Using Assert Classes and Methods in Unit Tests

If you have been following my blog posts on unit testing, you have used the Assert class to signify if a unit test is successful or not. The following are my previous posts on unit testing. If you are not familiar with unit testing, go back and read these posts.

- Introduction to Unit Testing in Visual Studio
- Avoid Hard-Coding in Unit Tests
- Unit Test Initialization and Cleanup
- Add Attributes to Unit Tests

You have used the Inconclusive, IsTrue, IsFalse and Fail methods. In this blog post you will learn about some of the other methods you can utilize in the Assert class. In addition you will learn about two additional assert classes you may take advantage of when writing unit tests.

Assert Class

There are many methods in the Assert class. I won't explain each one, but I will expose you to some so you can get an idea of what is available to use. You should search the MSDN documentation to see the complete list of properties and methods available to you in the Assert class.

Common Parameters to Assert Methods

Most of the methods you may invoke on the Assert class include an overload that allows you to specify a message to display in the test results. An additional overload lets you specify the message using the standard string.Format() tokens and a parameter array of the values to use to replace into the message.

```
[TestMethod]
public void FileNameDoesExistSimpleMessage() {
  FileProcess fp = new FileProcess();
  bool fromCall;

  fromCall = fp.FileExists(_GoodFileName);

  Assert.IsTrue(fromCall, "File Does Not Exist.");
}
```

When you run the above test, your Test Explorer window will show the hard-coded message after you click on the failed test (Figure 1).

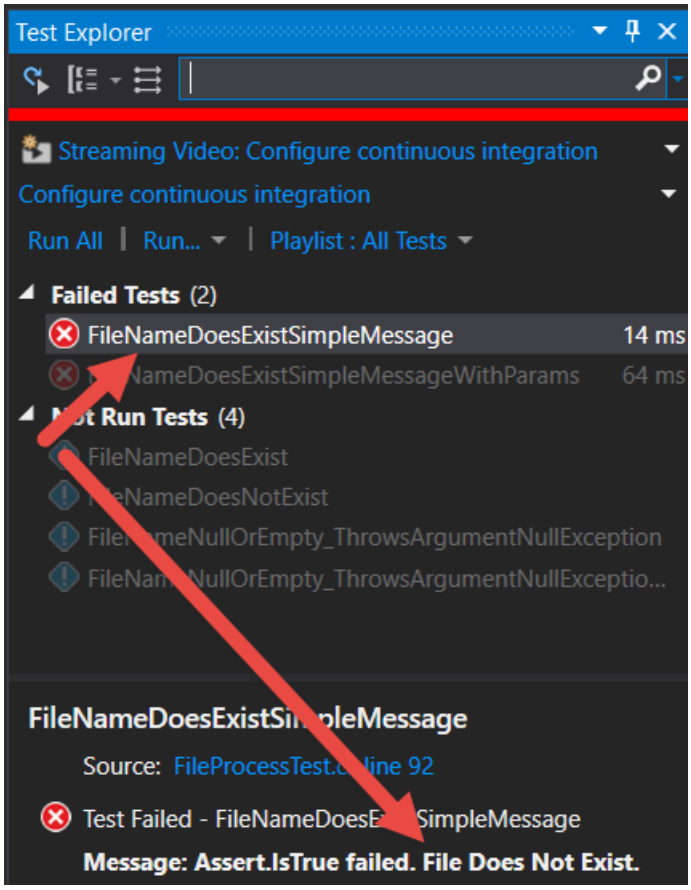

Figure 1: Display hard-coded messages into the results window.

To include some of the data you gathered during the test, use the format items just as you use in the string.Format() method.

```
[TestMethod]
public void FileNameDoesExistSimpleMessageWithParams() {
  FileProcess fp = new FileProcess();
  bool fromCall;

  fromCall = fp.FileExists(_GoodFileName);

  Assert.IsTrue(fromCall,
    "File {0} Does Not Exist.",
      _GoodFileName);
}
```

When you run this test, you get the message shown in Figure 2.

