**OrangeHRM Automation Framework - Data Retrieval Documentation**

**Overview**

This documentation covers the data retrieval process in the OrangeHRM automation framework. The framework uses a systematic approach to capture and validate data entered into forms by retrieving the displayed values and comparing them against the originally generated test data using Bogus library.

**Architecture Overview**

The data retrieval process consists of three main components:

* **Page Object Methods**: Functions that extract data from UI elements
* **Data Models**: Classes that structure the retrieved data
* **Test Assertions**: Methods that compare retrieved data with generated data

**Creating Retrieval Functions in Page Objects**

**Basic Structure**

Each page object should contain a dedicated retrieval method that returns a model containing all the relevant data from the page.

csharp

public class EmployeePersonalDetailsPage : BasePage

{

*// Constructor and locators...*

public EmployeePersonalDetailsModel GetEmployeePersonalDetails()

{

WaitForPageToLoad();

return new EmployeePersonalDetailsModel

{

FirstName = GetTextValue(firstNameInput),

LastName = GetTextValue(lastNameInput),

EmployeeId = GetTextValue(employeeIdInput),

DateOfBirth = GetTextValue(dateOfBirthInput),

Gender = GetSelectedDropdownText(genderDropdown),

MaritalStatus = GetSelectedDropdownText(maritalStatusDropdown),

Nationality = GetSelectedDropdownText(nationalityDropdown)

};

}

private string GetTextValue(IWebElement element)

{

try

{

return element.GetAttribute("value") ?? string.Empty;

}

catch (Exception ex)

{

Console.WriteLine($"Error retrieving text value: {ex.Message}");

return string.Empty;

}

}

private string GetSelectedDropdownText(IWebElement dropdown)

{

try

{

var selectElement = new SelectElement(dropdown);

return selectElement.SelectedOption.Text;

}

catch (Exception ex)

{

Console.WriteLine($"Error retrieving dropdown value: {ex.Message}");

return string.Empty;

}

}

}

**Advanced Retrieval Patterns**

**Handling Dynamic Lists**

csharp

public List<DependentModel> GetDependentsList()

{

var dependents = new List<DependentModel>();

var dependentRows = driver.FindElements(By.CssSelector(".dependent-row"));

foreach (var row in dependentRows)

{

dependents.Add(new DependentModel

{

Name = row.FindElement(By.CssSelector(".dependent-name")).GetAttribute("value"),

Relationship = GetDropdownText(row.FindElement(By.CssSelector(".relationship-dropdown"))),

DateOfBirth = row.FindElement(By.CssSelector(".dob-input")).GetAttribute("value")

});

}

return dependents;

}

**Handling Checkboxes and Radio Buttons**

csharp

public bool GetCheckboxState(IWebElement checkbox)

{

return checkbox.Selected;

}

public string GetSelectedRadioButtonValue(string radioGroupName)

{

var selectedRadio = driver.FindElement(By.CssSelector($"input[name='{radioGroupName}']:checked"));

return selectedRadio.GetAttribute("value");

}

**Data Retrieval Process**

**Step 1: Wait for Page Stability**

Always ensure the page is fully loaded before attempting data retrieval:

csharp

private void WaitForPageToLoad()

{

var wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));

wait.Until(driver => ((IJavaScriptExecutor)driver)

.ExecuteScript("return document.readyState").Equals("complete"));

}

**Step 2: Extract Data Using Appropriate Methods**

Choose the correct extraction method based on the element type:

* **Input Fields**: Use GetAttribute("value")
* **Dropdowns**: Use SelectElement.SelectedOption.Text
* **Text Areas**: Use GetAttribute("value") or Text property
* **Checkboxes/Radio**: Use Selected property
* **Display Text**: Use Text property

**Step 3: Handle Null and Empty Values**

Always implement null checking and default values:

csharp

public string SafeGetText(IWebElement element)

{

try

{

return element?.GetAttribute("value")?.Trim() ?? string.Empty;

}

catch (StaleElementReferenceException)

{

*// Re-find the element and try again*

RefreshElement(element);

return element?.GetAttribute("value")?.Trim() ?? string.Empty;

}

}

**Creating Data Models**

**Basic Model Structure**

csharp

public class EmployeePersonalDetailsModel

{

public string FirstName { get; set; } = string.Empty;

public string LastName { get; set; } = string.Empty;

public string EmployeeId { get; set; } = string.Empty;

public DateTime? DateOfBirth { get; set; }

public string Gender { get; set; } = string.Empty;

public string MaritalStatus { get; set; } = string.Empty;

public string Nationality { get; set; } = string.Empty;

*// Optional: Override Equals and GetHashCode for better comparison*

public override bool Equals(object obj)

{

if (obj is EmployeePersonalDetailsModel other)

{

return FirstName == other.FirstName &&

LastName == other.LastName &&

EmployeeId == other.EmployeeId &&

DateOfBirth == other.DateOfBirth &&

Gender == other.Gender &&

MaritalStatus == other.MaritalStatus &&

Nationality == other.Nationality;

