bjorn / tiled

TMX Map Format

The TMX (Tile Map XML) map format used by Tiled is a flexible way to describe a tile based map. It can describe maps with any tile size, any amount of layers, any number of tile sets and it allows custom properties to be set on most elements. Beside tile layers, it can also contain groups of objects that can be placed freely.

In this document we'll go through each element found in this map format. The elements are mentioned in the headers and the list of attributes of the elements are listed right below, followed by a short explanation. Attributes that are deprecated or unsupported by the current version of Tiled are formatted in italics.

<map>

- **version:** The TMX format version, generally 1.0.
- **orientation:** Map orientation. Tiled supports "orthogonal" and "isometric" 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.

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).

Can contain: properties, tileset, layer, objectgroup

<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.
- **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).

Can contain: image, tile

<image>

- *format:* To be used when embedding images. Deprecated and unsupported in Tiled Qt.
- *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).

As of the current version of Tiled Qt, each tileset has a single image associated with it, which is cut into smaller tiles based on the attributes defined on the tileset element. Later versions may add support for adding multiple images to a single tileset, as is possible in Tiled Java.

<tile>

- **id:** The local tile ID within its tileset.

Can contain: properties, *image*

<layer>

- **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.

Can contain: properties, 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 `firstgid`s.

When you use the tile flipping feature added in Tiled Qt 0.7.0, 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. These two bits have to be read and cleared *before* you can find out which tileset a tile belongs to.

The following C++ pseudo-code should make it all clear:

```
// Bits on the far end of the 32-bit global tile ID are used for tile flags
const unsigned FLIPPED_HORIZONTALLY_FLAG = 0x80000000;
const unsigned FLIPPED_VERTICALLY_FLAG   = 0x40000000;

...

// Extract the contents of the <data> element
string tile_data = ...

unsigned char *data = decompress(base64_decode(tile_data));
unsigned tile_index = 0;

// Here you should check that the data has the right size
// (map_width * map_height * 4)

for (int y = 0; y < map_height; ++y) {
  for (int x = 0; x < map_width; ++x) {
    unsigned global_tile_id = data[tile_index] |
                              data[tile_index + 1] << 8 |
                              data[tile_index + 2] << 16 |
                              data[tile_index + 3] << 24;
    tile_index += 4;

    // Read out the flags
    bool flipped_horizontally = (global_tile_id & FLIPPED_HORIZONTALLY_FLAG);
    bool flipped_vertically = (global_tile_id & FLIPPED_VERTICALLY_FLAG);
```

```
      // Clear the flags
      global_tile_id &= ~(FLIPPED_HORIZONTALLY_FLAG | FLIPPED_VERTICALLY_FLAG);

      // Resolve the tile
      for (int i = tileset_count - 1; i >= 0; --i) {
        Tileset *tileset = tilesets[i];

        if (tileset->first_gid() <= global_tile_id) {
          tiles[y][x] = tileset->tileAt(global_tile_id - tileset->first_gid());
          break;
        }
      }
    }
  }
```

(Since the above code was put together on this wiki page and can't be directly tested, please make sure to report any errors you encounter when basing your parsing code on it, thanks.)

Can contain: tile

<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.

<objectgroup>

- **name:** The name of the object 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.

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

Can contain: object

<object>

- **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.
- **height:** The height of the object in pixels.
- **gid:** An reference to a tile (optional).

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. Currently that means `width` and `height` are ignored for such objects. 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, *image*

<properties>

Can contain: 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.
- **value:** The value of the property.

When the 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.

Last edited by stefanbeller, August 17, 2011

- Training
- Site Status

Tools

- GitHub for Mac
- Issues for iPhone
- Gist: Code Snippets
- GitHub Enterprise
- Job Board

Extras

- GitHub Shop
- The Octodex

Documentation

- GitHub Help
- Developer API
- GitHub Flavored Markdown
- GitHub Pages

- Terms of Service
- Privacy
- Security

Markdown Cheat Sheet

Format Text

Headers

```
# This is an <h1> tag
## This is an <h2> tag
###### This is an <h6> tag
```

Text styles

```
*This text will be italic*
_This will also be italic_
**This text will be bold**
__This will also be bold__

*You **can** combine them*
```

Lists

Unordered

```
* Item 1
* Item 2
  * Item 2a
  * Item 2b
```

Ordered

```
1. Item 1
2. Item 2
3. Item 3
   * Item 3a
   * Item 3b
```

Miscellaneous

Images

```
![GitHub Logo](/images/logo.png)
Format: ![Alt Text](url)
```

Links

```
http://github.com - automatic!
[GitHub](http://github.com)
```

Blockquotes

```
As Kanye West said:
> We're living the future so
> the present is our past.
```

Code Examples in Markdown

Syntax highlighting with GFM

````
```javascript
function fancyAlert(arg) {
  if(arg) {
    $.facebox({div:'#foo'})
  }
}
```
````

Or, indent your code 4 spaces

```
Here is a Python code example
without syntax highlighting:

    def foo:
      if not bar:
        return true
```

Inline code for comments

```
I think you should use an
`<addr>` element here instead.
```

Something went wrong with that request. Please try again. Dismiss