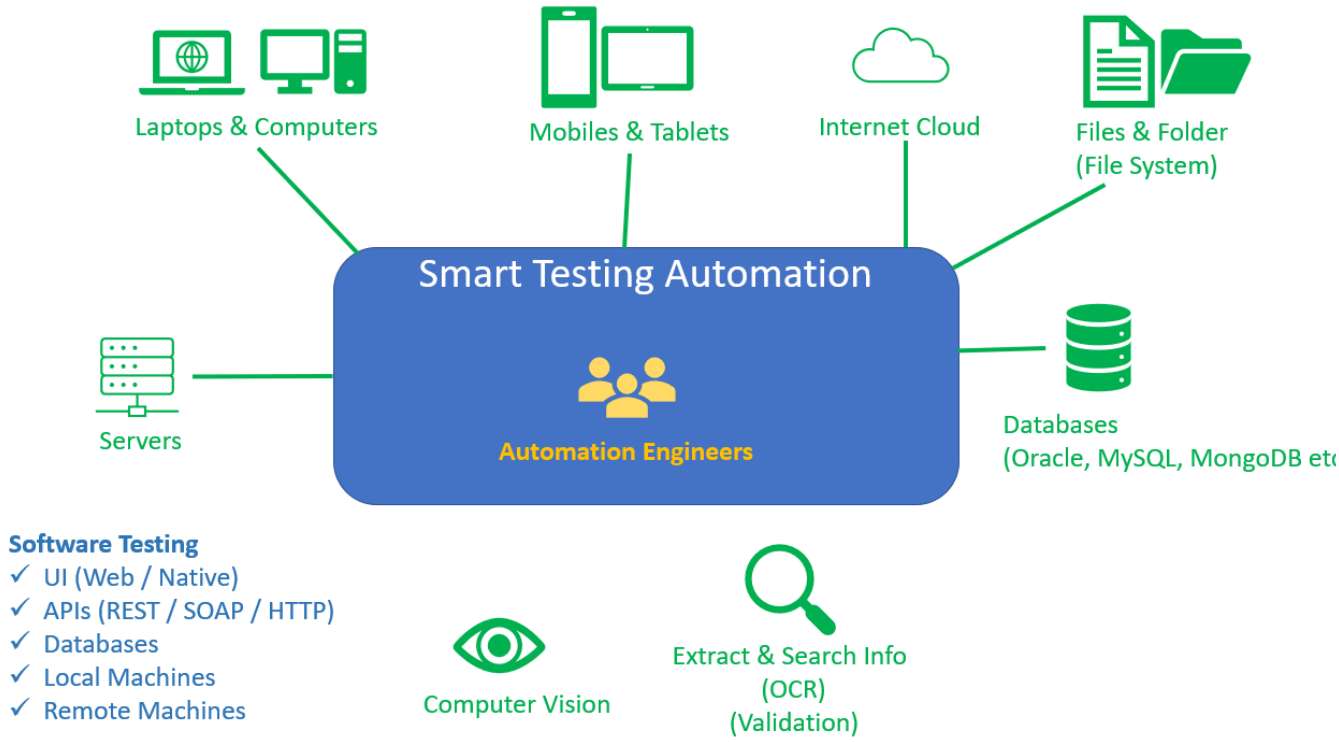

STAS CUCUMBER STEP DEFINITIONS - EN

Test Automation Developer's Guide

Abstract

This document lists the standard cucumber step definitions (English) provided by STAS tool.

Madhav Krishna

mkrishnacs20@gmail.com

Table of Contents

Table of Figures

1 Licensing and usage

This is developer's guide developed by the actual developer of Smart Test Automation Framework (STAF) / Smart TestAuto Studio tool (STAS). The purpose of this developer's guide is to help test automation engineers to understand the standard cucumber step definitions provided by STAS to automate the following types of testing: Regression Testing, Sanity Testing, Smoke Testing, End-to-End Testing, Functionality Testing, User Acceptance Testing etc.

This is a free and open-source tool and licensed under **Apache License 2.0 (https://www.apache.org/licenses/LICENSE-2.0)**. This tool and the guide only used to help develop testing automation and there is no liability on the author if there is any defect found in the system. But users can raise the defects on the Github URL (https://github.com/mkrishna4u/smart-testauto-cucumber-stepdefs-en/issues) so the community team can work on that defect and close as per their convenience. Also, if you are looking for any change or enhancement you can raise that on the same Github URL.

Also, the licensing of 3rd party tools integrated in this tool belong to the individual owner of the 3rd party tool. If there is any defect on third party tool, the defect should be raised on the respective 3rd party tool website.

