
tmxlib Documentation

Release 0.2.0

Petr Viktorin

January 16, 2015

1	Installation	3
2	Development	5
2.1	Tests	5
2.2	Documentation	5
3	Versioning & TODO	7
4	Contents	9
4.1	Overview	9
4.2	tmxlib Module Reference	11
5	Indices and tables	37
	Python Module Index	39

tmxlib is a Python library for handling TMX tile maps. It serves a relatively specific purpose: making it easy to write scripts for automatic handling of TMX files.

If you aren't familiar with TMX, or you just want to make some maps, install [Tiled](#), a GUI editor, and play around with it. Tiled's wiki and IRC channel are places to go if you have questions about the TMX format.

If you're looking to use maps in a game, chances are *tmxlib* won't help you much. Try [pytmxloader](#), [PyTMX](#), or one of the other projects listed on the [Tiled wiki](#).

Installation

To install `tmxlib`, you can use `pip`: `pip install --user tmxlib`. To install system-wide, leave out the `--user` option.

If you can't find `pip` on your system, look around. In Fedora, it's named `pip-python` and lives in the `python-pip` package.

Optionally, also install the `lxml` and `Pillow` packages to speed up XML and image handling, respectively. Linux distributions are likely to have them (in Fedora, `yum install python-lxml python-imaging`). If you can't find them, use `pip` to get them.

Development

The project is [hosted on Github](#) (as `pytmxlib`), free for anyone to file bugs, clone, fork, or otherwise help make it better.

To install the library for development, navigate to the source folder, and run `python setup.py develop`.

2.1 Tests

To run tests, `pip install pytest-cov`, and run `py.test`.

Tests can be run using [tox](#), to ensure cross-Python compatibility. Make sure you have all supported Pythons (2.6, 2.7, 3.3) installed, and run `tox`.

Nowadays we use Travis CI and Coveralls to run tests after each commit:

2.2 Documentation

This documentation is generated using [Sphinx](#). To build it, `pip install sphinx` and run `make` in the `doc/` directory.

Versioning & TODO

This package sports the [SemVer](#) versioning scheme. In this pre-1.0 version, that doesn't mean much.

Version 1.0 will include at least one generally useful command-line utility, most likely a crop/merge tool for maps.

4.1 Overview

Before doing anything else, import *tmxlib*.

```
>>> import tmxlib
```

4.1.1 Loading and saving

Loading a map from a file is quite easy:

```
>>> filename = 'desert.tmx'
>>> tmxlib.Map.open(filename)
<tmxlib.Map object at ...>
```

You can also load from a string:

```
>>> string = open('desert.tmx', 'rb').read()
>>> map = tmxlib.Map.load(string)
```

Saving is equally easy:

```
>>> map.save('saved.tmx')
>>> map_as_string = map.dump()
```

4.1.2 Maps

Map is *tmxlib*'s core class.

Each map has three “size” attributes: *size*, the size of the map in tiles; *tile_size*, the pixel size of one tile; and *pixel_size*, which is the product of the two. Each of these has *height* and *width* available as separate attributes; for example *pixel_height* would give the map's height in pixels.

A map's *orientation* is its fundamental mode. Tiled currently supports ‘orthogonal’ and ‘isometric’ orientations, but *tmxlib* will currently not object to any other orientation (as it does not need to actually draw maps). Orthogonal orientation is the default.

Each map has a dict of properties, with which you can assign arbitrary string values to string keys.

4.1.3 Tilesets

A map has one or more tilesets, which behave as lists of tiles.

```
>>> map.tilesets
[<ImageTileset 'Desert' at ...>]
>>> tileset = map.tilesets[0]
>>> len(tileset)
48
>>> tileset[0]
<TilesetTile #0 of Desert at ...>
>>> tileset[-1]
<TilesetTile #47 of Desert at ...>
```

As a convenience, tilesets may be accessed by name instead of number. A name will always refer to the first tileset with such name.

You can also remove tilesets (using either names or indexes). However, note that to delete a tileset, the map may not contain any of its tiles.

```
>>> del map.tilesets['Desert']
Traceback (most recent call last):
...
UsedTilesetError: Cannot remove <ImageTileset 'Desert' at ...>: map contains its tiles
Traceback (most recent call last):
...
UsedTilesetError: Cannot remove <ImageTileset 'Desert' at ...>: map contains its tiles
```

(If this causes you trouble when you need to move tilesets around, use the `map.tilesets.move` method)

Tilesets are not tied to maps, which means that several maps can share the same tileset object.

In map data, tiles are referenced by the `GID`, which uniquely determines the tile across all the map's tilesets.

```
>>> tile = tileset[10]
>>> tile.gid(map)
11
```

Each tileset within a map has a *first gid*, the `GID` of its first object. The `first_gid` is always *number of tiles in all preceding tilesets + 1*. (It is written to the `TMX` file to help loaders, but should not be changed there.)

Modifying the list of tilesets can cause the `first_gid` to change. All affected tiles in the map will automatically be renumbered in this case.

4.1.4 Layers

As with tilesets, each map has layers. Like tilesets, these can be accessed either by index or by name.

```
>>> map.layers[0]
<TileLayer #0: 'Ground' at ...>
>>> map.layers['Ground']
<TileLayer #0: 'Ground' at ...>
```

Creating layers directly can be a hassle, so `Map` provides an `add_layer` method that creates a compatible empty layer.

```
>>> map.add_layer('Sky')
<TileLayer #1: 'Sky' at ...>
>>> map.add_layer('Underground', before='Ground')
<TileLayer #0: 'Underground' at ...>
```

```
>>> map.layers
[<TileLayer #0: 'Underground' at ...>, <TileLayer #1: 'Ground' at ...>, <TileLayer #2: 'Sky' at ...>]
```

Layers come in two flavors: *tile layers*, which contain a rectangular grid of tiles, and *object layers*, which contain objects. This overview will only cover the former; object layers are explained in their documentation.

4.1.5 Tile layers

A tile layer is basically a 2D array of map tiles. Index the layer with the x and y coordinates to get a MapTile object.

```
>>> layer = map.layers['Ground']
>>> layer[0, 0]
<MapTile (0, 0) on Ground, gid=30 at ...>
>>> layer[6, 7]
<MapTile (6, 7) on Ground, gid=40 at ...>
```

The MapTile object is a reference to a particular place in the map. This means that changing the MapTile object (through its *value* attribute, for example) will update the map.

