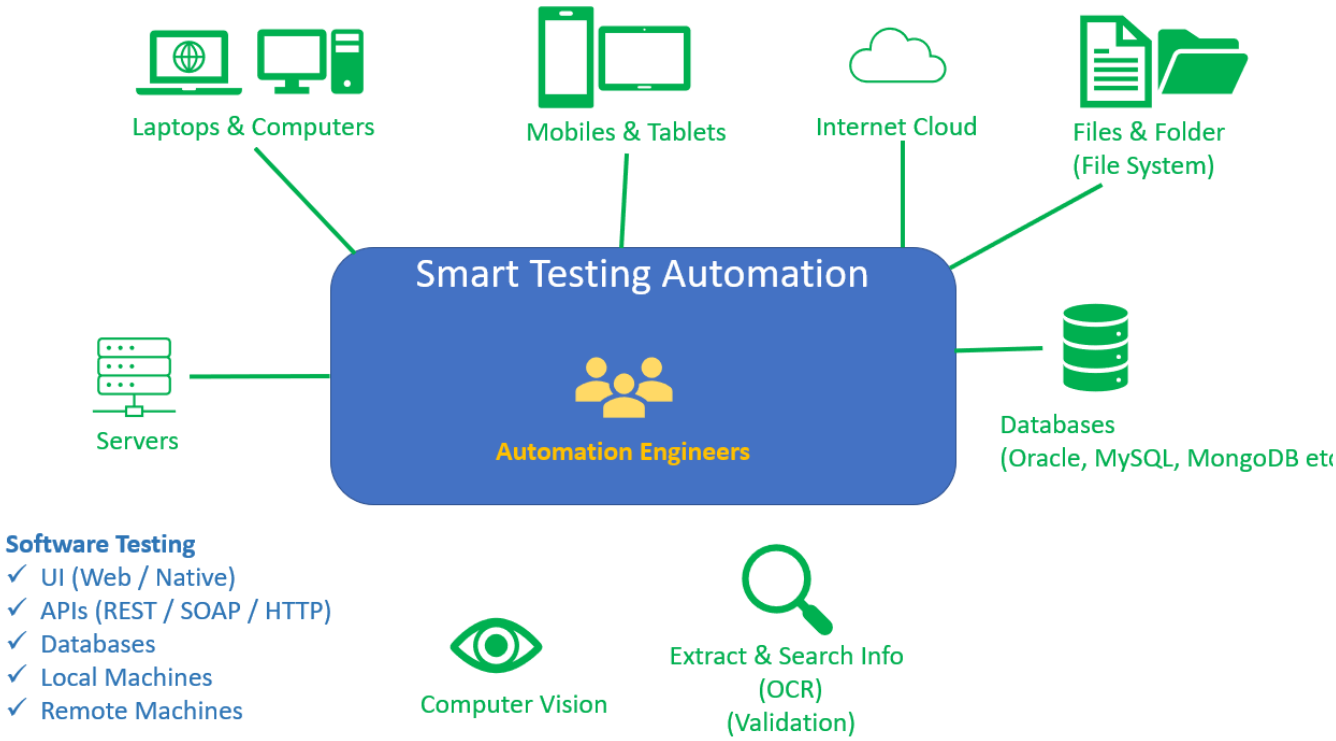

STAS CUCUMBER STEP DEFINITIONS – EN (ENGLISH)

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:

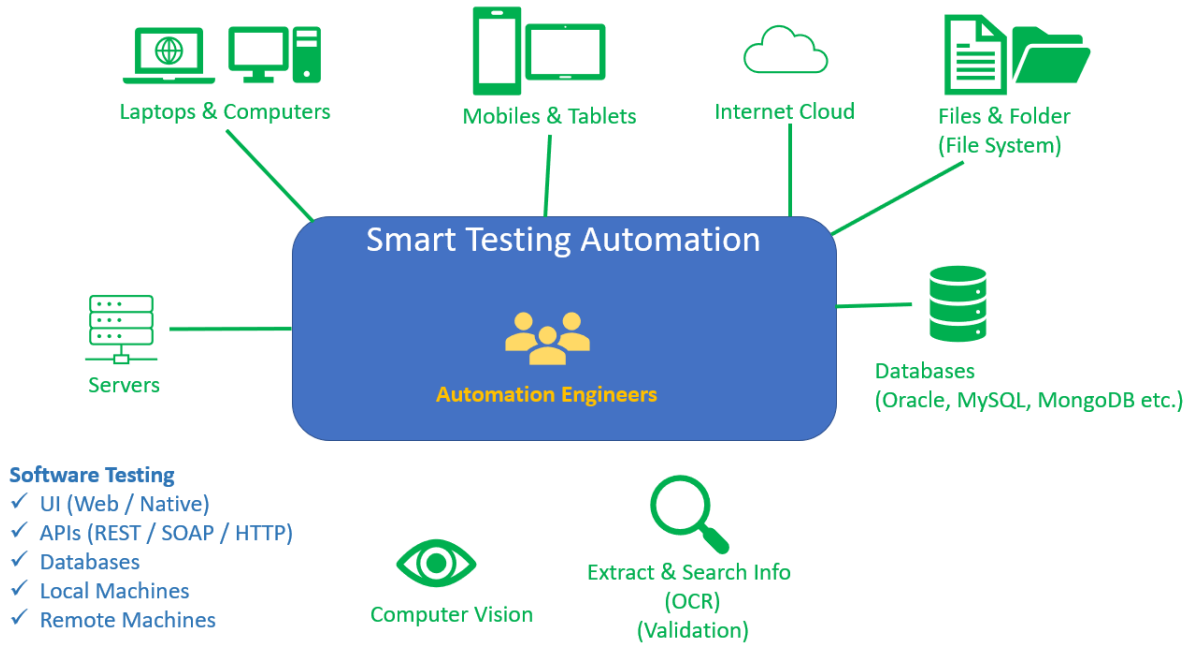

Figure 2-1 STAS Tool Connectivity

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/Gherkin 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 **Sample Scenario / Use Cases** section given below. 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')
      """
    And 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 value of "USERNAME_LATEST_DATA_VAR" variable "!=" value of "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 Verify downloaded file and its contents

Using STAS, downloading the file from the UI page and validation of the contents is simple. See the scenario given below:

```
Scenario: Download users list file and verify its contents.
    Given already connected to "myapp" application.
    And remove "users-list.xlsx" file from the local downloads directory.
    When open "http://host-name:port/myapp/user-list" URL.
    Then "URL" will be "opened successfully".
    When click on "myapp.UserListPO.Link_ExportUsersAsExcel" page element to "download users list as excel
file".
    Then wait for 10 seconds.
    And "the users list excel file" will be "downloaded in test-data/downloads folder".
    And verify that the "users-list" file with extension "xlsx" is downloaded [MaxTimeToWaitInSeconds=5].
    And verify that the downloaded file "users-list.xlsx" contains the following keywords
[KeywordDelimiter=";", InOrder="yes", ShouldPrint="no"]:
      """
      David; Martin Hu; Tom
      """
```

In the scenario given above, we are

- first deleting the expected downloaded file (users-list.xlsx)
- opening the UI page
- clicking in the export users link to download the users-list.xlsx file
- verifying the downloaded file name
- verifying the contents of the file using keywords mechanism. In this step system will check all keywords are present in order means David will come first then Martin Hu then Tom.

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

File upload from Web UI page can be performed as mentioned in the scenario below:

```
Scenario: Upload sample data file and verify its contents on remote machine.
    Given already connected to "myapp" application.
    And remove expected file(s) [RemoteDirectory="/data/uploads", ExpectedFileName="user-guide.docx",
FileNameMatchMechanism="exact-match-with-expected-value"] from remote machine [AppName="myapp",
RemoteMachineName:"test-server"].
    When open "http://host-name:port/myapp/uploads" URL.
    Then "URL" will be "opened successfully".
    When upload '["test-data\\sample-file.pdf"]' file(s) using "myapp.UploadsPO.InputFile_SelectFiles" page
element.
    And wait for 2 seconds.
    And click on "myapp.UploadsPO.Button_UploadFiles" page element to "upload files on the server".
    Then "the file" will be "uploaded on server".
    And verify that the expected files [RemoteDirectory="/data/uploads", ExpectedFileName="user-
guide.docx", FileNameMatchMechanism="exact-match-with-expected-value"] are uploaded on remote machine
[AppName="myapp", RemoteMachineName:"test-server", MaxTimeToWaitInSeconds=10].
```

In the scenario above, testing file upload, we should do the things in order given below to make sure file is uploaded correctly:

- Remove the uploading file first from the target server
- Open web page and upload the file by providing the file path and clicking on the Upload files buttons.
- Then verify the file presence on remote target server.

If we would like check the contents of the uploaded file then we have to download that file from remote target server to local server and we can be able to verify the contents as per the scenario steps given below:

NOTE: Delete the expecting downloading file from the local downloads directory to make sure existing file is not present.

```
    And remove "user-guide.docx" file from the local downloads directory.
    And download expected remote file(s) [RemoteDirectory="/data/uploads", ExpectedFileName="user-
guide.docx", FileNameMatchMechanism="exact-match-with-expected-value"] from remote machine
[AppName="myapp", RemoteMachineName:"test-server"].
    And verify that the downloaded file "user-guide.docx" contains the following keywords
[KeywordDelimiter=";", InOrder="yes", ShouldPrint="no"]:
        """
        Supported Platforms; Supported Web Browsers
        """
```

4.1.8 Switching windows / tabs to perform operations

In the scenario given below, we mentioned how to open new window and how to switch to new window to verify some contents and how to switch back to main window.

```
Scenario: Verify contents on different window / tab on web UI application.
    Given already connected to "myapp" application.
    When open "http://host-name:port/myapp/users-list" URL.
    Then "URL" will be "opened successfully with available users".
    When click on "myapp.UsersListPO.Checkbox_SelectFirstUser" page element to "select first user in users
list".
    And click on "myapp.UsersListPO.Button_UserInfo" page element to "open new window to display user".
    Then "the new window containing the selected user information" will be "opened".
    And switch to "UserInfoWindowHandle" window.
    And verify that the following page elements are visible on "User Information page":
      | Page Element |
      | {name: "myapp.UserInfoPO.PageTitle_UserInformation", maxTimeToWaitInSeconds: 6} |
    And switch to default content.
```

The highlighted section above describes the switching to newly opened window and switch back to main window.

