Unit Test Initialization and Cleanup

This blog post continues our look at unit testing techniques. See my previous two blog posts for the sample used for testing.

- Introduction to Unit Testing
- Avoid Hard-Coding in Unit Tests

In this blog post you learn about initialization and cleanup of the test environment. There are different methods of initialization and cleanup available to developers in Visual Studio. This blog post will introduce you to each and describe how to use each one.

Overview

There are six attributes you can use to perform initialization and cleanup. Which ones you use, depends on when you need to initialize something and/or clean something up. You can initialize/cleanup once for all classes within a unit test assembly. You can initialize/cleanup once for all test methods within a test class. You can initialize/cleanup before and after each test method runs within a class.

- AssemblyInitialize
- AssemblyCleanup
- ClassInitialize
- ClassCleanup
- TestInitialize
- TestCleanup

Add one or more of these attributes to a single method within your classes. You can see an example of how these attributes might be applied to a set of classes in your assembly displayed in Figure 1. The order in which these methods run is as follows.

1. AssemblyInitialize

2. ClassInitialize
3. TestInitialize
4. TestMethod, TestMethod, TestMethod, etc.
5. TestCleanup
6. ClassCleanup
7. AssemblyCleanup

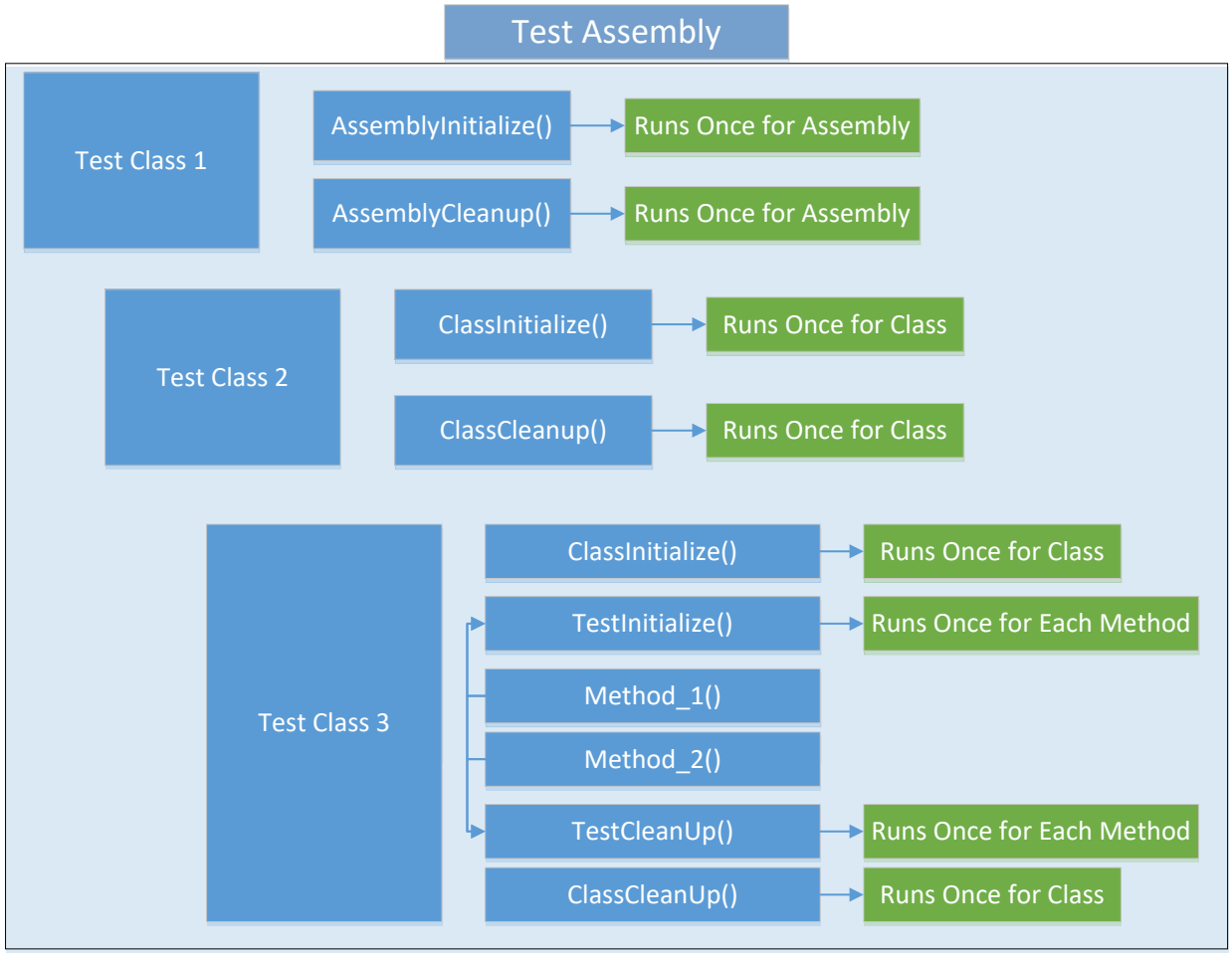

Figure 1: Overview of how test initialization and cleanup methods execute

Assembly Initialization and Clean Up

If you have several classes within a specific assembly and you need to create a database with some test records in some tables prior to running all, or the

majority, of tests within these classes, create that database within a method decorated with the [AssemblyInitialize] attribute.