2 Introduction

NOTE: We must read the STAS user guide before reading this user guide to know how to setup STAS project environment and different things. STAS user guide is present at the following location:

https://github.com/mkrishna4u/smart-testauto-framework/blob/main/smart-testauto-studio/latest/docs/STAS-TestDevelopersGuide.pdf

STAS provides the standard cucumber step definitions in English (**URL**: https://github.com/mkrishna4u/smart-testauto-cucumber-stepdefs-en) that can be used to create scenarios for testing automation. STAS definitions has connectivity with the different sub systems as mentioned in the diagram below:

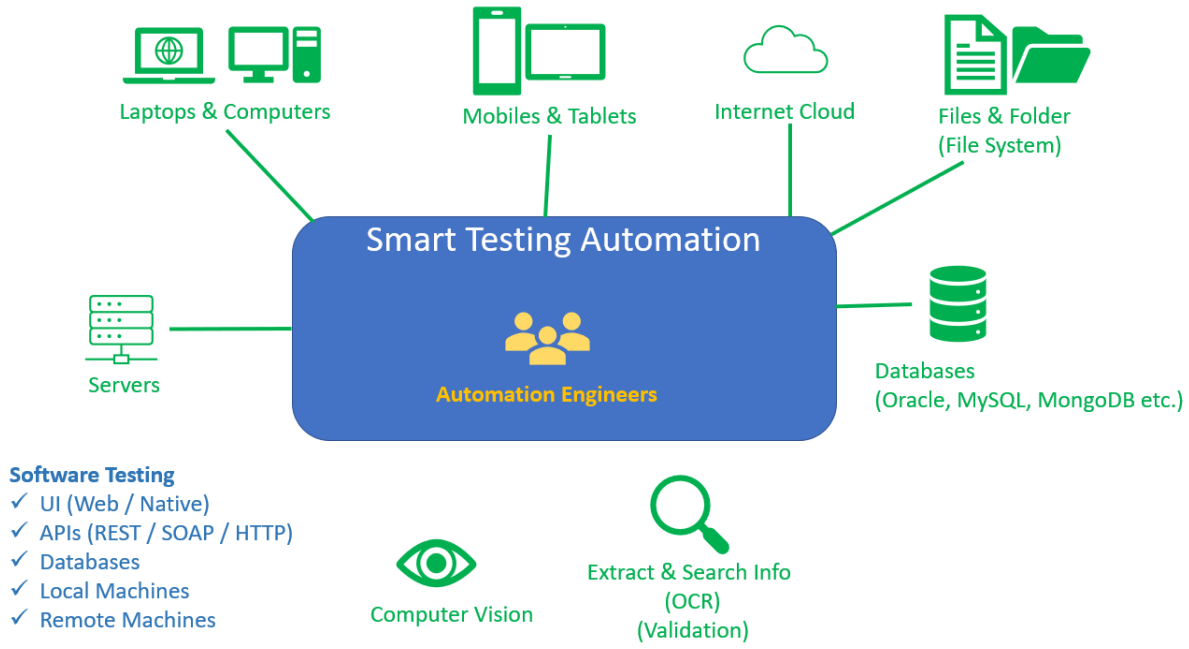

Figure 2-1 STAS Tool Connectivity

STAS tool connects with different types of applications (Web, Native, Mobile), servers (Database, file systems), remote machines (over SSH).

Using STAS, we can be able to perform the testing automation for:

A. Graphical User Interface (GUI) Testing
B. API Testing
C. Database Testing
D. Functional Testing
E. End-to-End Scenario Testing / Integration Testing (may include multiple applications and multiple sub systems)
F. Regression Testing
G. Smoke Testing

H. Cross applications testing
I. File contents verification testing i.e. PDF, MS Word, MS Excel, MS PowerPoint, Images etc.
J. File upload and download
K. OCR (Optical Character Recognition) Testing
L. Computer Vision / Image pattern matching

The following **approaches** are supported by STAS to write the test cases easily and manage them:

A. Data Driven Testing (DDT) Approach

Use different type of data file to feed the data to the system to verify the functionality. State of the art easy to use classes are provided to read the data from different types of files as given below:

1. ExcelFileReader
2. CSVFileReader
3. JsonDocumentReader
4. XmlDocumentReader
5. YamlDocumentReader
6. SmartDatabaseManager / SQLDatabaseActionHandler

B. Behavior Driven Testing (BDT) Approach

Use Cucumber Feature file to write the End-to-End scenarios and to provide the data to the system to verify the functionality.

C. Configuration Driven Testing (CDT) Approach

STAS provides standards configuration files in YAML format to configure your application related information (i.e. application config, user profile config, web driver config, database config, remote machine config etc.) that can be used to communicate with UI, REST Servers, Database Server and Remote Machines and also that can be used to validate the data on UI, REST Services, Database and Remote Machines.

