|  |
| --- |
| Sherwin-Williams | Color Excellence |
| Katalon Recorder |
| Installation and User Guide |

|  |
| --- |
| Terri L Freeman  6/5/2018 |



Table of Contents

Installation 2

Creating Automated Tests 4

Recording a Test Case 5

Manually Adding/Editing Commands 7

Pause and WaitFor Commands 10

Click and ClickAndWait Commands 11

Verify and Assert Commands 12

Common Test Functions 13

Saving Test Suites 15

Running Tests 16

Handling Step Failures 18

SherColor Web Testing 19

SherColor Web Application Tests 20

**Installation**

Katalon Recorder is a Selenium IDE that allows you to record actions, capture web elements on web applications, play automated test cases, and do reporting.

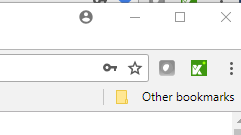
To get started:

Install Katalon Recorder by pasting the following address in Chrome:

https://chrome.google.com/webstore/detail/katalon-recorder-selenium/ljdobmomdgdljniojadhoplhkpialdid?hl=en-US

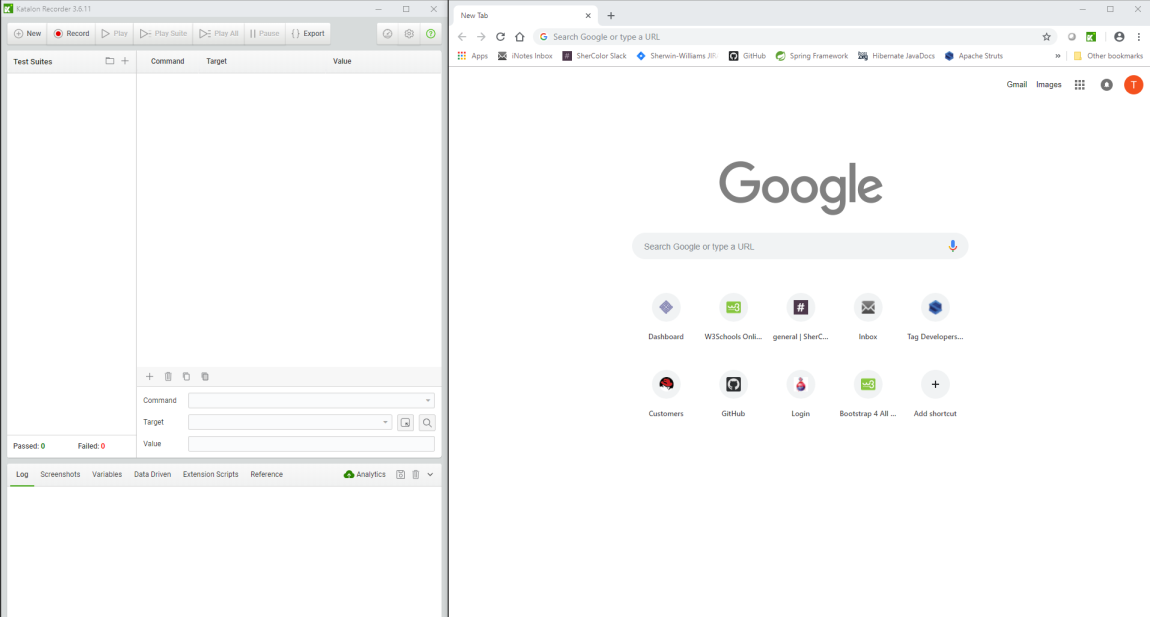


After Plug-In is installed, click the Katalon Recorder icon in the upper right hand corner of the Chrome browser window to open the recorder.

