# Tiled Tileset Format

The next thing that a map can contain is tilesets 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 <tilleset> you will see

#### <tileset>

* **firstgid:** The first global tile ID of this tileset (this global ID maps to the first tile in this tileset).
* **source:** If this tileset is stored in an external TSX (Tile Set XML) file, this attribute refers to that file. That TSX file has the same structure as the <tileset> element described here. (There is the **firstgid** attribute missing and this **source** attribute is also not there. These two attributes are kept in the TMX map, since they are map specific.)
* **name:** The name of this tileset.
* **tilewidth:** The (maximum) width of the tiles in this tileset.
* **tileheight:** The (maximum) height of the tiles in this tileset.
* **spacing:** The spacing in pixels between the tiles in this tileset (applies to the tileset image).
* **margin:** The margin around the tiles in this tileset (applies to the tileset image).
* **tilecount:** The number of tiles in this tileset (since 0.13)
* **columns:** The number of tile columns in the tileset. For image collection tilesets it is editable and is used when displaying the tileset. (since 0.15)

If there are multiple <tileset> elements, they are in ascending order of their firstgid attribute. The first tileset always has a firstgid value of 1. Since Tiled 0.15, image collection tilesets do not necessarily number their tiles consecutively since gaps can occur when removing tiles.

Can contain: [tileoffset](http://doc.mapeditor.org/reference/tmx-map-format/#tileoffset) (since 0.8), [properties](http://doc.mapeditor.org/reference/tmx-map-format/#properties) (since 0.8), [image](http://doc.mapeditor.org/reference/tmx-map-format/#image), [terraintypes](http://doc.mapeditor.org/reference/tmx-map-format/#terraintypes) (since 0.9), [tile](http://doc.mapeditor.org/reference/tmx-map-format/#tile)

## Description

Notice that a tile set has tileoffset, properties, image, terraintypes, and tile. Each one of these contains multiple attributes and child elements. We will be doing each section individually; properties have already been completed. Also note that this project does not support the use of external tile sets (a Tile set with a source of tilesetName.tsx) is not supported. We have already created the place holders for all of the different elements that are needed in 001 Initial Unity Setup.

If you open a map with a tileset up in a text editor or xml editor you will see something similar to this.

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

<image attributes/>

<terraintypes>terrain elements</terraintypes>

<tile/>

………………

</tileset>

Notice that the firstgid is a number, name is a string, tilewidth is a number, tileheight is a number tilecount is a number, and columns is a number. Not displayed here but spacing and margin are numbers, and source is a string. All of these are attributes so will need the attribute tag.

The elements are all normal except for the properties this is the same as we have already seen done in the map.

## Code

### TMXTileset.cs

using System.Collections.Generic;

using System.Xml.Serialization;

namespace TileMapXML.Tileset

{

/// <summary>

/// <tileset>

/// • firstgid: The first global tile ID of this tileset (this global ID maps to the first tile in this tileset).

/// • source: If this tileset is stored in an external TSX (Tile Set XML) file, this attribute refers to that file. That TSX file has the same structure as the <tileset> element described here. (There is the firstgid attribute missing and this source attribute is also not there. These two attributes are kept in the TMX map, since they are map specific.)

/// • name: The name of this tileset.

/// • tilewidth: The (maximum) width of the tiles in this tileset.

/// • tileheight: The (maximum) height of the tiles in this tileset.

/// • spacing: The spacing in pixels between the tiles in this tileset (applies to the tileset image).

/// • margin: The margin around the tiles in this tileset (applies to the tileset image).

/// • tilecount: The number of tiles in this tileset (since 0.13)

/// • columns: The number of tile columns in the tileset. For image collection tilesets it is editable and is used when displaying the tileset. (since 0.15)

///

/// If there are multiple <tileset> elements, they are in ascending order of their firstgid attribute.

/// The first tileset always has a firstgid value of 1.

/// Since Tiled 0.15, image collection tilesets do not necessarily number their tiles consecutively since gaps can occur when removing tiles.

///

/// Can contain: tileoffset (since 0.8), properties (since 0.8), image, terraintypes (since 0.9), tile

/// </summary>

public class TMXTileset

{

#region attributes

/// <summary>

/// The first global tile ID of this tileset (this global ID maps to the first tile in this tileset).

/// </summary>

[XmlAttribute]

public int firstgid;

/// <summary>

/// If this tileset is stored in an external TSX (Tile Set XML) file, this attribute refers to that file.

