Testing using xUnit in C#

Overview

In this tutorial, we will create a xUnit Test class with multiple Test Cases, implemented in C#.

Description

This simple example is based on a sample xUnit tutorial, which provides some tests using xUnit Fact and Theory concepts.

There are different tests created using the different facilities provided by xUnit, including parameterized tests.

Start by creating a working project and add the necessary dependencies.

```
dotnet new classlib
dotnet add package xunit
dotnet add package XunitXml.TestLogger
dotnet test --test-adapter-path:. --logger:xunit
```

Your project should have a configuration similar to the following one.
csharp-xunit-calc.csproj

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <TargetFramework>netcoreapp2.1</TargetFramework>
    <IsPackable>false</IsPackable>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.NET.Test.Sdk" Version="15.9.0" />
    <PackageReference Include="xunit" Version="2.4.1" />
    <PackageReference Include="xunit.runner.visualstudio" Version="2.4.1" />
    <PackageReference Include="XunitXml.TestLogger" Version="2.1.26" />
  </ItemGroup>

</Project>
```

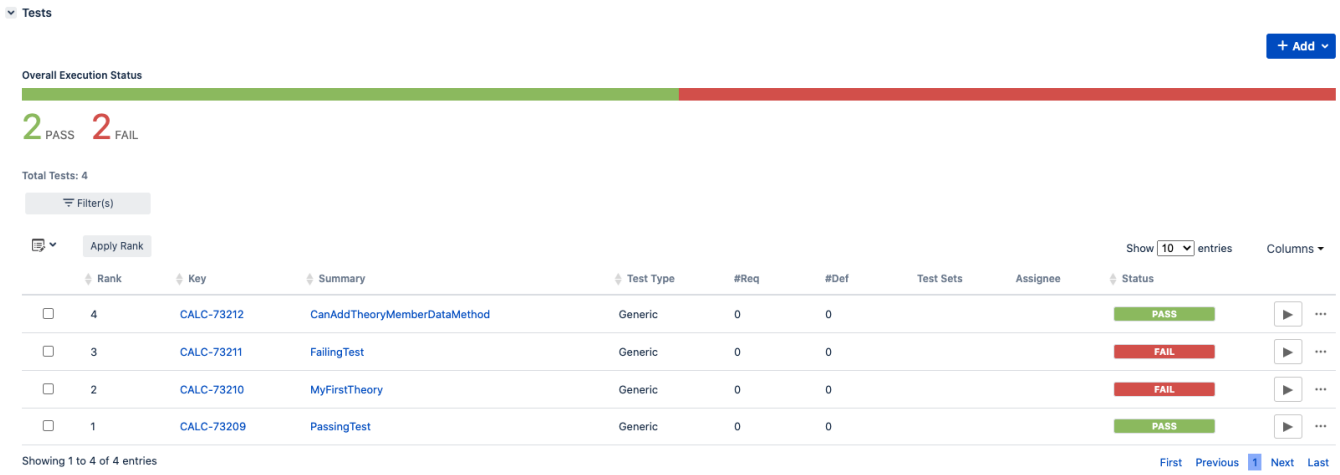
Write your test methods in a class, using Fact and/or Theory.
test/CalcTests.cs

```
using System;
using Xunit;
using System.Collections.Generic;

namespace csharp_xunit_calc
{

    public class CalcTests
    {
        [Fact]
        [Trait("labels","core UI")]
        public void PassingTest()
        {
            Assert.Equal(4, Add(2, 2));
        }