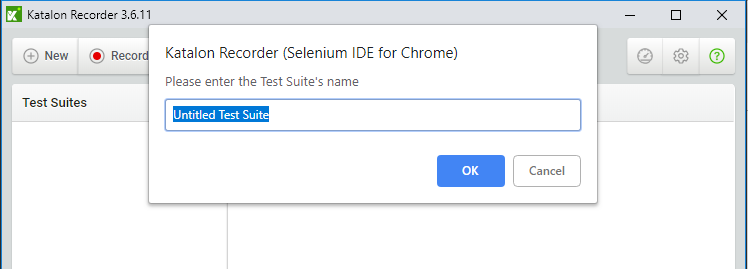


### **Creating Automated Tests**

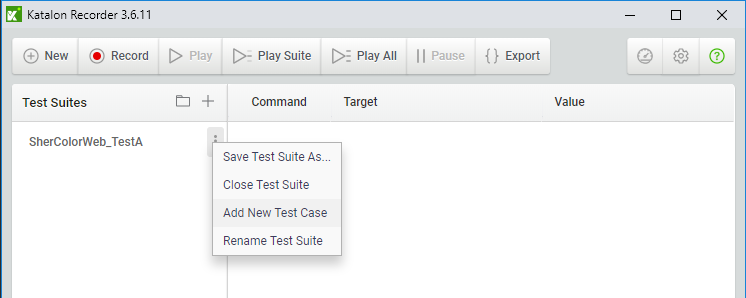
Open Katalon Recorder from Google Chrome. Since both Chrome and Katalon need to be open simultaneously while creating tests, displaying the windows side-by-side provides the most optimal workspace .



To create a new Test Suite, click the **+** icon in the **Test Suites** pane within Katalon. Enter the Test Suite name in the prompt and click **OK**.



To create new Test Cases (test steps) within the Test Suite, click the ellipsis next to the Test Suite and click the **Add New Test Case** option. Enter the Test Case name in the prompt and click **OK**.



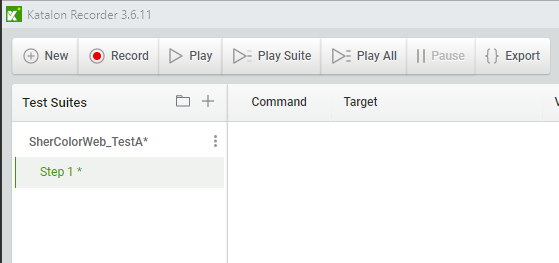
Navigate to the Sher-Color website and sign in to begin creating the automated test.

Katalon will record the actions as Selenium commands as the test is performed. Most commands will record entirely, but some will need editing or additional information. Some commands will need to be manually added to the recorder, for example, in the case of verifying or asserting information on the page. When editing or adding commands, please refer to the [Katalon Selenese Documentation](https://docs.katalon.com/katalon-recorder/docs/selenese-selenium-ide-commands-reference.html) for support when creating test scripts. The [Kantu Selenium IDE Commands](https://a9t9.com/kantu/docs/selenium-ide) page can also be helpful to learn more about Selenium commands.

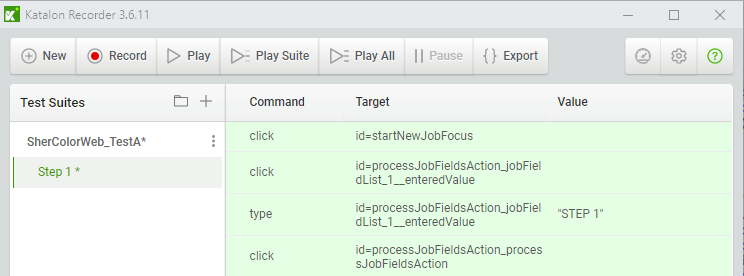
##### Recording a Test Case

To begin creating a Test Case, the first step is to start recording the actions taken during the test step. At this point, the only commands that will need to be manually added are pause and waitFor. Also, the commands created during the recording may need to be edited or deleted.

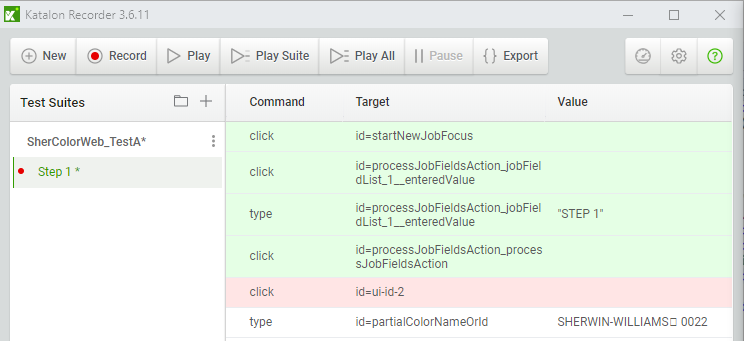
To get started recording the Test Case, click the **Record** button at the top of the Katalon Recorder window and execute the application test step. If generating a test suite from a SherColor POS system test, an important thing to know is that SherColor Web does not offer all of the same functions as SherColor POS. Try to translate the POS test steps into Web test steps as closely as possible.



While recording, Katalon will generate test commands in the command window, and a popup will appear on the bottom right of the screen indicating an action has been performed. The recording can be stopped and started at any time during the test. As long as the web page is not left after the recording is stopped, the test recording can resume uninterrupted by clicking the record button again. It is helpful to repeatedly stop and start recording during the test. This will ensure actions and/or verifications have not been missed, especially when commands have to be manually added.