/// That TSX file has the same structure as the <tileset> element described here.

/// There is the firstgid attribute missing and this source attribute is also not there.

/// These two attributes are kept in the TMX map, since they are map specific.

/// </summary>

[XmlAttribute]

public string source;

/// <summary>

/// The name of this tileset.

/// </summary>

[XmlAttribute]

public string name;

/// <summary>

/// The (maximum) width of the tiles in this tileset.

/// </summary>

[XmlAttribute]

public int tileWidth;

/// <summary>

/// The (maximum) height of the tiles in this tileset.

/// </summary>

[XmlAttribute]

public int tileHeight;

/// <summary>

/// The spacing in pixels between the tiles in this tileset (applies to the tileset image).

/// </summary>

[XmlAttribute]

public int spacing = 0;

/// <summary>

/// The margin around the tiles in this tileset (applies to the tileset image).

/// </summary>

[XmlAttribute]

public int margin = 0;

/// <summary>

/// The number of tiles in this tileset (since 0.13)

/// </summary>

[XmlAttribute]

public int tilecount;

/// <summary>

/// The number of tile columns in the tileset.

/// For image collection tilesets it is editable and is used when displaying the tileset. (since 0.15)

/// </summary>

[XmlAttribute]

public int columns;

#endregion

public TMXTileOffset tileOffset;

/// <summary>

/// Wraps any number of custom properties.

/// </summary>

[XmlArray("properties")]

[XmlArrayItem("property")]

public List<TMXProperty> properties;

/// <summary>

/// The image for this tile set

/// </summary>

public TMXImage image;

/// <summary>

/// The terrain types

/// </summary>

public TMXTerraintypes terraintypes;

/// <summary>

/// The tile information of the tile set

/// </summary>

[XmlElement("tile")]

public List<TMXTilesetTile> tiles;

}//public class TMXTileset

}//namespace TileMapXML.Tileset

Nothing new here. This is done the same as the TMXMap.cs

### TMXTest.cs

Add the following test code.

[Test]

public void TMXTilesetsLoaded()

{

// If there are no tilesets then fail

if(tmx.map.tilesets.Count < 1)

Assert.Fail("Map must contain at least one tileset");

int firstGID = 1;

// Loop through all of the tile sets and make sure that they loaded correctly

foreach(TMXTileset tileset in tmx.map.tilesets)

{

// The first gid should be >= firstGID

Assert.GreaterOrEqual(tileset.firstgid, firstGID, "First Gid must be >= " + firstGID);

// External Tilesets is not suported

Assert.IsNullOrEmpty(tileset.source, "External tilesets are not supported");

// The name sould not be null or empty

Assert.IsNotNullOrEmpty(tileset.name, "name not loaded");

// The tile width should be > 0

Assert.Greater(tileset.tilewidth, 0, "tilewidth not loaded");

// The width of the tile should be >= the map.tilewidth

Assert.GreaterOrEqual(tileset.tilewidth, tmx.map.tilewidth, "The width of a tile must be >= the maps tilewidth");

// The tile height should be > 0

Assert.Greater(tileset.tileheight, 0, "tileheight not loaded");

// The height of the tile should be >= the map.tileheight

Assert.GreaterOrEqual(tileset.tileheight, tmx.map.tileheight, "The height of a tile must be >= the maps tileheight");

// The tile count should be > 0

Assert.Greater(tileset.tilecount, 0, "tilecount not loaded");

// The tile count should be > 0

Assert.Greater(tileset.columns, 0, "columns not loaded");

}//foreach(TMXTileset tileset in tmx.map.tilesets)

}//void TMXTilesetsLoaded()

[Test]

public void TMXTilesetsPropertiesLoaded()

{

foreach(TMXTileset tileset in tmx.map.tilesets)

{

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

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

foreach(TMXProperty property in tileset.properties)

{

TMXPropertyLoaded(property);

}//foreach(TMXProperty property in tileset.properties)

}//foreach(TMXTileset tileset in tmx.map.tilesets)

}//void TMXTilesetsPropertiesLoaded()

Notice that we are only testing the attributes on a tileset. We are not testing the spacing or margin values, these attributes may not be set by Tiled.

We check to make sure that you are not trying to use an external tile set with the Assert.IsNullOrEmpty(tileset.source, "External tilesets are not supported");.