I think it is a good idea to create a separate class that is only used for assembly initialization and cleanup. Create a new class with the name of **MyClassesTestInitialization** within your test project. Add the following method to this class.

```
[AssemblyInitialize()]
public static void AssemblyInitialize(TestContext tc) {
  // TODO: Initialize for all tests within an assembly
  tc.WriteLine("In AssemblyInitialize");
}
```

Only one method within your entire assembly may be decorated with the [AssemblyInitialize] attribute. This method must be static and the test framework will pass in an instance of the TestContext object.

If you wish to delete any files or a database after all methods have run within an assembly, place the code to perform these operations within a method decorated with the [AssemblyCleanup] attribute. Only one method within your entire assembly may be decorated with the [AssemblyCleanup] attribute. This method must be declared as static.

```
[AssemblyCleanup()]
public static void AssemblyCleanup() {
  // TODO: Clean up after all tests in an assembly
}
```

Class Initialization and Clean Up

Just as you create classes in your application to encapsulate a specific set of functionality, organize your unit test classes the same way. In the test program for this series of blog posts, I have a FileProcess class. There is currently only one method in that class, but in a real application, there would probably be many methods all dealing with file IO.

In the FileProcessTest class you only have unit test methods that perform testing on methods within the FileProcess class. Within the unit test class, you may need to create some files (or a database, or other data) that is needed when running all or the majority of the test methods. If that is the case, create a method within that class that is decorated with the

[ClassInitialize] attribute. Only one method within your test class may be decorated with the [ClassInitialize] attribute.

```
[ClassInitialize()]
public static void ClassInitialize(TestContext tc) {
  // TODO: Initialize for all tests within a class
  tc.WriteLine("In ClassInitialize");
}
```

If you create a file (or a database, or other data), you might want to delete that file, or other data after all of the tests have run in that class. Add a method to perform this cleanup within a method decorated with the [ClassCleanup] attribute. Only one method within your class may be decorated with the [ClassCleanup] attribute.

```
[ClassCleanup()]
public static void ClassCleanup() {
  // TODO: Clean up after all tests within this class
}
```

Test Initialization and Clean Up

The method decorated with the [ClassInitialize] attribute only runs once the first time a class is created. A method decorated with the [TestInitialize] attribute will run before **each** test method within that class. If you have four methods in a test class, the TestInitialize method will run four times. As an example, if you need a specific file name to exist before all, or the majority, of tests within a class are run, create a method like the one shown below.

```
[TestInitialize()]
public void TestInitialize() {
  // Create the Test.txt file.
  File.AppendAllText("Test.txt", "Some Text");
}
```

The above code will run before each test method is executed. If you did not clean up after each test, then you would have four lines of text within the Test.txt file. To clean up after each method, write a method that has the [TestCleanup] attribute attached to it.

```
[TestCleanup()]
public void TestCleanup() {
  // Delete Text.txt file
  if (File.Exists("Test.txt")) {
    File.Delete("Test.txt");
  }
}
```

Since this method runs after each unit test method, each new test method that runs will have a file available to it that only has a single line of text within it.

Check for Test Name

Within the TestInitialize method you can check the TestName property of the TestContext object to see if a specific test is being run. If it is, you could do a specific initialization just for that test. In the following sample, you check to see if the current test being run is “FileNameDoesExist” and if it does, you create the file contained within the _GoodFileName property. Of course, you are already doing this in the FileNameDoesExist method already, so you do not need to do it in the TestInitialize method, but it does show an example of what you could do.

```
[TestInitialize]
public void TestInitialize() {
  TestContext.WriteLine("In TestInitialize");

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

Summary

In this blog post you learned about initialization and clean up for unit tests. You have three ways to perform initialization and clean up in the unit test framework. You just need to decide where in the process you need to perform

the initialization. Then decide how much clean up you need, and write the appropriate methods, and decorate with the appropriate attributes.

Sample Code

You can download the code for this sample at www.pdsa.com/downloads. Choose the category “PDSA Blogs”, then locate the sample **Unit Test Initialization and Cleanup**.