4.1.9 Switching frames in web page to perform operations

In the scenario given below, we mentioned how to open selected user record into new frame and switching to frame to verify the contents inside the frame.

```
Scenario: Verify contents on different frame/iframe of web UI page.
    Given already connected to "myapp" application.
    When open "http://host-name:port/myapp/users-list" URL.
    Then "URL" will be "opened successfully with available users".
    When click on "myapp.UsersListPO.Checkbox_SelectFirstUser" page element to "select first user in users
list".
    And click on "myapp.UsersListPO.Button_UserInfo" page element to "open user information below the users
table".
    Then "the user information section for selected user" will be "displayed under users table. This
section is present in different iframe named userInfoFrame.".
    And switch to "userInfoFrame" frame.
    And verify that the following page elements are visible on "User Information page":
```

```
      | Page Element                                                                  |
      | {name: "myapp.UserInfoPO.PageTitle_UserInformation", maxTimeToWaitInSeconds: 6} |
    And switch to "parent" frame.
```

The highlighted section above describes the switching to frame and switch back to parent frame. Also we can use the following step to switch back to parent frame.

```
    And switch to default content.
```

4.1.10 Change App Driver / Web Driver settings in scenarios

We can be able to change the app driver / web driver properties as the first step in the scenario. So that all the other scenario steps can run the updated driver settings. See the highlighted step in the scenario given below:

```
Scenario: Change application driver / web driver settings and perform UI operations.
    Given set the following app driver properties for [AppName="myapp", WebBrowser="chrome"]:
      """
      { unexpectedAlertBehaviour: "dismiss and notify", driverCapabilities: { "download.default_directory":
"test-results/downloads"}}
      """
    And 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                          |
```

NOTE: For native application, we have to set WebBrowser="not-applicable" Or use the following step to set the driver settings:

```
    Given set the following app driver properties for [AppName="myapp"]:
          """
          { driverCapabilities: {deviceName: "emulator-5554", platformName: "android", automationName:
    "UIAutomator2", appPackage: "com.android.calculator2", appActivity: ".Calculator"} }
          """
```

4.1.11 Perform shortcut keys operations on page element

We can use the selenium web driver key controls to perform multiple keys operations together to perform shortcut key task on any of the web page. See the scenario below for an example:

```
Scenario: Perform shortcut key (ctrl + T) operation to open a new window.
    Given already connected to "myapp" application.
    When open "http://host-name:port/myapp/users-list" URL.
    Then "URL" will be "opened successfully with available users".
    When use '["CONTROL", "T"]' key(s) on "myapp.CommonPO.BodyElement" page element to "open new tab / window".
    Then "the new tab / window" will be "opened".
```

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.

NOTE: API target servers must be configured in application ApiConfig.yaml file.

4.2.1 Verify request and response using HTTP GET

Sample Response Data in JSON format:

```
[
  {
    "userName": "User 1",
```

```
        "phone": "999-888-7777",
        "email": "user1@uitnet.org",
        "city": "Centreville",
        "state": "VA",
        "country": "US",
        "zip": "20120"
    },
    {
        "userName": "User 2",
        "phone": "999-888-7777",
        "email": "user2@uitnet.org",
        "city": "Fairfax",
        "state": "VA",
        "country": "US",
        "zip": "20230"
    }
]
```


To perform the HTTP GET API call, we can use the following steps as mentioned in scenario below:

```
Scenario: Verify get users list based on parameters using HTTP GET.
    When make HTTP GET request on target server [AppName="myapp", TargetServer="myapp-services",
TargetURL="get-users?country=US&state=VA"] using [UserProfile="StandardUserProfile"] with header info
[Accept="application/json"] and variable info [RespVar="HTTP_RESP_VAR"].
    Then verify "HTTP_RESP_VAR" HTTP response contains HTTPStatusCode=200.
    And verify "HTTP_RESP_VAR" HTTP response contains following header information:
      | Header Name  | Expected Value   | Text Match Mechanism                |
      | Content-Type | application/json | ic-exact-match-with-expected-value |
    And verify "HTTP_RESP_VAR" HTTP response contains JSON data with the following expected params
information:
      | Parameter/JSON Path                              | Operator    | Expected Information
|
      | {path: "$[*].country", valueType: "string-list"}  | =           | US
|
      | {path: "$[*].state", valueType: "string-list"}    | starts-with | {ev: "VA", textMatchMechanism:
"starts-with-expected-value"} |
      | {path: "$[*].userName", valueType: "string-list"} | !=          | {ev: ""}
|      | {path: "$[*].userName", valueType: "string-list"} | !=          | {ev: ""}
|
```

In the example scenario given above, highlighted step is used to make HTTP GET call to get the users list from myapp-services API server using parameterized URL. Here user profile is used to provide credentials so that server can authenticate the user. This call is expecting response to be in JSON format in response data. The response is going to get stored in HTTP_RESP_VAR variable so that this information can be used in next steps to verify the response information. As part of the verification, the above scenario is doing the following verification:

- HTTP Response Code
- Response header parameters
- Response body

To verify the JSON data, we have to use the JSON path mechanism to retrieve the data from JSON document and check with the expected information as mentioned above.

