Unit Testing Guide

Prepared By: Architecture Team
Revision 1.0

Table of Contents

Introduction

UiPathCoreFrameworkv1.1 helps developer to write their own Unit Test to check their workflows. The unit test for each block of UiPathCoreFrameworkv1.1 has been already created and can be run by running the *RunUnitTest.xaml* file.

Quick Guide for Running Pre-Created Unit Test workflows:

1. Install custom activity package (*"ActivityAssertUnitTest.1.x.x.nupkg"*) in the UiPath Studio. Steps for the same are here.
2. Run *RunUnitTest.xaml*.

ActivityAssertUnitTest.1.x.x.nupkg:

ActivityAssertUnitTest.1.x.x.nupkg package is the core part of testing workflows. It contains custom **Assert Unit Test** activity that must be used when making **Unit Tests.**

Assert Unit Test activity is used as a container for writing **Assert methods** (imported from *ActivityAssertUnitTest* namespace) which are used to evaluate unit test outputs.

Following methods are available in the *ActivityAssertUnitTest* namespace.

- Assert.AreEqual(expected,actual)
- Assert.AreNotEqual(expected,actual)
- Assert.Contains(expected,actual)
- Assert.NotContains(expected,actual)
- Assert.IsNull(actual)
- Assert.IsNotNull(actual)

The description of the above methods is present here.

Assert methods throw custom exceptions that are used in the **UiPathCoreFramework** to log test results and give important information in case of a test failure.

- AssertException is main exception thrown by Assert methods.
- AssertNullException is exception thrown when Assert.IsNull or Assert.IsNotNull method returns false.
- CustomAssertException is generic exception that is thrown when random boolean expression that returns false is passed to Assert Unit Test activity

Creating Custom Unit Tests:

To create Unit Tests the developer should use *1_UnitTest_WorkflowName.xaml* template workflow which is present in ***Tests_Repository\UnitTestTemplate.***

The template has been divided into three parts:

- Arrange
- Act
- Assert

Arrange Stage

Purpose of the arrange stage is to define input parameters for the test:

- Manually define parameters and data
- (If needed) Define expected result
- (If needed) Read configuration file

Act Stage

Purpose of act stage is to run the code that is being tested (by invoking the code and supplying it previously arranged data).

Output of this stage should be actual value that code produces assigned to the actual variable.

Assert Stage

Purpose of the assert stage is to evaluate test outputs (usually this is done by comparing actual value gotten by running the code with the **expected value** that was defined in the arrange stage, but is not the only type of test that can be done).

For UiPathCoreFrameworkv1.1 to consider a Unit Test as valid, it must implement Assert Unit Test activity. This activity is meant to be used along with, one of the many, Assert methods from *ActivityAssertUnitTest* namespace.

Assert methods make writing of Unit Tests much easier and exceptions thrown by these methods serve a big role in Framework logging.

Assert Methods

1. **Assert.AreEqual()**

   This method compares the actual and the expected value to check if the unit test has passed or not. Returns true if actual and expected results are not equal. The method is overloaded to support different types and number of arguments.

   a) *public static bool AreEqual(string expected, string actual):* Takes two string arguments i.e. actual and expected Result. Returns true if both the strings are equal. It uses Ordinal Culture to compare the strings.
   b) *public static bool AreEqual(string expected, string actual, string exceptionMessage) :* Takes result arguments as well as custom exception message from the user. Returns true if both the strings are equal or else throws *AssertException* with the user provided exception message.
   c) *public static bool AreEqual(string expected, string actual, bool ignoreCase):* Compares two string using OrdinalIgnoreCase culture if the third argument is true.

d) *public static bool AreEqual(string expected, string actual, bool ignoreCase, string exceptionMessage):* Combination of the above two methods.
e) *public static bool AreEqual(string expected, string actual, StringComparison culture):* Takes in result arguments as well as culture which should be use for comparison. Returns true if actual and expected are equal.
f) *public static bool AreEqual (string expected, string actual, StringComparison culture, string exceptionMessage):* Takes in result arguments, culture and exception message from the user. Performs string comparison using the user defined culture and returns true if both actual and expected arguments are equal or else throws *AssertException* if are unequal with the custom exception message.
g) *public static bool AreEqual(double expected, double actual, int precision):*Takes in double numbers and the precision as arguments and makes comparison of the two numbers i.e. actual and the expected. Returns true if both the double numbers are equal to the given precision.
h) *public static bool AreEqual(double expected, double actual, int precision, string exceptionMessage):* Same as above method. It also takes in custom Exception Message. Returns true if actual is equal to expected.
i) *public static bool AreEqual(double expected, double actual, double delta)*: Takes in double numbers as arguments and delta value to compare the distance between two numbers. Returns True if calculated distance is less than or equal to the given delta value.
j) *public static bool AreEqual(double expected, double actual, double delta, string exceptionMessage):* Same as above method. Takes in an extra argument i.e. custom exception message from the user.
k) *public static bool AreEqual(Decimal expected, Decimal actual, int precision):* Takes in decimal numbers and the precision as arguments and makes comparison of the two numbers i.e. actual and the expected. Returns true if both the decimal numbers are equal to the given precision.
l) *public static bool AreEqual(Decimal expected, Decimal actual, int precision, string exceptionMessage):* Same as above method. It also takes in custom Exception Message. Returns true if actual is equal to expected.

