to using these URL calls for integration with Sahi. The guide is

divided into sections that correspond to the steps required for executing a script or suite from an external tool. Users can modify parameters in the guide to generate a URL specific to their environment.

The Sahi server typically runs on localhost:9999 by default. If you're using a Sahi server from another machine, you'll need to modify the host and port accordingly and then click the "Update" button.

Get Script Directory:

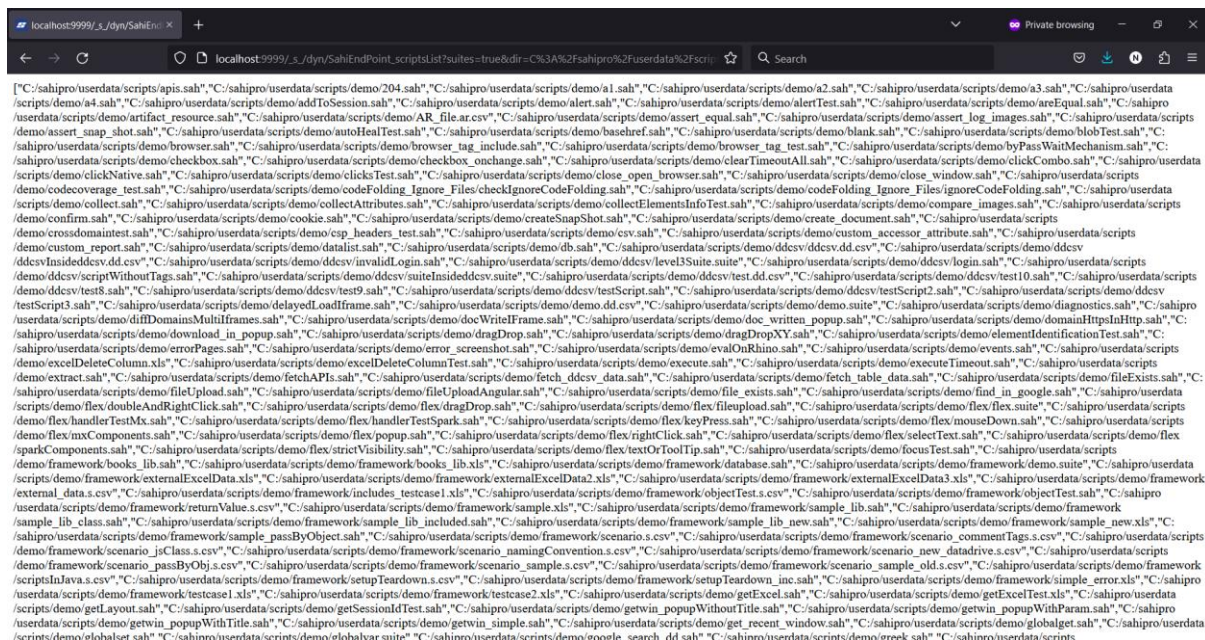
To get the script path specified in the sahi.properties file, you can use the following Request URL. You will get the scripts path as response as shown below.



```
["C:/SahiPro/userdata/scripts/"]
```

Get All Scripts Path in the given Directory:

http://localhost:9999/_s_/dyn/SahiEndPoint_scriptsElementsList?suites=true&dir=C%3A%2Fsahipro%2Fuserdata%2Fscripts%2F



Get Available Modes:

Request URL: http://localhost:9999/_s_/dyn/SahiEndPoint_getStartWithOptions

Response:

```
["GENERIC","BROWSER","ANDROID_BROWSER","IOS_BROWSER"]
```

Request URL:

```
http://localhost:9999/_s_/dyn/SahiEndPoint_getStartWithOptions
```

Response:

```
["GENERIC","BROWSER","ANDROID_BROWSER","IOS_BROWSER"]
```

Get All Available Browsers:

Request URL:

```
http://localhost:9999/_s_/dyn/SahiEndPoint_getBrowserList
```

Response:

```
["firefox","ie","chrome","edgenew","brave"]
```

Test

Get All Available Android Devices/Emulators

Name	Description	Value
refresh	If set to true, devices list will be refreshed.	true ▾

Generate URL

Request URL:

```
http://localhost:9999/_s_/dyn/SahiEndPoint_getAndroidDeviceList?refresh=true
```

Response:

```
[]
```

Test

Get All Available iOS Devices/Simulators:

Name	Description	Value
refresh	If set to true, devices list will be refreshed.	true ▾

Generate URL

Request URL:

```
http://localhost:9999/_s_/dyn/SahiEndPoint_getIOSDeviceList?refresh=true
```

Response:

```
[]
```

Test

Running the Script/Suite:

To execute a suite or script, you can use a URL like this:

```
http://localhost:9999/_s_/dyn/SahiEndPoint_run?a=a
&suite=C%3A%2Fsahi_pro%2Fuserdata%2Fscripts%2Fdemo%2Fsmall.suite
&startWith=BROWSER
&browserType=chrome
&baseUrl=http%3A%2F%2Fsahitest.com%2Fdemo%2F
&androidDevice=XXXX
&iOSDevice=XXXX
&threads=5
&isSingleSession=false
&jsCodeCoverage=false
&abortedRetryCount=0
&failureRetryCount=0
&logsInfo=
&tags=
&userDefinedId=2June2015__12_1_36_338
&suiteTimeOut=0.0
&isAutoHealEnabled=false
```

Users can visit this link to learn more about Sahi Pro integration with other software using REST APIs: <https://resources.sahipro.com/docs/using-sahi/sahi-integration.html>

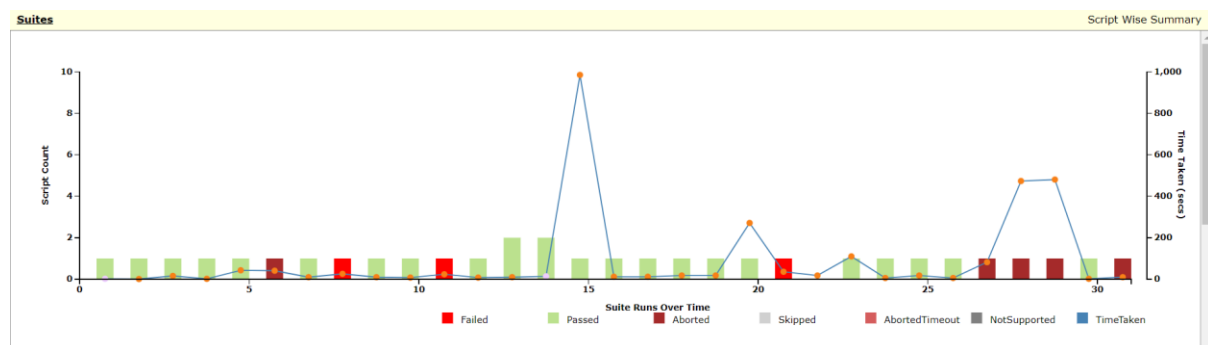
Testing Overall

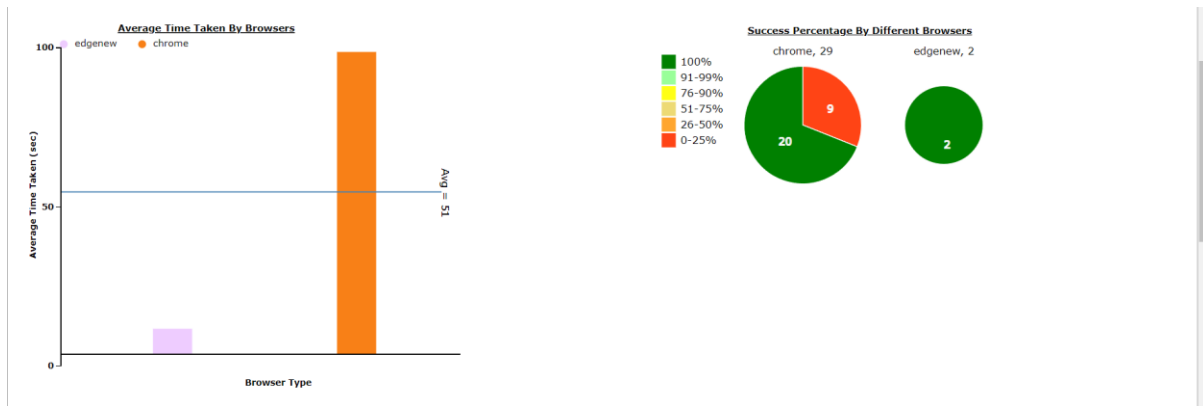
Automation is the strong point in Sahi Pro. User can just record any scenario on application under test and use playback to test it with different data.

Weakness: Sahi does one thing i.e., Automation and it's good at doing it optimally without overhead to maintain test scripts. Users must just maintain data related to the test cases correctly. No naming conventions are proposed in the Sahi docs itself.

Reports

Reports are generated after each execution. User can reach logs site to verify reports generated.