When finished recording the test case, click play to run the test. The test is set to run at the fastest speed, keep that setting and run it. The commands that are successful will be highlighted in green. If a command fails, it will be highlighted in red. When the command fails, the test will not be able to continue. The failed commands will either have to be deleted (if not needed) or edited for accuracy.

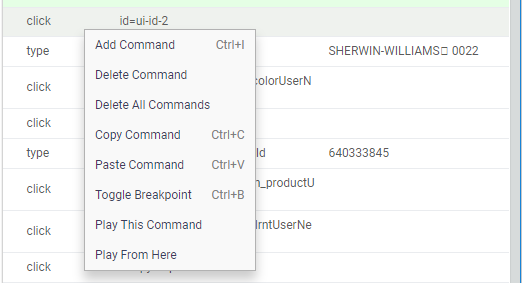


# Successful Command

# Failed Command

To determine whether the test case needs to be edited or deleted, run the test at a slower speed[[1]](#footnote-1) and watch each command as it is executed. Breakpoints can be set at the command which is failing to check the execution to that point. The test will pause at that point and will not continue until the breakpoint is removed.

To set a breakpoint, right click the command, then click Toggle Breakpoint. A green arrow will appear to the left of the command.

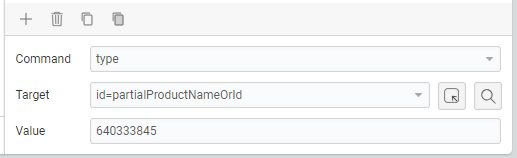


It is possible to set many breakpoints and play the test steps from anywhere within the test case. There is even an option to play one test step at a time (see above figure). All of these functions are intended to assist in creating a thorough and accurate test case, so be sure to utilize all of them.

In this case, the **Click** command for **id=ui-id-2** target failed because it is not needed within the test case. A command that is created during the recording that is not actually needed can be safely deleted. Delete a command by right-clicking the command and then click Delete Command.

#### Manually Adding/Editing Commands

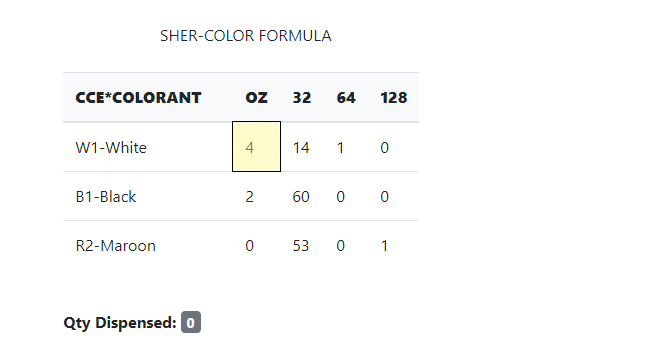
To edit a test command, Katalon Recorder provides editing fields for the command data. Here, the command, target, and value of the test command can be manually edited.



Select Element Icon

The **Command** refers to the Selenium command and can be manually changed by typing or selecting an option from the drop-down menu. See both Katalon Recorder and Selenium documentation for details about these commands.

The **Target** refers to the HTML element and can be manually changed by typing, selecting an option form the drop-down menu, or clicking the **Select Element** icon to search for elements directly on the web page. The element will highlight yellow, click to select.

