# Tiled Properties Format

The first thing that a map can contain is properties so this will be the next thing to take a look at. If you take a look at <http://doc.mapeditor.org/reference/tmx-map-format> and look at <properties> you will see

## <properties>

Can contain: [property](http://doc.mapeditor.org/reference/tmx-map-format/#property)

Wraps any number of custom properties. Can be used as a child of the map, tile (when part of a tileset), layer, objectgroup and object elements.

### <property>

* **name:** The name of the property.
* **type:** The type of the property. Can be string (default), int, float or bool. (since 0.16)
* **value:** The value of the property.

Boolean properties have a value of either "true" or "false".

When a string property spans contains newlines, the current versions of Tiled Java and Tiled Qt will write out the value as characters contained inside the property element rather than as the value attribute. However, it is at the moment not really possible to edit properties consisting of multiple lines with Tiled.

It is possible that a future version of the TMX format will switch to always saving property values inside the element rather than as an attribute.

## Description

Now if you create different properties save the map and then open it 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>

………………

</map>

Notice that the properties child element contains child elements all called property. This is how a list or an array of child elements is written in xml. Everything that has a property will contain only one child element called properties, all of the properties will be contained in a child element called property. All of the property items are contained in the property as an attribute.

There are two ways that we can load in properties. The first way is to create a properties object that holds a list or array of property objects. You would just have the properties object in your class like so.

public TMXProperties properties;

And then in the TMXProperties class load in a list of property objects like so

[XmlElement("property")]

public List<TMXProperty> properties;

The other option is to have the list of property objects on the main class itself and then load that list in like so.

[XmlArray("properties")]

[XmlArrayItem("property", IsNullable = false)]

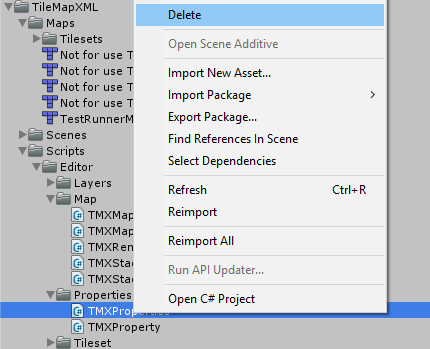
public List<TMXProperty> properties;

The tag is a special tag that tells the xml serializer/deserializer that this element is an array or list. And the tag tells the xml serializer/deserializer that the element is an item in the array or list.

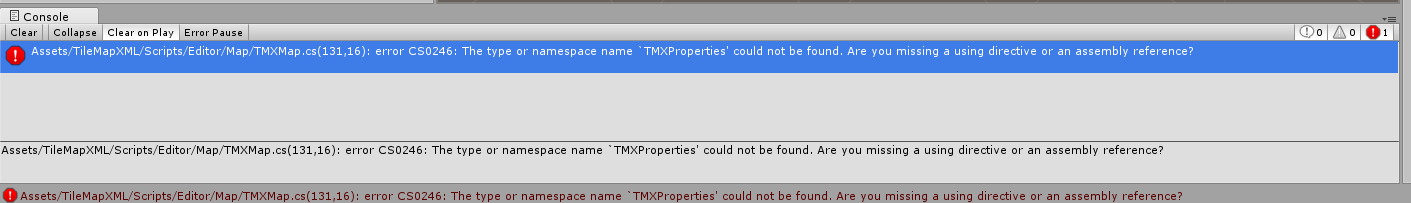
We will be using the second option.

## Code

In Unity delete the C# scripts in TileMapXML->Scripts->Properties called TMXProperties. Just Right click on it and select delete. This was only there as a place holder script until we got to loading the properties.



Now this causes Unity to complain and gives you the following error.



If you double click on this error you will be taken to the public TMXProperties properties variable which needs to be changed to.

### TMXMap.cs

/// <summary>

/// Wraps any number of custom properties.

/// </summary>

[XmlArray("properties")]

[XmlArrayItem("property", IsNullable = false)]

public List<TMXProperty> properties;

### TMXProperty.cs

using System.Xml.Serialization;

namespace TileMapXML

{

/// <summary>

/// <property>

/// • name: The name of the property.

/// • type: The type of the property.Can be string (default), int, float or bool. (since 0.16)

/// • value: The value of the property.

///

/// Boolean properties have a value of either "true" or "false".