2. <u>**Assert.AreNotEqual()**</u>

This method compares the actual and the expected value to check if the unit test has passed or not. Returns true if actual and expected results are not equal. The method is overloaded to support different types and number of arguments.

a) *public static bool AreNotEqual(string expected, string actual):* Takes two string arguments i.e. actual and expected Result. Returns true if both the strings are not equal. It uses Ordinal Culture to compare the strings.
b) *public static bool AreNotEqual(string expected, string actual, string exceptionMessage) :* Takes result arguments as well as custom exception message from the user. Returns true if both the strings are not equal or else throws *AssertException* with the user provided exception message.

c) *public static bool AreNotEqual(string expected, string actual, bool ignoreCase):* Compares two string using OrdinalIgnoreCase culture if the third argument is true.
d) *public static bool AreNotEqual(string expected, string actual, bool ignoreCase, string exceptionMessage):* Combination of the above two methods.
e) *public static bool AreNotEqual(string expected, string actual, StringComparison culture):* Takes in result arguments as well as culture which should be use for comparison. Returns true if actual and expected are not equal.
f) *public static bool AreNotEqual (string expected, string actual, StringComparison culture, string exceptionMessage):* Takes in result arguments, culture and exception message from the user. Performs string comparison using the user defined culture and returns true if both actual and expected arguments are unequal or else throws *AssertException* if are equal with the custom exception message.
g) *public static bool AreNotEqual(double expected, double actual, int precision):*Takes in double numbers and the precision as arguments and makes comparison of the two numbers i.e. actual and the expected. Returns true if both the double numbers are unequal to the given precision.
h) *public static bool AreNotEqual(double expected, double actual, int precision, string exceptionMessage):* Same as above method. It also takes in custom Exception Message. Returns true if actual is not equal to expected.
i) *public static bool AreNotEqual(double expected, double actual, double delta)*: Takes in double numbers as arguments and delta value to compare the distance between two numbers. Returns True if calculated distance is more than to the given delta value.
j) *public static bool AreNotEqual(double expected, double actual, double delta, string exceptionMessage):* Same as above method. Takes in an extra argument i.e. custom exception message from the user.
k) *public static bool AreNotEqual(Decimal expected, Decimal actual, int precision):* Takes in decimal numbers and the precision as arguments and makes comparison of the two numbers i.e. actual and the expected. Returns true if both the decimal numbers are not equal to the given precision.
l) *public static bool AreNotEqual(Decimal expected, Decimal actual, int precision, string exceptionMessage):* Same as above method. It also takes in custom Exception Message. Returns true if actual is not equal to expected.

3. Assert. Contains()

This method checks if the supplied substring is present in the base string. It will return True if Substring is present in the base string otherwise throw *AssertException*.

a) *public static bool Contains(string ExpectedSubstring, string baseString):*Takes in two string arguments i.e. substring and the base string. Returns true if substring is present in the base string.
b) *public static bool Contains(string ExpectedSubstring, string baseString, string exceptionMessage):* Same as above method. Takes an extra argument of custom exception message. If the substring is not present in the base string, throws *AssertException* with this Custom Exception Message.

c) *public static bool Contains(string ExpectedSubstring, string baseString, StringComparison culture):* Takes in substring , base string and the culture as input. The culture is used to make the comparison. Returns true if substring is present in the base string or else throws *AssertException.*
d) *public static bool Contains(string ExpectedSubstring, string baseString, StringComparison culture, string exceptionMessage):* Combination of the above two methods.

4. **Assert.NotContains()**

This method checks if the supplied substring is present in the base string. It will return True if Substring is not present in the base string otherwise throw *AssertException*.

e) *public static bool NotContains(string notExpectedSubstring, string baseString):*Takes in two string arguments i.e. substring and the base string. Returns true if substring is not presented in the base string.
f) *public static bool NotContains(string notExpectedSubstring, string baseString, string exceptionMessage):* Same as above method. Takes an extra argument of custom exception message. If the substring is present in the base string, throws *AssertException* with this Custom Exception Message.
g) *public static bool NotContains(string notExpectedSubstring, string baseString, StringComparison culture):* Takes in substring , base string and the culture as input. The culture is used to make the comparison. Returns true if substring is not present in the base string or else throws *AssertException.*
h) *public static bool NotContains(string notExpectedSubstring, string baseString, StringComparison culture, string exceptionMessage):* Combination of the above two methods.

5. **Assert.IsNull()**

This method is use to check if the input argument is null or not. Returns true if the input argument is null otherwise throws *AssertNullException.*

a) *public static bool IsNull(object value):* Takes in the variable which is to be checked. Returns true if the variable is null.
b) *public static bool IsNull(object value, string exceptionMessage):*Same as above method. Takes in an extra argument of custom exception message. If the variable is not null throws *AssertNullException* with this custom exception message.

6. **Assert.IsNotNull()**

This method is use to check if the input argument is null or not. Returns true if the input argument is not null otherwise throws *AssertNullException.*

c) *public static bool IsNotNull(object value):* Takes in the variable which is to be checked. Returns true if the variable is not null.

d) *public static bool IsNotNull(object value, string exceptionMessage):*Same as above method. Takes in an extra argument of custom exception message. If the variable is null throws *AssertNullException* with this custom exception message.