We are also making sure that the tile height and width are loaded in. These values should be greater than or equal to the tile width and tile height set by the map. If they are not the map will not display correctly in Unit.

For the properties on the tilesets we are only checking to make sure they are loaded in correctly.

The rest of the child elements on the tilesets we will add checks for when we get to loading them in.

# Tiled Tileset Tileoffset Format

The first thing that a tileset can contain is a tileoffset 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 <tileoffset> you will see

#### <tileoffset>

* **x:** Horizontal offset in pixels
* **y:** Vertical offset in pixels (positive is down)

This element is used to specify an offset in pixels, to be applied when drawing a tile from the related tileset. When not present, no offset is applied.

## Description

The x and y are attributes of tile offset. I do not think very many people are going to be using this feature of Tiled but I will be loading it in so If you want it use it then you can.

## Code

### TMXTileOffset.cs

using System.Xml.Serialization;

namespace TileMapXML.Tileset

{

/// <summary>

/// <tileoffset>

/// • x: Horizontal offset in pixels

/// • y: Vertical offset in pixels(positive is down)

///

/// This element is used to specify an offset in pixels,

/// to be applied when drawing a tile from the related tileset.

/// When not present, no offset is applied.

/// </summary>

public class TMXTileOffset

{

#region attributes

/// <summary>

/// Horizontal offset in pixels

/// </summary>

[XmlAttribute]

public float x;

/// <summary>

/// Vertical offset in pixels(positive is down)

/// </summary>

[XmlAttribute]

public float y;

#endregion

}//public class TMXTileOffset

}//namespace TileMapXML.Tileset

The tile offset only contains to XmlAttribute tags.

TMXTest.cs

To test to make sure that the offset was loaded in correctly add

[Test]

public void TMXTilesetsTileoffsetLoaded()

{

foreach(TMXTileset tileset in tmx.map.tilesets)

{

// If the tile set has a tile offset

if(tileset.tileoffset != null)

{

Assert.NotNull(tileset.tileoffset.x);

Assert.NotNull(tileset.tileoffset.y);

}

}//foreach(TMXTileset tileset in tmx.map.tilesets)

}//void TMXTilesetsTileoffsetLoaded()

Note that there is no check to see if a tile set has a tile offset and was not loaded. We can have a false positive.

# Tiled Tileset Image Format

The next thing that a tileset can contain is an Image 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 <image> you will see

#### <image>

* **format:** Used for embedded images, in combination with a data child element. Valid values are file extensions like png, gif, jpg, bmp, etc. (since 0.9)
* *id:* Used by some versions of Tiled Java. Deprecated and unsupported by Tiled Qt.
* **source:** The reference to the tileset image file (Tiled supports most common image formats).
* **trans:** Defines a specific color that is treated as transparent (example value: "#FF00FF" for magenta). Up until Tiled 0.12, this value is written out without a # but this is planned to change.
* **width:** The image width in pixels (optional, used for tile index correction when the image changes)
* **height:** The image height in pixels (optional)

Can contain: [data](http://doc.mapeditor.org/reference/tmx-map-format/#data) (since 0.9)

## Description

If you notice that an image can contain data. I have not seen this child element or the format attribute used in Tiled, if you know how or where it is used or why feel free to let me know. At this time I will not be including this child element.

If you open a map with a tileset up in a text editor or xml editor you will see something similar to this.

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

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

Take note that Tiled sets the source of an image relative to the map file path. This is why it is important to make sure that when creating your maps that you make sure everything is within the Assets folder in Unity. When we get to displaying the map in Unity you will end up with errors if you did not follow the instructions in the Initial Unity Set and Tileset Textures tutorials. There will also be errors if you drag and drop your tmx files into Unity, we will be creating an editor script in a future tutorial that you can use to import your map, this script will make sure that the source files get set correctly.

## Code

### TMXImage.cs

using System.Xml.Serialization;

namespace TileMapXML.Tileset

{

/// <summary>

/// <image>

/// • format: Used for embedded images, in combination with a data child element.Valid values are file extensions like png, gif, jpg, bmp, etc. (since 0.9)

/// • id: Used by some versions of Tiled Java.Deprecated and unsupported by Tiled Qt.

/// • source: The reference to the tileset image file (Tiled supports most common image formats).

/// • trans: Defines a specific color that is treated as transparent(example value: "#FF00FF" for magenta). Up until Tiled 0.12, this value is written out without a # but this is planned to change.

