# Tiled Map Format

If you take a look at <http://doc.mapeditor.org/reference/tmx-map-format> and look at <map> you will see

## <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](http://doc.mapeditor.org/reference/tmx-map-format/#properties), [tileset](http://doc.mapeditor.org/reference/tmx-map-format/#tileset), [layer](http://doc.mapeditor.org/reference/tmx-map-format/#layer), [objectgroup](http://doc.mapeditor.org/reference/tmx-map-format/#objectgroup), [imagelayer](http://doc.mapeditor.org/reference/tmx-map-format/#imagelayer)

## Description

Now if you create different types of maps with different settings, layers, tilesets, and properties save them and then open them in a text editor or xml editor you will see something similar to this.

<?xml version="1.0" encoding="UTF-8"?>

<map version="1.0" orientation="orthogonal" renderorder="right-down" width="5" height="5" tilewidth="32" tileheight="32" backgroundcolor="#04010203" nextobjectid="1">

<properties>

<property name="boolPropertyFalse" type="bool" value="false"/>

<property name="boolPropertyTrue" type="bool" value="true"/>

<property name="floatProperty" type="float" value="0.01"/>

<property name="floatProperty0" type="float" value="0"/>

<property name="intProperty" type="int" value="1"/>

<property name="intProperty0" type="int" value="0"/>

<property name="stringProperty" value="Hello"/>

</properties>

<tileset firstgid="1" name="outdoor" tilewidth="32" tileheight="32" tilecount="162" columns="9">

<image source="Tilesets/Liberated Pixel Cup/outdoor.png" trans="ffaa55" width="288" height="576"/>

</tileset>

<layer name="Tile Layer 1" width="5" height="5">

<data>

<tile gid="0"/>

<tile gid="0"/>

<tile gid="0"/>

<tile gid="0"/>

<tile gid="0"/>

<tile gid="0"/>

<tile gid="0"/>

<tile gid="0"/>

<tile gid="0"/>

<tile gid="0"/>

<tile gid="0"/>

<tile gid="0"/>

<tile gid="0"/>

<tile gid="0"/>

<tile gid="0"/>

<tile gid="0"/>

<tile gid="0"/>

<tile gid="0"/>

<tile gid="0"/>

<tile gid="0"/>

<tile gid="0"/>

<tile gid="0"/>

<tile gid="0"/>

<tile gid="0"/>

<tile gid="0"/>

</data>

</layer>

</map>

Remember we added the [XmlRoot("map")] tag TMXMap class, We will Now be using the [XmlAttribute()] tag. Notice that all of the things with built points on the tile map format documentation are all listed before the “>”, these are referred to as attributes in xml. And the things listed as a map can contain are all in between <map ….> and </map> these are called child elements. Elements in an xml document will look like

<ElementName AttributeName=”value” >

<ChildElementName></ChildElementName>

</ElementName>

Or

<ElementName AttributeName=”value” />(if no child elements exist)

<ElementName is knowen as the starting tag for an element and the </ElementName> or /> is knowen as the ending tag for an element. Xml is done this way so it is easy to figure out where one element starts and where it ends as well as which elements contain children and where the attributes of the elements are. When writing code in C# to get xml attributes to load in when using serialization you need to add [XmlAttribute()] tag if your variable name is the same as your attribute name or [XmlAttribute("AttributeName")] tag if the variable name is not the same.

For example for the version attribute you would use

[XmlAttribute()]

public float version;

or

[XmlAttribute("version")]

public float versionNumber;

For child elements you need to use a public variable with the same name or use the [XmlElement("ElementName")] tag. You can also use [XmlElement("layer", typeof(LayerElement))] if you want to load several different types into a list when using inheritance. I will show you an example of this when we get to loading in the different layer’s that a map can contain.

