# NUnit Testing

Now that we have our tmx file being loaded by the xml serializer it is now time to test the data that was loaded in. For the testing we will be using Unity’s built in testing feature, which uses NUnit.Framework. For more information please see <http://docs.unity3d.com/Manual/testing-editortestsrunner.html>

From Unity answers <http://answers.unity3d.com/questions/1191898/which-version-of-nunit-ships-with-unity-cant-find.html>

"The Editor Tests Runner is an implementation of the open source NUnit library - a well-known unit testing library for .Net languages. More information about NUnit can be found on [http://www.nunit.org/.](http://www.nunit.org/) The implementaion is based on **version 2.6.4**."

We will be using the TMXTest script for all of our editor test. We need to make sure that we have the using statement for TileMapXML and NUnit Framework.

using TileMapXML;

using NUnit.Framework;

We need a variable to hold our tmx file we will also be using a const string variable to tell where our map is located. Using a const variable just prevents it from being changed in code, this keeps you from assigning a new string to the variable by mistake. If you have been following along with this tutorial series the path to your map file will be Assets/TileMapXML/Maps/NameOfYourMap.tmx If you have a different folder structure it will be Assets/DirectoryOfMapFile/NameOfYourMap.tmx or Assets/NameOfYourMap.tmx Every Asset in Unity is stored in the Assets directory, when using code to load a file you need to include the Assets folder in the path. The name of the map that I am using for this is TestRunnerMap so my file path will be Assets/TileMapXML/TestRunnerMap.tmx

const string TMX\_FILE\_PATH = @"Assets/TileMapXML/Maps/TestRunnerMap.tmx";

Our tmx variable we will make public to access it but make it so you can only assign to it in the script itself. We will be using this script latter when we get to importing our maps from outside Unity.

public TMX tmx { get; private set; }

Now we need a method to load the tmx file, this method will take a string for the path of the file to load

public void LoadTMXFile(string filePath)

{

tmx = new TMX();

tmx.Load(filePath);

}

Like the XML serialization namespace the NUnit Framework name space has special tags that we can use, we need a method that has the [SetUp] tag, this will be run before every test that is run using the test runner in unity. This ensures that we have a clean setup for each test. In this method we will call LoadTMXFile passing in the file to load which is what is stored in the const variable TMX\_File\_PATH.

[SetUp]

public void Init()

{

LoadTMXFile(TMX\_FILE\_PATH);

}

Next we will add 2 test methods that using the [Test] tag, this is how the test runner knows what test to run. The First test will make sure that the tmx variable is not null and the second test will make sure that the map is not null.

[Test]

public void TMXIsNotNull()

{

Assert.IsNotNull(tmx, "Failed to load tmx file, " + TMX\_FILE\_PATH);

}

[Test]

public void TMXMapIsNotNull()

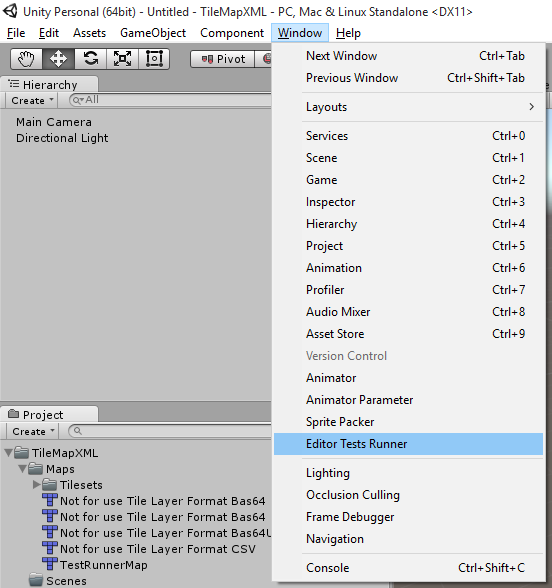
{

Assert.IsNotNull(tmx.map, "Failed to load map from " + TMX\_FILE\_PATH);