The easiest way to change the map, though, is to assign a tileset tile to a location on the map.

```
>>> layer[6, 7] = map.tilesets['Desert'][29]
```

Map tiles can also be flipped around, using Tiled's three flipping flags: horizontal (H), vertical (V), and diagonal (D) flip.

```
>>> tile = layer[6, 7]
>>> tile.flipped_horizontally = True
>>> tile
<MapTile (6, 7) on Ground, gid=30 H at ...>
>>> tile.vflip()
>>> tile
<MapTile (6, 7) on Ground, gid=30 HV at ...>
>>> tile.rotate()
>>> tile
<MapTile (6, 7) on Ground, gid=30 VD at ...>
```

Map tiles are true in a boolean context iff they're not empty (i.e. their *gid* is not 0).

4.1.6 Images and pixels

The library has some basic support for working with tile images.

If tmxlib can't find [Pillow](#) (or [PIL](#)), it will use the pure-python [png](#) package. This is very slow when reading the pictures, and it can only handle PNG files. For this reason, it's recommended that you install PIL to work with images.

```
>>> map.tilesets['Desert'][0].get_pixel(0, 0)
(1.0, 0.8156862..., 0.5803921..., 1.0)
>>> map.layers['Ground'][0, 0].get_pixel(0, 0)
(1.0, 0.8156862..., 0.5803921..., 1.0)
```

4.2 tmxlib Module Reference

The main module exports the most important classes directly:

- `Map`, the main object
- Layer objects: `ImageLayer`, `ObjectLayer`, and `TileLayer`
- `MapTile`
- `ImageTileset`, the only kind of tileset so far, and `TilesetTile`
- Map object classes: `PolygonObject`, `PolylineObject`, `RectangleObject`, and `EllipseObject`
- Exceptions, `UsedTilesetError` and `TilesetNotInMapError`

See submodule documentation for more details:

4.2.1 The `tmxlib.map` module

Map

class `tmxlib.map.Map` (*size*, *tile_size*, *orientation='orthogonal'*, *background_color=None*,
base_path=None, *render_order=None*)

A tile map, `tmxlib`'s core class

init arguments, which become attributes:

size

a (height, width) pair specifying the size of the map, in tiles

tile_size

a pair specifying the size of one tile, in pixels

orientation

The orientation of the map (`'orthogonal'`, `'isometric'`, or `'staggered'`)

background_color

The background color for the map, as a triple of floats (0..1)

Other attributes:

tilesets

A `TilesetList` of tilesets this map uses

layers

A `LayerList` of layers this map uses

properties

A dict of properties, with string (or unicode) keys & values

pixel_size

The size of the map, in pixels. Not settable directly: use *size* and *tile_size* for that.

end_gid

The first GID that is not available for tiles. This is the `end_gid` for the map's last tileset.

Unpacked size attributes:

Each “size” property has corresponding “width” and “height” properties.

height

width

tile_height

tile_width

pixel_height**pixel_width**

Methods:

add_layer (*name*, *before=None*, *after=None*, *layer_class=None*)

Add an empty layer with the given name to the map.

By default, the new layer is added at the end of the layer list. A different position may be specified with either of the *before* or *after* arguments, which may be integer indices or names.

layer_class defaults to `TileLayer`

add_tile_layer (*name*, *before=None*, *after=None*)

Add an empty tile layer with the given name to the map.

See `add_layer`

add_object_layer (*name*, *before=None*, *after=None*)

Add an empty object layer with the given name to the map.

See `add_layer`

add_image_layer (*name*, *image*, *before=None*, *after=None*)

Add an image layer with the given name and image to the map.

See `add_layer`

all_tiles ()

Yield all tiles in the map, including tile objects

all_objects ()

Yield all objects in the map

get_tiles (*x*, *y*)

For each tile layer, yield the tile at the given position.

check_consistency ()

Check that this map is okay.

Most checks are done when reading a map, but if more are required, call this method after reading. This will do a more expensive check than what's practical from within readers.

Loading and saving (see `tmxlib.fileio.ReadWriteBase` for more information):

classmethod open (*filename*, *shared=False*)**classmethod load** (*string*)**save** (*filename*)**dump** (*string*)**to_dict** ()

Export to a dict compatible with Tiled's JSON plugin

You can use e.g. a JSON or YAML library to write such a dict to a file.

classmethod from_dict (*dct*, **args*, ***kwargs*)

Import from a dict compatible with Tiled's JSON plugin

Use e.g. a JSON or YAML library to read such a dict from a file.

Parameters

- **dct** – Dictionary with data
- **base_path** – Base path of the file, for loading linked resources

4.2.2 The `tmxlib.layer` module

Layer

class `tmxlib.layer.Layer` (*map, name, visible=True, opacity=1*)

Base class for map layers

init arguments, which become attributes:

map

The map this layer belongs to. Unlike tilesets, layers are tied to a particular map and cannot be shared.

name

Name of the layer

visible

A boolean setting whether the layer is visible at all. (Actual visibility also depends on *opacity*)

opacity

Floating-point value for the visibility of the layer. (Actual visibility also depends on *visible*)

Other attributes:

properties

Dict of properties with string (or unicode) keys and values.

type

'tiles' if this is a tile layer, 'objects' if it's an object layer, 'image' for an object layer.

index

Index of this layer in the layer list

A Layer is false in a boolean context iff it is empty, that is, if all tiles of a tile layer are false, or if an object layer contains no objects.

Methods:

all_objects ()

Yield all objects in this layer

all_tiles ()

Yield all tiles in this layer, including empty ones and tile objects

Dict import/export:

to_dict ()

Export to a dict compatible with Tiled's JSON plugin

classmethod from_dict (*dct, *args, **kwargs*)

Import from a dict compatible with Tiled's JSON plugin

TileLayer

class `tmxlib.layer.TileLayer` (*map, name, visible=True, opacity=1, data=None*)

A tile layer

Acts as a 2D array of MapTile's, indexed by [x, y] coordinates. Assignment is possible either via numeric values, or by assigning a TilessetTile. In the latter case, if the tileset is not on the map yet, it is added.

See [Layer](#) documentation for most init arguments.

Other init arguments, which become attributes:

data

Optional list (or array) containing the values of tiles in the layer, as one long list in row-major order. See `TileLikeObject.value` for what the numbers will mean.

Methods:

all_objects()

Yield all objects in this layer

all_tiles()

Yield all tiles in this layer, including empty ones.