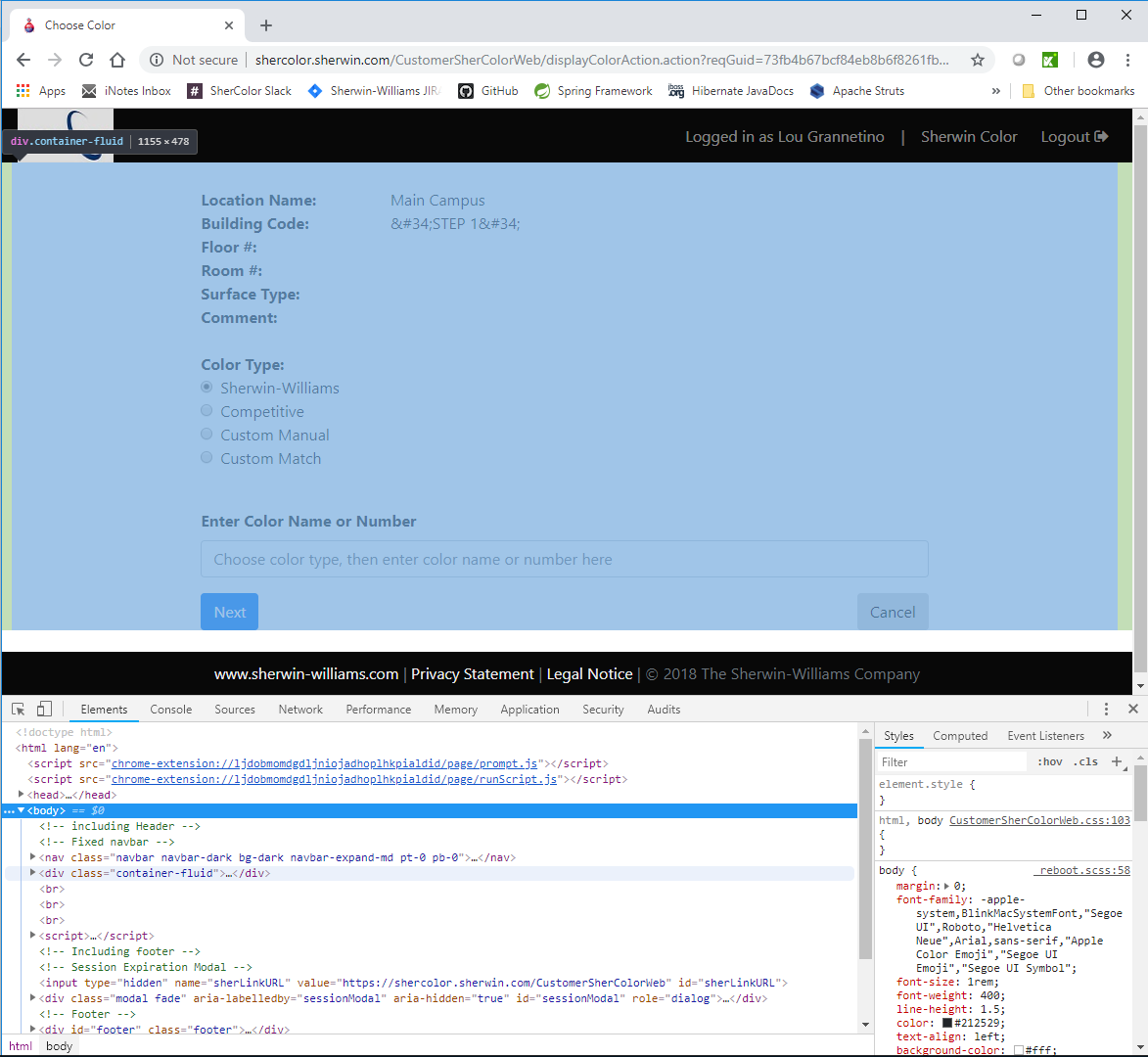


Highlighted Element

The **Value** refers to the text or input value used by Katalon to either compare with or enter into the target specified. This can be manually changed only by typing (or pasting) the data into the field.

When manually adding commands to the test case, it may be helpful to use Chrome’s Developer Tools by pressing Ctrl+Shift+I within the active browser window.

Normally, the dev tools will be docked at the bottom of the active window, but can be moved around as needed. These tools can be useful when searching for a reference of a particular HTML element, since the Selenium commands need to locate the target element when executing. The Elements tab of the dev tools displays all HTML elements, and will highlight on the rendered page when the mouse hovers over the element.