        [Fact]
        [Trait("requirement","CALC-1")]
        public void FailingTest()
        {
            Assert.Equal(5, Add(2, 2));
        }

        int Add(int x, int y)
        {
            return x + y;
        }

        [Theory]
        [InlineData(3)]
        [InlineData(5)]
        [InlineData(6)]
        public void MyFirstTheory(int value)
        {
            Assert.True(IsOdd(value));
        }

        bool IsOdd(int value)
        {
            return value % 2 == 1;
        }


        [Theory]
        [MemberData(nameof(GetData))]
        public void CanAddTheoryMemberDataMethod(int value1, int value2, int expected)
        {
            //var calculator = new Calculator();
            //var result = calculator.Add(value1, value2);
            var result = value1 + value2;
            Assert.Equal(expected, result);
        }

        public static IEnumerable<object[]> GetData()
        {
            var allData = new List<object[]>
            {
                new object[] { 1, 2, 3 },
                new object[] { -4, -6, -10 },
                new object[] { -2, 2, 0 },
                new object[] { int.MinValue, -1, int.MaxValue },
            };

            return allData;
        }

    }


}
```

You can then run your tests using the xUnit logger which will produce a XML report inside the subfolder "TestResults".

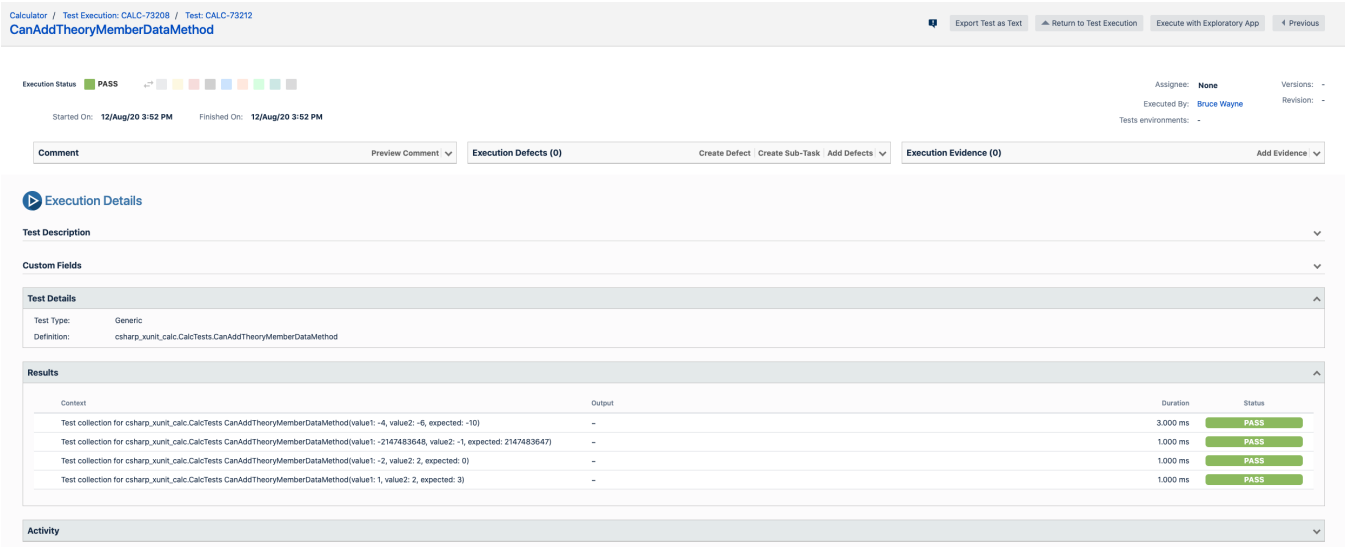
```
dotnet test --test-adapter-path:. --logger:xunit
```

After successfully running the Test Case and generating the xUnit XML report (e.g., TestResults.xml), it can be imported to Xray (by using either the REST API or the **Import Execution Results** action within the Test Execution).



Each xUnit's test is mapped to a Generic Test in Jira, and the **Generic Test Definition** field contains the namespace, the name of the class, and the method name that implements the Test case.

The Execution Details of the Generic Test contains information about the context, which in this case corresponds to the Test case method, along with the different input values that were validated.



References

- https://xunit.github.io/docs/getting-started/netcore/cmdline
- http://ikeptwalking.com/writing-data-driven-tests-using-xunit/
- https://andrewlock.net/creating-parameterised-tests-in-xunit-with-inlinedata-classdata-and-memberdata/