Rapise Automation

Guide to Assertion Methods

Version History

Version #	Author	Date	Version Description
1.0	Alwin M. Mangampo	March 20, 2024	

Table of Contents

Version History 1
Table of Contents 2
1. Introduction 3
  1.1. Purpose of Document 3
2. Rapise Asserts and Definitions 3
3. Common Validations and Appropriate Assert Methods 5

1. Introduction

1.1. Purpose of Document

The document is designed with the aim to:

a. Provide guidance and information on assertion methods in Rapise automation. It aims to educate users on the different assertion techniques available in Rapise, their significance in automated testing, and how to effectively utilize them to verify the correctness of test results.

b. Provide sample scenarios or use cases to demonstrate the practical application of assertion methods in automated testing workflows.

Overall, the document seeks to empower users with the knowledge and skills needed to enhance the quality and reliability of their automated tests using Rapise.

2. Rapise Asserts and Definitions

The following table provides a list of Rapise assert methods along with their definitions:

Assert	Description
Tester.Assert()	● Asserting a condition is true.
Tester.AssertContains()	● Asserts that a string contains another string.
Tester.AssertEndsWith()	● Asserts that a string ends with another string.
Tester.AssertEqual()	● Asserts that two values are equal.
Tester.AssertFalse()	● Asserts that a condition is false.
Tester.AssertGreater()	● Asserts that one value is greater than another.
Tester.AssertGreaterOrEqual()	● Asserts that one value is greater than or equal to another.
Tester.AssertImage()	● Asserts that an image matches an expected image.
Tester.AssertLess()	● Asserts that one value is less than another.
Tester.AssertLessOrEqual()	● Asserts that one value is less than or equal to another.

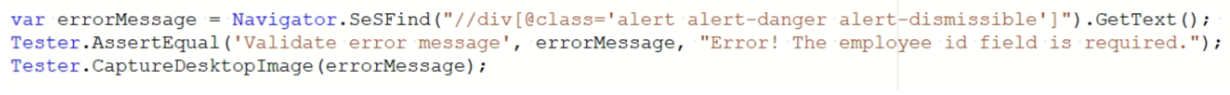
Tester.AssertNotContains()	● Asserts that a string does not contain another string.
Tester.AssertNotEqual()	● Asserts that two values are not equal.
Tester.AssertNull()	● Asserts that a value is null.
Tester.AssertNotSet()	● Asserts that a value is not set.
Tester.AssertSet()	● Asserts that a value is set.
Tester.AssertStartsWith()	● Asserts that a string starts with another string.

3. Common Validations and Appropriate Assert Methods

1. Text Verification vs Expected Output

Sample Scenario:

Validate that the error message "Error! The employee id field is required." is displayed when the user clicks the “Save” without populating the employee id text field.



```
var errorMessage = Navigator.SeSFind("//div[@class='alert alert-danger alert-dismissible']").GetText();
Tester.AssertEqual('Validate error message', errorMessage, "Error! The employee id field is required.");
Tester.CaptureDesktopImage(errorMessage);
```

var errorMessage = Navigator.SeSFind("//div[@class='alert alert-danger alert-dismissible']").GetText();

This line of code finds the error message displayed on the web page. It uses the Navigator.SeSFind method to locate an HTML element with the specified class attribute ('alert alert-danger alert-dismissible') which typically represents an error message. Then, it retrieves the text content of that element using the GetText() method and assigns it to the variable ‘errorMessage’.

Tester.AssertEqual('Validate error message', errorMessage, "Error! The employee id field is required.");

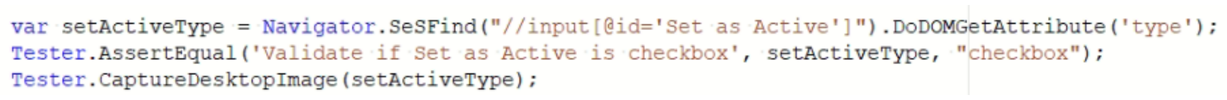
This line performs an assertion to validate whether the actual error message (errorMessage) matches the expected error message ("Error! The employee id field is required."). If the actual message matches the expected message, the assertion passes. Otherwise, it fails.