/// • width: The image width in pixels (optional, used for tile index correction when the image changes)

/// • height: The image height in pixels (optional)

///

/// Can contain: data (since 0.9)

/// </summary>

public class TMXImage

{

#region attributes

/// <summary>

/// The reference to the tileset image file (Tiled supports most common image formats).

/// </summary>

[XmlAttribute]

public string source;

/// <summary>

/// Defines a specific color that is treated as transparent(example value: "#FF00FF" for magenta).

/// Up until Tiled 0.12, this value is written out without a # but this is planned to change.

/// </summary>

[XmlAttribute]

public string trans;

/// <summary>

/// The image width in pixels (optional, used for tile index correction when the image changes)

/// </summary>

[XmlAttribute]

public int width;

/// <summary>

/// The image height in pixels (optional)

/// </summary>

[XmlAttribute]

public int height;

#endregion

}//public class TMXImage

}//namespace TileMapXML.Tileset

### TMXTest.cs

[Test]

public void TMXTilesetsImageLoaded()

{

foreach(TMXTileset tileset in tmx.map.tilesets)

{

TMXImageLoaded(tileset.image);

}//foreach(TMXTileset tileset in tmx.map.tilesets)

}//void TMXTilesetsImageLoaded()

void TMXImageLoaded(TMXImage image)

{

// There is a source

Assert.IsNotNullOrEmpty(image.source, "Needs a source in order to display");

// The width and height loaded in

Assert.Greater(image.width, 0, "width not loaded");

Assert.Greater(image.height, 0, "height not loaded");

}

# Tiled Tileset Terrain Types Format

The first thing that a tileset can contain is a tileoffset 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 < terraintypes > you will see

#### <terraintypes>

This element defines an array of terrain types, which can be referenced from the terrain attribute of the tile element.

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

#### <terrain>

* **name:** The name of the terrain type.
* **tile:** The local tile-id of the tile that represents the terrain visually.

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

## Description

If you open a map with a tileset that contains terrains up in a text editor or xml editor you will see something similar to this.

<terraintypes>

<terrain name="Grass" tile="31"/>

<terrain name="Dirt" tile="28"/>

<terrain name="Water" tile="51"/>

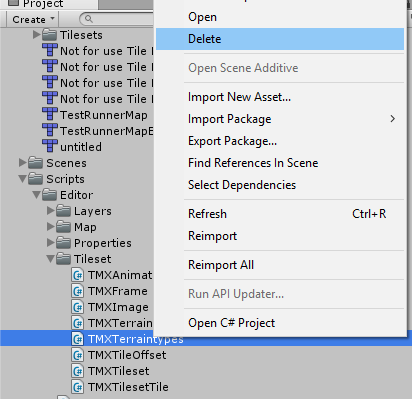
<terrain name="Hole" tile="82"/>

</terraintypes>

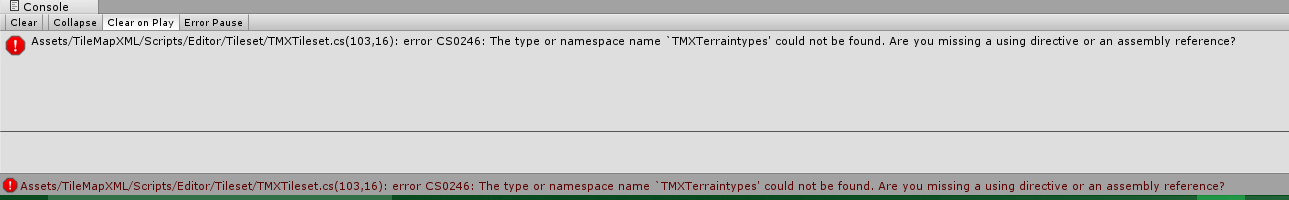
Notice that terraintypes contains a list of terrain elements. We can do this the same way we did properties.

## Code

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



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 TMXTerraintypes terraintypes variable which needs to be changed to.

### TMXTileset.cs

/// <summary>

/// The terrain types

/// </summary>

[XmlArray("terraintypes")]

[XmlArrayItem("terrain")]

public List<TMXTerrain> terraintypes;

### TMXTerrain.cs

using System.Xml.Serialization;

namespace TileMapXML.Tileset

{

/// <summary>

/// <terrain>

/// • name: The name of the terrain type.

/// • tile: The local tile-id of the tile that represents the terrain visually.