Tile access:

__getitem__(*pos*)

Get a `MapTile` representing the tile at the given position.

Supports negative indices by wrapping in the obvious way.

__setitem__(*pos, value*)

Set the tile at the given position

The set value can be either a raw integer value, or a `TilesetTile`. In the latter case, any tileset not in the map yet will be added to it.

Supports negative indices by wrapping in the obvious way.

Methods to be overridden in subclasses:

value_at(*pos*)

Return the value at the given position

See `MapTile` for an explanation of the value.

set_value_at(*pos, new*)

Sets the raw value at the given position

See `MapTile` for an explanation of the value.

Dict import/export:

to_dict()

Export to a dict compatible with Tiled's JSON plugin

classmethod from_dict(*dct, *args, **kwargs*)

Import from a dict compatible with Tiled's JSON plugin

ObjectLayer

class `tmxlib.layer.ObjectLayer`(*map, name, visible=True, opacity=1, color=None*)

A layer of objects.

Acts as a `named list` of objects. This means semantics similar to layer/tileset lists: indexing by name is possible, where a name references the first object of such name.

See `Layer` for inherited init arguments.

ObjectLayer-specific init arguments, which become attributes:

color

The intended color of objects in this layer, as a triple of floats (0..1)

Methods:

all_objects ()
Yield all objects in this layer (i.e. return self)

all_tiles ()
Yield all tile objects in this layer, in order.

Dict import/export:

to_dict ()
Export to a dict compatible with Tiled's JSON plugin

classmethod from_dict (dct, *args, **kwargs)
Import from a dict compatible with Tiled's JSON plugin

ImageLayer

class `tmxlib.layer.ImageLayer` (map, name, visible=True, opacity=1, image=None)
An image layer

See [Layer](#) documentation for most init arguments.

Other init arguments, which become attributes:

image
The image to use for the layer

Dict import/export:

to_dict ()
Export to a dict compatible with Tiled's JSON plugin

classmethod from_dict (dct, *args, **kwargs)
Import from a dict compatible with Tiled's JSON plugin

LayerList

class `tmxlib.layer.LayerList` (map, lst=None)
A list of layers.

Allows indexing by name, and can only contain layers of a single map.

See [NamedElementList](#) for LayerList's methods.

4.2.3 The tmxlib.tile module

TileLikeObject

class `tmxlib.tile.TileLikeObject`
Bases: `tmxlib.helpers.TileMixin`

Base tile-like object: regular tile or tile object.

Has an associated layer and value, and can be flipped, etc.

A TileLikeObject is “true” iff there's a tile associated with it. Empty, “false” tiles have a GID of zero.

Note: Subclasses should use the `_value` attribute for your own purposes. The `value` allows setting itself to

TilesetTiles, has checks, etc.

Tile attributes & methods:

tileset

Get the referenced tileset

value

Numeric value of a tile, representing the image and transformations.

See the TMX format for a hopefully more detailed specification. The upper bits of this number are used for flags:

- 0x80000000: tile is flipped horizontally.
- 0x40000000: tile is flipped vertically.
- 0x20000000: tile is flipped diagonally (axes are swapped).
- 0x10000000: tmxlib reserves this bit for now, just because 0xFFFFFFFF is a nice round number.

The rest of the value is zero if the layer is empty at the corresponding spot (or an object has no associated tile image), or it holds the GID of the tileset-tile.

The GID can be computed as $1 + X + Y$ where X is the number of tiles in all tilesets preceding the tile's, and Y is the number of the tile within its tileset.

The individual parts of value are reflected in individual properties:

- flipped_horizontally (0x80000000)
- flipped_vertically (0x40000000)
- flipped_diagonally (0x20000000)
- gid (0x0FFFFFFF)

The properties themselves have a *value* attribute, e.g. `tmxlib.MapTile.flipped_diagonally.value == 0x20000000`.

gid

flipped_horizontally

flipped_vertically

flipped_diagonally

See [value](#)

tileset_tile

Get the referenced tileset tile

number

Get the number of the referenced tileset tile

image

Get the image of the tile. (N.B. see full docstring!)

N.B. No transformations are applied to the image. This can change in future versions. Use `self.tileset_tile.image` for future-safe behavior.

get_pixel (x, y)

Get the pixel at the given x, y coordinates.

Handles negative indices in the obvious way.

tile_to_image_coordinates (x, y)

Transform map-tile pixel coordinates to tileset-tile pixel coords.

Handles negative indices in the obvious way.

Flipping helpers:

hflip()

Flip the tile horizontally

vflip()

Flip the tile vertically

rotate(*degrees=90*)

Rotate the tile clockwise by the specified number of degrees

Note that tiles can only be rotated in 90-degree increments.

Inherited:

map

size

Size of the referenced tile, taking rotation into account. The size is given in map tiles, i.e. “normal” tiles are 1x1. A “large tree” tile, twice as big as a regular tile, would have a size of (1, 2). The size will be given as floats.

Empty tiles have (0, 0) size.

MapTile

class `tmxlib.tile.MapTile(layer, pos)`

References a particular spot on a tile layer

MapTile object can be hashed and they compare equal if they refer to the same tile of the same layer.

init arguments, which become attributes:

layer

The associated layer.

pos

The associated coordinates, as (x, y), in tile coordinates.

See [TileLikeObject](#) for attributes and methods shared with tile objects.

properties

Properties of the *referenced* tileset-tile

Note: Changing this will change properties of all tiles using this image. Possibly even across more maps if tilesets are shared.

See [TilesetTile](#).

4.2.4 The `tmxlib.tileset` module

Tileset

class `tmxlib.tileset.Tileset(name, tile_size)`

Base class for a tileset: bank of tiles a map can use.

There are two kinds of tilesets: external and internal. Internal tilesets are specific to a map, and their contents are saved inside the map file. External tilesets are saved to their own file, so they may be shared between several maps. (Of course, any tileset can be shared between maps at the Python level; this distinction only applies to what happens on disk.) External tilesets have the file path in their *source* attribute; internal ones have *source* set to None.

tmxlib will try to ensure that each external tileset gets only loaded once, and the resulting Python objects are shared. See `ReadWriteBase.open()` for more information.

init arguments, which become attributes:

name

Name of the tileset

tile_size:

A (width, height) pair giving the size of a tile in this tileset. In cases where a tileset can have unequally sized tiles, the tile size is not defined. This means that this property should not be used unless working with a specific subclass that defines `tile_size` better.

source

For external tilesets, the file name for this tileset. None for internal ones.