STAS provides **LOW CODE / NO CODE** model, it means test engineers have to write low code or no code based on the scenario description. But they have to write scenarios in Cucumber Gherkin language as per the standardized step definitions provided by this tool.

This tool supports the **real environment** for software testing (Similar way our manual tester performs software testing like data preparation, run test cases (data-driven), data verification and generate reports etc.). Here using this tool, we can automate data preparation, test cases execution, data verification and report generation easily.

2.1 What you need to know to work on this tool?

There are many tools integrated in this tool to make the test engineers life easy. The most import things that every test engineer must know are:

1. **Cucumber Gherkin Language:** Study about it on the following URL: https://cucumber.io/docs/gherkin/reference/

2. **JSON and JSON Path:** JSON is a very powerful data format that is used while writing cucumber scenario using STAS tool. For JSON path, please refer https://github.com/json-path/JsonPath link. JSON Path is used to modify the JSON data or retrieve field data from JSON data.
3. **Basic Knowledge of Java:** Basic knowledge of Java programming is needed to create the page object classes. It only require, how to create class object using parameterized constructor or how to call class methods. Other basic knowledge of Java programming will be useful if you are planning to write your customized cucumber step definitions.
4. **Different type of data files:** If you are writing data driven test scenarios that requires to read data from Excel, CSV, TXT, XML, JSON, YAML etc. file then you should know the format of these data files. This tool provide ready to use step definition to read the data from these types of files.
5. **XML and XPATH:** Knowledge of XML file contents are mandatory to work on the user interface and web services (API testing) that uses XML data format. Using XPATH mechanism you can access any element in XML document or modify the contents of XML document. XPATH references: https://www.w3.org/TR/1999/REC-xpath-19991116/
6. **HTML and XPATH:** For user interface testing automation, STAS tool uses Selenium / Appium internally to perform operation on page object elements like Textbox, Button etc. You can use different type of locators to locate element on the user interface like ID, AutomationID, AccessibilityID, Name, CSS Selector, LinkText, XPATH etc. By default STAS tool uses XPATH mechanism to locate element on user interface as XPATH mechanism is much solid and we can identify any element on user interface. XPATH references: https://www.w3.org/TR/1999/REC-xpath-19991116/
7. **Basic knowledge of Shell Scripting:** Since STAS uses command line to run the test scenarios. So, knowledge of how to run shell script (windows or linux) is required to run the maven commands or STAS provided scripts using command line.
8. **YAML file:** All the configuration in STAS tool is given in YAML format. It is very simple and standard format to specify the configuration. To know the YAML file format, you can refer any online document.

2.2 Supported Platform Types

The following platforms are supported to perform the software testing automation:

1. windows
2. linux
3. mac
4. android-mobile
5. ios-mobile

2.3 Supported Application Types

The following application types are supported to perform the software testing automation:

1. **native-app**: Like Desktop applications (like calculator) on any platform like windows, mac, android, iOS etc.

2. **web-app**: Like applications running on web browsers (like Github UI) on any platform like windows, mac, android, iOS etc.

2.4 Supported Web Browsers

The following web browsers are supported to perform the software testing automation:

1. chrome
2. firefox
3. edge
4. opera
5. safari
6. internetExplorer
7. remoteWebDriver
8. notApplicable: This is set for native applications.

3 Standard Cucumber Step Definitions (Provided by STAS)

To find the standard step definitions supported by the STAS tool, please visit source directory of https://github.com/mkrishna4u/smart-testauto-cucumber-stepdefs-en link. If you have configured your project in Eclipse or IntellijIDEA code editor then during scenario writing, inside feature file you can see the code completion for the steps. Only thing is that you have to configure Cucumber plugin and make your project as Cucumber project. Also specify the following paths as glue code.

```
stepdefs,org.uitnet.testing.smartfwk.core.stepdefs.en
```

We have different type of standard step definitions files using that you can automate any scenario. For more information, please refer **TBD** document. High level overview is given below:

1. **SmartStepDefs:** This is a main hook file that defines the @Before and @After methods definition that used to run before each scenario and after each scenario completion respectively.
2. **SmartUiBasicAppOperationsStepDefs:** This defines basic UI operations to perform operations on UI like connect to UI application using specified user profile. User profiles are configured in test-config/apps-config/ directory to its specific app. Also it contains step definitions related to opening URL, take screenshot etc.
3. **SmartUiFormElementOperationsStepDefs:** This defines the UI operations like operations on the form elements / page objects like enter the information on page elements, type text on textbox or textarea, extract information from the element, verify information of the elements (i.e. visibility of the element, default information of the element) etc.
4. **SmartUiMouseOperationsStepDefs:** This defines the mouse operations related step definitions that user can perform on user interface. It includes operations like click, double click, click hold and release, right click, hoverover etc.
5. **SmartUiTouchScreenOperationsStepDefs:** This defines touch pad / touch screen operations on user interface like tap, swipe, zoom in, zoom out etc operations.
6. **SmartUiWindowAndFrameOperationsStepDefs:** This defines the step definitions to handle multiple windows, iframes etc. like switching to windows and switching to frame etc.
7. **SmartUiKeyboardOperationsStepDefs:** This defines the step definitions to handle keyboard related operations like keydown, keyup, keypress, combo keys operations like (CTRL + a, CTRL + v, CTRL + click) etc.
8. **SmartFileUploadStepDefs:** This defines file upload related operations on UI.
9. **SmartApiStepDefs:** This defines API Testing related operations and response data verifications. Like HTTP GET, POST, PUT, DELETE etc.

10. **SmartDatabaseManagementStepDefs:** This defines Database Management related step definition like to access the database information using SQL, update database information using SQL query etc.
11. **SmartLocalFileManagementStepDefs:** This defines step definitions to verify the downloaded files from UI or API.
12. **SmartRemoteFileManagementStepDefs:** This defines step definitions to verify the uploaded files on remote remote server using SSH (Secured Shell) protocol.
13. **SmartExcelDataManagementStepDefs:** This defines step definitions to read and verify the contents of Excel File.
14. **SmartCsvDataManagementStepDefs:** This defines step definitions to read and verify the contents of CSV (Comma Separated Value) File.
15. **SmartJsonDataManagementStepDefs:** This defines step definitions to read and verify the contents of JSON File or data. It also defines the steps to update the JSON data.
16. **SmartYamlDataManagementStepDefs:** This defines step definitions to read and verify the contents of YAML File or data. It also defines the steps to update the YAML data.
17. **SmartXmlDataManagementStepDefs:** This defines step definitions to read and verify the contents of XML File or data. It also defines the steps to update the XML data.
18. **SmartTextualDataManagementStepDefs:** This defines step definitions to read and verify the contents of text File or data.
19. **SmartVariableManagementStepDefs:** This defines step definitions to read and verify the contents of the variables.
20. **SmartConditionManagementStepDefs:** This defines the steps to add conditional blocks in scenario to run some steps conditionality. You can also be able to clear the condition. Refer section ***4.6-Cucumber feature file: Run steps conditionally*** for details on conditional block.
21. **SmartDataGeneratorStepDefs:** This defines the steps to auto generate the text data based on the inputs provided. This helps in reducing the hardcoded data in steps.

Using the smart step definition (specified above) we can automate

1. Any user interface like **web or native user interface**.
2. Automate **API Testing** like REST APIs.
3. Automate **database testing**.
4. Automate **file download or upload testing**.
5. Manage multiple windows and frames for web based applications.
6. Remote machine contents verification
7. Visualization testing
8. OCR testing
9. Automate mobile native and web applications (Android and iOS both).

4 Use cases

To perform the testing automation using STAS, there are few things that must be configured in STAS project as per your need, given below:

A. Configure applications
    a. Configure application driver
    b. Configure user profiles
    c. Configure environments
B. Configure databases and database profiles
C. Configure API Target Servers
D. Configure remote machines

Please refer the following document for STAS project configuration: https://github.com/mkrishna4u/smart-testauto-framework/blob/main/smart-testauto-studio/latest/docs/STAS-TestDevelopersGuide.pdf

The uses cases given below are dependent on the STAS project configuration. Once the proper configuration is done then it is easy to write the test scenarios in Cucumber feature file. While we are preparing test scenarios using STAS provided cucumber step definitions then we do not have to write any code.

NOTE: In STAS tool, we can use different types of steps in a single scenario / scenario outline (Part of Cucumber feature file preparation). These steps on a high level may relate to the following categories:

- UI automation steps
- Database automation steps (CRUD operations)
- API automation steps
- Variable management steps
- External file (Excel, CSV, JSON, YAML, XML etc.) data reading steps
- Remote machine data verification steps
- Local machine data verification steps
- Etc…

4.1 UI Testing Automation

When we write test scenarios for UI screen / page functionality verification then we may have to do the following things:

1. Data preparation: We can use SQL step definitions to prepare the data.
2. Verification of default data on UI page when the page is just opened
3. Retrieve data from database to check on UI
4. Fill data on UI page and submit data on server
    a. After UI data submission,
        i. verify the data on database
        ii. verify the data on remote machine
        iii. verify the success/failure messages on UI page
        iv. verify the tabular data on UI page based on the search criteria
5. Upload file using UI
6. Download file using UI
7. Search data