4.2.2 Verify request and response using HTTP POST

Sample Response Data in JSON format:


```
[
   {
      "userName": "User 1",
      "phone": "999-888-7777",
      "email": "user1@uitnet.org",
      "city": "Centreville",
      "state": "VA",
      "country": "US",
      "zip": "20120"
   },
   {
      "userName": "User 2",
      "phone": "999-888-7777",
      "email": "user2@uitnet.org",
      "city": "Fairfax",
      "state": "VA",
      "country": "US",
      "zip": "20230"
   }
```

]

To perform the HTTP POST API call, we can use the following steps as mentioned in scenario below:

```
Scenario: Verify get users list based on JSON request using HTTP POST.
    When make HTTP POST request using the following request body on target server [AppName="myapp",
TargetServer="myapp-services", TargetURL="get-users"] using [UserProfile="StandardUserProfile"] with header
info [ContentType="application/json", Accept="application/json"] and variable info [ReqVar="HTTP_REQ_VAR",
RespVar="HTTP_RESP_VAR"]:
    """
    {
        "state": "VA",
        "country": "US"
    }
    """
    Then verify "HTTP_RESP_VAR" HTTP response contains HTTPStatusCode=200.
    And verify "HTTP_RESP_VAR" HTTP response contains following header information:
      | Header Name  | Expected Value   | Text Match Mechanism               |
      | Content-Type | application/json | ic-exact-match-with-expected-value |
    And verify "HTTP_RESP_VAR" HTTP response contains JSON data with the following expected params
information:
      | Parameter/JSON Path                            | Operator    | Expected Information
|
      | {path: "$[*].country", valueType: "string-list"}  | =           | US
|
      | {path: "$[*].state", valueType: "string-list"}    | starts-with | {ev: "VA", textMatchMechanism:
"starts-with-expected-value"} |
      | {path: "$[*].userName", valueType: "string-list"} | !=          | {ev: ""}
|
```

In the above example scenario, highlighted step is used to perform the HTTP POST call using the JSON body to search users from the server. After that we can verify the response received from the server.

4.2.3 Verify request and response using HTTP PUT

HTTP PUT call is similar to HTTP POST (as mentioned in previous section) but we have to use the HTTP PUT method in highlighted step as given below:

```
Scenario: Verify get users list based on JSON request using HTTP PUT.
    When make HTTP PUT request using the following request body on target server [AppName="myapp",
TargetServer="myapp-services", TargetURL="get-users"] using [UserProfile="StandardUserProfile"] with header
info [ContentType="application/json", Accept="application/json"] and variable info [ReqVar="HTTP_REQ_VAR",
RespVar="HTTP_RESP_VAR"]:
    """
    {
        "state": "VA",
        "country": "US"
    }
    """
    Then verify "HTTP_RESP_VAR" HTTP response contains HTTPStatusCode=200.
    And verify "HTTP_RESP_VAR" HTTP response contains following header information:
      | Header Name  | Expected Value   | Text Match Mechanism                |
      | Content-Type | application/json | ic-exact-match-with-expected-value |
    And verify "HTTP_RESP_VAR" HTTP response contains JSON data with the following expected params
information:
      | Parameter/JSON Path                              | Operator    | Expected Information
|
      | {path: "$[*].country", valueType: "string-list"}  | =           | US
|
      | {path: "$[*].state", valueType: "string-list"}    | starts-with | {ev: "VA", textMatchMechanism:
"starts-with-expected-value"} |
      | {path: "$[*].userName", valueType: "string-list"} | !=          | {ev: ""}
|
```

4.2.4 Verify request and response using HTTP DELETE

Sample Response:

{ message: "User deleted successfully."}

This example scenarios shows how to delete the user using HTTP DELETE API call.

```
Scenario: Verify delete user using HTTP DELETE.
    When make HTTP DELETE request on target server [AppName="myapp", TargetServer="myapp-services",
TargetURL="delete-user?userName=User 1"] using [UserProfile="StandardUserProfile"] with header info
[Accept="application/json"] and variable info [RespVar="HTTP_RESP_VAR"].
    Then verify "HTTP_RESP_VAR" HTTP response contains HTTPStatusCode=200.
    And verify "HTTP_RESP_VAR" HTTP response contains following header information:
      | Header Name  | Expected Value   | Text Match Mechanism                |
      | Content-Type | application/json | ic-exact-match-with-expected-value |
    And verify "HTTP_RESP_VAR" HTTP response contains JSON data with the following expected params
information:
      | Parameter/JSON Path | Operator | Expected Information       |
      | {path: "$.message"} | =        | User deleted successfully. |
```

4.2.5 Verify request and response using HTTP HEAD

The following example show how to use HTTP HEAD API call and verify the response data.

```
Scenario: Verify response header information using HTTP HEAD.
    When make HTTP HEAD request on target server [AppName="myapp", TargetServer="myapp-services",
TargetURL="delete-user?userName=User 1"] using [UserProfile="StandardUserProfile"].
    Then verify "HTTP_RESP_VAR" HTTP response contains HTTPStatusCode=200.
    And verify "HTTP_RESP_VAR" HTTP response contains following header information:
      | Header Name  | Expected Value   | Text Match Mechanism                |
      | Content-Type | application/json | ic-exact-match-with-expected-value |
```