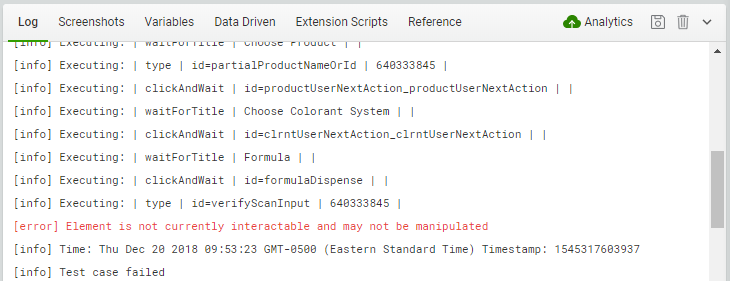


# Developer Tools

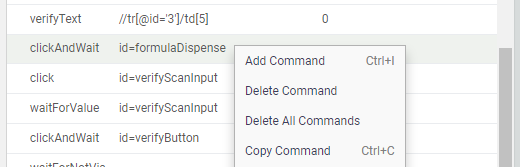
# Highlight from hover

# Hovered Element

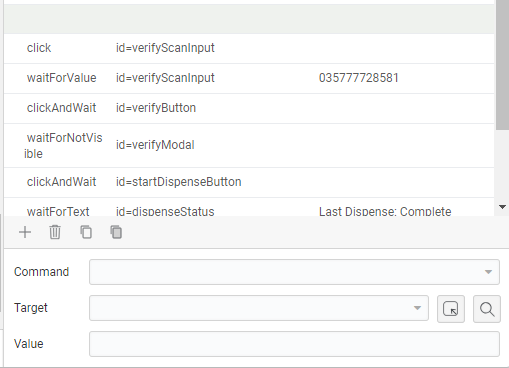
Another helpful tool for manually adding commands is the Katalon Recorder Log. This displays information about the execution of the Selenium commands. It provides clues on what changes need to be made to create a successful command.



To add commands, right-click the command above the placement of the command to be added and click **Add Command**.



A new, blank command will be added to the test case.



# Command Editing Fields

Configure this command within the editing fields.

**Pause and WaitFor Commands**

Adding manual **Pause** and **WaitFor** commands are essential to creating a successful test case. They are used in the test cases to suspend the test until a condition is met. The main difference between the two is that the **Pause** command’s only condition is the time set in the **Value** field.

Frequently used commands for pause and waitFor:

**Pause –** pauses the test for a defined period. **Value**: time, in milliseconds.

Use this sparingly, as there are more accurate ways to implement explicit waiting. The best way to utilize this command is to add it temporarily until a more efficient command can replace it. Also, it is unavoidable in some cases to use a slight pause before a modal becomes visible so subsequent commands can execute.

**WaitForNotVisible –** pauses execution until an element is not visible. **Target**: the element the command is waiting to not be visible. **Value**: none.

This command is useful when waiting for a modal to finish processing in order to continue with the test.

**WaitForText** – pauses execution until text within an element matches the value of the target. **Target**: the HTML element that contains the text. **Value**: the exact value of the text.

This command can be used when text within an HTML element changes after an action has been completed and subsequent commands cannot be executed until the text appears.

**WaitForTitle –** pauses execution until the title of the web page matches the value of the target. **Target**: exact value of the page title as is listed within the Elements tab in the dev tools. **Value**: none.

This is necessary since it is important that the test is on the correct page before the subsequent commands can execute.

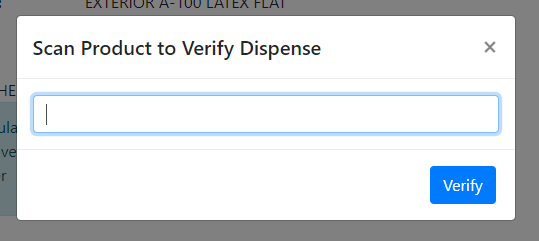
**WaitForVisible** – pauses execution until an element is visible. **Target**: the element the command is waiting to be visible. **Value**: none.

This command is useful when waiting for a modal to become visible so subsequent commands can execute.

##### Click and ClickAndWait Commands

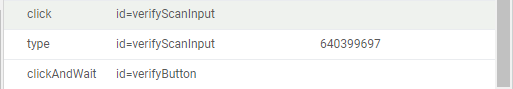
It is recommended to change all **Click** commands for buttons and other input types to **ClickAndWait** because explicit waiting is often needed while an action within the application is being processed. This usually takes place when a button or link is clicked. Timeouts are often a cause of test failures, since implicit waiting is sometimes inadequate. This can occur if the test attempts to move on before the action has finished processing.

Most click commands will be added at recording time, some will be necessary to keep and some can be deleted. Always keep click commands related to buttons, checkboxes and radio buttons. These particular commands should also be changed to clickAndWait. Click commands related to input fields for text on the web page and in modals rely on client-side scripts to activate. If the text field is in focus when the page loads, the click command is unnecessary. This is usually common in text fields that are the first input form field on the page, but not always. If the input field is not in focus (active), the type command may possibly fail. To be safe, add a click command before typing into input fields, especially modal input fields.



# Click Command Activates Input Field

# Modal



**Select and SelectAndWait Commands**

The select command is used to select an option from a drop-down menu. Use selectAndWait if the page loads after the option is selected.

The **Target** is the HTML select element. The **Value** can be any value that uniquely identifies the select option: the value of the option, the id of the option element, etc.

**Type and TypeAndWait Commands**

The type command is used to type a value into a text input field. Use typeAndWait if the text field uses an autocomplete script.

The **Target** is the HTML input field element. The **Value** is the exact value of the text within the field.

