# Tiled Object Group Format

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

## <objectgroup>

* **name:** The name of the object group.
* **color:** The color used to display the objects in this group.
* *x:* The x coordinate of the object group in tiles. Defaults to 0 and can no longer be changed in Tiled Qt.
* *y:* The y coordinate of the object group in tiles. Defaults to 0 and can no longer be changed in Tiled Qt.
* *width:* The width of the object group in tiles. Meaningless.
* *height:* The height of the object group in tiles. Meaningless.
* **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 object group in pixels. Defaults to 0. (since 0.14)
* **offsety:** Rendering offset for this object group in pixels. Defaults to 0. (since 0.14)
* **draworder:** Whether the objects are drawn according to the order of appearance ("index") or sorted by their y-coordinate ("topdown"). Defaults to "topdown".

The object group is in fact a map layer, and is hence called "object layer" in Tiled Qt.

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

#### <object>

* **id:** Unique ID of the object. Each object that is placed on a map gets a unique id. Even if an object was deleted, no object gets the same ID. Can not be changed in Tiled Qt. (since Tiled 0.11)
* **name:** The name of the object. An arbitrary string.
* **type:** The type of the object. An arbitrary string.
* **x:** The x coordinate of the object in pixels.
* **y:** The y coordinate of the object in pixels.
* **width:** The width of the object in pixels (defaults to 0).
* **height:** The height of the object in pixels (defaults to 0).
* **rotation:** The rotation of the object in degrees clockwise (defaults to 0). (since 0.10)
* **gid:** An reference to a tile (optional).
* **visible:** Whether the object is shown (1) or hidden (0). Defaults to 1. (since 0.9)

While tile layers are very suitable for anything repetitive aligned to the tile grid, sometimes you want to annotate your map with other information, not necessarily aligned to the grid. Hence the objects have their coordinates and size in pixels, but you can still easily align that to the grid when you want to.

You generally use objects to add custom information to your tile map, such as spawn points, warps, exits, etc.

When the object has a gid set, then it is represented by the image of the tile with that global ID. The image alignment currently depends on the map orientation. In orthogonal orientation it's aligned to the bottom-left while in isometric it's aligned to the bottom-center.

Can contain: [properties](http://doc.mapeditor.org/reference/tmx-map-format/#properties), [ellipse](http://doc.mapeditor.org/reference/tmx-map-format/#ellipse) (since 0.9), [polygon](http://doc.mapeditor.org/reference/tmx-map-format/#polygon), [polyline](http://doc.mapeditor.org/reference/tmx-map-format/#polyline), *image*

##### <ellipse>

Used to mark an object as an ellipse. The existing x, y, width and height attributes are used to determine the size of the ellipse.

##### <polygon>

* **points:** A list of x,y coordinates in pixels.

Each polygon object is made up of a space-delimited list of x,y coordinates. The origin for these coordinates is the location of the parent object. By default, the first point is created as 0,0 denoting that the point will originate exactly where the object is placed.

##### <polyline>

* **points:** A list of x,y coordinates in pixels.

A polyline follows the same placement definition as a polygon object.

## Description

Now if you create different object layers with different objects on them then open it in a text editor or xml editor you will see something similar to this.

<objectgroup color="#ff0000" name="Collision Layer">

<object id="5" x="0" y="0" width="32" height="32"/>

<object id="6" x="64" y="128" width="32" height="32">

<ellipse/>

</object>

<object id="7" x="160" y="160">

<polygon points="0,0 -64,0 0,-64"/>

</object>

<object id="8" x="160" y="64">

<polyline points="0,0 -32,0 -32,32 -64,32 -64,0 -64,-32 -32,-64 0,-64 0,-32 0,0"/>

</object>

</objectgroup>

<objectgroup name="Object Layer 2">

<object id="11" gid="139" x="64" y="160" width="32" height="32"/>

<object id="12" gid="4" x="0" y="96" width="32" height="32"/>

<object id="13" gid="2147483652" x="32" y="96" width="32" height="32"/>

<object id="14" gid="1073741828" x="96" y="96" width="32" height="32"/>

<object id="15" gid="3221225476" x="128" y="96" width="32" height="32"/>

</objectgroup>

## Code

### TMXObjectGruop.cs

using System.Collections.Generic;

using System.Xml.Serialization;

using TileMapXML.Layers.Objects;

namespace TileMapXML.Layers

{

/// <summary>

/// <objectgroup>

/// • name: The name of the object group.