4.2.6 Verify file download using API

The example scenario given below is used to download the user list excel file using HTTP GET and verifies the contents of the file:

```
Scenario: Verify downloaded user list excel file contents using API Call.
    When make HTTP GET request on target server [AppName="myapp", TargetServer="myapp-services",
TargetURL="export-users-as-excel?country=US&state=VA"] using [UserProfile="StandardUserProfile"] with
header info [Accept="application/vnd.openxmlformats-officedocument.spreadsheetml.sheet"] and variable info
[RespVar="HTTP_RESP_VAR"].
    Then verify "HTTP_RESP_VAR" HTTP response contains HTTPStatusCode=200.
    And verify "HTTP_RESP_VAR" HTTP response contains following header information:
      | Header Name  | Expected Value   | Text Match Mechanism                |
```

```
      | Content-Type | application/vnd.openxmlformats-officedocument.spreadsheetml.sheet | ic-exact-match-
with-expected-value |
    And verify downloaded file as part of "HTTP_RESP_VAR" HTTP response contains following expected name:
      """
      {expectedFileName: "user-list.xlsx", textMatchMechanism: "starts-with-expected-value", deleteFile:
false}
      """
    And verify downloaded file as part of "HTTP_RESP_VAR" HTTP response contains following keywords in its
contents:
      """
      { keywords: ["User 1", "User 2"], inOrder: "yes"}
      """
```

We can use any HTTP API to download the file and same way we can verify the contents of the downloaded file.

4.2.7 Verify file upload using API

Sample Response:

{ message: " File(s) uploaded successfully."}

The example scenario given below is used to upload the file on the remote server using API call:

```
Scenario: Verify uploaded file on remote machine using API Call.
    When make HTTP POST request to upload the following files on target server [AppName="myapp",
TargetServer="myapp-services", TargetURL="upload-files"] using [UserProfile="StandardUserProfile"] with
header info [ContentType="multipart/form-data", Accept="application/json"] and variable info
[ReqVar="HTTP_REQ_VAR", RespVar="HTTP_RESP_VAR"]:
      | Part Name      | File Name      | Content Type    | Apply Variables | Relative File Path (Relative
to project directory) |
      | SampleFile.pdf | SampleFile.pdf | application/pdf | no              | test-data/uploads/Sample.pdf
|
    Then verify "HTTP_RESP_VAR" HTTP response contains HTTPStatusCode=200.
    And verify "HTTP_RESP_VAR" HTTP response contains following header information:
      | Header Name  | Expected Value   | Text Match Mechanism                |
      | Content-Type | application/json | ic-exact-match-with-expected-value |
```

```
    And verify "HTTP_RESP_VAR" HTTP response contains JSON data with the following expected params
information:
      | Parameter/JSON Path | Operator | Expected Information           |
      | {path: "$.message"} | =        | File(s) uploaded successfully. |
    And verify that the expected files [RemoteDirectory="/data/uploads/",
ExpectedFileName="SampleFile.pdf", FileNameMatchMechanism="exact-match-with-expected-value"] are uploaded
on remote machine [AppName="myapp", RemoteMachineName:"api-server", MaxTimeToWaitInSeconds=10].
```

NOTE: Also we can be able to upload multiple files in a single API call.

4.3 Relational Database Testing Automation

Mostly time to time we have to verify the database information is correct. Sometimes what happens like if there is any defect in application then database is not updated properly due to that data in the database become inconsistent or invalid, that creates lot of problem. There are many scenarios for database testing automation like:

A. Verify column data is correct
B. Read valid data from database and verify the UI page functionality or API functionality
C. Prepare data (Insert new data) before executing the scenario to verify API functionality or UI page functionality
D. Delete data from database table to make sure some records do not exist so that we can create those records from UI page or API
E. Update records information in database to some other value so that we can update that particular information using UI page or API

To handle all these kinds of database CRUD (Create, Read, Update, Delete) operations, we have the following high level example scenarios that can help handling database table operations.

4.3.1 Verify invalid values in database column

This is a sample scenario that validates the null value in database column,

```
Scenario: Verify invalid values in database table column.
    When get "record count with not null phone numbers" from "USERS" table using query below and store into
"REC_COUNT_WITH_NOT_NUL_PHONE_NUMBER_VAR" variable. Target DB Info [AppName="myapp",
DatabaseProfileName="sample-db"]:
      """
      select count(1) from USERS where PHONE_NUMBER is null
```

```
      """
    Then verify value of "REC_COUNT_WITH_NOT_NUL_PHONE_NUMBER_VAR" variable "=" "0".
```

4.3.2 Read data from database

This sample scenario is used to get the EMAIL column data from USERS table present in sample-db database and store it into EMAIL_LIST_VAR variable, then verifies each element of the EMAIL_LIST_VAR variable contains '@' text.

```
Scenario: Verify email column in USERS table that must contain '@' character.
    When get all entries of "EMAIL" column from "USERS" table using query below and store into
"EMAIL_LIST_VAR" variable. Target DB Info [AppName="myapp", DatabaseProfileName="sample-db"]:
      """
      select EMAIL from USERS
      """
    Then verify each element of "EMAIL_LIST_VAR" list variable matches with "@" text where
TextMatchMechanism="contains-expected-value".
```