##### Verify and Assert Commands

Verification commands will need to be added for test accuracy. Information on the web pages needs to either be asserted or verified by the automation program since it is not being performed by a manual tester.

Katalon has two ways to check for conditional values: asserting and verifying. The difference between the two is that asserting will cause the test to fail if it does not match the value of the test command. The values that did not match the verify command will highlight red during the test, but the test will not fail.

Frequently used commands to verify values:

**VerifyText –** compares text on the web page with the value of the target element. **Target**: use the inspect tool to select the element on the page that contains the value to be verified. **Value**: exact value of the text to be verified.

This is used mostly to verify that the formula matches correctly.

**AssertText** – compares text on the web page with the value of the target element. If the two do not match, the test will fail. **Target**: use the inspect tool to select the element on the page that contains the value to be asserted. **Value**: exact value of the text to be asserted.

This command can be used to assert important informational messages, such as warnings.

Once the verification commands have been added, run the test at full speed. Correct any failed test steps by running the test slower, toggling breakpoints, and searching the Log for errors. After the test case runs successfully at full speed, the test case has been completed.

**Common Test Functions**

Listed here are commands for functions that are common across some or all tests and can be reused.

**Dispensing**

|  |  |  |
| --- | --- | --- |
| **Command** | **Target** | **Value** |
| clickAndWait | Id=formulaDispense |  |
| waitForVisible | Id=setDispenseQuantityModal |  |
| type |  | [quantity to dispense] |
| clickAndWait | Id=setDispenseQuantityButton |  |
| waitForVisible | Id=verifyModal |  |
| type | Id=verifyScanInput | [product number] |
| clickAndWait | Id=verifyButton |  |
| waitForNotVisible | Id=verifyModal |  |
| pause |  | 1000 |
| clickAndWait | id=startDispenseButton |  |
| waitForText | id=dispenseStatus | Last Dispense: Complete |
| waitForNotVisible | id=tinterInProgressModal |  |
| Pause |  | 500 |
| verifyText | id=qtyDispensed | [quantity dispensed] |

Replace values within [] with values from step instructions

Repeat these steps for each qty dispense

**Dispense Addition**

|  |  |  |
| --- | --- | --- |
| **Command** | **Target** | **Value** |
| pause |  | 1000 |
| clickAndWait | id=dispenseAdd |  |
| waitForVisible | Id=positionContainerModal |  |
| clickAndWait | Id=startDispenseButton |  |
| waitForNotVisible | Id=tinterInProgressModal |  |
| waitForText | Id=currentPrompt | Currently Correcting Container\* |
| clickAndWait | Id=acceptContainer |  |
| waitForText | Id=currentPrompt | \*Completed\* |
| clickAndWait | Id=mainForm\_displayFormulaAction |  |
| waitForTitle | Formula |  |

Be sure to include asterisks in **Value** field

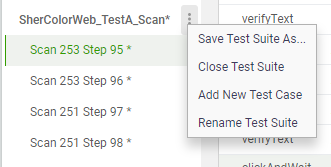
**Printing**

|  |  |  |
| --- | --- | --- |
| **Command** | **Target** | **Value** |
| clickAndWait | id=formulaPrint |  |
| waitForVisible | id=printLabelModal |  |
| pause |  | 1000 |
| clickAndWait | id=printLabelPrint |  |
| waitForNotVisible | id=printLabelModal |  |

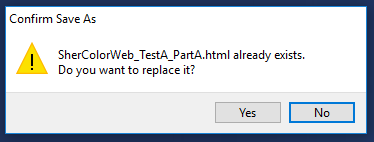
**Saving Test Suites**

Save test suites often to avoid losing work. Once Katalon is closed, if work has not been saved, it is most likely recent work will be lost upon reopening the application. In Katalon, there is only one way to save work on a test suite.

Click the elipsis next to the test suite name, and click **Save Test Suite As…**.



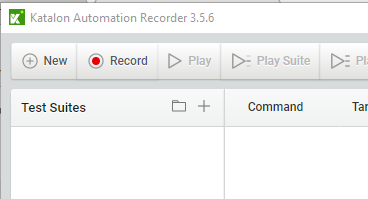
Navigate to the directory where the test is stored. The test suite name will be populated in the file name field. Do not change the name if the goal is to write over the already existing test suite to update.