2. Validation of Form Controls or Input Controls Types.

Sample Scenario:

Verify if the following fields and buttons shall display when adding new Citizenship records.

• Citizenship Name - Textfield
• Set as Active - Checkbox
• Save Citizenship - Button



```
var setActiveType = Navigator.SeSFind("//input[@id='Set as Active']").DoDOMGetAttribute('type');
Tester.AssertEqual('Validate if Set as Active is checkbox', setActiveType, "checkbox");
Tester.CaptureDesktopImage(setActiveType);
```

var setActiveType = Navigator.SeSFind("//input[@id='Set as Active']").DoDOMGetAttribute('type');

This line of code locates an HTML input element with the ID "Set as Active" using the XPath selector //input[@id='Set as Active'].

Once the element is found, the method DoDOMGetAttribute('type') is invoked to retrieve its type attribute's value. This value is assigned to the variable setActiveType.

Tester.AssertEqual('Validate if Set as Active is checkbox', setActiveType, "checkbox");

This line of code performs an assertion to validate whether the value of setActiveType is equal to "checkbox".
If the value of setActiveType matches "checkbox", the assertion passes. Otherwise, it fails.

Note: Ensure that the XPath includes an attribute named "type".

3. **Validate if Value is in a Dropdown or Picklist Field**

Sample Scenario:

Validate if the created prefix("Mr.") reflects in the prefix dropdown in 201 File.

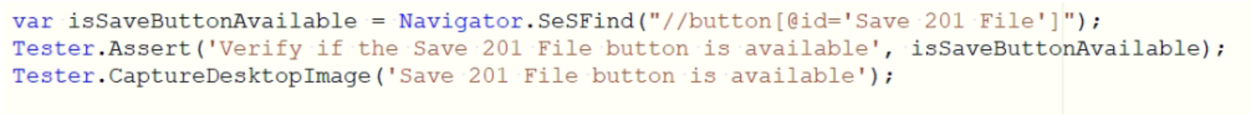
```
function validateOptionInDropdown(dropdownXpath, textToFind) {
  /*
  Created By: Alwin 02.05.24

  dropdownXpath must be the select tagname not the option tag (e.g path =
'//select[@name="name_prefix_id"]');
  textToFind is the text that you are looking in the dropdown (e.g 'Mr.')

  Example usage:
  var dropdownXPath = '//select[@name="name_prefix_id"]';
  validateOptionInDropdown(dropdownXPath, 'Mr.');
  */

  // Initialize a variable to store the text of the option
  Global.textInOption;

  // Get the count of options within the dropdown
  var optionCount = Navigator.SeSFind(dropdownXpath).DoDOMChildrenCount();
  Tester.Message("Option Count: " + optionCount);

  /// Initialize a variable to check if the option is found
        var optionFound = false;

        // Iterate through each option in the dropdown
        for (var i = 0; i < optionCount; i++) {
          // Build the XPath for the current option
          var xpath = dropdownXpath + "/option[" + (i + 1) + "]";
```

```
            // Get the text of the current option
            Global.textInOption = Navigator.SeSFind(xpath).GetText();

            // Check if the text matches the desired textToFind
            if (Global.textInOption === textToFind) {
                Tester.Message(textToFind + " is in the dropdown.");
                optionFound = true; // Set the flag to true if the option is found
                Navigator.SeSFind(dropdownXpath).DoSelect(textToFind);
                Tester.CaptureDesktopImage(textToFind);
                break; // Exit the loop if the option is found
            }
        }

        // Check if the option is not found
        if (!optionFound) {
            Tester.Message(textToFind + " is not in the dropdown.");
            Tester.FailTest(textToFind + " is not in the dropdown.");
        }

}
```