4.3.3 Update data in database

The sample scenario given below is first updating the USERS table data for some columns and then makes the HTTP GET API call to get the same updated user information and verifies the API response contains the expected information that should be same as updated into the database for the same user.

```
Scenario: Verify updated user information in database is correctly retrieved in API call.
    Given update "USERS" table data using query below. Target DB Info [AppName="myapp",
DatabaseProfileName="sample-db"]:
      """
        update USERS set EMAIL='testuser1@uitnet.org', PHONE_NUMBER='999-888-7777' where
USERNAME='testuser1'
      """
    When make HTTP GET request on target server [AppName="myapp", TargetServer="myapp-services",
TargetURL="get-user?username=testuser1"] using [UserProfile="StandardUserProfile"] with header info
[Accept="application/json"] and variable info [RespVar="HTTP_RESP_VAR"].
    And verify "HTTP_RESP_VAR" HTTP response contains HTTPStatusCode=200.
    And verify "HTTP_RESP_VAR" HTTP response contains JSON data with the following expected params
information:
      | Parameter/JSON Path | Operator | Expected Information |
      | $.email             | =        | testuser1@uitnet.org |
      | $.phoneNumber       | =        | 999-888-7777         |
```

4.3.4 Delete data from database

The sample scenario given below is first deleting the user from database so that we can successfully register the same user from User Registration UI page without getting any error.

```
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}     |
```

4.3.5 Insert new data into database

The sample scenario given below describes how to perform the unsuccessful user registration when the user is already present in the database. To do so, in the beginning, we delete the **testuser1** user and re-insert into the database table to make sure user is already present in the system and the from the User Registration page enter the testuser1 information and submit the data. In this case the user registration request should get failed as registering user is already present into the database table.

```
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')
      """
    And 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}  |
```

4.4 Local Machine Data: Read data from files and verify contents

As part of the local machine (Local file system) data verification, on a high level, we do the following things:

1. Read data from different types of files i.e. Excel, CSV, JSON, YAML, XML, Text files.
2. Verify the downloaded file and its contents

4.4.1 Read data from Microsoft EXCEL file

Sample USA-States.xlsx file data:

Sheet Name: USA States Information

USA State Code	USA State Name	USA State Abbreviation
01	Alabama	AL
…	…	…

The example scenario given below is reading the USA State's Code information from **USA-States.xlsx** file and verifies the states code returned from API call are from the USA States code list that we retrieved from USA-States.xlsx file.

```
Scenario: Verify user's State Code information as part of API call.
    Given read "USA State Information" sheet data of "test-data/USA-States.xlsx" excel file into tabular
form and store into "USA_STATES_SHEET_DATA_VAR" excelsheet variable.
    And read "USA State Code" column data from "USA_STATES_SHEET_DATA_VAR" excelsheet variable and store
into "USA_STATE_CODE_LIST_VAR" variable.
    And convert "USA_STATE_CODE_LIST_VAR" list variable value into plain text using joiner=",",
valuePrefix='"', valueSuffix='"' and store into "USA_STATE_CODE_LIST_AS_CSV_VAR" variable.
    When make HTTP GET request on target server [AppName="myapp", TargetServer="myapp-services",
TargetURL="get-all-users"] using [UserProfile="StandardUserProfile"] with header info
[Accept="application/json"] and variable info [RespVar="HTTP_RESP_VAR"].
    And verify "HTTP_RESP_VAR" HTTP response contains HTTPStatusCode=200.
    And verify "HTTP_RESP_VAR" HTTP response contains JSON data with the following expected params
information:
```

```
      | Parameter/JSON Path                                  | Operator | Expected Information
|
      | {path: "$[*].stateCode", valueType: "string-list"} | in       | {ev:
[USA_STATE_CODE_LIST_AS_CSV_VAR], valueType: "string-list"} |
```

4.4.2 Read data from CSV file

Sample USA-States.csv file data:

USA State Code, USA State Name, USA State Abbreviation

01, Alabama, AL

… … …

… … …

The example scenario given below is reading the USA State's Code information from **USA-States.csv** file and verifies the states code returned from API call are from the USA States code list that we retrieved from USA-States.csv file.

```
Scenario: Verify user's State Code information as part of API call.
    Given read "test-data/USA-States.csv" CSV file into tabular form and store into
"USA_STATES_TABLE_DATA_VAR" CSV variable.
    And read "USA State Code" column data from "USA_STATES_TABLE_DATA_VAR" CSV variable and store into
"USA_STATE_CODE_LIST_VAR" variable.
    And convert "USA_STATE_CODE_LIST_VAR" list variable value into plain text using joiner=",",
valuePrefix='"', valueSuffix='"' and store into "USA_STATE_CODE_LIST_AS_CSV_VAR" variable.
    When make HTTP GET request on target server [AppName="myapp", TargetServer="myapp-services",
TargetURL="get-all-users"] using [UserProfile="StandardUserProfile"] with header info
[Accept="application/json"] and variable info [RespVar="HTTP_RESP_VAR"].
    And verify "HTTP_RESP_VAR" HTTP response contains HTTPStatusCode=200.
    And verify "HTTP_RESP_VAR" HTTP response contains JSON data with the following expected params
information:
      | Parameter/JSON Path                                  | Operator | Expected Information
|
      | {path: "$[*].stateCode", valueType: "string-list"} | in       | {ev:
[USA_STATE_CODE_LIST_AS_CSV_VAR], valueType: "string-list"} |
```