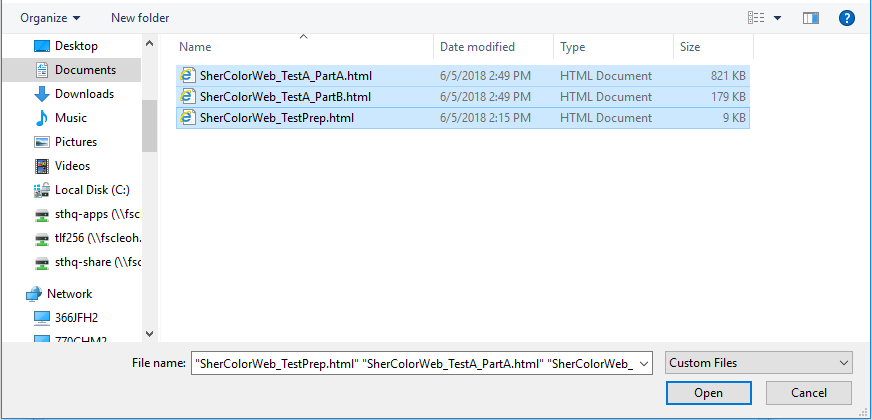
To save a different copy in the same location, change the test suite name.

### **Running Tests**

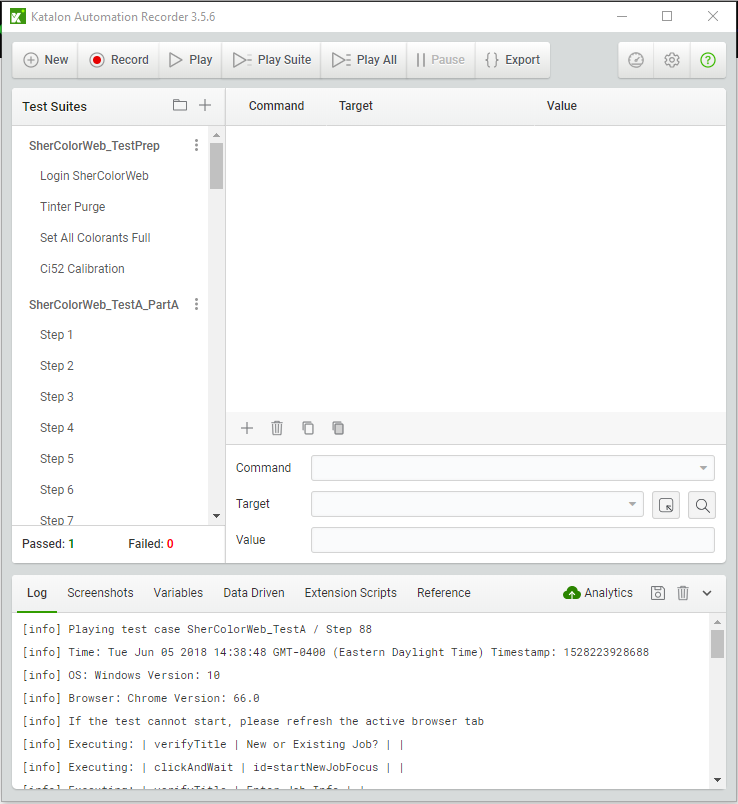
Open Katalon Recorder from Google Chrome. In the Test Suites pane, click the folder icon to open saved Test Suites.



Browse to the location of the Test Suite HTML files and select all suites needed for testing, click Open.



All selected suites will appear in the Test Suites pane, but not necessarily in the correct order. The Katalon IDE gives you the flexibility to decide what test suites are played and when. Test cases are steps within the suite and can be played together or individually. Additionally, the test suites can be moved up or down by dragging the suite to the proper location. Any newly opened suites will locate to the bottom of the Test Suites pane.



# Test case

# Test suite

# Play current suite

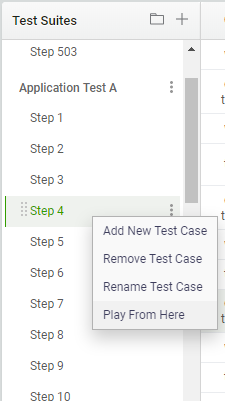
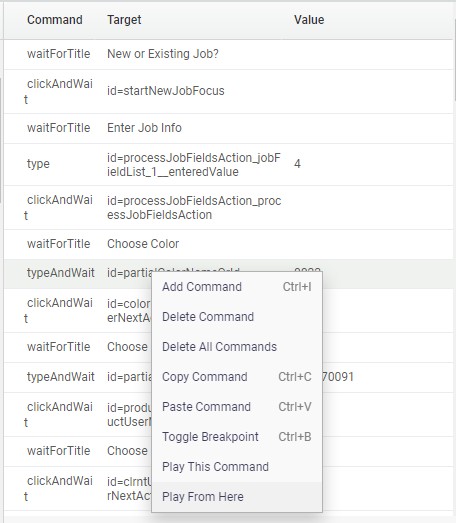
# Play all suites

Be sure your computer’s screen saver will not interrupt the testing process and do not leave the active browser window! If there are times within the test where there will be no interaction with the system, or if the browser window becomes inactive, the test will fail.