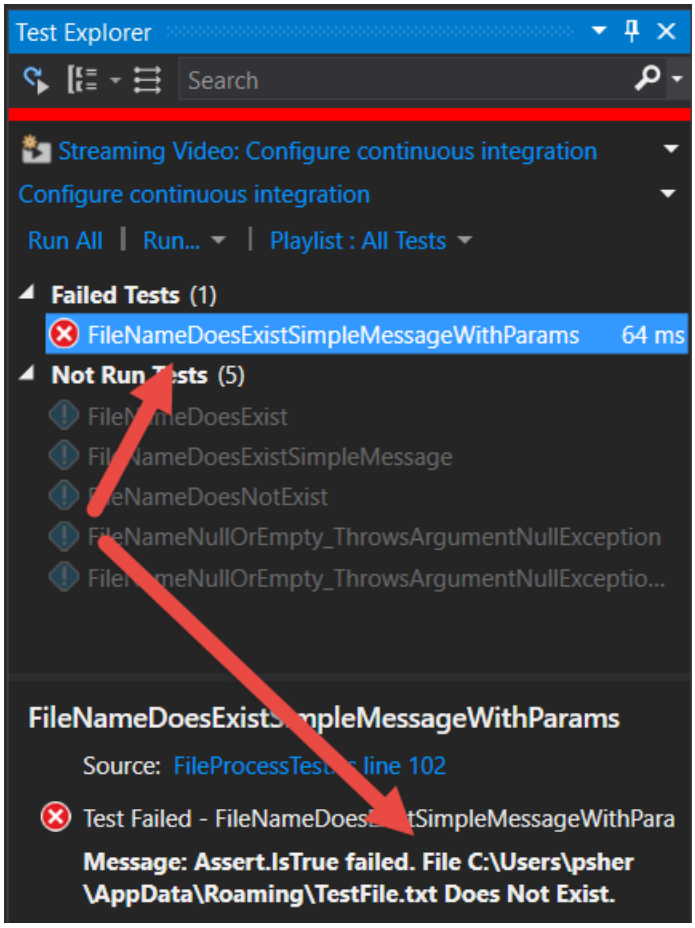

Figure 2: Display messages with data from the unit test itself.

AreEqual Method

The AreEqual method compares two variables of the same data type to determine if they are equivalent. There are overloads for each of the different data types such as double, single, int, etc. You may add your own custom

message, and specify format items as explained earlier. Below are just a few of the data types you can compare.

```
Assert.AreEqual(int, int);
Assert.AreEqual(bool, bool);
Assert.AreEqual(double, double);
```

Here is an example using integer data types.

```
[TestMethod]
public void AreEqualTest() {
  int x = 1;
  int y = 1;

  Assert.AreEqual(x, y);
}
```

When the two variables you wish to compare are of a string data type, you may also specify a CultureInfo object to handle string comparisons based on the language of the user. You may also specify a boolean value to perform a case-sensitive or case-insensitive comparison of the strings.

```
AreEqual(string, string)               // case-insensitive
AreEqual(string, string, true)         // case-sensitive
AreEqual(string, string, CultureInfo)  // Use a culture
```

AreNotEqual Method

To compare two values to see if they are not equal you use the AreNotEqual method. This method, like the AreEqual method, has several overloads you can use based on the different data types. Below is a sample of using the AreNotEqual method.

```
[TestMethod]
public void AreNotEqualTest() {
  int x = 1;
  int y = 2;

  Assert.AreNotEqual(x, y);
}
```

AreSame Method

To compare two objects to see if they are the same object. For example, if you write the following test:

```
[TestMethod]
public void AreSameTest() {
  FileProcess x = new FileProcess();
  FileProcess y = new FileProcess();

  Assert.AreSame(x, y);
}
```

This test will fail, because the two objects point to two different objects. If you change this test to look like the following, where you assign the variable x to the variable y, then this test will succeed.

```
[TestMethod]
public void AreSameTest() {
  FileProcess x = new FileProcess();
  FileProcess y = x;

  Assert.AreSame(x, y);
}
```

AreNotSame Method

To compare two objects to see if they are NOT the same object. In this case, the following test would succeed.

```
[TestMethod]
public void AreNotSameTest() {
  FileProcess x = new FileProcess();
  FileProcess y = new FileProcess();

  Assert.AreNotSame(x, y);
}
```

IsInstanceOfType Method

You can use this method to determine if an object returned from a method is of a certain type. For example, you may have a method that returns an interface or a base class. When you call this method from your unit test, you might wish to compare the type of instance that is returned against the type you are expecting.

Look at the class diagram in Figure 3 in which you have a Person class with two properties. Both the Employee and the Supervisor classes inherit from the Person class. The PersonManager class has a method named CreatePerson that returns a Person object. Depending on the parameters you

pass to the CreatePerson method, determines the type of object passed back. Either an Employee or a Supervisor object is returned.