Other attributes:

properties

A dict with string (or unicode) keys and values. Note that the official TMX format does not support tileset properties (yet), so editors like Tiled will remove these. (tmxlib saves and loads them just fine, however.)

terrains

A `TerrainList` of terrains belonging to this tileset. Note that tileset tiles reference these by index, and the indices are currently not updated when the `TerrainList` is modified. This may change in the future.

tile_offset

An offset in pixels to be applied when drawing a tile from this tileset.

Unpacked versions of tuple attributes:

tile_width

tile_height

tile_offset_x

tile_offset_y

Loading and saving (see `tmxlib.fileio.ReadWriteBase` for more information):

classmethod open (*filename*, *shared=False*)

classmethod load (*string*)

save (*filename*)

dump (*string*)

to_dict (***kwargs*)

Export to a dict compatible with Tiled's JSON plugin

classmethod from_dict (*dct*, *base_path=None*)

Import from a dict compatible with Tiled's JSON plugin

List-like access:

__getitem__ (*n*)

Get tileset tile with the given number.

Supports negative indices by wrapping around, as one would expect.

`__len__()`

Return the number of tiles in this tileset.

Subclasses need to override this method.

`__iter__()`

Iterate through tiles in this tileset.

Overridable methods:

`tile_image(number)`

Return the image used by the given tile.

Usually this will be a region of a larger image.

Subclasses need to override this method.

GID calculation methods:

Note: `TilesetList` depends on the specific GID calculation algorithm provided by these methods to renumber a map's tiles when tilesets are moved around. Don't override these unless your subclass is not used with vanilla `TilesetLists`.

`first_gid(map)`

Return the first gid used by this tileset in the given map

`end_gid(map)`

Return the first gid after this tileset in the given map

ImageTileset

class `tmxlib.tileset.ImageTileset` (*name*, *tile_size*, *image*, *margin=0*, *spacing=0*, *source=None*,
base_path=None)

A tileset whose tiles form a rectangular grid on a single image.

This is the default tileset type in Tiled.

init arguments, which become attributes:

`name`

`tile_size`

`source`

see `Tileset`

`image`

The `Image` this tileset is based on.

`margin`

Size of a border around the image that does not contain tiles, in pixels.

`spacing`

Space between adjacent tiles, in pixels.

Other attributes:

`column_count`

Number of columns of tiles in the tileset

`row_count`

Number of rows of tiles in the tileset

See `Tileset` for generic tileset methods.

ImageTileset methods:

tile_image (*number*)
Return the image used by the given tile

TilesetTile

class `tmxlib.tileset.TilesetTile` (*tileset, number*)

Reference to a tile within a tileset

init arguments, which become attributes:

tileset
the tileset this tile belongs to

number
the number of the tile

Other attributes:

pixel_size
The size of the tile, in pixels. Also available as (`pixel_width`, `pixel_height`).

properties
A string-to-string dictionary holding custom properties of the tile

image
Image this tile uses. Most often this will be a `region` of the tileset's image.

terrain_indices
List of indices to the tileset's terrain list for individual corners of the tile. See the TMX documentation for details.

terrains
Tuple of terrains for individual corners of the tile. If no terrain is given, `None` is used instead.

probability
The probability that this tile will be chosen among others with the same terrain information. May be `None`.

Methods:

gid (*map*)
Return the GID of this tile for a given map

The GID is a map-specific identifier unique for any tileset-tile the map uses.

get_pixel (*x, y*)
Get a pixel at the specified location.

Pixels are returned as RGBA 4-tuples.

TilesetList

class `tmxlib.tileset.TilesetList` (*map, lst=None*)

A list of tilesets.

Allows indexing by name.

Whenever the list is changed, GIDs of tiles in the associated map are renumbered to match the new set of tilesets.

See `NamedElementList` for `TilesetList`'s methods.

modification_context (*args, **kws)

Context manager that “wraps” modifications to the tileset list

While this manager is active, the map's tiles are invalid and should not be touched. After all modification_contexts exit, tiles are renumbered to match the new tileset list. This means that multiple operations on the tileset list can be wrapped in a modification_context for efficiency.

If a used tileset is removed, an exception will be raised whenever the outermost modification_context exits.

4.2.5 The `tmxlib.mapobject` module

MapObject

class `tmxlib.mapobject.MapObject` (layer, pixel_pos, name=None, type=None)

A map object: something that's not placed on the fixed grid

Has several subclasses.

Can be either a “tile object”, which has an associated tile much like a map-tile, or a regular (non-tile) object that has a settable size.

init arguments, which become attributes:

layer

The layer this object is on

pixel_pos

The pixel coordinates

pixel_size

Size of this object, as a (width, height) tuple, in pixels.

Only one of `pixel_size` and `size` may be specified.

size

Size of this object, as a (width, height) tuple, in units of map tiles.

name

Name of the object. A string (or unicode)

type

Type of the object. A string (or unicode). No semantics attached.

Other attributes:

objtype

Type of the object: 'rectangle', 'tile' or 'ellipse'

properties

Dict of string (or unicode) keys & values for custom data

pos

Position of the object in tile coordinates, as a (x, y) float tuple

map

The map associated with this object

Unpacked position attributes:

x

y
pixel_x
pixel_y

Methods:

to_dict (*y=None*)
 Export to a dict compatible with Tiled's JSON plugin

classmethod from_dict (*dct, layer*)
 Import from a dict compatible with Tiled's JSON plugin

RectangleObject

class `tmxlib.mapobject.RectangleObject` (*layer, pixel_pos, size=None, pixel_size=None, name=None, type=None, value=0*)

A rectangle object, either blank (sized) or a tile object

See `MapObject` for inherited members.

Extra init arguments, which become attributes:

pixel_size
 Size of this object, as a (width, height) tuple, in pixels. Must be specified for non-tile objects, and must *not* be specified for tile objects (unless the size matches the tile).
 Similar restrictions apply to setting the property (and `width` & `height`).

size
 Size of this object, as a (width, height) tuple, in units of map tiles.
 Shares setting restrictions with `pixel_size`. Note that the constructor will nly accept one of `size` or `pixel_size`, not both at the same time.

value
 Value of the tile, if it's a tile object.

See `tmxlib.tile.TileLikeObject` for attributes and methods shared with tiles.