/// • color: The color used to display the objects in this group.

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

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

/// • width: The width of the object group in tiles. Meaningless.

/// • height: The height of the object group in tiles. Meaningless.

/// • 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 object group in pixels.Defaults to 0. (since 0.14)

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

/// • draworder: Whether the objects are drawn according to the order of appearance ("index") or sorted by their y-coordinate ("topdown"). Defaults to "topdown".

///

/// The object group is in fact a map layer, and is hence called "object layer" in Tiled Qt.

///

/// Can contain: properties, object

/// </summary>

public class TMXObjectGroup : TMXLayer

{

#region attributes

/// <summary>

/// The color used to display the objects in this group.

/// </summary>

[XmlAttribute]

public string color;

/// <summary>

/// Whether the objects are drawn according to the order

/// of appearance ("index") or sorted by their y-coordinate ("topdown").

/// Defaults to "topdown".

/// </summary>

[XmlAttribute]

public string draworder;

#endregion

/// <summary>

/// The objects this group contains

/// </summary>

[XmlElement("object")]

public List<TMXObject> objects;

}//public class TMXObjectGroup

}//namespace TileMapXML.Layers

This script inherits from the TMXLayer script. This means that this script has everything the TMXLayer script has. The color and draworder attributes are the only 2 attributes that are not in the TMXLayer.

The TMXLayer script already has the properties element so all we need to do is add the object element

### TMXObject.cs

using System.Collections.Generic;

using System.Xml.Serialization;

namespace TileMapXML.Layers.Objects

{

/// <summary>

/// <object>

/// • id: Unique ID of the object. Each object that is placed on a map gets a unique id. Even if an object was deleted, no object gets the same ID. Cannot be changed in Tiled Qt. (since Tiled 0.11)

/// • name: The name of the object. An arbitrary string.

/// • type: The type of the object. An arbitrary string.

/// • x: The x coordinate of the object in pixels.

/// • y: The y coordinate of the object in pixels.

/// • width: The width of the object in pixels(defaults to 0).

/// • height: The height of the object in pixels(defaults to 0).

/// • rotation: The rotation of the object in degrees clockwise(defaults to 0). (since 0.10)

/// • gid: A reference to a tile(optional).

/// • visible: Whether the object is shown(1) or hidden(0). Defaults to 1. (since 0.9)

///

/// While tile layers are very suitable for anything repetitive aligned to the tile grid,

/// sometimes you want to annotate your map with other information,

/// not necessarily aligned to the grid.

/// Hence the objects have their coordinates and size in pixels,

/// but you can still easily align that to the grid when you want to.

/// You generally use objects to add custom information to your tile map,

/// such as spawn points, warps, exits, etc.

/// When the object has a gid set,

/// then it is represented by the image of the tile with that global ID.

/// The image alignment currently depends on the map orientation.

/// In orthogonal orientation it's aligned to the bottom-left

/// while in isometric it's aligned to the bottom-center.

///

/// Can contain: properties, ellipse (since 0.9), polygon, polyline, image

/// </summary>

public class TMXObject

{

#region attributes

/// <summary>

/// Unique ID of the object.

/// Each object that is placed on a map gets a unique id.

/// Even if an object was deleted, no object gets the same ID.

/// Cannot be changed in Tiled Qt. (since Tiled 0.11)

/// </summary>

[XmlAttribute]

public int id;

/// <summary>

/// The name of the object. An arbitrary string.

/// </summary>

[XmlAttribute]

public string name;

/// <summary>

/// The type of the object. An arbitrary string.

/// </summary>

[XmlAttribute]

public string type;

/// <summary>

/// The x coordinate of the object in pixels.