4.4.3 Read data from JSON file

Sample JSON file (Name: USA-States.json):

[

{usaStateCode: "01", usaStateName: "Alabama", usaStateAbbreviation: "AL"},

… … …

]

The example scenario given below is reading the USA State's Code information from **USA-States.json** file and verifies the states code returned from API call are from the USA States code list that we retrieved from USA-States.json file.

```
Scenario: Verify user's State Code information as part of API call.
    Given read "test-data/USA-States.json" JSON file contents and store into "USA_STATES_JSON_DATA_VAR"
variable.
    And read "$[*].usaStateCode" parameter value from JSON object
[JSONObjRefVariable="USA_STATES_JSON_DATA_VAR"] and store into "USA_STATE_CODE_LIST_VAR" variable.
    And convert "USA_STATE_CODE_LIST_VAR" list variable value into plain text using joiner=",",
valuePrefix='"', valueSuffix='"' and store into "USA_STATE_CODE_LIST_AS_CSV_VAR" variable.
    When make HTTP GET request on target server [AppName="myapp", TargetServer="myapp-services",
TargetURL="get-all-users"] using [UserProfile="StandardUserProfile"] with header info
[Accept="application/json"] and variable info [RespVar="HTTP_RESP_VAR"].
    And verify "HTTP_RESP_VAR" HTTP response contains HTTPStatusCode=200.
    And verify "HTTP_RESP_VAR" HTTP response contains JSON data with the following expected params
information:
      | Parameter/JSON Path                                | Operator | Expected Information
|
      | {path: "$[*].stateCode", valueType: "string-list"} | in       | {ev:
[USA_STATE_CODE_LIST_AS_CSV_VAR], valueType: "string-list"} |
```

4.4.4 Read data from XML file

Sample XML file (Name: USA-States.xml):

<usa-states>

<usa-state state-code="01" state-name="Alabama" state-abbr="AL"/>

… … …

</usa-states>

The example scenario given below is reading the USA State's Code information from **USA-States.xml** file and verifies the states code returned from API call are from the USA States code list that we retrieved from USA-States.xml file.

```
Scenario: Verify user's State Code information as part of API call.
    Given read "test-data/USA-States.xml" XML file contents and store into "USA_STATES_XML_DATA_VAR"
variable.
    And read "//usa-states/usa-state/@state-code" parameter value from XML object
[XMLObjRefVariable="USA_STATES_JSON_DATA_VAR"] and store into "USA_STATE_CODE_LIST_VAR" variable.
    And convert "USA_STATE_CODE_LIST_VAR" list variable value into plain text using joiner=",",
valuePrefix='"', valueSuffix='"' and store into "USA_STATE_CODE_LIST_AS_CSV_VAR" variable.
    When make HTTP GET request on target server [AppName="myapp", TargetServer="myapp-services",
TargetURL="get-all-users"] using [UserProfile="StandardUserProfile"] with header info
[Accept="application/json"] and variable info [RespVar="HTTP_RESP_VAR"].
    And verify "HTTP_RESP_VAR" HTTP response contains HTTPStatusCode=200.
    And verify "HTTP_RESP_VAR" HTTP response contains JSON data with the following expected params
information:
      | Parameter/JSON Path                                | Operator | Expected Information
|
      | {path: "$[*].stateCode", valueType: "string-list"} | in       | {ev:
[USA_STATE_CODE_LIST_AS_CSV_VAR], valueType: "string-list"} |
```

4.4.5 Read data from YAML file

Sample JSON file (Name: USA-States.yaml):

usaStates:

- usaState:

  stateCode: 01

  stateName: Alabama

stateAbbreviaton: AL

- usaState

… … …

--- -- ---

The example scenario given below is reading the USA State's Code information from **USA-States.json** file and verifies the states code returned from API call are from the USA States code list that we retrieved from USA-States.json file.

```
Scenario: Verify user's State Code information as part of API call.
    Given read "test-data/USA-States.yaml" YAML file contents and store into "USA_STATES_YAML_DATA_VAR"
variable.
    And read "$.usaStates.usaState[*].stateCode" parameter value from YAML object
[YAMLObjRefVariable="USA_STATES_YAML_DATA_VAR"] and store into "USA_STATE_CODE_LIST_VAR" variable.
    And convert "USA_STATE_CODE_LIST_VAR" list variable value into plain text using joiner=",",
valuePrefix='"', valueSuffix='"' and store into "USA_STATE_CODE_LIST_AS_CSV_VAR" variable.
    When make HTTP GET request on target server [AppName="myapp", TargetServer="myapp-services",
TargetURL="get-all-users"] using [UserProfile="StandardUserProfile"] with header info
[Accept="application/json"] and variable info [RespVar="HTTP_RESP_VAR"].
    And verify "HTTP_RESP_VAR" HTTP response contains HTTPStatusCode=200.
    And verify "HTTP_RESP_VAR" HTTP response contains JSON data with the following expected params
information:
      | Parameter/JSON Path                                | Operator | Expected Information
|
      | {path: "$[*].stateCode", valueType: "string-list"} | in       | {ev:
[USA_STATE_CODE_LIST_AS_CSV_VAR], valueType: "s
```