///

/// Can contain: properties

/// </summary>

public class TMXTerrain

{

#region attributes

[XmlAttribute]

public string name;

[XmlAttribute]

public int tile;

#endregion

}//public class TMXTerrain

}//namespace TileMapXML.Tileset

### TMXTest.cs

Did not create any test for this, the terrain types are realy something that is only used internally for Tiled.

# Tiled Tileset Tile Format

The last thing that a tileset can contain is a Tile 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 <tile> you will see

#### <tile>

* **id:** The local tile ID within its tileset.
* **terrain:** Defines the terrain type of each corner of the tile, given as comma-separated indexes in the terrain types array in the order top-left, top-right, bottom-left, bottom-right. Leaving out a value means that corner has no terrain. (optional) (since 0.9)
* **probability:** A percentage indicating the probability that this tile is chosen when it competes with others while editing with the terrain tool. (optional) (since 0.9)

Can contain: [properties](http://doc.mapeditor.org/reference/tmx-map-format/#properties), [image](http://doc.mapeditor.org/reference/tmx-map-format/#image) (since 0.9), [objectgroup](http://doc.mapeditor.org/reference/tmx-map-format/#objectgroup) (since 0.10), [animation](http://doc.mapeditor.org/reference/tmx-map-format/#animation) (since 0.10)

## Description

If you open a map with a tileset that contains tiles up in a text editor or xml editor you will see something similar to this.

<tile id="50" terrain="0,0,0,0"/>

<tile id="51" terrain="2,2,2,2">

<objectgroup draworder="index">

<object id="1" x="0" y="0" width="32" height="32"/>

</objectgroup>

<animation>

<frame tileid="51" duration="300"/>

<frame tileid="52" duration="300"/>

<frame tileid="53" duration="300"/>

<frame tileid="52" duration="300"/>

</animation>

</tile>

<tile id="55" terrain="3,3,3,1"/>

The only thing to note is that the objectgroup element is the same as the objectgroup layer. The placeholder TMXObjectGroup will take care of this until we get to the layers section.

## Code

### TMXTilesetTile.cs

using System.Collections.Generic;

using System.Xml.Serialization;

using TileMapXML.Layers;

namespace TileMapXML.Tileset

{

/// <summary>

/// <tile>

/// • id: The local tile ID within its tileset.

/// • terrain: Defines the terrain type of each corner of the tile, given as comma-separated indexes in the terrain types array in the order top-left, top-right, bottom-left, bottom-right.Leaving out a value means that corner has no terrain. (optional) (since 0.9)

/// • probability: A percentage indicating the probability that this tile is chosen when it competes with others while editing with the terrain tool. (optional) (since 0.9)

///

/// Can contain: properties, image (since 0.9), objectgroup (since 0.10), animation (since 0.10)

/// </summary>

public class TMXTilesetTile

{

#region attributes

/// <summary>

/// The local tile ID within its tileset.

/// </summary>

[XmlAttribute]

public int id;

/// <summary>

/// Defines the terrain type of each corner of the tile,

/// given as comma-separated indexes in the terrain types array

/// in the order top-left, top-right, bottom-left, bottom-right.

///

/// Leaving out a value means that corner has no terrain. (optional) (since 0.9)

/// </summary>

[XmlAttribute]

public string terrain;

/// <summary>

/// A percentage indicating the probability that this tile is chosen when it competes with others while editing with the terrain tool. (optional) (since 0.9)

/// </summary>

[XmlAttribute]

public float probability;

#endregion

/// <summary>

/// Wraps any number of custom properties.

/// </summary>

[XmlArray("properties")]

[XmlArrayItem("property")]

public List<TMXProperty> properties;

/// <summary>

/// An image contained in this tile

/// </summary>

public TMXImage image;

/// <summary>

/// Contains a list of objects that this tile can have

/// </summary>

public TMXObjectGroup objectgroup;

/// <summary>

/// Contains a list of animation frames.

/// As of Tiled 0.10, each tile can have exactly one animation associated with it.

/// In the future, there could be support for multiple named animations on a tile.

/// </summary>

[XmlElement("animation")]

public List<TMXAnimation> animation;

}//public class TMXTilesetTile

}//namespace TileMapXML.Tileset

The reason for having the List<TMXAnimation> animation is because in the future Tiled might change it so that a tile can contain multiple animations. This will be a feature that will be really good to implement in Unity. You can have a chest in your tile sheet that has an open, close, idle opened, and idle closed animation. This may also be a good place to make use of properties.