/// </summary>

[XmlAttribute]

public float x;

/// <summary>

/// The y coordinate of the object in pixels.

/// </summary>

[XmlAttribute]

public float y;

/// <summary>

/// The width of the object in pixels(defaults to 0).

/// </summary>

[XmlAttribute]

public int width = 0;

/// <summary>

/// The height of the object in pixels(defaults to 0).

/// </summary>

[XmlAttribute]

public int height = 0;

/// <summary>

/// The rotation of the object in degrees clockwise(defaults to 0). (since 0.10)

/// </summary>

[XmlAttribute]

public float rotation = 0;

/// <summary>

/// A reference to a tile(optional).

/// stored as a long(use bitwise to get gid that has been flipped horizontaly or verticaly)

/// </summary>

[XmlAttribute]

public long gid;

/// <summary>

/// Whether the object is shown(1) or hidden(0). Defaults to 1. (since 0.9)

/// </summary>

[XmlAttribute]

public int visible = 1;

#endregion

/// <summary>

/// Wraps any number of custom properties.

/// </summary>

[XmlArray("properties")]

[XmlArrayItem("property")]

public List<TMXProperty> properties;

/// <summary>

/// The ellipse data

/// Will be null unless this object is an ellipse

/// </summary>

public TMXEllipse ellipse;

/// <summary>

/// The polygon data

/// Will be null unless this object is a polygon

/// </summary>

public TMXpolygon polygon;

/// <summary>

/// The polyline data

/// Will be null unless this object is a polyline

/// </summary>

public TMXPolyline polyline;

/// <summary>

/// The image data

/// Will be null unless this object is an image

/// </summary>

public Tileset.TMXImage image;

}//public class TMXObject

}//namespace TileMapXML.Layers.Objects

Tiled has different types of objects that an object can be, a rectangle, an ellipse, a polygon, a polyline, or an image. The type of object this is depends on what child element is not null, see TMXTest.TMXObjectLoaded to see how this is done. You can use this to create your own objects in Unity to represent this object. I will show you how to do this in a later tutorial.

The gid is stored as a long Tiled uses bits for tile flipping.

About tile flipping from <http://doc.mapeditor.org/reference/tmx-map-format>

#### Tile flipping

When you use the tile flipping feature added in Tiled Qt 0.7, the highest two bits of the gid store the flipped state. Bit 32 is used for storing whether the tile is horizontally flipped and bit 31 is used for the vertically flipped tiles. And since Tiled Qt 0.8, bit 30 means whether the tile is flipped (anti) diagonally, enabling tile rotation. These bits have to be read and cleared *before* you can find out which tileset a tile belongs to.

When rendering a tile, the order of operation matters. The diagonal flip (x/y axis swap) is done first, followed by the horizontal and vertical flips.

### TMXEllipse.cs

namespace TileMapXML.Layers.Objects

{

/// <summary>

/// <ellipse>

/// Used to mark an object as an ellipse.

///

/// The existing x, y, width and height attributes are used to determine the size of the ellipse.

/// </summary>

public class TMXEllipse

{

#region attributes

#endregion

}//public class TMXEllipse

}//namespace TileMapXML.Layers.Objects

This is an empty class. Everything that is needed to create an ellipse is in the TMXObject file. This class is simply here to mark an object as an ellipse

### TMXPolygon.cs

using System.Xml.Serialization;

namespace TileMapXML.Layers.Objects

{

/// <summary>

/// <polygon>

/// • points: A list of x, y coordinates in pixels.

///

/// Each polygon object is made up of a space-delimited list of x, y coordinates.

/// The origin for these coordinates is the location of the parent object.

/// By default, the first point is created as 0,0 denoting that the point will

/// originate exactly where the object is placed.

/// </summary>

public class TMXpolygon

{

#region attributes

/// <summary>

/// A list of x, y coordinates in pixels.

/// </summary>

[XmlAttribute]

public string points;

#endregion

}//public class TMXpolygon

}//namespace TileMapXML.Layers.Objects

To create a polygon object in Unity you will have to parse these points. The test in TMXTest. TMXObjectPointsLoaded method will make sure that these points can be parsed into something useful.