EllipseObject

class `tmxlib.mapobject.EllipseObject` (*layer, pixel_pos, size=None, pixel_size=None, name=None, type=None*)

An ellipse object

Extra init arguments, which become attributes:

pixel_size
 Size of this object, as a (width, height) tuple, in pixels. Must be specified for non-tile objects, and must *not* be specified for tile objects (unless the size matches the tile).
 Similar restrictions apply to setting the property (and `width` & `height`).

size
 Size of this object, as a (width, height) tuple, in units of map tiles.
 Shares setting restrictions with `pixel_size`. Note that the constructor will nly accept one of `size` or `pixel_size`, not both at the same time.

Unpacked size attributes:

width
height
pixel_width
pixel_height

PolygonObject

class `tmxlib.mapobject.PolygonObject` (*layer, pixel_pos, size=None, pixel_size=None, name=None, type=None, points=()*)

A polygon object

See [MapObject](#) for inherited members.

Extra init arguments, which become attributes:

points
Size of this object, as a (width, height) tuple, in pixels. Must be specified for non-tile objects, and must *not* be specified for tile objects (unless the size matches the tile).
The format is list of iterables: [(x0, y0), (x1, y1), ..., (xn, yn)]

PolylineObject

class `tmxlib.mapobject.PolylineObject` (*layer, pixel_pos, size=None, pixel_size=None, name=None, type=None, points=()*)

A polygon object

Behaves just like [PolygonObject](#), it's not closed when drawn. Has the same `points` attribute/argument as [PolygonObject](#).

4.2.6 The `tmxlib.terrain` module

Terrain

class `tmxlib.terrain.Terrain` (*name, tile*)

Represents a Tiled terrain

Init arguments, which become attributes:

name
The name of the terrain
tile
The tile that represents the terrain visually. Should be from the same tileset.

TerrainList

class `tmxlib.terrain.TerrainList` (*lst=None*)

append_new (*name, tile*)
Append a newly created Terrain to the list

4.2.7 The tmxlib.image modules

Image loading

The `open()` function provides a high-level interface to reading images.

`tmxlib.image.open(filename, trans=None, size=None)`

Open the given image file

Uses `preferred_image_class`.

Parameters

- **filename** – Name of the file to load the image from
- **trans** – Optional color that should be loaded as transparent

Note: Currently, loading images that use color-key transparency is very inefficient. If possible, use the alpha channel instead.

- **size** – Optional (width, height) tuple. If specified, the file will not be read from disk when the image size needs to be known. If and when the image is loaded, the given size is checked and an exception is raised if it does not match.

Returns

An `Image`

Note that the file is not opened until needed. This makes it possible to use maps and tilesets that refer to nonexistent images.

`tmxlib.image.preferred_image_class`

The type of the object `open()` returns depends on the installed libraries. If `Pillow` (or `PIL`) is installed, the faster `PilImage` is used; otherwise `tmxlib` falls back to `PngImage`, which works anywhere but may be lower and only supports PNG files. Both wrappers offer the same API.

`tmxlib.image.image_classes`

A list of all available image classes, listed by preference. `preferred_image_class` is the first element in this list.

Image base classes

Image Base

class `tmxlib.image_base.ImageBase`

Image base class

This defines the basic image API, shared by `Image` and `ImageRegion`.

Pixels are represented as (r, g, b, a) float tuples, with components in the range of 0 to 1.

`__getitem__(pos)`

Get a pixel or region

With a pair of integers, this returns a pixel via `get_pixel()`:

Parameters `pos` – pair of integers, (x, y)

Returns pixel at (x, y) as a (r, g, b, a) float tuple

With a pair of slices, returns a sub-image:

Parameters `pos` – pair of slices, (left:right, top:bottom)

Returns a `ImageRegion`

Image

class `tmxlib.image_base.Image` (*data=None, trans=None, size=None, source=None*)
An image. Conceptually, an 2D array of pixels.

Note: This is an abstract base class. Use `tmxlib.image.open()` or `tmxlib.image.preferred_image_class` to get a usable subclass.

init arguments that become attributes:

data

Data of this image, as read from disk.

size

Size of the image, in pixels.

If given in constructor, the image doesn't have to be decoded to get this information, somewhat speeding up operations that don't require pixel access.

If it's given in constructor and it does not equal the actual image size, an exception will be raised as soon as the image is decoded.

source

The file name of this image, if it is to be saved separately from maps/tilesets that use it.

trans

A color key used for transparency

Note: Currently, loading images that use color-key transparency is very inefficient. If possible, use the alpha channel instead.

Images support indexing (`img[x, y]`); see `tmxlib.image_base.ImageBase.__getitem__()`

get_pixel (*x, y*)

Get the color of the pixel at position (*x, y*) as a RGBA 4-tuple.

Supports negative indices by wrapping around in the obvious way.

Methods interesting for subclasses:

load_image ()

Load the image from `self.data`, and set `self._size`

If `self._size` is already set, assert that it equals

ImageRegion

class `tmxlib.image_base.ImageRegion` (*parent, top_left, size*)
A rectangular region of a larger image

init arguments that become attributes:

parent

The "parent" image

top_left

The coordinates of the top-left corner of the region. Will also be available as `x` and `y` attributes.

size

The size of the region. Will also available as `width` and `height` attributes.

Except for the constructor and attributes, *ImageRegion* supports the same external API as *Image*:

get_pixel (*x*, *y*)

Get the color of the pixel at position (*x*, *y*) as a RGBA 4-tuple.

Supports negative indices by wrapping around in the obvious way.

4.2.8 The `tmxlib.canvas` and `draw` modules

Provides image drawing/modification capabilities

This module requires `PIL` (or `Pillow`) to be installed.

class `tmxlib.canvas.Canvas` (*size*=(0, 0), *commands*=(), *color*=(0, 0, 0, 0))

A mutable image

Acts as a regular *Image*, except it allows modifications.

Some operations, such as taking an *ImageRegion*, will work on an immutable copy of the canvas.

Parameters **commands** – An iterable of drawing commands to apply on the canvas right after creation

init arguments that become attributes:

size

The size of this Canvas. Will also available as `width` and `height` attributes.

color

The initial color the canvas will have

drawing methods:

draw_image (*image*, *pos*=(0, 0), *opacity*=1)

Paste the given image at the given position

conversion:

to_image ()

Take an immutable copy of this Canvas

Returns an *Image*

internals:

pil_image

The PIL image underlying this Canvas.

