# Tiled Layer Format

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

## <layer>

All <tileset> tags shall occur before the first <layer> tag so that parsers may rely on having the tilesets before needing to resolve tiles.

* **name:** The name of the layer.
* *x:* The x coordinate of the layer in tiles. Defaults to 0 and can no longer be changed in Tiled Qt.
* *y:* The y coordinate of the layer in tiles. Defaults to 0 and can no longer be changed in Tiled Qt.
* *width:* The width of the layer in tiles. Traditionally required, but as of Tiled Qt always the same as the map width.
* *height:* The height of the layer in tiles. Traditionally required, but as of Tiled Qt always the same as the map height.
* **opacity:** The opacity of the layer as a value from 0 to 1. Defaults to 1.
* **visible:** Whether the layer is shown (1) or hidden (0). Defaults to 1.
* **offsetx:** Rendering offset for this layer in pixels. Defaults to 0. (since 0.14)
* **offsety:** Rendering offset for this layer in pixels. Defaults to 0. (since 0.14)

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

#### <data>

* **encoding:** The encoding used to encode the tile layer data. When used, it can be "base64" and "csv" at the moment.
* **compression:** The compression used to compress the tile layer data. Tiled Qt supports "gzip" and "zlib".

When no encoding or compression is given, the tiles are stored as individual XML tile elements. Next to that, the easiest format to parse is the "csv" (comma separated values) format.

The base64-encoded and optionally compressed layer data is somewhat more complicated to parse. First you need to base64-decode it, then you may need to decompress it. Now you have an array of bytes, which should be interpreted as an array of unsigned 32-bit integers using little-endian byte ordering.

Whatever format you choose for your layer data, you will always end up with so called "global tile IDs" (gids). They are global, since they may refer to a tile from any of the tilesets used by the map. In order to find out from which tileset the tile is you need to find the tileset with the highest firstgid that is still lower or equal than the gid. The tilesets are always stored with increasing firstgids.

Can contain: [tile](http://doc.mapeditor.org/reference/tmx-map-format/#tile_1)

#### <tile>

* **gid:** The global tile ID.

Not to be confused with the tile element inside a tileset, this element defines the value of a single tile on a tile layer. This is however the most inefficient way of storing the tile layer data, and should generally be avoided.

## Description

Now if you create maps with layers then open it in a text editor or xml editor you will see something similar to this.

<layer name="Tile Layer 1" width="5" height="5" offsetx="-9" offsety="-1">

<data>

<tile gid="32"/>

<tile gid="32"/>

………………………………………..

</data>

</layer>

<layer name="Tile Layer 2" width="5" height="5" visible="0">

<data>

...

</data>

</layer>

## Code

### TMXLayer.cs

using System.Collections.Generic;

using System.Xml.Serialization;

namespace TileMapXML.Layers

{

/// <summary>

/// <layer>

/// All<tileset> tags shall occur before the first<layer> tag so that parsers may rely on having the tilesets before needing to resolve tiles.

/// • name: The name of the layer.

/// • x: The x coordinate of the layer in tiles.Defaults to 0 and can no longer be changed in Tiled Qt.

/// • y: The y coordinate of the layer in tiles.Defaults to 0 and can no longer be changed in Tiled Qt.

/// • width: The width of the layer in tiles.Traditionally required, but as of Tiled Qt always the same as the map width.

/// • height: The height of the layer in tiles.Traditionally required, but as of Tiled Qt always the same as the map height.

/// • opacity: The opacity of the layer as a value from 0 to 1. Defaults to 1.

/// • visible: Whether the layer is shown (1) or hidden(0). Defaults to 1.

/// • offsetx: Rendering offset for this layer in pixels.Defaults to 0. (since 0.14)

/// • offsety: Rendering offset for this layer in pixels. Defaults to 0. (since 0.14)

///

/// Can contain: properties, data

/// </summary>

public class TMXLayer

{

#region attributes

/// <summary>

/// The name of the layer.

/// </summary>

[XmlAttribute]

public string name;

/// <summary>

/// The width of the layer in tiles.

/// Traditionally required, but as of Tiled Qt always the same as the map width.

/// </summary>

[XmlAttribute]

public float width;

/// <summary>

/// The height of the layer in tiles.

/// Traditionally required, but as of Tiled Qt always the same as the map height.

/// </summary>

[XmlAttribute]

public float height;

/// <summary>

/// The opacity of the layer as a value from 0 to 1.

/// Defaults to 1.

/// </summary>

[XmlAttribute]

public float opacity = 1;

/// <summary>

/// Whether the layer is shown (1) or hidden(0).

/// Defaults to 1.

/// </summary>

[XmlAttribute]

public int visible = 1;

/// <summary>

/// Rendering offset for this layer in pixels.

/// Defaults to 0. (since 0.14)

/// </summary>

[XmlAttribute]

public float offsetx = 0;

/// <summary>

/// Rendering offset for this layer in pixels.

/// Defaults to 0. (since 0.14)

/// </summary>

[XmlAttribute]

public float offsety = 0;

#endregion

/// <summary>

/// Wraps any number of custom properties.

/// </summary>

[XmlArray("properties")]

[XmlArrayItem("property")]

public List<TMXProperty> properties;

public TMXData data;

}//public class TMXLayer

}//namespace TileMapXML.Layers

We assign the optional attributes their default value. This makes sure that we have a value for them, Tiled will only write these values to the tmx file if they are not set to their defaults.

### TMXData.cs

using System.Collections.Generic;

using System.Xml.Serialization;

namespace TileMapXML.Layers

{

/// <summary>

/// <data>

/// • encoding: The encoding used to encode the tile layer data.

/// When used, it can be "base64" and "csv" at the moment.

/// • compression: The compression used to compress the tile layer data.