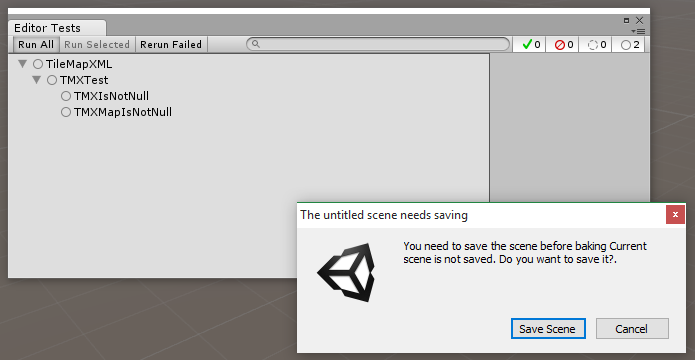
}

In TMXIsNotNull method we are asserting that the tmx variable is not null by using Assert.IsNotNull. This just states that the tmx variable should have been created. In TMXMapIsNotNull method we verify that the map did load. If it failed to load the tmx.map variable would be null.

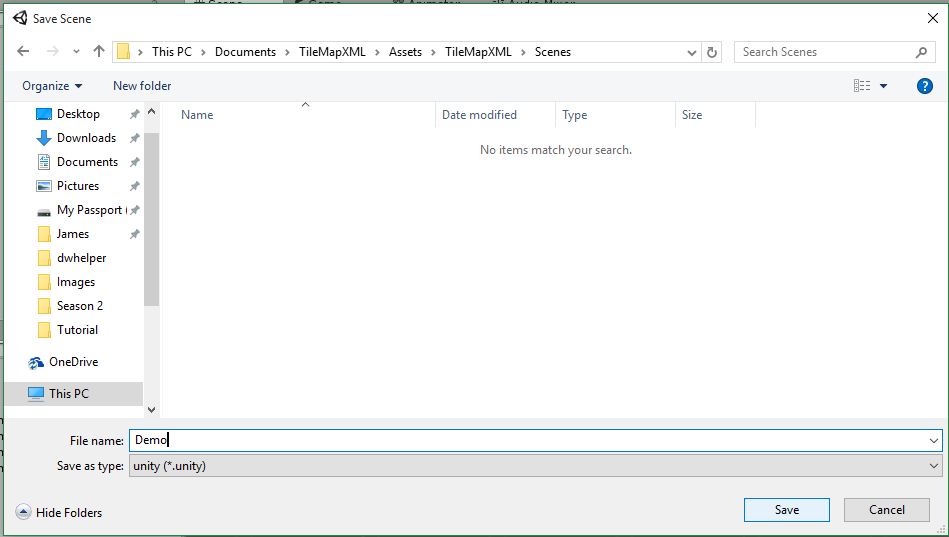
Now let’s run our test and make sure that everything passes. In Unity on the menu bar select Window->Editor Tests Runner



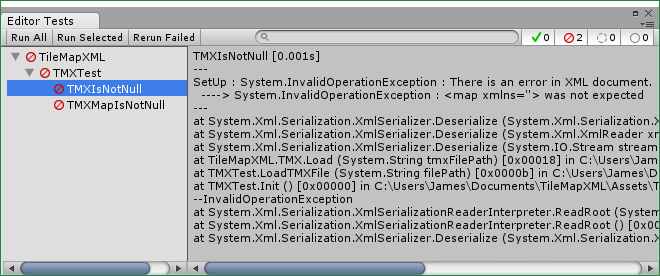
This should bring up the editor test runner window. Now if you select run all test this will run all of the editor test that we have created. You will get a dialog window telling you that your scene needs to be saved. So lets do that now. Select Save Scene.



In the save scene dialog create a new folder in TileMapXML called Scenes and save your scene as Demo in TileMapXML->Scenes.



Once the scene is saved the test will run. Both of them fail.



If you notice the test failed during the set up portion,

SetUp : System.InvalidOperationException : There is an error in XML document.

----> System.InvalidOperationException : <map xmlns=''> was not expected

This is because the root element in the xml document is <map> and the xml serializer is expecting a root element of <TMXMap>

To fix this we need to add the [XmlRoot("ElementName")] tag to our TMXMap script.