For now, let’s take a look at the different xml attributes that a map can contain and get them to load into our map. Every attribute is a string so you can load these all as string’s if you want to, but we are going to use types that are as close as possible to represent our attributes, this will make it easier on us when we go to displaying our map in Unity.

version is a number and in code you have different number data types, int, float, decimal, just to name a few. We will be using float in this tutorial. As of Tiled 0.16.1 which is the current version of Tiled as of this writing the version number will always be 1.0 unless you manually change it after saving the map.

orientation is orthogonal, isometric, staggered, or hexagonal. We will create our own enum for this.

renderorder is right-down, right-up, left-down, left-up. We will creating an enum for this using the [XmlEnum("xmlEnumName")] cannot use the – character in an enum. We will not be using this to draw our map in Unity, this is used in Tiled to control the order in which the tiles are drawn to the screen. We will be looping thru our tiles on the tile layer right down, a tile with a position of 0,0 will be the first tile or the top left of your map and position n, -n will be the last tile or the bottom right of your map. This is how Tiled saves out the tiles when you set the Tile Layer Format to xml, regardless of the render order.

width, height, tilewidth, tileHeight, and hexsidelength are all whole numbers so we will use the int type for these.

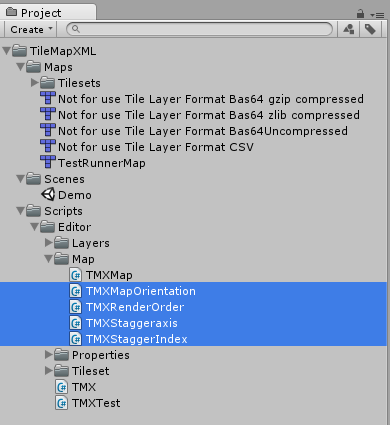
staggeraxis is either x or y and staggerindex is either even or odd so we will be making our own enum types for these.

backgroundcolor is a hex value in the form of #AARRGGBB since we are using xml serialization this will be loaded in as a string. I may show you how to convert this to a color and use this value to set the background color of our scene when drawing the map in Unity.

nextobjectid is a whole number so we will use the int type for this. Again this is really just for use in Tiled when creating new objects in Tiled, we will more than likely not be using this within unity.

## Code

In Unity Create 4 new C# scripts in TileMapXML->Scripts->Map called TMXMapOrientation, TMXRenderOrder, TMXStaggeraxis, and TMXStaggerIndex. Open these four scripts and delete all of the generated code, we are going to use these to hold our enums. You can skip this step and put all of your enums in TMXMap.cs outside of the class if you want. I just like to keep my enums in their own file so I can easily find them if I want to make a change to them.



### TMXMapOrientation.cs

namespace TileMapXML.Map

{

/// <summary>

/// orientation: Map orientation.

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

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

///

/// Tiled Map Editor Version 0.16.2

/// orthogonal

/// isometric

/// staggered-isometric staggered

/// hexagonal-hexagonal staggered

/// </summary>

[System.Serializable]

public enum TMXMapOrientation

{

none,

orthogonal,

isometric,

staggered,

hexagonal,

}//enum MapOrientation

}//namespace TileMapXML.Map

Nothing special here except we are using the [Systm.Serializable] tag and this is a public enum instead of a class. Also notice that the namespace is TileMapXML.Map. The Serializable tag is more for the internal workings of Unity and this being displayed in the inspector automatically when we use it in other classes derived from Monobehavior or ScriptableObject. Notice that the first entry is none which is not listed in the Tiled documentation. This is for us to use in error checking later on, we will also be assigning none as the default when this variable is used. This will be the same for TMXRenderOrder, TMXStaggerAxis and TMXStaggerIndex.

### TMXRenderOrder.cs

using System.Xml.Serialization;

namespace TileMapXML.Map

{

/// <summary>

/// 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)

/// </summary>

[System.Serializable]

public enum TMXRenderOrder

{

none,

[XmlEnum("right-down")]

RightDown,

[XmlEnum("right-up")]

RightUp,

[XmlEnum("left-down")]

LeftDown,

[XmlEnum("left-up")]

LeftUp,

}//enum TMXRenderOrder

}//namespace TileMapXML.Map

Notice the use of the [XmlEnum("xmlEnumName")] tag. Enums do not allow the use of the ‘-‘ character but Tiled writes the render order out as right-down, so we need the [XmlEnum("right-down ")] tag so the xml deserializer knows how to read in our data.