/// Tiled Qt supports "gzip" and "zlib".

///

/// When no encoding or compression is given, the tiles are stored as individual XML tile elements.

/// Next to that, the easiest format to parse is the "csv" (comma separated values) format.

/// The base64-encoded and optionally compressed layer data is somewhat more complicated to parse.

/// First you need to base64-decode it, then you may need to decompress it.

/// Now you have an array of bytes, which should be interpreted as an array of unsigned 32-bit integers using little-endian byte ordering.

/// Whatever format you choose for your layer data, you will always end up with so called "global tile IDs" (gids).

/// They are global, since they may refer to a tile from any of the tilesets used by the map.

/// In order to find out from which tileset the tile is you need to find the tileset with the highest firstgid that is still lower or equal than the gid.

/// The tilesets are always stored with increasing firstgids.

///

/// Can contain: tile

/// </summary>

public class TMXData

{

#region attributes

/// <summary>

/// The encoding used to encode the tile layer data.

/// When used, it can be "base64" and "csv" at the moment.

/// Must be null or empty for use in unity

/// </summary>

public string encoding;

/// <summary>

/// The compression used to compress the tile layer data.

/// Tiled Qt supports "gzip" and "zlib".

/// Must be null or empty for use in unity

/// </summary>

public string compression;

#endregion

/// <summary>

/// List of tiles on a tile layer.

/// </summary>

[XmlElement("tile")]

public List<TMXLayerTile> tiles;

}//public class TMXData

}//namespace TileMapXML.Layers

### TMXLayerTile.cs

using System.Xml.Serialization;

namespace TileMapXML.Layers

{

/// <summary>

/// <tile>

/// • gid: The global tile ID.

///

/// Not to be confused with the tile element inside a tileset,

/// this element defines the value of a single tile on a tile layer.

/// This is however the most inefficient way of storing the tile layer data,

/// and should generally be avoided.

/// </summary>

public class TMXLayerTile

{

#region attributes

[XmlAttribute]

public int gid = -1;

#endregion

}//public class TMXLayerTile

}//namespace TileMapXML.Layers

We assign a default value of -1, this is just to make sure the gid’s are actually loaded in

### TMXTest.cs

Add the following test code

#region Layer Loaded

[Test]

public void TMXLayersLoaded()

{

bool hasTileLayer = false;

foreach(var layer in tmx.map.layers)

{

if(layer is TMXObjectGroup)

TMXObjectGroupLoaded(layer as TMXObjectGroup);

else if(layer is TMXImageLayer)

TMXImageLayerLoaded(layer as TMXImageLayer);

else if(layer is TMXLayer)

{

hasTileLayer = true;

TMXTileLayerLoaded(layer as TMXLayer);

}//else if(layer is TMXLayer)

}//foreach(var layer in tmx.map.layers)

//There must be at least one layer

Assert.True((tmx.map.layers.Count > 0) && (hasTileLayer), "There needs to be at least on tile layer");

}//void TMXLayersLoaded()

#region Tile Layer Loaded

public void TMXTileLayerLoaded(TMXLayer layer)

{

//Name of the layer must not be null

Assert.IsNotNullOrEmpty(layer.name, "Layer must have a name");

//The data is loaded correctly

TMXLayerDataLoaded(layer.data);

}//void TMXTileLayerLoaded(TMXLayer layer)

#region TMXLayerData Loaded

public void TMXLayerDataLoaded(TMXData data)

{

// The encoding and compression must be null

Assert.IsNullOrEmpty(data.encoding, "Invalid encoding used must be XML");

Assert.IsNullOrEmpty(data.compression, "Invalid compression used must be uncompressed");

foreach(TMXLayerTile tile in data.tiles)

{

TMXLayerTileLoaded(tile);

}//foreach(TMXLayerTile tile in data.tiles)

}//void TMXLayerDataLoaded(TMXData data)

#region TMXLayerTile Loaded

public void TMXLayerTileLoaded(TMXLayerTile tile)

{

//The gid should be > -1

Assert.Greater(tile.gid, -1, "gid not loaded correctly");

}//void TMXLayerTileLoaded(TMXLayerTile tile)

#endregion

#endregion

#endregion

#region Object Group Loaded

private void TMXObjectGroupLoaded(TMXObjectGroup tMXObjectGroup)

{

}

#endregion

#region Image Layer Loaded

private void TMXImageLayerLoaded(TMXImageLayer tMXImageLayer)

{

}

#endregion

#endregion

In the TMXLayersLoaded() method we create a bool variable to make sure there is at least one Tile Layer in our map. We then loop through all of the layers in our map. If it is We then check if the layer is an objectLayer and if it is we check to make sure the object layer is loaded correctly TMXObjectGroupLoaded(layer as TMXObjectGroup), we will fill in this method when we load in the objectgoup. We then check to see if the layer is an imageLayer and if it is we check to make sure the imagelayer is loaded correctly ,we will fill in this method when we load in the imagelayer. Then we check if the layer is a tile layer and if it is we set the hasTileLayer to true, and then check to make sure the Tile Layer was Load correctly TMXTileLayerLoaded(layer as TMXLayer). We do these check in this order because all of our layers are going to inherit from TMXLayer. We use the else if to check if something is true only when the if statement before returned false. The last thing we do is make sure the map has at least one tile layer.

The TMXTileLayerLoaded method we make sure that the tile layer has a name, and then make sure that the data is correct.

The TMXLayerDataLoaded method we make sure that the encoding and compression strings are empty or null, these strings will only get set if you set the tile layer format option in Tiled to cvs or one of the Base64 options. We then make sure that the tile was loaded correctly.

The tile method we just make sure that the gid of the tile is greater than -1. If it is not then the data did not get loaded in correctly.