T4 TEMPLATE

What is T4 template?

A Text Template Transformation Toolkit (T4) template is a general-purpose template engine. using T4 we can generate C#, VB code, XML, HTML, or text of any kind. The code generation is used in Visual Studio in technologies like MVC, Entity Framework, LINQ to SQL, and many others that use those templates. The Entity Framework supports extension or manipulation of the code generation using the T4 template.

Visual Studio T4 Text Template can be used to generate test methods in this situation. T4 Text Templates are [WYSISYG](https://en.wikipedia.org/wiki/WYSIWYG) text mixed with C# or Visual Basic control blocks that get executed at compile time. They are used to generate text files programmatically. Since the templates simply generate a text file, they could be used to generate tests or scripts for pretty much any test framework, but they integrate best with writing and running tests in the programming languages supported by Visual Studio.

What is the goal of this research on T4 template?

The goal of this research is to create a plugin using the concept of T4 template which will generate Mock classes for Unit Test Cases.

What is purpose of this research?

The plugin that we will create using this research will automatically create mock classes for unit test cases with just a click which will save developer time in writing unit test cases.

Advantage of this research?

1. After developing plugin using this research developer no longer will have to put extra effort it writing mocking classes in unit test cases since developer will able to generate the former in UT cases with just one click using plugin.
2. In premise no. of test cases created are limited but in real time using this approach we can create n no. of unit test cases.

Drawback of this research?

The premise used in this research contain hard coded values but in future while developing we need to make sure to make code generic so that it works in all cases

Our premise for the research -

Considering two enumerations

public enum Colour

{

Red,

Orange,

Yellow,

Green,

Blue,

Violet

}

public enum Shape

{

Circle,

Square

}

Now we will use c# control blocks to iterate over the Colour and Shape enumerations to generate C# test methods that test drawing all combinations of the coloured shapes

<#@ template debug="false" hostspecific="false" language="C#" #>

<#@ assembly name="System.Core" #>

<#@ assembly name="$(TargetPath)" #>

<#@ assembly name="$(SolutionDir)packages\\Drawing\\Drawing.dll" #>

<#@ import namespace="System.Linq" #>

<#@ import namespace="System.Text" #>

<#@ import namespace="System.Collections.Generic" #>

<#@ import namespace="Drawing" #>

<#@ output extension=".Generated.cs" #>

using Drawing;

using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace DrawingTests

{

[TestClass]

public partial class ColouredShapesTests

{<#

foreach(var colour in System.Enum.GetNames(typeof(Colour)))

{

foreach(var shape in System.Enum.GetNames(typeof(Shape)))

{#>

[TestMethod]

public void Draw<#= string.Format("{0}{1}", colour, shape) #>()

{

DrawingTest(<#= string.Format("Colour.{0}", colour) #>, <#=

string.Format("Shape.{0}", shape) #>);

}

<#

}

}#>

}

}

The <# control blocks contain C# statements for flow control and the <#= control blocks contain C# statements that output strings that will be written to the output file. This template will generate a file called ColouredShapesTests.Generated.cs that contains a partial class with all of the test methods.

using Drawing;

using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace DrawingTests

{

[TestClass]

public partial class ColouredShapesTests

{

[TestMethod]

public void DrawRedCircle()

{

DrawingTest(Colour.Red, Shape.Circle);

}

[TestMethod]

public void DrawRedSquare()

{

DrawingTest(Colour.Red, Shape.Square);

}

[TestMethod]

public void DrawOrangeCircle()

{

DrawingTest(Colour.Orange, Shape.Circle);

}

[TestMethod]

public void DrawOrangeSquare()

{

DrawingTest(Colour.Orange, Shape.Square);

}

[TestMethod]

public void DrawYellowCircle()

{

DrawingTest(Colour.Yellow, Shape.Circle);

}

[TestMethod]

public void DrawYellowSquare()

{

DrawingTest(Colour.Yellow, Shape.Square);

}

[TestMethod]

public void DrawGreenCircle()

{

DrawingTest(Colour.Green, Shape.Circle);

}

[TestMethod]

public void DrawGreenSquare()

{

DrawingTest(Colour.Green, Shape.Square);

}

[TestMethod]

public void DrawBlueCircle()

{

DrawingTest(Colour.Blue, Shape.Circle);

}

[TestMethod]

public void DrawBlueSquare()

{

DrawingTest(Colour.Blue, Shape.Square);

}

[TestMethod]

public void DrawVioletCircle()

{

DrawingTest(Colour.Violet, Shape.Circle);

}

[TestMethod]

public void DrawVioletSquare()

{

DrawingTest(Colour.Violet, Shape.Square);

}

}

}

Now we can see here how using T4 template we are able to generate unit test cases using similar premise we would be able to generate mock classes with in unit test cases.