### TMXStaggerAxis.cs

namespace TileMapXML.Map

{

/// <summary>

/// staggeraxis: For staggered and hexagonal maps,

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

/// </summary>

[System.Serializable]

public enum TMXStaggerAxis

{

none,

x,

y

}//enum TMXMapStaggerAxis

}//namespace TileMapXML.Map

### TMXStaggerIndex.cs

namespace TileMapXML.Map

{

/// <summary>

/// staggerindex: For staggered and hexagonal maps,

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

/// </summary>

[System.Serializable]

public enum TMXStaggerIndex

{

none,

odd,

even

}//enum TMXStaggerIndex

}//namespace TileMapXML.Map

### *TMXMap.cs*

using System.Collections.Generic;

using System.Xml.Serialization;

using TileMapXML.Layers;

using TileMapXML.Map;

using TileMapXML.Properties;

using TileMapXML.Tileset;

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

/// <summary>

/// The TMX format version, generally 1.0

/// </summary>

[XmlAttribute()]

public float version = 0;

/// <summary>

/// Map orientation.

/// Tiled supports "orthogonal", "isometric", "staggered" (since 0.9) and "hexagonal" (since 0.11).

/// </summary>

[XmlAttribute()]

public TMXMapOrientation orientation = TMXMapOrientation.none;

/// <summary>

/// 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)

/// </summary>

[XmlAttribute()]

public TMXRenderOrder renderorder = TMXRenderOrder.none;

/// <summary>

/// The map width in tiles

/// </summary>

[XmlAttribute()]

public int width = 0;

/// <summary>

/// The map height in tiles.

/// </summary>

[XmlAttribute()]

public int height = 0;

/// <summary>

/// The width of a tile.

/// </summary>

[XmlAttribute()]

public int tilewidth = 0;

/// <summary>

/// The height of a tile.

/// </summary>

[XmlAttribute()]

public int tileheight = 0;

/// <summary>

/// Only for hexagonal maps.

/// Determines the width or height (depending on the staggered axis) of the tile's edge, in pixels.

/// </summary>

[XmlAttribute()]

public int hexsidelength = -1;

/// <summary>

/// For staggered and hexagonal maps, determines which axis("x" or "y") is staggered. (since 0.11)

/// </summary>

[XmlAttribute()]

public TMXStaggerAxis staggeraxis = TMXStaggerAxis.none;

/// <summary>

/// For staggered and hexagonal maps, determines whether the "even" or "odd" indexes along the staggered axis are shifted. (since 0.11)

/// </summary>

[XmlAttribute()]

public TMXStaggerIndex staggerindex = TMXStaggerIndex.none;

/// <summary>

/// The background color of the map. (since 0.9, optional, may include alpha value since 0.15 in the form #AARRGGBB)

/// </summary>

[XmlAttribute()]

public string backgroundcolor = "";// = new Color32(128, 128, 128, 255);

/// <summary>

/// 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)

/// </summary>

[XmlAttribute()]

public int nextobjectid = 0;

#endregion

/// <summary>

/// Wraps any number of custom properties.

/// </summary>

public TMXProperties properties;

/// <summary>

/// The Tilesets that this map contains

/// </summary>

[XmlElement("tileset")]

public List<TMXTileset> tilesets;

/// <summary>

/// The Layers on this map

/// </summary>

[XmlElement("layer", typeof(TMXLayer))]

[XmlElement("objectgroup", typeof(TMXLayer))]

[XmlElement("imagelayer", typeof(TMXLayer))]

public List<object> layers;

}//public class TMXMap

}//namespace TileMapXML

Notice how every attribute has the [XmlAttribute()] tag. Also notice that I have initialized every variable when declaring them, this is to prevent null reference errors and also for error/type checking latter on. Also notice that I have included every attribute that can be included in a TMX file. If you do not want to support loading in staggered, isometric, hexagonal or background color you do not have to include the related variables in your code. Just be aware that by using xml serialization to load in our TMX files that if one of these attributes is included in the TMX file you may get an error when trying to load the file if you do not have it included in your C# script. These errors can be hard to track down latter if your maps are not loading correctly.

Also notice that the tile sets have the [XmlElement("tileset")] tag because it is a list and also in the xml document it is called tileset.

The Layers is a special case. We created a public list of objects called layers and added a tag for each layer type. The order the layer is listed in the xml document is the order it will be added to the layers list.