Note: Modifying the returned image is not guaranteed to have an effect.

In the future there might be Canvas implementations not based on PIL; their `pil_image` attribute might only give a copy of the data. If PIL is not installed, the attribute won't be present at all.

Drawing commands

Operations on a Canvas may be specified by command objects. This allows one to flexibly filter or modify a stream of drawing operations.

class `tmxlib.draw.DrawCommand`

A draw command

draw (*canvas*)

Apply this operation to the given Canvas

class `tmxlib.draw.DrawImageCommand` (*image, pos=(0, 0), opacity=1*)

Command to draw an image

init arguments that become attributes:

image

The image to draw

pos

Position at which to draw the image. Will also available as `x` and `y` attributes.

4.2.9 The `tmxlib.helpers` module

Exceptions

exception `tmxlib.helpers.UsedTilesetError`

Raised when trying to remove a tileset from a map that uses it

exception `tmxlib.helpers.TilesetNotInMapError`

Used when trying to use a tile from a tileset that's not in the map

NamedElementList

class `tmxlib.helpers.NamedElementList` (*lst=None*)

A list that supports indexing by element name, as a convenience, etc

`lst[some_name]` means the first *element* where `element.name == some_name`. The dict-like `get` method is provided.

Additionally, `NamedElementList` subclasses can use several hooks to control how their elements are stored or what is allowed as elements.

get (*index_or_name, default=None*)

Same as `__getitem__`, but returns default if not found

insert (*index_or_name, value*)

Same as `list.insert`, except indices may be names instead of ints.

insert_after (*index_or_name, value*)

Insert the new value after the position specified by `index_or_name`

For numerical indexes, the same as `insert(index + 1, value)`. Useful when indexing by strings.

move (*index_or_name, amount*)

Move an item by the specified number of indexes

amount can be negative. For example, "move layer down" translates to `layers.move(idx, -1)`

The method will clamp out-of-range amounts, so, for example, `lst.move(0, -1)` will do nothing.

Hooks for subclasses:

modification_context (**args, **kwargs*)

Context in which all modifications take place.

The default implementation nullifies the modifications if an exception is raised.

Note that the manager may nest, in which case the outermost one should be treated as an atomic operation.

retrieved_value (*item*)

Called when an item is being retrieved from the list.

Return the object that will actually be retrieved.

This method must undo any modifications that `stored_value` does.

stored_value (*item*)

Called when an item is being inserted into the list.

Return the object that will actually be stored.

To prevent incompatible items, subclasses may raise an exception here.

This method must undo any modifications that `retrieved_value` does.

Internal helpers and mixins

Dict conversion helpers

`tmxlib.helpers.from_dict_method` (*func*)

Decorator for `from_dict` classmethods

Takes a copy of the second argument (`dct`), and makes sure it is empty at the end.

`tmxlib.helpers.assert_item` (*dct, key, expected_value*)

Asserts that `dct[key] == expected_value`

Helpers for tuple properties

`tmxlib.helpers.unpacked_properties` (*full_prop_name, count=2*)

Return properties that “unpack” a tuple property

For example, if a class defines:

```
x, y = unpacked_properties('pos')
```

then `obj.x` will return the same as `obj.pos[0]`, and setting `obj.x` will update `obj.pos` to (new x, old y).

class `tmxlib.helpers.SizeMixin`

width

Getter/setter for `self.size[0]`

height

Getter/setter for `self.size[1]`

_wrap_coords (*x, y*)

Normalize coordinates for indexing/slicing

Positive values are unchanged, negative ones wrap around using `self.width & self.height`

Other mixins

class `tmxlib.helpers.LayerElementMixin`
Provides a *map* attribute extracted from the object's *layer*.

class `tmxlib.helpers.TileMixin`
Provides *size* based on *pixel_size* and the map

width
Getter/setter for `self.size[0]`

height
Getter/setter for `self.size[1]`

tile_width
Getter/setter for `self.tile_size[0]`

tile_height
Getter/setter for `self.tile_size[1]`

pixel_x
Getter/setter for `self.pixel_pos[0]`

pixel_y
Getter/setter for `self.pixel_pos[1]`

x
Getter/setter for `self.pos[0]`

y
Getter/setter for `self.pos[1]`

Helpers

class `tmxlib.helpers.Property`
Trivial subclass of the *property* builtin. Allows custom attributes.

4.2.10 Extra members of tmxlib classes

To avoid clutter, some members aren't mentioned in their respective classes' documentation. This page documents such members, so that they can be linked.

(And also to make the doc coverage tool happy.)

class `tmxlib.layer.TileLayer`

Layer methods

`TileLayer.to_dict()`
Export to a dict compatible with Tiled's JSON plugin

classmethod `TileLayer.from_dict(dct, *args, **kwargs)`
Import from a dict compatible with Tiled's JSON plugin

class `tmxlib.layer.ObjectLayer`

NamedList methods

`ObjectLayer.__getitem__(index_or_name)`
Same as list's, except non-slice indices may be names.

`ObjectLayer.__setitem__(index_or_name, value)`
 Same as list's, but non-slice indices may be names instead of ints.

`ObjectLayer.__contains__(item_or_name)`
item_or_name in *self*

NamedElementLists can be queried either by name or by item.

`ObjectLayer.count(value) → integer` – return number of occurrences of value

`ObjectLayer.append(value)`
S.append(object) – append object to the end of the sequence

`ObjectLayer.extend(values)`
S.extend(iterable) – extend sequence by appending elements from the iterable

`ObjectLayer.pop([index]) → item` – remove and return item at index (default last).
 Raise `IndexError` if list is empty or index is out of range.

`ObjectLayer.remove(value)`
S.remove(value) – remove first occurrence of value. Raise `ValueError` if the value is not present.

`ObjectLayer.reverse()`
S.reverse() – reverse *IN PLACE*

`ObjectLayer.insert(index_or_name, value)`
 Same as `list.insert`, except indices may be names instead of ints.

`ObjectLayer.insert_after(index_or_name, value)`
 Insert the new value after the position specified by *index_or_name*

For numerical indexes, the same as `insert(index + 1, value)`. Useful when indexing by strings.

`ObjectLayer.move(index_or_name, amount)`
 Move an item by the specified number of indexes

amount can be negative. For example, “move layer down” translates to `layers.move(idx, -1)`

The method will clamp out-of-range amounts, so, for example, `lst.move(0, -1)` will do nothing.