}

return false;

}

}

**Complex Models with Nested Objects**

csharp

public class EmployeeCompleteModel

{

public EmployeePersonalDetailsModel PersonalDetails { get; set; }

public List<DependentModel> Dependents { get; set; } = new List<DependentModel>();

public EmployeeContactModel ContactDetails { get; set; }

public EmployeeJobModel JobDetails { get; set; }

}

**Creating Tests with Retrieval Functions**

**Basic Test Structure**

csharp

[Test]

public void VerifyEmployeePersonalDetailsDataRetrieval()

{

*// Arrange*

var faker = new Faker();

var testData = new EmployeePersonalDetailsModel

{

FirstName = faker.Name.FirstName(),

LastName = faker.Name.LastName(),

EmployeeId = faker.Random.Number(1000, 9999).ToString(),

DateOfBirth = faker.Date.Past(30, DateTime.Today.AddYears(-18)),

Gender = faker.PickRandom("Male", "Female"),

MaritalStatus = faker.PickRandom("Single", "Married", "Divorced"),

Nationality = faker.PickRandom("American", "Canadian", "British")

};

*// Act*

var personalDetailsPage = new EmployeePersonalDetailsPage(driver);

personalDetailsPage.FillPersonalDetails(testData);

personalDetailsPage.SaveDetails();

*// Retrieve the data*

var retrievedData = personalDetailsPage.GetEmployeePersonalDetails();

*// Assert*

Assert.AreEqual(testData.FirstName, retrievedData.FirstName, "First name mismatch");

Assert.AreEqual(testData.LastName, retrievedData.LastName, "Last name mismatch");

Assert.AreEqual(testData.EmployeeId, retrievedData.EmployeeId, "Employee ID mismatch");

Assert.AreEqual(testData.Gender, retrievedData.Gender, "Gender mismatch");

Assert.AreEqual(testData.MaritalStatus, retrievedData.MaritalStatus, "Marital status mismatch");

Assert.AreEqual(testData.Nationality, retrievedData.Nationality, "Nationality mismatch");

}

**Advanced Test with Custom Assertions**

csharp

[Test]

public void VerifyCompleteEmployeeDataWithCustomAssertions()

{

*// Arrange*

var testEmployee = GenerateTestEmployee();

*// Act*

var employeePage = new EmployeePage(driver);

employeePage.CreateEmployee(testEmployee);

var retrievedEmployee = employeePage.GetCompleteEmployeeDetails();

*// Assert using custom assertion methods*

AssertEmployeePersonalDetails(testEmployee.PersonalDetails, retrievedEmployee.PersonalDetails);

AssertEmployeeDependents(testEmployee.Dependents, retrievedEmployee.Dependents);

AssertEmployeeContactDetails(testEmployee.ContactDetails, retrievedEmployee.ContactDetails);

}

private void AssertEmployeePersonalDetails(EmployeePersonalDetailsModel expected, EmployeePersonalDetailsModel actual)

{

Assert.Multiple(() =>

{

Assert.AreEqual(expected.FirstName, actual.FirstName);

Assert.AreEqual(expected.LastName, actual.LastName);

Assert.AreEqual(expected.EmployeeId, actual.EmployeeId);

Assert.AreEqual(expected.DateOfBirth?.Date, actual.DateOfBirth?.Date);

Assert.AreEqual(expected.Gender, actual.Gender);

});

}

**Parameterized Tests for Multiple Scenarios**

csharp

[Test, TestCaseSource(nameof(GetEmployeeTestData))]

public void VerifyEmployeeDataRetrieval\_MultipleScenarios(EmployeePersonalDetailsModel testData)

{

*// Act*

var personalDetailsPage = new EmployeePersonalDetailsPage(driver);

personalDetailsPage.FillPersonalDetails(testData);

var retrievedData = personalDetailsPage.GetEmployeePersonalDetails();

*// Assert*

Assert.That(retrievedData, Is.EqualTo(testData).Using(new EmployeePersonalDetailsComparer()));

}

private static IEnumerable<EmployeePersonalDetailsModel> GetEmployeeTestData()

{

var faker = new Faker();

for (int i = 0; i < 5; i++)

{

yield return new EmployeePersonalDetailsModel

{

FirstName = faker.Name.FirstName(),

LastName = faker.Name.LastName(),

EmployeeId = faker.Random.Number(1000, 9999).ToString(),

Gender = faker.PickRandom("Male", "Female"),

MaritalStatus = faker.PickRandom("Single", "Married")

};

}

}

**Troubleshooting Common Issues**

**Issue 1: Stale Element Reference**

**Problem**: Elements become stale after page refreshes or DOM updates.

**Solution**:

csharp

private IWebElement RefreshElement(By locator)

{

try

{

return driver.FindElement(locator);

}

catch (NoSuchElementException)