### TMXTest.cs

Now all that is left is for you to create your NUnit test, add the following test code to TMXTest.cs

[Test]

public void TMXMapLoaded()

{

// Every map has these

// The version of the map should be 1.0 unless you manually change it

Assert.AreEqual(1.0f, tmx.map.version);

// The orientation of the map should not be none

Assert.AreNotEqual(TileMapXML.Map.TMXMapOrientation.none, tmx.map.orientation, "Failed to load in map orientation");

// The render-order should not be none

Assert.AreNotEqual(tmx.map.renderorder, TileMapXML.Map.TMXRenderOrder.none, "Failed to load in render order");

// The width should be > 0

Assert.Greater(tmx.map.width, 0, "Failed to load the map width");

// The height should be > 0

Assert.Greater(tmx.map.height, 0, "Failed to load the map height");

// The tilewidth should be > 0

Assert.Greater(tmx.map.tilewidth, 0, "Failed to load the maps tile width");

// The tileheight should be > 0

Assert.Greater(tmx.map.tileheight, 0, "Failed to load the maps tile height");

// The nextobjectid should be > 0

Assert.Greater(tmx.map.nextobjectid, 0, "Failed to load the maps next object id");

// check values based on the maps orientation

switch(tmx.map.orientation)

{

case TileMapXML.Map.TMXMapOrientation.orthogonal:

// hexsidelength is only for hexagonal maps

Assert.AreEqual(-1, tmx.map.hexsidelength, "hexsidelength is only for hexagonal maps");

// staggeraxis is only for hexagonal and staggered maps

Assert.AreEqual(0, (int)tmx.map.staggeraxis, "staggeraxis is only for hexagonal and staggered maps");

// staggerindex is only for hexagonal and staggered maps

Assert.AreEqual(0, (int)tmx.map.staggerindex, "staggerindex is only for hexagonal and staggered maps");

break;

case TileMapXML.Map.TMXMapOrientation.isometric:

// hexsidelength is only for hexagonal maps

Assert.AreEqual(-1, tmx.map.hexsidelength, "hexsidelength is only for hexagonal maps");

// staggeraxis is only for hexagonal and staggered maps

Assert.AreEqual(0, (int)tmx.map.staggeraxis, "staggeraxis is only for hexagonal and staggered maps");

// staggerindex is only for hexagonal and staggered maps

Assert.AreEqual(0, (int)tmx.map.staggerindex, "staggerindex is only for hexagonal and staggered maps");

break;

case TileMapXML.Map.TMXMapOrientation.staggered:

// hexsidelength is only for hexagonal maps

Assert.AreEqual(-1, tmx.map.hexsidelength, "hexsidelength is only for hexagonal maps");

// staggeraxis is only for hexagonal and staggered maps

Assert.Greater((int)tmx.map.staggeraxis, 0, "staggeraxis is only for hexagonal and staggered maps");

// staggerindex is only for hexagonal and staggered maps

Assert.Greater((int)tmx.map.staggerindex, 0, "staggerindex is only for hexagonal and staggered maps");

break;

case TileMapXML.Map.TMXMapOrientation.hexagonal:

// hexsidelength is only for hexagonal maps

Assert.Greater(tmx.map.hexsidelength, -1, "failed to load in a hexsidelength");

// staggeraxis is only for hexagonal and staggered maps

Assert.Greater((int)tmx.map.staggeraxis, 0, "staggeraxis is only for hexagonal and staggered maps");

// staggerindex is only for hexagonal and staggered maps

Assert.Greater((int)tmx.map.staggerindex, 0, "staggerindex is only for hexagonal and staggered maps");

break;

default:

Assert.Fail("The orientation is not recognized by the system");

break;

}

}//void MapLoaded

Here we check that every attribute was loaded in correctly. Notice that the switch statement we check to make sure that for maps that are not supposed to have a value do not. Also notice that when checking the enums for the staggerindex and stageraxis we cast them to an int value, this is one of the benefits of using enum types and also why the first entry is none. You could have also done

Assert.AreEqual(TileMapXML.Map.TMXStaggerAxis.none, tmx.map.staggeraxis); and Assert.AreNotEqual(TileMapXML.Map.TMXStaggerAxis.none, tmx.map.staggeraxis);

Also notice that the test does not check the backgroundcolor attribute, this is the one case where it is only set if you change it in Tiled so it is possible that it will be an empty string, or it may have a value of some sort. You could add a check to see if it is empty then check to see if it is in the right format #AARRGGBB. I will leave that up to you to figure out how to do on your own.

We are also not checking the child elements of the map, properties, tilesets, and layers; we will check these in separate test cases.

At this point if you run the test in Unity all of your test should pass no matter what map you are using and how it is configured.