### TMXTest.cs

[Test]

public void TMXTilesetsTilesLoaded()

{

foreach(TMXTileset tileset in tmx.map.tilesets)

{

foreach(TMXTilesetTile tile in tileset.tiles)

{

// The id is the only thing that we need to make sure we have

// it must be > 0

Assert.Greater(tile.id, 0, "id failed to load");

}//foreach(TMXTilesetTile tile in tileset.tiles)

}//foreach(TMXTileset tileset in tmx.map.tilesets)

}//void TMXTilesetsTilesLoaded()

[Test]

public void TMXTilesetsTilesPropertiesLoaded()

{

foreach(TMXTileset tileset in tmx.map.tilesets)

{

foreach(TMXTilesetTile tile in tileset.tiles)

{

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

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

foreach(TMXProperty property in tile.properties)

{

TMXPropertyLoaded(property);

}//foreach(TMXProperty property in tile.properties)

}//foreach(TMXTilesetTile tile in tileset.tiles)

}//foreach(TMXTileset tileset in tmx.map.tilesets)

}//void TMXTilesetsTilesPropertiesLoaded()

The only attribute that we need to make sure gets loaded here is the id

# Tiled Tileset Tile Animation and Frame Format

The last thing that a tileset can contain is a Tile 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 <animation> and <frame> you will see

#### <animation>

Contains a list of animation frames.

As of Tiled 0.10, each tile can have exactly one animation associated with it. In the future, there could be support for multiple named animations on a tile.

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

##### <frame>

* **tileid**: The local ID of a tile within the parent [tileset](http://doc.mapeditor.org/reference/tmx-map-format/#tileset).
* **duration**: How long (in milliseconds) this frame should be displayed before advancing to the next frame.

## Description

If you open a map with a tileset that contains tiles with animations up in a text editor or xml editor you will see

<animation>

<frame tileid="51" duration="300"/>

<frame tileid="52" duration="300"/>

<frame tileid="53" duration="300"/>

<frame tileid="52" duration="300"/>

</animation>

## Code

### TMXAnimation.cs

using System.Collections.Generic;

using System.Xml.Serialization;

namespace TileMapXML.Tileset

{

/// <summary>

/// <animation>

/// Contains a list of animation frames.

/// As of Tiled 0.10, each tile can have exactly one animation associated with it.

/// In the future, there could be support for multiple named animations on a tile.

///

/// Can contain: frame

/// </summary>

public class TMXAnimation

{

#region attributes

#endregion

/// <summary>

/// The frames in the animation

/// </summary>

[XmlElement("frame")]

public List<TMXFrame> frames;

}//public class TMXAnimation

}//namespace TileMapXML.Tileset

As of right now there are no attributes, this may change when Tiled supports more than one animation.

### TMXFrame.cs

using System.Xml.Serialization;

namespace TileMapXML.Tileset

{

/// <summary>

/// <frame>

/// • tileid: The local ID of a tile within the parent tileset.

/// • duration: How long (in milliseconds) this frame should be displayed before advancing to the next frame.

/// </summary>

public class TMXFrame

{

#region attributes

/// <summary>

/// The local ID of a tile within the parent tileset.

/// </summary>

[XmlAttribute]

public int tileid;

/// <summary>

/// How long (in milliseconds) this frame should be displayed before advancing to the next frame.

/// </summary>

[XmlAttribute]

public float duration;

#endregion

}//public class TMXFrame

}//namespace TileMapXML.Tileset

### TMXTest.cs

[Test]

public void TMXTilesetsTilesAnimationsLoaded()

{

foreach(TMXTileset tileset in tmx.map.tilesets)

{

foreach(TMXTilesetTile tile in tileset.tiles)

{

foreach(TMXAnimation animation in tile.animation)

{

foreach(TMXFrame frame in animation.frames)

{

// The tile Id must be > 0

Assert.Greater(frame.tileid, 0, "tileid failed to load");

// The duration must not be null

Assert.IsNotNull(frame.duration, "duration failed to load");

}//foreach(TMXFrame frame in animation.frames)

}//foreach(TMXAnimation animation in tile.animation)

}//foreach(TMXTilesetTile tile in tileset.tiles)

}//foreach(TMXTileset tileset in tmx.map.tilesets)

}//void TMXTilesetsTilesAnimationsLoaded()