### TMXPolyLine.cs

using System.Xml.Serialization;

namespace TileMapXML.Layers.Objects

{

/// <summary>

/// <polyline>

/// • points: A list of x, y coordinates in pixels.

///

/// A polyline follows the same placement definition as a polygon object.

/// Each polyline object is made up of a space-delimited list of x, y coordinates.

/// The origin for these coordinates is the location of the parent object.

/// By default, the first point is created as 0,0 denoting that the point will

/// originate exactly where the object is placed.

/// </summary>

public class TMXPolyline

{

#region attributes

/// <summary>

/// A list of x, y coordinates in pixels.

/// </summary>

[XmlAttribute]

public string points;

#endregion

}//public class TMXPolyline

}//namespace TileMapXML.Layers.Objects

To create a polyline object in Unity you will have to parse these points. The test in TMXTest. TMXObjectPointsLoaded will make sure that these points can be parsed into something useful.

### TMXTest.cs

Fill in the TMXObjectGroupLoaded method

#region Object Group Loaded

private void TMXObjectGroupLoaded(TMXObjectGroup objectgroup)

{

//Name of the objectGroup must not be null

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

foreach(TMXObject tmxObject in objectgroup.objects)

{

// check the object loaded correctly

TMXObjectLoaded(tmxObject);

}//foreach(TMXObject tmxObject in objectGroup.objects)

}//void TMXObjectGroupLoaded(TMXObjectGroup objectGroup)

and add these methods

#region TMXObject Loaded

void TMXObjectLoaded(TMXObject tmxObject)

{

// The id must be a valid id

Assert.Greater(tmxObject.id, 0, "id not loaded");

// if this object is an elipse

if(tmxObject.ellipse != null)

{

// it must have a width and a height

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

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

}//object is an elipse

// else if this object is a polygon

else if(tmxObject.polygon != null)

{

// check the points string is valid

TMXObjectPointsLoaded(tmxObject.polygon.points);

}//object is a polygon

// else if this object is a polyline

else if(tmxObject.polyline != null)

{

// check the points string is valid

TMXObjectPointsLoaded(tmxObject.polyline.points);

}//object is a polyline

// else if this object is an image

else if(tmxObject.image != null)

{

// it must have a valid gid

Assert.Greater(tmxObject.gid, 0);

// it must have a width and a height

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

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

// make sure the image is vaild

TMXImageLoaded(tmxObject.image);

}//object is an image

// else this object is a rectangle

else

{

// it must have a width and a height

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

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

}//object is a rectangle

}//void TMXObjectLoaded(TMXObject tmxObject)

void TMXObjectPointsLoaded(string pointsList)

{

// seperate the points into pairs of x,y coordinates

string[] points = pointsList.Split(' ');

// loop trhough the points array

foreach(string point in points)

{

// seprate the x and y coordinates

// x=coord[0], y=coord[1]

string[] coord = point.Split(',');

// floats to store the x and y coords

float x, y;

// make sure the x coord can be parased

Assert.True(float.TryParse(coord[0], out x), "x coord is invalid");

// make sure the y coord can be parased

Assert.True(float.TryParse(coord[1], out y), "y coord is invalid");

}//foreach(string point in points)

}//TMXObjectPointsLoaded(string pointsList)

[Test]

public void TMXObjectGroupsObjectsPropertiesLoaded()

{

// loop through all of the layers on the map

foreach(var layer in tmx.map.layers)

{

// if the layer is an objectgroup

if(layer is TMXObjectGroup)

{

// get a reference to the layer as an objectgroup

TMXObjectGroup objectgroup = layer as TMXObjectGroup;

foreach(TMXObject tmxObject in objectgroup.objects)

{

// check the properties are valid

foreach(TMXProperty property in tmxObject.properties)

{

//check the property is valid

TMXPropertyLoaded(property);

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

}//foreach(TMXObject tmxObject in objectGroup.objects)

}//if(layer is TMXObjectGroup)

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

}//void TMXObjectGroupsObjectsPropertiesLoaded()

#endregion

#endregion