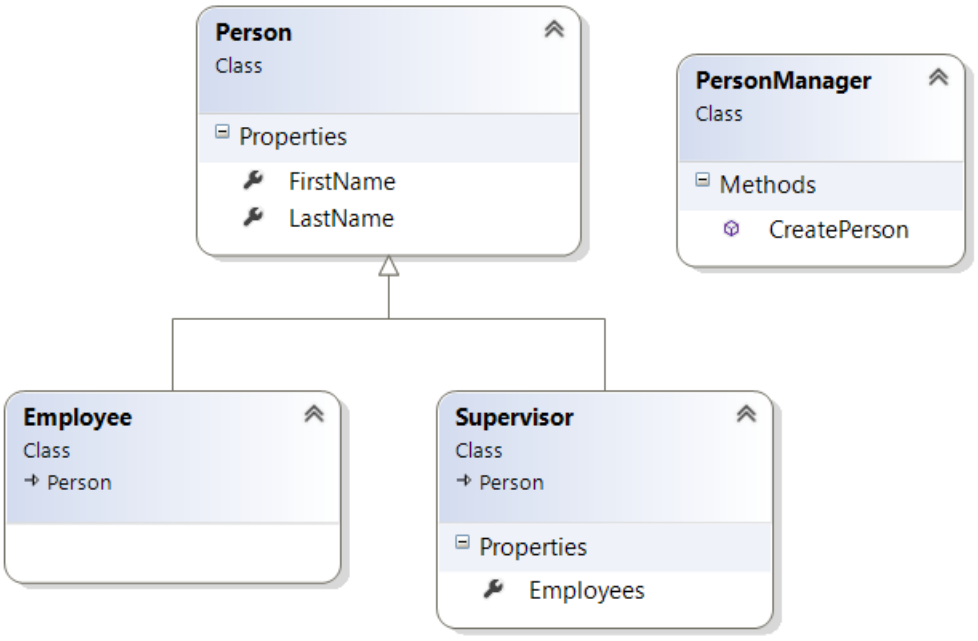

Figure 3: A class diagram for our example

You can write a unit test to determine what the type is. In the unit test below, you pass a true value to the CreatePerson method. This value tells CreatePerson to return a Supervisor object. You use the IsInstanceOfType method to compare the variable per against the typeof(Supervisor).

```
[TestMethod]
public void IsInstanceOfTypeTest() {
  PersonManager mgr = new PersonManager();
  Person per;

  per = mgr.CreatePerson("Paul", "Sheriff", true);

  Assert.IsInstanceOfType(per, typeof(Supervisor));
}
```

IsNotInstanceOfType Method

This method is the same as the IsInstanceOfType, but checks to see if the instance returned is not of a specific type.

IsNull Method

Call this method to compare the return result from a method against null. If the return result is a null, then the test succeeds. In the unit test below, if you pass an empty string as the first name to the CreatePerson method, that method returns a null object.

```
[TestMethod]
public void IsNullTest() {
  PersonManager mgr = new PersonManager();
  Person per;

  per = mgr.CreatePerson("", "Sheriff", true);

  Assert.IsNull(per);
}
```

IsNotNull Method

This method is the same as the IsNull method, but checks to see if the value returned is not null.

StringAssert

When working with strings, you commonly need to check to see if one string is contained within another, or if one string matches a regular expression, or a string starts or ends with a specific character or other string value. To test these in a unit test, use the StringAssert class with any of the following methods.

- Contains
- DoesNotContain
- Matches
- DoesNotMatch
- StartsWith
- EndsWith

CollectionAssert

Many methods you write in your applications deal with collections of data. This data could be retrieved from a database, an XML file, or simply arrays of objects you create in your code. The CollectAssert class allows you to test a known set of data in a collection against the collection returned from the method you are testing. You do not need to do any looping through the collections, the CollectionAssert class will do all of that for you. Below is a list of the various methods you can use.

- AllItemsAreInstancesOfType
- AllItemsAreNotNull
- AllItemsAreUnique
- AreEqual
- AreNotEqual
- AreEquivalent
- AreNotEquivalent
- Contains
- DoesNotContain
- IsSubsetOf
- IsNotSubsetOf

Summary

In this post, you learned more about the different methods in the Assert class. You also learned about two additional classes to help you test strings and collections. All methods in the various Assert classes contain overloads to allow you to add your own custom message to display in the test results. With this many methods available to you, writing your unit tests should go quickly.

Sample Code

You can download the code for this sample at www.pdsa.com/downloads. Choose the category “PDSA Blogs”, then locate the sample **Using Assert Classes and Methods in Unit Tests**.