Suites										Script Wise Summary		
Icon	Script Name	URL	Browser	Start Time	End Time	Duration	Attempts	Passes	Failures	Summary	Details	Icon
■	demo.sah	http://sahit...	chrome	Apr 28, 2023 11:32:43 AM	Apr 28, 2023 11:32:54 AM	00:00:10.748	1	Aborted:1	USER_ABORTED			ⓘ
■	demo.sah	http://sahit...	chrome	Apr 28, 2023 11:32:32 AM	Apr 28, 2023 11:32:34 AM	00:00:02.650	1	Passed:1	SUCCESS			ⓘ
■	ShoppingCart.s.csv	http://sahit...	chrome	Apr 27, 2023 10:23:36 PM	Apr 27, 2023 10:31:37 PM	00:08:00.254	1	Aborted:1	USER_ABORTED			ⓘ
■	ShoppingCart.s.csv	http://sahit...	chrome	Apr 27, 2023 10:15:25 PM	Apr 27, 2023 10:23:19 PM	00:07:53.836	1	Aborted:1	USER_ABORTED			ⓘ
■	ShoppingCart.s.csv	http://sahit...	chrome	Apr 27, 2023 10:13:36 PM	Apr 27, 2023 10:14:58 PM	00:01:22.706	1	Aborted:1	USER_ABORTED			ⓘ
■	ShoppingCartScript...	http://sahit...	chrome	Apr 27, 2023 10:13:17 PM	Apr 27, 2023 10:13:23 PM	00:00:06.160	1	Passed:1	SUCCESS			ⓘ
■	ShoppingCart.s.csv	http://sahit...	chrome	Apr 27, 2023 09:38:07 PM	Apr 27, 2023 09:38:25 PM	00:00:18.065	1	Passed:1	SUCCESS			ⓘ
■	ShoppingCartScript...	http://sahit...	chrome	Apr 27, 2023 09:37:46 PM	Apr 27, 2023 09:37:52 PM	00:00:06.049	1	Passed:1	SUCCESS			ⓘ
■	ShoppingCart.s.csv	http://sahit...	chrome	Apr 27, 2023 09:32:11 PM	Apr 27, 2023 09:34:02 PM	00:01:50.436	1	Passed:1	SUCCESS			ⓘ
■	ShoppingCart.s.csv	http://sahit...	chrome	Apr 27, 2023 09:31:42 PM	Apr 27, 2023 09:32:00 PM	00:00:18.079	1	Error:1	FAILURE			ⓘ
■	ShoppingCart.s.csv	http://sahit...	chrome	Apr 27, 2023 09:30:20 PM	Apr 27, 2023 09:30:56 PM	00:00:36.178	1	Failed:1	FAILURE			ⓘ
■	ShoppingCart.s.csv	http://sahit...	chrome	Apr 27, 2023 09:25:03 PM	Apr 27, 2023 09:29:34 PM	00:04:31.137	1	Passed:1	SUCCESS			ⓘ
■	ShoppingCart.s.csv	http://sahit...	chrome	Apr 27, 2023 07:57:55 PM	Apr 27, 2023 07:58:13 PM	00:00:18.090	1	Passed:1	SUCCESS			ⓘ
■	ShoppingCart.s.csv	http://sahit...	chrome	Apr 27, 2023 07:56:47 PM	Apr 27, 2023 07:57:05 PM	00:00:18.124	1	Passed:1	SUCCESS			ⓘ
■	ShoppingCart.s.csv	http://sahit...	chrome	Apr 27, 2023 06:43:47 PM	Apr 27, 2023 06:43:59 PM	00:00:12.055	1	Passed:1	SUCCESS			ⓘ
■	ShoppingCart.s.csv	http://sahit...	chrome	Apr 27, 2023 06:43:29 PM	Apr 27, 2023 06:43:41 PM	00:00:12.075	1	Passed:1	SUCCESS			ⓘ
■	ShoppingCart.s.csv	http://sahit...	chrome	Apr 27, 2023 06:24:34 PM	Apr 27, 2023 06:40:59 PM	00:16:24.287	1	Passed:1	SUCCESS			ⓘ
■	cart_test_suite.dd.csv	http://sahit...	edgewise	Apr 25, 2023 05:34:34 PM	Apr 25, 2023 05:34:48 PM	00:00:14.050	2	Passed:2	SUCCESS			ⓘ
■	cart_test_suite.dd.csv	http://sahit...	chrome	Apr 25, 2023 05:33:15 PM	Apr 25, 2023 05:33:25 PM	00:00:10.048	2	Passed:2	SUCCESS			ⓘ
■	cart.s.csv	http://sahit...	chrome	Apr 25, 2023 05:14:29 PM	Apr 25, 2023 05:14:37 PM	00:00:08.053	1	Passed:1	SUCCESS			ⓘ
■	sahil_sample_run.sah	http://sahit...	chrome	Apr 25, 2023 04:49:13 PM	Apr 25, 2023 04:49:37 PM	00:00:24.102	1	Failed:1	FAILURE			ⓘ

Contributions

Naga Vara Pradeep Yendluri – Sahi Record feature, Data Drive, Sahi script research and its usage in automating various testing methods, BDTA execution.

Varuna Sri Budidi – Playback feature, Data driven test case execution

Yogesh – Sahi Economics & usage of automation and identifying elements in web applications

Hari Prasad – Sahi History, BDTA in Sahi

Sreenivas – Sahi Editor, BDTA applying in Sahi Scripts using scenarios.

Sumanth – Automation using Sahi. Parallel execution in Sahi using Data Driven Suites

Vamshi – Logging in Sahi, creating library functions in Sahi.

Ishitha – Suites in Sahi, suites execution.