`ObjectLayer.retrieved_value(item)`
 Called when an item is being retrieved from the list.

Return the object that will actually be retrieved.

This method must undo any modifications that `stored_value` does.

`ObjectLayer.stored_value(item)`

Layer methods

`ObjectLayer.to_dict()`
 Export to a dict compatible with Tiled's JSON plugin

classmethod `ObjectLayer.from_dict(dct, *args, **kwargs)`
 Import from a dict compatible with Tiled's JSON plugin

class `tmxlib.layer.ImageLayer`

Layer methods

`ImageLayer.to_dict()`
Export to a dict compatible with Tiled's JSON plugin

classmethod `ImageLayer.from_dict(dct, *args, **kwargs)`
Import from a dict compatible with Tiled's JSON plugin

`class tmxlib.layer.LayerList`

NamedList methods

`LayerList.__getitem__(index_or_name)`
Same as list's, except non-slice indices may be names.

`LayerList.__setitem__(index_or_name, value)`
Same as list's, but non-slice indices may be names instead of ints.

`LayerList.__contains__(item_or_name)`
item_or_name in *self*

NamedElementLists can be queried either by name or by item.

`LayerList.index(value) → integer` – return first index of value.
Raises `ValueError` if the value is not present.

`LayerList.count(value) → integer` – return number of occurrences of value

`LayerList.append(value)`
`S.append(object)` – append object to the end of the sequence

`LayerList.extend(values)`
`S.extend(iterable)` – extend sequence by appending elements from the iterable

`LayerList.pop([index]) → item` – remove and return item at index (default last).
Raise `IndexError` if list is empty or index is out of range.

`LayerList.remove(value)`
`S.remove(value)` – remove first occurrence of value. Raise `ValueError` if the value is not present.

`LayerList.reverse()`
`S.reverse()` – reverse *IN PLACE*

`LayerList.insert(index_or_name, value)`
Same as `list.insert`, except indices may be names instead of ints.

`LayerList.insert_after(index_or_name, value)`
Insert the new value after the position specified by *index_or_name*

For numerical indexes, the same as `insert(index + 1, value)`. Useful when indexing by strings.

`LayerList.move(index_or_name, amount)`
Move an item by the specified number of indexes

amount can be negative. For example, “move layer down” translates to `layers.move(idx, -1)`

The method will clamp out-of-range amounts, so, for example, `lst.move(0, -1)` will do nothing.

`LayerList.modification_context(*args, **kws)`
Context in which all modifications take place.

The default implementation nullifies the modifications if an exception is raised.

Note that the manager may nest, in which case the outermost one should be treated as an atomic operation.

`LayerList.retrieved_value(item)`

Called when an item is being retrieved from the list.

Return the object that will actually be retrieved.

This method must undo any modifications that `stored_value` does.

`LayerList.stored_value(layer)`

Prevent layers that aren't from this map.

class `tmxlib.tileset.TilesetList`

NamedList methods

`TilesetList.__getitem__(index_or_name)`

Same as list's, except non-slice indices may be names.

`TilesetList.__setitem__(index_or_name, value)`

Same as list's, but non-slice indices may be names instead of ints.

`TilesetList.__contains__(item_or_name)`

item_or_name in *self*

NamedElementLists can be queried either by name or by item.

`TilesetList.index(value) → integer` – return first index of value.

Raises `ValueError` if the value is not present.

`TilesetList.count(value) → integer` – return number of occurrences of value

`TilesetList.append(value)`

`S.append(object)` – append object to the end of the sequence

`TilesetList.extend(values)`

`S.extend(iterable)` – extend sequence by appending elements from the iterable

`TilesetList.pop([index]) → item` – remove and return item at index (default last).

Raise `IndexError` if list is empty or index is out of range.

`TilesetList.remove(value)`

`S.remove(value)` – remove first occurrence of value. Raise `ValueError` if the value is not present.

`TilesetList.reverse()`

`S.reverse()` – reverse *IN PLACE*

`TilesetList.insert(index_or_name, value)`

Same as `list.insert`, except indices may be names instead of ints.

`TilesetList.insert_after(index_or_name, value)`

Insert the new value after the position specified by `index_or_name`

For numerical indexes, the same as `insert(index + 1, value)`. Useful when indexing by strings.

`TilesetList.move(index_or_name, amount)`

Move an item by the specified number of indexes

amount can be negative. For example, “move layer down” translates to `layers.move(idx, -1)`

The method will clamp out-of range amounts, so, for example, `lst.move(0, -1)` will do nothing.

`TilesetList.retrieved_value(item)`

Called when an item is being retrieved from the list.

Return the object that will actually be retrieved.

This method must undo any modifications that `stored_value` does.

`TilesetList.stored_value(item)`

Called when an item is being inserted into the list.

Return the object that will actually be stored.

To prevent incompatible items, subclasses may raise an exception here.

This method must undo any modifications that `retrieved_value` does.

class `tmxlib.tileset.ImageTileset`

Load/save methods (see `tmxlib.fileio.ReadWriteBase`):

classmethod `ImageTileset.open(filename, shared=False)`

Load an object of this class from a file

Parameters

- **filename** – The file from which to load
- **shared** – Objects loaded from a single file with `shared=True` will be reused. Modifications to this shared object will, naturally, be visible from all variables that reference it. (External tilesets are loaded as *shared* by default.)

classmethod `ImageTileset.load(string)`

Load an object of this class from a string.

Parameters **string** – String containing the XML description of the object, as it would be read from a file.

`ImageTileset.save(filename)`

Save this object to a file

Parameters **filename** – Name of the file to save to.

`ImageTileset.dump(string)`

Save this object as a string

Returns String with the representation of the object, suitable for writing to a file.

`ImageTileset.to_dict(**kwargs)`

Export to a dict compatible with Tiled's JSON plugin

classmethod `ImageTileset.from_dict(dct, *args, **kwargs)`

Import from a dict compatible with Tiled's JSON plugin

Overridden methods (see `tmxlib.tileset.Tileset`):

`ImageTileset.tile_image(number)`

Return the image used by the given tile

GID calculation methods (see `tmxlib.tileset.Tileset`):

Note: `TilesetList` depends on the specific GID calculation algorithm provided by these methods to renumber a map's tiles when tilesets are moved around. Don't override these unless your subclass is not used with vanilla `TilesetList`s.