There could be many more different types of scenarios. Some of the useful use cases are given below:

4.1.1 Sample Data

For an example, we have "Search Users" UI page that has following search filter fields:

1. **Username**: textbox
2. **Phone Number**: textbox
3. **Department Name**: combobox, multi selectable dropdown. By default, empty. Supported values: Admin, HR, Finance
4. **Status**: combobox, single selectable dropdown. Default value is Active. Supported values: Active, Inactive. Deleted
5. **Search button**
6. **Reset button**
7. **Search Results table**: This displays the search results. The following columns are present:

    Username, First Name, Last Name, Phone Number, Email, Departments, Status

    a. Below table there is a text to display number of records like

        Found 10 records.

4.1.2 Verify default information on UI page when the page is just opened

Please refer the sample data for UI page mentioned in ***4.1.1-Sample Data*** section.

```
Scenario: Verify the default information of Search Users page when it is just opened.
    Given user is already logged in using "StandardUserProfile" user profile on "myapp" application.
    When click "myapp.MenuItemPO.Menu_SearchUsers" page element to "open Search Users page".
    Then "the Search Users page" will be "opened".
    And verify that the following page elements are visible on "Search Users page":
      | Page Element                                                          |
      | {name: "myapp.SearchUsersPO.Label_PageTitle", maxTimeToWaitInSeconds: 6} |
      | myapp.SearchUsersPO.Textbox_Username                                  |
      | myapp.SearchUsersPO.Label_SearchResults                               |
    And verify the selected value(s) of the following page elements on "Search Users page":
      | Page Element                                | Operator | Expected Information |
      | {name: "myapp.SearchUsersPO.Combobox_Status"} | =        | Active               |
```

4.1.3 Verify search functionality on UI page

Please refer the sample data for UI page mentioned in ***4.1.1-Sample Data*** section.

Let's say if we would like to search a user by valid username then it is important to know the valid username that are present in the system. To know the valid user present in the system we can use database query to get the one valid user name and then we can search on UI page to get the results. Example scenario:

```
Scenario: Verify the search functionality on Search Users page with valid username.
    Given user is already logged in using "StandardUserProfile" user profile on "myapp" application.
    When click "myapp.MenuItemPO.Menu_SearchUsers" page element to "open Search Users page".
    Then "the Search Users page" will be "opened".
    And verify that the following page elements are visible on "Search Users page":
      | Page Element                                                          |
      | {name: "myapp.SearchUsersPO.Label_PageTitle", maxTimeToWaitInSeconds: 6} |
    When get "one username" from "USERS" table using query below and store into "VALID_USERNAME_VAR"
variable. Target DB Info [AppName="myapp", DatabaseProfileName="sample-db"]:
      """
      select USERNAME from USERS fetch first 1 row only
```

```
      """
    When type "VALID_USERNAME_VAR" text in "myapp.SearchUsersPO.Textbox_Username" page element.
    And click "myapp.SearchUsersPO.Button_Search" page element to "search users".
    Then verify text part of each element of '{name: "myapp.SearchUsersPO.Column_LabelType",
maxTimeToWaitInSeconds: 10, params: {colName: "Username", colId: "username"}}' page element matches
"VALID_USERNAME_VAR" text where TextMatchMechanism="exact-match-with-expected-value".
```

In the example above, we first got valid user name and stored its value into VALID_USERNAME_VAR variable. After that this variable value is used to type into the Username textbox and the same value is verified in Username table column.

Now, if we would like to check the "Found <N> records." information also using STAS then the following example is helpful to frame the scenario:

```
Scenario: Verify the search functionality on Search Users page with valid username.
    Given user is already logged in using "StandardUserProfile" user profile on "myapp" application.
    When click "myapp.MenuItemPO.Menu_SearchUsers" page element to "open Search Users page".
    Then "the Search Users page" will be "opened".
    And verify that the following page elements are visible on "Search Users page":
      | Page Element                                                                  |
      | {name: "myapp.SearchUsersPO.Label_PageTitle", maxTimeToWaitInSeconds: 6} |
    When get "one username" from "USERS" table using query below and store into "VALID_USERNAME_VAR"
variable. Target DB Info [AppName="myapp", DatabaseProfileName="sample-db"]:
      """
      select USERNAME from USERS fetch first 1 row only
      """
    And get "Search results records count" from "USERS" table using query below and store into
"EXPECTED_USER_COUNT_VAR" variable. Target DB Info [AppName="myapp", DatabaseProfileName="sample-db"]:
      """
      select count(1) from USERS where USERNAME = 'VALID_USERNAME_VAR'
      """
    When type "VALID_USERNAME_VAR" text in "myapp.SearchUsersPO.Textbox_Username" page element.
    And click "myapp.SearchUsersPO.Button_Search" page element to "search users".
    Then verify text part of each element of '{name: "myapp.SearchUsersPO.Column_LabelType",
maxTimeToWaitInSeconds: 10, params: {colName: "Username", colId: "username"}}' page element matches
"VALID_USERNAME_VAR" text where TextMatchMechanism="exact-match-with-expected-value".
    And verify text part of each element of "myapp.SearchUsersPO.Label_FoundNRecords" page element matches
"EXPECTED_USER_COUNT_VAR" text where TextMatchMechanism="contains-expected-value".
```