///

/// When a string property spans contains newlines,

/// the current versions of Tiled Java and Tiled Qt will write out the

/// value as characters contained inside the property element rather than

/// as the value attribute.However, it is at the moment not really possible

/// to edit properties consisting of multiple lines with Tiled.

/// It is possible that a future version of the TMX format will switch to

/// always saving property values inside the element rather than as an attribute.

/// </summary>

public class TMXProperty

{

#region attributes

/// <summary>

/// The name of the property

/// </summary>

[XmlAttribute()]

public string name;

/// <summary>

/// The type of the property.Can be string (default), int, float or bool. (since 0.16)

/// </summary>

[XmlAttribute()]

public string type = "string";

/// <summary>

/// The value of the property

/// </summary>

[XmlAttribute()]

public string value;

#endregion

}//public class TMXProperty

}//namespace TileMapXML

Notice that every variable in this class is in the attributes region and contains the [XmlAttribute()] tag.

### 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 TMXMapPropertiesLoaded()

{

// If the map contains no properties then this test fails

if(tmx.map.properties.Count == 0)

{

// IF the map has no properties then properties may not have been loaded correctly.

Assert.Fail("This map contains no properties, if this is intended then ignore this failure.");

}

foreach(TMXProperty property in tmx.map.properties)

TMXPropertyLoaded(property);

// If you are using properties to set a value in your map that you need for use in Unity

// add a check here to make sure that it is included in your map

Assert.Pass("Map contains " + tmx.map.properties.Count + " if you have more properties they all did not load in");

}//void TMXMapPropertiesLoaded()

void TMXPropertyLoaded(TMXProperty property)

{

// Make sure that the property has a name

Assert.IsNotNullOrEmpty(property.name, "Failed to load a name for the property");

// Make sure that the property has a type

// Should default to string if the type in the xml attribute was missing

Assert.IsNotNullOrEmpty(property.type, " Failed to load a type for the property");

// Make sure that the property has a value

Assert.IsNotNullOrEmpty(property.value, " Failed to load the value for the property");

switch(property.type)

{

case "int":

int intValue;

Assert.True(int.TryParse(property.value, out intValue));

break;

case "float":

float floatValue;

Assert.True(float.TryParse(property.value, out floatValue));

break;

case "bool":

bool boolValue;

Assert.True(bool.TryParse(property.value, out boolValue));

break;

case "string":

// A string type should not convet into an int float or bool

// if it does then the property did not load in corrrctly

int intString;

Assert.False(int.TryParse(property.value, out intString),property.name + "=" + property.value + " is an int make sure you set the type correctly");

float floatString;

Assert.False(float.TryParse(property.value, out floatString), property.name + "=" + property.value + " is a float make sure you set the type correctly");

bool boolString;

Assert.False(bool.TryParse(property.value, out boolString), property.name + "=" + property.value + " is a bool make sure you set the type correctly");

break;

default:

Assert.Fail(property.type + " is not a vaild type");

break;

}//void TMXPropertyLoaded(TMXProperty property)

We have two methods here, TMXMapPropertiesLoaded() and the TMXPropertyLoaded(TMXProperty property). Notice that the TMXMapPropertiesLoaded has the [Test] tag , this is the test that the test runner will run in Unity. We put all of the testing code for the property itself in a separate method, this is because almost everything in the tmx file can contain properties, we will be calling this method from those tests.

TMXMapPropertiesLoaded will fail if the map that you are using has no properties. You can change this to Assert.Pass("Message") if you do not like to see test fail; I choose to use fail because I did not want to create a false positive if your map has properties and they failed to load in. If I used pass instead then you can easily miss the fact that the properties did not load. When doing unit testing you have to be careful with creating false positives.

It then tests each property that was loaded in to make sure the property loaded in correctly.

The last thing there is a pass message, this is added to make you aware that the test could possibly be a false pass, if you have 8 properties and there are only 7 properties loaded in, this test will pass, the message lets you know how many properties where actually loaded in.

The method makes sure that all of the variables contain a string. The switch statement makes sure the value is correct format for the type. The TryParse returns true if the string can successfully be changed into that type. Notice that in all but the string case we are asserting that this should be true. In the string case we are asserting that the TryParse should be false. The reason we check to make sure that we do this extra test is to make sure that the property did in fact load in correctly. Tiled dose not write out a type if the type is string so when we create a property the type is set to string. If the type did not load in correctly, then the property’s type will be set to string. The default gives a fail message, the type is not a type that Tiled will write.