Avoid Hard-Coding in Unit Tests

In my previous blog post, I introduced you to creating unit tests with Visual Studio. A method named FileExists was created to which you pass a file name to see if it exists. In the tests you created, you use hard-coded file names to test. Just as you wouldn't hard-code values in a normal application, you should not do this with unit tests either. In this blog post you will learn to use constants, a configuration file, and how to create and delete test files.

Please go read the previous blog post and create the project, or download the project at http://www.pdsa.com/downloads and select "**Introduction to Unit Testing**" from the list.

Use a Constant

Constants are a great way to centralize hard-coded data that would otherwise be repeated throughout an application. In this case, you are going to replace the hard-coded file name used in the FileNameDoesNotExist method with a constant. At the top of the FileProcessTest class, add the following constant.

```
private const string BAD_FILE_NAME = @"C:\NotExists.bad";
```

Modify the FileNameDoesNotExist to use this new constant as shown in the code snippet below.

```
public void FileNameDoesNotExist() {
  FileProcess fp = new FileProcess();
  bool fromCall;

  fromCall = fp.FileExists(BAD_FILE_NAME);

  Assert.IsFalse(fromCall);
}
```

Use a Configuration File

A constant is a good option for the “bad” file name. For the “good” file name you wish to test to see exists, let’s add that to a configuration file so it can be modified easily. In fact, let’s add a replaceable token called [AppPath] that will figure out the appropriate path to use based on the machine the test is running upon.

Right mouse click on the FileProcessTest project and select **Add** | **New Item…** from the menu. From the template dialog select **General** | **Application Configuration File**. The name should already be set to App.config, so click on the Add button.

Within the <configuration> element add an <appSettings> section. Within the <appSettings> section add a key called GoodFileName with a value of [AppPath]\TestFile.text as shown below.

```
<appSettings>
  <add key="GoodFileName" value="[AppPath]\TestFile.txt"/>
</appSettings>
```

In order to retrieve this value from the configuration file, you need to use the ConfigurationManager class from the System.Configuration namespace. By default, the System.Configuration DLL is not added to a test project. Right mouse click on References folder in your FileProcessTest project and select **Add Reference** from the menu. From the dialog select **Assemblies** | **Framework**. Locate the **System.Configuration** dll and select the check box. Click the OK button to add this DLL to your test project.

At the top of the FileProcessTest class, add a using statement for the System.Configuration namespace.

```
using System.Configuration;
```

Add private field to FileProcessTest class to hold the value you are going to retrieve from the configuration file. Set the name of this field to _GoodFileName as shown below.

```
private string _GoodFileName;
```

Add a constructor to the FileProcessTest class in which you will read the GoodFileName value from the configuration file. Within this constructor you will replace the token [AppPath] with the value from the Environment.SpecialFolder.ApplicationData. This enumeration, supplied by .NET, when passed to the GetFolderPath, returns the pre-defined path for any data you need to store for this application. The location may vary from OS to OS, but on Windows 10 it is C:\\Users\\YOUR_USERNAME\\AppData\\Roaming. Write the constructor as shown below.

```
public FileProcessTest() {
  _GoodFileName =
     ConfigurationManager.AppSettings["GoodFileName"];
  if (_GoodFileName.Contains("[AppPath]")) {
     _GoodFileName = _GoodFileName.Replace("[AppPath]",
        Environment.GetFolderPath(
          Environment.SpecialFolder.ApplicationData));
  }
}
```

Locate and modify the FileNameDoesExist method to use this new property name.

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

  fromCall = fp.FileExists(_GoodFileName);
  Assert.AreEqual(true, fromCall);
}
```

Create / Delete File

Instead of you having to create the file name in the location specified, prior to running the FileNameDoesExist test, you should create the file name within the method itself. You should delete the file after you have performed the FileExists call so you don't keep files around you don't need. Modify the FileNameDoesExist method to look like the following.

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

  if (!string.IsNullOrEmpty(_GoodFileName)) {
    // Create the 'Good' file.
    File.AppendAllText(_GoodFileName, "Some Text");
  }

  fromCall = fp.FileExists(_GoodFileName);

  // Delete file
  if (File.Exists(_GoodFileName)) {
    File.Delete(_GoodFileName);
  }

  Assert.IsTrue(fromCall);
}
```

TestContext

When the unit test framework creates an instance of a test class (a class marked with the [TestClass] attribute), the test framework creates a TestContext object. This object contains properties and methods related to testing. To access this TestContext object, you must create a public property named TestContext in each of your test classes. The test framework checks for this property and inserts the instance of this TestContext into your property.

```
private TestContext _TestInstance;
public TestContext TestContext
{
    get { return _TestInstance; }
    set { _TestInstance = value; }
}
```

One of the things you can do with this TestContext is to use the WriteLine method to add some output into the test results. Add the lines shown in bold below to write some messages about what file you are creating into the output area of the test results.