In the example above, we got the username count from database based on the search parameters and stored into EXPECTED_USER_COUNT_VAR variable. Then after search from UI, this expected information is matched with the information present in the label under the table.

4.1.4 Fill UI form information and submit form

Let's say we have user registration form where we would like to fill the data and submit on the server. So, there are 2 cases we are going to mention here:

A. Successful user registration
B. Unsuccessful user registration

Sample **User Registration** page information:

1. **Username**: textbox
2. **First Name:** textbox
3. **Last Name:** textbox
4. **Gender:** combobox, supported dropdown options: Male, Female
5. **Phone Number**: textbox
6. **Email ID:** textbox
7. **Submit button**
8. **Reset button**

Case A: Successful user registration scenario

```
Scenario: Verify successful user registration when user does not exist in the system.
    Given already connected to "myapp" application.
    When open "http://host-name:port/myapp/register-user" URL.
    Then "URL" will be "opened successfully".
    When delete "USERS" table data using query below. Target DB Info [AppName="myapp",
DatabaseProfileName="sample-db"]:
      """
      delete USERS where USERNAME="testuser1"
      """
    Then "the existing testuser1" will be "deleted from USERS table".
    When enter the following form fields information present on "User Registration page":
```

```
      | Page Object / Field Info                                                | Input value(s)
|
      | {name: "myapp.UserRegPO.Textbox_Username", maxTimeToWaitInSeconds: 10} | testuser1
|
      | myapp.UserRegPO.Textbox_FirstName                                       | {value: "Test"}
|
      | myapp.UserRegPO.Textbox_LastName                                        | {value: "User1"}
|
      | myapp.UserRegPO.Combobox_Gender                                         | {value: ["Male"], action:
"select"} |
      | myapp.UserRegPO.Textbox_PhoneNumber                                     | 999-888-7777
|
      | myapp.UserRegPO.Textbox_EmailID                                         | testuser1@uitnet.org
|
    And click "myapp.UserRegPO.Button_Submit" page element to "submit the user registration data".
    Then "the User Registration page data" will be "submitted".
    And verify that the following page elements are visible on "Search Users page":
      | Page Element                                                          |
      | {name: "myapp.UserRegPO.Message_Success", maxTimeToWaitInSeconds: 6}  |
```

In the above example, to test successful registration of user, we first have to make sure that the **testuser1** does not exist in the database. So, scenario is first deleting the user from the database table before submitting the user registration information from UI page for the same user.

Case B: Un-successful user registration scenario

```
Scenario: Verify unsuccessful user registration when user already exist in the system.
    Given already connected to "myapp" application.
    When open "http://host-name:port/myapp/register-user" URL.
    Then "URL" will be "opened successfully".
    When delete "USERS" table data using query below. Target DB Info [AppName="myapp",
DatabaseProfileName="sample-db"]:
      """
      delete USERS where USERNAME="testuser1"
      """
    Then "the existing testuser1" will be "deleted from USERS table".
```

```
    When insert new data into "USERS" table using query below. Target DB Info [AppName="myapp",
DatabaseProfileName="sample-db"]:
      """
      insert into USERS(USERNAME, FIRST_NAME, LAST_NAME, GENDER, PHONE_NUMBER, EMAIL_ID, STATUS)
      values('testuser1', 'Test', 'User1', 'Male', '999-888-7777', 'testuser1@uitnet.org', 'Active')
      """
    When enter the following form fields information present on "User Registration page":
      | Page Object / Field Info                                                | Input value(s)
|
      | {name: "myapp.UserRegPO.Textbox_Username", maxTimeToWaitInSeconds: 10} | testuser1
|
      | myapp.UserRegPO.Textbox_FirstName                                       | {value: "Test"}
|
      | myapp.UserRegPO.Textbox_LastName                                        | {value: "User1"}
|
      | myapp.UserRegPO.Combobox_Gender                                         | {value: ["Male"], action:
"select"} |
      | myapp.UserRegPO.Textbox_PhoneNumber                                     | 999-888-7777
|
      | myapp.UserRegPO.Textbox_EmailID                                         | testuser1@uitnet.org
|
    And click "myapp.UserRegPO.Button_Submit" page element to "submit the user registration data".
    Then "the User Registration page data" will be "submitted".
    And verify that the following page elements are visible on "Search Users page":
      | Page Element                                                                |
      | {name: "myapp.UserRegPO.Message_UserAlreadyExists", maxTimeToWaitInSeconds: 6}     |
```