{

Thread.Sleep(1000); *// Wait briefly and retry*

return driver.FindElement(locator);

}

}

public string GetTextWithRetry(By locator, int maxRetries = 3)

{

for (int i = 0; i < maxRetries; i++)

{

try

{

var element = driver.FindElement(locator);

return element.GetAttribute("value") ?? string.Empty;

}

catch (StaleElementReferenceException)

{

if (i == maxRetries - 1) throw;

Thread.Sleep(500);

}

}

return string.Empty;

}

**Issue 2: Timing Issues with Dynamic Content**

**Problem**: Attempting to retrieve data before elements are fully loaded.

**Solution**:

csharp

public string WaitAndGetText(By locator, int timeoutSeconds = 10)

{

var wait = new WebDriverWait(driver, TimeSpan.FromSeconds(timeoutSeconds));

var element = wait.Until(ExpectedConditions.ElementToBeClickable(locator));

return element.GetAttribute("value") ?? string.Empty;

}

**Issue 3: Dropdown Values Not Retrieving Correctly**

**Problem**: Selected dropdown values returning incorrect text.

**Solution**:

csharp

public string GetDropdownSelectedValue(IWebElement dropdown)

{

try

{

*// Try SelectElement first (for standard select elements)*

var selectElement = new SelectElement(dropdown);

return selectElement.SelectedOption.Text;

}

catch (UnexpectedTagNameException)

{

*// Handle custom dropdowns*

return dropdown.FindElement(By.CssSelector(".selected-option, .current-selection"))?.Text ?? string.Empty;

}

}

**Issue 4: Data Type Conversion Issues**

**Problem**: String to DateTime or other type conversions failing.

**Solution**:

csharp

public DateTime? SafeParseDate(string dateString)

{

if (string.IsNullOrWhiteSpace(dateString))

return null;

*// Try multiple date formats*

string[] formats = { "MM/dd/yyyy", "dd/MM/yyyy", "yyyy-MM-dd", "dd-MM-yyyy" };

foreach (string format in formats)

{

if (DateTime.TryParseExact(dateString, format, CultureInfo.InvariantCulture, DateTimeStyles.None, out DateTime result))

{

return result;

}

}

*// Fallback to general parsing*

if (DateTime.TryParse(dateString, out DateTime fallbackResult))

{

return fallbackResult;

}

return null;

}

**Issue 5: Empty or Null Values in Retrieved Data**

**Problem**: Retrieved data contains unexpected empty values.

**Solution**:

csharp

public void ValidateRetrievedData(object model)

{

var properties = model.GetType().GetProperties();

var emptyProperties = new List<string>();

foreach (var prop in properties)

{

var value = prop.GetValue(model);

if (value == null || (value is string str && string.IsNullOrWhiteSpace(str)))

{

emptyProperties.Add(prop.Name);

}

}

if (emptyProperties.Any())

{

Console.WriteLine($"Warning: Empty properties found: {string.Join(", ", emptyProperties)}");

*// Take screenshot for debugging*

TakeScreenshot($"EmptyData\_{DateTime.Now:yyyyMMdd\_HHmmss}");

}

}

**Issue 6: Performance Issues with Large Data Sets**

**Problem**: Slow retrieval when dealing with large tables or lists.

**Solution**:

csharp

public List<T> GetDataWithPagination<T>(Func<IWebElement, T> rowMapper, By rowsLocator, By nextButtonLocator)

{

var allData = new List<T>();

bool hasNextPage = true;

while (hasNextPage)

{

var rows = driver.FindElements(rowsLocator);

allData.AddRange(rows.Select(rowMapper));

try

{

var nextButton = driver.FindElement(nextButtonLocator);

if (nextButton.Enabled)

{

nextButton.Click();

WaitForPageToLoad();

}

else

{

hasNextPage = false;

}

}

catch (NoSuchElementException)

{

hasNextPage = false;

}

}

return allData;

}

**Best Practices**

1. **Always implement wait strategies** before data retrieval
2. **Use try-catch blocks** for robust error handling
3. **Implement retry mechanisms** for flaky elements
4. **Validate retrieved data** before assertions
5. **Use meaningful variable names** and comments
6. **Create reusable utility methods** for common operations
7. **Log retrieval operations** for debugging purposes
8. **Handle different data types** appropriately
9. **Implement custom comparers** for complex objects
10. **Take screenshots** on retrieval failures for debugging

**Debugging Tips**

1. **Add logging** to retrieval methods to track what values are being extracted
2. **Use browser developer tools** to verify element properties and values
3. **Implement debug modes** that pause execution for manual verification
4. **Create helper methods** that dump all retrieved data to console
5. **Use breakpoints** strategically in retrieval methods
6. **Verify element visibility** and state before retrieval attempts

This documentation provides a comprehensive guide to implementing and troubleshooting the data retrieval process in your OrangeHRM automation framework. The examples and solutions should help you build robust and reliable data retrieval mechanisms.