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

  if (!string.IsNullOrEmpty(_GoodFileName)) {
    TestContext.WriteLine("Creating file: " + _GoodFileName);
    // Create the 'Good' file.
    File.AppendAllText(_GoodFileName, "Some Text");
  }

  TestContext.WriteLine("Checking file: " + _GoodFileName);
  fromCall = fp.FileExists(_GoodFileName);

  // Delete file
  if (File.Exists(_GoodFileName)) {
    TestContext.WriteLine("Deleting file: " + _GoodFileName);
    File.Delete(_GoodFileName);
  }

  Assert.IsTrue(fromCall);
}
```

Run this test, once it is complete, click on the FileNameDoesExist test in the Test Explorer window, locate the **Output** link at the bottom of the window and

click on it. You should then see your messages appear in an output window as shown in Figure 1.

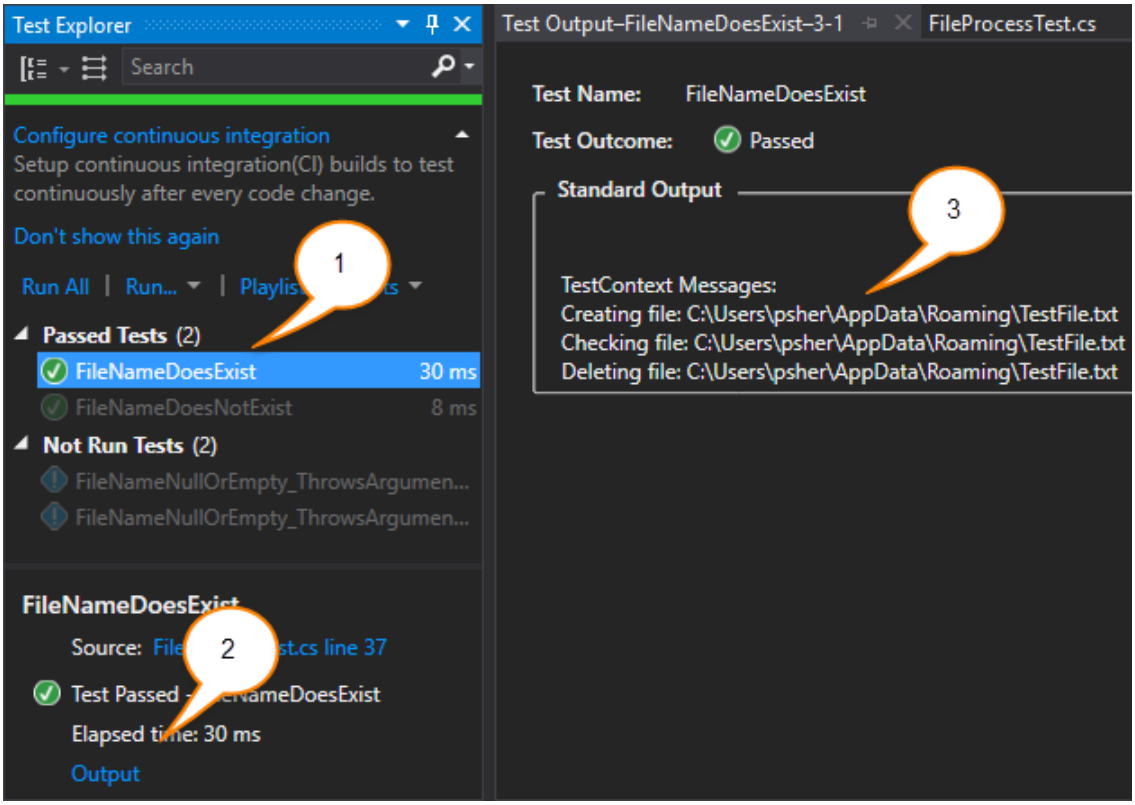

Figure 1: Write output messages using the TestContext property

Summary

In this blog post a constant was used in place of a hard-coded file name. You placed another hard-coded file name into the App.config file in the test project. Within a test method you created a file, tested that file's existence, then deleted the file. This helps keep that method self-contained and avoids manual setup of files prior to running these tests. Finally, you added a TestContext property to access the WriteLine method of the TestContext property. This allows you to add additional messages into the output of the test results.

Sample Code

You can download the code for this sample at www.pdsa.com/downloads. Choose the category “PDSA Blogs”, then locate the sample **Avoid Hard-Coding in Unit Tests**.