In the example given above, to test the unsuccessful user registration, scenario is first deleting the **testuser1** and reinserting again to make sure that the user exists in the system before submitting the same **testuser1** data from UI page for registration.

4.1.5 Verify UI form field data like max length, unacceptable characters etc.

Let's say we have user registration form where we would like to verify the form field data.

Sample **User Registration** page information:

9. **Username**: textbox
10. **First Name:** textbox

11. **Last Name:** textbox
12. **Gender:** combobox, supported dropdown options: Male, Female
13. **Phone Number**: textbox
14. **Email ID:** textbox
15. **Submit button**
16. **Reset button**

```
Scenario: Verify form field (Phone Number) limitations on user registration form.
    Given already connected to "myapp" application.
    When open "http://host-name:port/myapp/register-user" URL.
    Then "URL" will be "opened successfully".
    When delete "USERS" table data using query below. Target DB Info [AppName="myapp",
DatabaseProfileName="sample-db"]:
      """
      delete USERS where USERNAME="testuser1"
      """
    Then "the existing testuser1" will be "deleted from USERS table".
    When auto generate "invalid Phone Number [Expected: 10 digits + 2 hyphens(-)]" data based on the JSON
input given below and store into "INVALID_PHONE_NUMBER_VAR" variable:
      """
      {length: 13, maxWordLength: 13, includeAlphabetsLower: true, includeAlphabetsUpper: true,
      alphabetsLower: "abcdefghijklmnopqrstuvwxyz", alphabetsUpper: "ABCDEFGHIJKLMNOPQRSTUVWXYZ",
      includeNumbers: true, numbers: "1234567890", includeSpecialCharacters: true,
      specialCharacters: "`~!@#$%^&*()+={[]}\\|;:'\",<>/?", includeNewLine: false, includeWhiteSpaces:
true,
      includeLeadingWhiteSpace: false, includeTrailingWhiteSpace: false}
      """
    And type "INVALID_PHONE_NUMBER_VAR" text in "myapp.UserRegPO.Textbox_PhoneNumber" page element.
    And get input text value of "myapp.UserRegPO.Textbox_Username" page element and store into
"USERNAME_LATEST_DATA_VAR" variable.
    Then verify valueOf("USERNAME_LATEST_DATA_VAR") variable "!=" valueOf("INVALID_USERNAME_DATA_VAR")
variable.
```

In the above scenario, it is first generating the invalid phone number and that we type on the phone number UI field. (NOTE: If system is not allowing to fill the invalid characters on phone number field, then when we type invalid phone number then it will not allow to

enter all the incorrect characters.) After typing, we read the typed text and matching with the invalid phone number text. It should not match. If both numbers match then text scenario will fail.

Another way of testing the UI form field limitations is by entering the wrong data on a field and submit the form data on server using Submit button and then server will raise an error message, then we can verify the error message in the scenario as mentioned below:

```
Scenario: Verify form field (Phone Number) limitations on user registration form.
    Given already connected to "myapp" application.
    When open "http://host-name:port/myapp/register-user" URL.
    Then "URL" will be "opened successfully".
    When delete "USERS" table data using query below. Target DB Info [AppName="myapp",
DatabaseProfileName="sample-db"]:
      """
      delete USERS where USERNAME="testuser1"
      """
    Then "the existing testuser1" will be "deleted from USERS table".
    When auto generate "invalid Phone Number [Expected: 10 digits + 2 hyphens(-)]" data based on the JSON
input given below and store into "INVALID_PHONE_NUMBER_VAR" variable:
      """
      {length: 13, maxWordLength: 13, includeAlphabetsLower: true, includeAlphabetsUpper: true,
      alphabetsLower: "abcdefghijklmnopqrstuvwxyz", alphabetsUpper: "ABCDEFGHIJKLMNOPQRSTUVWXYZ",
      includeNumbers: true, numbers: "1234567890", includeSpecialCharacters: true,
      specialCharacters: "`~!@#$%^&*()+={[]}\\|;:'\",<>/?", includeNewLine: false, includeWhiteSpaces:
true,
      includeLeadingWhiteSpace: false, includeTrailingWhiteSpace: false}
      """
    When enter the following form fields information present on "User Registration page":
      | Page Object / Field Info                                                | Input value(s)
|
      | {name: "myapp.UserRegPO.Textbox_Username", maxTimeToWaitInSeconds: 10} | testuser1
|
      | myapp.UserRegPO.Textbox_FirstName                                       | {value: "Test"}
|
      | myapp.UserRegPO.Textbox_LastName                                        | {value: "User1"}
|
      | myapp.UserRegPO.Combobox_Gender                                         | {value: ["Male"], action:
"select"} |
      | myapp.UserRegPO.Textbox_PhoneNumber                                     | 999-888-7777
|
```

```
      | myapp.UserRegPO.Textbox_EmailID                                    | INVALID_PHONE_NUMBER_VAR
|
    And click "myapp.UserRegPO.Button_Submit" page element to "submit the user registration data".
    Then "the User Registration page data" will be "submitted".
    And verify that the following page elements are visible on "User Registration page":
      | Page Element                                                                                |
      | {name: "myapp.UserRegPO.Message_ErrorWithInvalidPhoneNumber", maxTimeToWaitInSeconds: 6}    |
```