**Handling step failures**

Should the test fail at any point:

* **If the test step fails at a verify command:** be sure to record as much information about the failure as soon as it happens. Since it is a verify command, the test will continue to run, but the failed command(s) will highlight red. It is possible to go back and check the step after the test completes – the failed commands will still be highlighted if Katalon is not closed and the test is not reloaded or stopped during execution.
* **If the test step fails at any other command:** The step will not only fail, but it will stop running at that point. Record the failure, and if possible, continue the test at the next possible command by using the “play from here” option in the command step window. This menu option can be accessed by right-clicking the command the test is to be restarted at, then choosing **play from here**. Additionally, it is possible to continue the automated test by right-clicking the next test step (the step after the failed step) in the Test Suites window and selecting the **play from here** menu option.



# Continue playing the test step commands

# Continue playing the test steps

**SherColor Web Testing**

**Open** [**JIRA**](http://jira.sherwin.com:8080/secure/Dashboard.jspa)**,** in the top menu, click on Issues > Search for Issues. Then in the Search options click on Project > PSG SherColor Web; Type > All; Status > Open, In Progress, Reopened, Resolved; type CSW in the “contains text” field and click Search. This will return a list of known issues for CustomerShercolorWeb. You can save this search criteria as “CSW Issues” so that it can be quickly accessed. Keep JIRA open and check these issues against any issues found during testing to prevent duplicate bug reporting.

**Navigate to SherColor Web and login using either the test login or your user login** – this part of testing is manual since the login ID’s are different for each person.

**Configure and/or initialize tinter** – this part of testing is manual, for now, and is located under the menu icon > Tinter Menu > Add New Tinter. The Initialize Tinter menu option can be chosen if there are communication issues, otherwise, SherColor Web will automatically initialize the tinter either after login or, in the case of an unconfigured tinter, after configuration. The initialization process needs to be complete before the Test Prep suite can run successfully.

**Upload tinter calibration file (Ecal)** – this part of testing is manual, for now, and is located under the menu icon > Tinter Menu > Calibration Manager. Upload Calibration for your tinter.

**Configure printer** – this part of testing is manual, for now, and is located under the menu icon > Printer Menu > Configure. Select the ZDesigner printer from the options and check Auto Print on Dispense. Set the number of labels to 1 and save the settings.

**Play SherColorWeb Test Prep** - this suite prepares your system for the automated test by:

* Purging the tinter
* Setting all colorants full (running out of colorant will cause the test to fail)
* Calibrating the Color Eye

This test suite should be completed before any automated test is run. This suite is fully automated and does not need to be completed if your system is already prepared in this way.

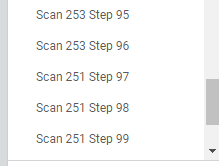
## SherColor Web Application Tests

Before running any Application Tests, be sure you have completed all steps under **SherColor Web Testing**. If color samples and/or quart containers are needed, they should be out and ready for scanning.

SherColor Web does not currently support adjusting formula to new product size by changing product dynamically, so the test re-enters the color information with the new product number for each step.

Application tests may include one or more of the following test suites:

* **Application Test [A, B, C, D, E, F]** - these suites run through all steps relevant to SherColor Web that do not utilize the Color Eye or Scan gun, they are completely automated. These test steps also include the scanning steps without the need to use the scan gun.
* **Application Test [A, B, C, D, E, F] ColorEye** - these suites run through all Color Eye steps. These are assisted (partially automated) and will need manual input from the color eye.
* **Application Test [A, B] Scan** - these suites run through all scanning steps. These are assisted (partially automated) and will need manual input from the scanner gun. On the Scan Steps, the container to scan is named in the test case for quick reference.



The SherColor Web application does not support adjusting the formula to new product size, or color re-match, so the color sample will need scanned for each step.

It is important to be ready when prompted for Ci52 and Scanner input, since the automated test runs quickly. Do not start these suites until prepared to do so, to ensure the test will not fail because the command has timed out. The test case default timeout is set at 30,000ms, and the test will fail after that time has elapsed.

1. Running the test at a slower speed should be used with caution. The goal is to create the test to be run at the highest speed and when the test is run slower the location to insert wait commands may be missed. [↑](#footnote-ref-1)