4.4.6 Read data from plain text file

The highlighted step given below is used to read the JSON file contents as plain text. The read data is stored in a variable that can be used in further steps to complete the scenario.

```
Scenario: Verify user's State Code information as part of API call.
    Given read "test-data/user-registration-request.json" file data as text and store into
"USER_REG_JSON_DATA_VAR" variable.
```

```
    And apply existing variables value on the plain text data present in "USER_REG_JSON_DATA_VAR" variable
and store into "USER_REG_JSON_DATA_VAR" variable.
    # ... ... ...
    # ... ... ...
```

4.4.7 Verify downloaded file and its contents

The sample scenario given below is used to do download the **users-list.xlsx** file and verifies its contents.

```
Scenario: Download users list file and verify its contents.
    Given already connected to "myapp" application.
    And remove "users-list.xlsx" file from the local downloads directory.
    When open "http://host-name:port/myapp/user-list" URL.
    Then "URL" will be "opened successfully".
    When click on "myapp.UserListPO.Link_ExportUsersAsExcel" page element to "download users list as excel
file".
    Then wait for 10 seconds.
    And "the users list excel file" will be "downloaded in test-data/downloads folder".
    And verify that the "users-list" file with extension "xlsx" is downloaded [MaxTimeToWaitInSeconds=5].
    And verify that the downloaded file "users-list.xlsx" contains the following keywords
[KeywordDelimiter=";", InOrder="yes", ShouldPrint="no"]:
      """
      David; Martin Hu; Tom
      """
```

4.5 Remote Machine Data Verification

The remote machines are the machines generally where the application server is deployed. So based on that the following scenarios may be applicable:

A. Upload files and verify uploaded files are present on remote server
B. Verify uploaded file contents
C. Download files from the remote machine to local server

4.5.1 Verify remote file exists

The highlighted step in the sample scenario is used to verify the uploaded file is present on remote server.

```
Scenario: Upload sample data file and verify its contents on remote machine.
    Given already connected to "myapp" application.
    And remove expected file(s) [RemoteDirectory="/data/uploads", ExpectedFileName="user-guide.docx",
FileNameMatchMechanism="exact-match-with-expected-value"] from remote machine [AppName="myapp",
RemoteMachineName:"test-server"].
    When open "http://host-name:port/myapp/uploads" URL.
    Then "URL" will be "opened successfully".
    When type the following text in "myapp.UploadsPO.InputFile_Uploads" page element:
      """
      test-data/uploads/user-guide.docx
      """
    When click on "myapp.UploadsPO.Button_UploadFiles" page element to "upload files on the server".
    And wait for 10 seconds.
    Then "the file" will be "uploaded on server".
    And verify that the expected files [RemoteDirectory="/data/uploads", ExpectedFileName="user-
guide.docx", FileNameMatchMechanism="exact-match-with-expected-value"] are uploaded on remote machine
[AppName="myapp", RemoteMachineName:"test-server", MaxTimeToWaitInSeconds=10].
```

4.5.2 Upload file on remote machine and verify its contents

The sample scenario given below is used to upload the **user-guide.pdf** file on the remote server and then it makes HTTP GET API call to download the same PDF file and then verify the file name and the contents inside the file.

```
Scenario: Upload file on remote machine and verify that the file is present in API call.
    When upload local file(s) [LocalDirectory="test-data", ExpectedFileName="user-guide.pdf",
FileNameMatchMechanism="exact-match-with-expected-value"] on remote machine [AppName="myapp",
RemoteMachineName:"test-server", RemoteDirectory="/data/uploads"].
    Then "the file" will be "uploaded on test-server".
    When make HTTP GET request on target server [AppName="myapp", TargetServer="myapp-services",
TargetURL="get-file?name=user-guide.pdf"] using [UserProfile="StandardUserProfile"] with header info
[Accept="application/pdf"] and variable info [RespVar="HTTP_RESP_VAR"].
    Then verify downloaded file as part of "HTTP_RESP_VAR" HTTP response contains following expected name:
      """
      {expectedFileName: "(user-guide.*.pdf)", textMatchMechanism: "match-with-regular-expression",
deleteFile: false}
```

```
      """
    Then verify downloaded file as part of "HTTP_RESP_VAR" HTTP response contains following keywords in its
contents:
      """
      { keywords: ["Supported Platforms", "Supported Web Browsers"], inOrder: "yes" }
      """
```

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.

4.7 Cucumber feature file: prepare scenario outline to execute set of steps for different input datasets

Cucumber gherkin language is used to create the scenario outlines that can run the steps for different input data. See the example below:

The scenario outline given below is used to verify the element visibility based on different logged in user.

```
Scenario Outline: Verify the home page information based on different user profiles/roles.
    When connect to "myapp" application using "<User Profile Name>" user profile.
    And click on "myapp.HomePO.Link_Home" page element to "open Home page".
    Then verify "<Visible Element>" page element is visible on "home page".
    And verify "<Hidden Element>" page element is hidden on "home page".
    Examples:
      | User Profile Name   | Visible Element                | Hidden Element                  |
      | StandardUserProfile | myapp.HomePO.List_Users        | myapp.HomePO.Button_UpdateUsers |
      | AdminUserProfile    | myapp.HomePO.Button_UpdateUsers | myapp.HomePO.List_Announcements |
```