4.1.6 File download and verify downloaded file and its contents

TBD(documentation is in progress…)

4.1.7 File upload and verify uploaded file and its contents on remote machine

TBD(documentation is in progress…)

4.2 API Testing Automation

To perform the API testing automation, generally we test the HTTP response based on the input that we specified in URL or in request body. The following type of HTTP methods are supported to perform request on server:

A. HTTP GET
B. HTTP POST
C. HTTP PUT
D. HTTP DELETE
E. HTTP HEAD

The sub sections given below lists some examples to perform the API testing.

4.2.1 Verify request and response using HTTP GET

4.3 Relational Database Testing Automation

TBD (documentation is in progress…)

4.4 Remote Machine Data Verification

TBD (documentation is in progress…)

4.5 Downloaded File Data Verification

TBD (documentation is in progress…)

4.6 Cucumber feature file: Run steps conditionally

STAS support conditional steps using that we can create the scenarios in which we can write some steps that can be execute conditionally.

Example:

```
Scenario: Verify the search functionality on Search Users page with valid username.
    Given user is already logged in using "StandardUserProfile" user profile on "myapp" application.
    When click "myapp.MenuItemPO.Menu_SearchUsers" page element to "open Search Users page".
    Then "the Search Users page" will be "opened".
    And verify that the following page elements are visible on "Search Users page":
      | Page Element                                                                  |
      | {name: "myapp.SearchUsersPO.Label_PageTitle", maxTimeToWaitInSeconds: 6} |
    When get "one username" from "USERS" table using query below and store into "VALID_USERNAME_VAR"
variable. Target DB Info [AppName="myapp", DatabaseProfileName="sample-db"]:
      """
      select USERNAME from USERS fetch first 1 row only
      """
    And get "Search results records count" from "USERS" table using query below and store into
"EXPECTED_USER_COUNT_VAR" variable. Target DB Info [AppName="myapp", DatabaseProfileName="sample-db"]:
      """
      select count(1) from USERS where USERNAME = 'VALID_USERNAME_VAR'
      """
    When type "VALID_USERNAME_VAR" text in "myapp.SearchUsersPO.Textbox_Username" page element.
    And click "myapp.SearchUsersPO.Button_Search" page element to "search users".
    Then verify text part of each element of "myapp.SearchUsersPO.Label_FoundNRecords" page element matches
"EXPECTED_USER_COUNT_VAR" text where TextMatchMechanism="contains-expected-value".
```

```
    And condition="CheckExpectedUserCount"-Start> if ["EXPECTED_USER_COUNT_VAR" "!=" "0"] condition is true
then execute the steps below.
    And verify text part of each element of '{name: "myapp.SearchUsersPO.Column_LabelType",
maxTimeToWaitInSeconds: 10, params: {colName: "Username", colId: "username"}}' page element matches
"VALID_USERNAME_VAR" text where TextMatchMechanism="exact-match-with-expected-value".
    And condition="CheckExpectedUserCount"-End.
```

In the example above, the steps mentioned between condition block (Highlighted above) will be executed if and only if the condition is true. When condition is false then the steps listed between the conditional body will get executed with message (will be displayed in report) that:

```
This step is not executed due to false value of condition="CheckExpectedUserCount".
```

This message will inform the user that the step real code is not executed.

NOTE: We can also be able to add nested conditions also. Just to make sure condition name should be unique when it is nested condition.