namespace TileMapXML

{

[XmlRoot("map")]

public class TMXMap

{

#region attributes

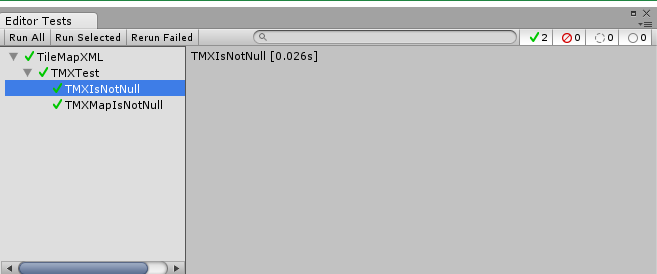
#endregion

}

}

[XmlRoot("map")] This is telling the xml serializer that the class is going to be the root element in our xml file and the name of the element is “map”. This is very important to keep in mind when loading xml files. The name of your elements need to match what the deserializer is expecting. I left the [XmlRoot] tag out on purpose so you can see the type of failures you may see in your projects. And also to show you how to read the test message.

Now if you select Run All in the Editor Test Runner Window you will see that all of the test pass.



### TMXTest.cs

using NUnit.Framework;

using TileMapXML;

public class TMXTest

{

/// <summary>

/// The path to the map file to use for testing

/// </summary>

const string TMX\_FILE\_PATH = @"Assets/TileMapXML/Maps/TestRunnerMap.tmx";

/// <summary>

/// The tmx file

/// public get private set

/// </summary>

public TMX tmx { get; private set; }

/// <summary>

/// Load the passed in file into tmx

/// </summary>

/// <param name="filePath">The path of the file to load</param>

public void LoadTMXFile(string filePath)

{

tmx = new TMX();

tmx.Load(filePath);

}//public void LoadTMXFile

/// <summary>

/// Runs before every NUnit test, this makes sure that we have a clean environment.

/// </summary>

[SetUp]

public void Init()

{

LoadTMXFile(TMX\_FILE\_PATH);

}//void Init()

/// <summary>

/// Verify that the tmx variable is not null

/// </summary>

[Test]

public void TMXIsNotNull()

{

Assert.IsNotNull(tmx, "Failed to load tmx file, " + TMX\_FILE\_PATH);

}//void TMXIsNotNull()

/// <summary>

/// Verify that the map is not null

/// </summary>

[Test]

public void TMXMapIsNotNull()

{

Assert.IsNotNull(tmx.map, "Failed to load map from " + TMX\_FILE\_PATH);

}//void TMXMapIsNotNull

}//public class TMXTest

### TMXMap.cs

namespace TileMapXML

{

/// <summary>

/// <map>

/// • version: The TMX format version, generally 1.0.

/// • orientation: Map orientation.

/// Tiled supports "orthogonal", "isometric", "staggered" (since 0.9)

/// and "hexagonal" (since 0.11).

/// • renderorder: The order in which tiles on tile layers are rendered.

/// Valid values are right-down(the default), right-up, left-down and left-up.

/// In all cases, the map is drawn row-by-row.

/// (since 0.10, but only supported for orthogonal maps at the moment)

/// • width: The map width in tiles.

/// • height: The map height in tiles.

/// • tilewidth: The width of a tile.

/// • tileheight: The height of a tile.

/// • hexsidelength: Only for hexagonal maps.

/// Determines the width or height (depending on the staggered axis)

/// of the tile's edge, in pixels.

/// • staggeraxis: For staggered and hexagonal maps,

/// determines which axis("x" or "y") is staggered. (since 0.11)

/// • staggerindex: For staggered and hexagonal maps,

/// determines whether the "even" or "odd" indexes along the staggered axis are shifted. (since 0.11)

/// • backgroundcolor: The background color of the map.

/// (since 0.9, optional, may include alpha value since 0.15 in the form #AARRGGBB)

/// • nextobjectid: Stores the next available ID for new objects.

/// This number is stored to prevent reuse of the same ID after objects have been removed. (since 0.11)

///

/// The tilewidth and tileheight properties determine the general grid size of the map.

/// The individual tiles may have different sizes.

/// Larger tiles will extend at the top and right(anchored to the bottom left).

///

/// A map contains three different kinds of layers.

/// Tile layers were once the only type, and are simply called layer,

/// object layers have the objectgroup tag and image layers use the imagelayer tag.

/// The order in which these layers appear is the order in which the layers are rendered by Tiled.

///

/// Can contain: properties, tileset, layer, objectgroup, imagelayer

/// </summary>

[XmlRoot("map")]

public class TMXMap

{

#region attributes

#endregion

}//public class TMXMap

}//namespace TileMapXML