4. **Validation of Form Controls or Input Controls Availability**

Sample Scenario:

Validate if the “Save 201 File” button is available on the Edit 201 File page.



```
var isSaveButtonAvailable = Navigator.SeSFind("//button[@id='Save 201 File']");
Tester.Assert('Verify if the Save 201 File button is available', isSaveButtonAvailable);
Tester.CaptureDesktopImage('Save 201 File button is available');
```

var isSaveButtonAvailable

This line declares a variable named isSaveButtonAvailable to store the result of the check for the existence of the "Save 201 File" button.

N*avigator.CheckObjectExists("//button[@id='Save 201 File']")*

This part checks if an HTML button element with the ID "Save 201 File" exists in the current web page. The CheckObjectExists function of the Navigator object is used for this purpose. If the button exists, this function returns true; otherwise, it returns false.

Tester.Assert('Verify if the Save 201 File button is available, isSaveButtonAvailable)

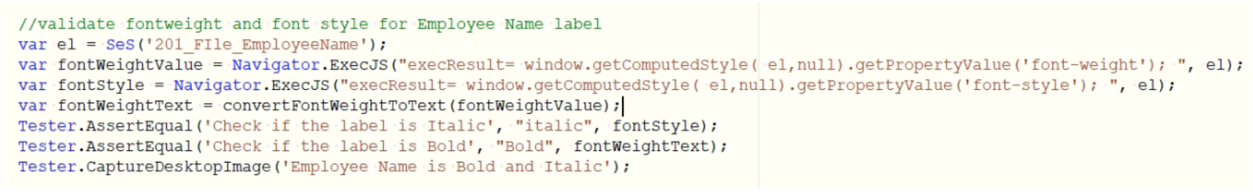
This line asserts whether the "Save 201 File" button is available based on the value stored in isSaveButtonAvailable. If isSaveButtonAvailable is true, the assertion passes, indicating that the button is available. If it's false, the assertion fails, indicating that the button is not available.

5. Validations of Font Weight and Font Style

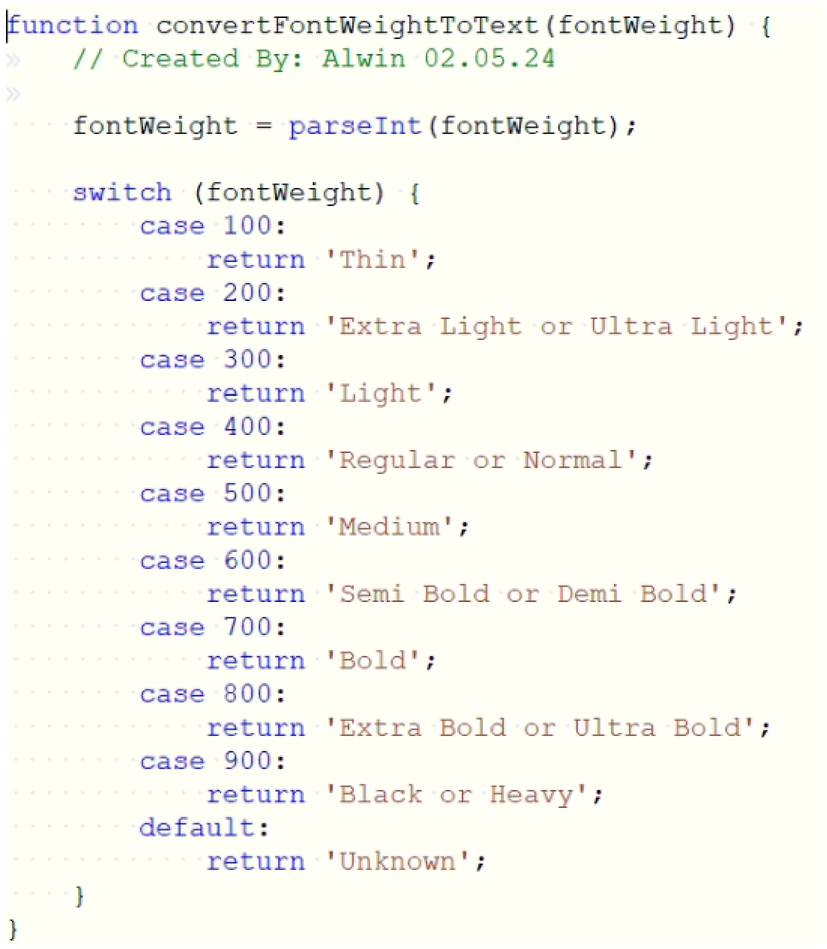
Sample Scenario:

Validate if the following labels in Personal Information:

1. Employee Name: - this shall be the group label name in Bold, and Italicized formats
2. Other Information: - this shall be the group label name in Bold, and Italicized formats
3. Current Address: - this shall be the group label name in Bold and italicized formats



```
//validate fontweight and font style for Employee Name label
var el = SeS('201_FIle_EmployeeName');
var fontWeightValue = Navigator.ExecJS("execResult= window.getComputedStyle( el,null).getPropertyValue('font-weight'); ", el);
var fontStyle = Navigator.ExecJS("execResult= window.getComputedStyle( el,null).getPropertyValue('font-style'); ", el);
var fontWeightText = convertFontWeightToText(fontWeightValue);
Tester.AssertEqual('Check if the label is Italic', "italic", fontStyle);
Tester.AssertEqual('Check if the label is Bold', "Bold", fontWeightText);
Tester.CaptureDesktopImage('Employee Name is Bold and Italic');
```



```
function convertFontWeightToText(fontWeight) {
    // Created By: Alwin 02.05.24

    fontWeight = parseInt(fontWeight);

    switch (fontWeight) {
        case 100:
            return 'Thin';
        case 200:
            return 'Extra Light or Ultra Light';
        case 300:
            return 'Light';
        case 400:
            return 'Regular or Normal';
        case 500:
            return 'Medium';
        case 600:
            return 'Semi Bold or Demi Bold';
        case 700:
            return 'Bold';
        case 800:
            return 'Extra Bold or Ultra Bold';
        case 900:
            return 'Black or Heavy';
        default:
            return 'Unknown';
    }
}
```

var el = SeS('201_FIle_EmployeeName');

Assign the object '201_FIle_EmployeeName' to variable el.

var fontWeightValue = Navigator.ExecJS("execResult= window.getComputedStyle(el,null).getPropertyValue('font-weight'); ", el);

Here, JavaScript is executed within the browser context to retrieve the computed value of the font-weight CSS property for the selected element (el). The value is stored in the variable fontWeightValue.

var fontStyle = Navigator.ExecJS("execResult= window.getComputedStyle(el,null).getPropertyValue('font-style'); ", el);

Similar to the previous line, this JavaScript code retrieves the computed value of the font-style CSS property for the selected element (el). The value is stored in the variable fontStyle.

var fontWeightText = convertFontWeightToText(fontWeightValue);

This line calls a custom function convertFontWeightToText and passes the fontWeightValue as an argument. The purpose of this function is to convert the numeric font weight value into a descriptive text format (e.g., "bold", "normal") for easier interpretation. The result is stored in the variable fontWeightText.

Tester.AssertEqual('Check if the label is Italic', "italic", fontStyle);

This assertion compares the value of the font-style property retrieved in fontStyle with the expected value "italic". It checks if the font style of the selected element (el) is italic. If the assertion fails, it indicates that the label is not italic.

Tester.AssertEqual('Check if the label is Bold', "Bold", fontWeightText);

Similarly, this assertion compares the value of the font-weight property, converted to text format, with the expected value "Bold". It checks if the font weight of the selected element (el) is bold. If the assertion fails, it indicates that the label is not bold.

6.