`ImageTileset.first_gid(map)`

Return the first gid used by this tileset in the given map

ImageTileset.**end_gid**(*map*)

Return the first gid after this tileset in the given map

class `tmxlib.mapobject.RectangleObject`

MapObject methods

RectangleObject.**to_dict**()

classmethod RectangleObject.**from_dict**(*dct*, **args*, ***kwargs*)

class `tmxlib.mapobject.EllipseObject`

MapObject methods

EllipseObject.**to_dict**()

classmethod EllipseObject.**from_dict**(*dct*, **args*, ***kwargs*)

class `tmxlib.draw.DrawImageCommand`

DrawCommand methods

DrawImageCommand.**draw**(*canvas*)

Indices and tables

- *genindex*
- *modindex*
- *search*

t

- `tmxlib`, [11](#)
- `tmxlib.canvas`, [27](#)
- `tmxlib.draw`, [27](#)
- `tmxlib.helpers`, [28](#)
- `tmxlib.image`, [25](#)
- `tmxlib.image_base`, [25](#)
- `tmxlib.layer`, [14](#)
- `tmxlib.terrain`, [24](#)
- `tmxlib.tile`, [16](#)
- `tmxlib.tileset`, [18](#)

Symbols

__contains__() (tmxlib.layer.LayerList method), 32
 __contains__() (tmxlib.layer.ObjectLayer method), 31
 __contains__() (tmxlib.tileset.TilesetList method), 33
 __getitem__() (tmxlib.image_base.ImageBase method), 25
 __getitem__() (tmxlib.layer.LayerList method), 32
 __getitem__() (tmxlib.layer.ObjectLayer method), 30
 __getitem__() (tmxlib.layer.TileLayer method), 15
 __getitem__() (tmxlib.tileset.Tileset method), 19
 __getitem__() (tmxlib.tileset.TilesetList method), 33
 __iter__() (tmxlib.tileset.Tileset method), 20
 __len__() (tmxlib.tileset.Tileset method), 19
 __setitem__() (tmxlib.layer.LayerList method), 32
 __setitem__() (tmxlib.layer.ObjectLayer method), 30
 __setitem__() (tmxlib.layer.TileLayer method), 15
 __setitem__() (tmxlib.tileset.TilesetList method), 33
 __wrap_coords__() (tmxlib.helpers.SizeMixin method), 29

A

add_image_layer() (tmxlib.map.Map method), 13
 add_layer() (tmxlib.map.Map method), 13
 add_object_layer() (tmxlib.map.Map method), 13
 add_tile_layer() (tmxlib.map.Map method), 13
 all_objects() (tmxlib.layer.Layer method), 14
 all_objects() (tmxlib.layer.ObjectLayer method), 16
 all_objects() (tmxlib.layer.TileLayer method), 15
 all_objects() (tmxlib.map.Map method), 13
 all_tiles() (tmxlib.layer.Layer method), 14
 all_tiles() (tmxlib.layer.ObjectLayer method), 16
 all_tiles() (tmxlib.layer.TileLayer method), 15
 all_tiles() (tmxlib.map.Map method), 13
 append() (tmxlib.layer.LayerList method), 32
 append() (tmxlib.layer.ObjectLayer method), 31
 append() (tmxlib.tileset.TilesetList method), 33
 append_new() (tmxlib.terrain.TerrainList method), 24
 assert_item() (in module tmxlib.helpers), 29

B

background_color (Map attribute), 12

C

Canvas (class in tmxlib.canvas), 27
 Canvas.pil_image (in module tmxlib.canvas), 27
 check_consistency() (tmxlib.map.Map method), 13
 color (tmxlib.canvas.Canvas attribute), 27
 color (tmxlib.layer.ObjectLayer attribute), 15
 column_count (tmxlib.tileset.ImageTileset attribute), 20
 count() (tmxlib.layer.LayerList method), 32
 count() (tmxlib.layer.ObjectLayer method), 31
 count() (tmxlib.tileset.TilesetList method), 33

D

data (tmxlib.image_base.Image attribute), 26
 data (tmxlib.layer.TileLayer attribute), 15
 draw() (tmxlib.draw.DrawCommand method), 28
 draw() (tmxlib.draw.DrawImageCommand method), 35
 draw_image() (tmxlib.canvas.Canvas method), 27
 DrawCommand (class in tmxlib.draw), 27
 DrawImageCommand (class in tmxlib.draw), 28
 dump() (Map method), 13
 dump() (tmxlib.tileset.ImageTileset method), 34
 dump() (tmxlib.tileset.Tileset method), 19

E

EllipseObject (class in tmxlib.mapobject), 23
 end_gid (Map attribute), 12
 end_gid() (tmxlib.tileset.ImageTileset method), 34
 end_gid() (tmxlib.tileset.Tileset method), 20
 extend() (tmxlib.layer.LayerList method), 32
 extend() (tmxlib.layer.ObjectLayer method), 31
 extend() (tmxlib.tileset.TilesetList method), 33

F

first_gid() (tmxlib.tileset.ImageTileset method), 34
 first_gid() (tmxlib.tileset.Tileset method), 20
 flipped_diagonally (tmxlib.tile.TileLikeObject attribute), 17
 flipped_horizontally (tmxlib.tile.TileLikeObject attribute), 17

flipped_vertically (tmxlib.tile.TileLikeObject attribute), 17
from_dict() (tmxlib.ImageLayer class method), 32
from_dict() (tmxlib.layer.ImageLayer class method), 16
from_dict() (tmxlib.layer.Layer class method), 14
from_dict() (tmxlib.layer.ObjectLayer class method), 16, 31
from_dict() (tmxlib.layer.TileLayer class method), 15
from_dict() (tmxlib.map.Map class method), 13
from_dict() (tmxlib.mapobject.EllipseObject class method), 35
from_dict() (tmxlib.mapobject.MapObject class method), 23
from_dict() (tmxlib.mapobject.RectangleObject class method), 35
from_dict() (tmxlib.TileLayer class method), 30
from_dict() (tmxlib.tileset.ImageTileset class method), 34
from_dict() (tmxlib.tileset.Tileset class method), 19
from_dict_method() (in module tmxlib.helpers), 29

G

get() (tmxlib.helpers.NamedElementList method), 28
get_pixel() (tmxlib.image_base.Image method), 26
get_pixel() (tmxlib.image_base.ImageRegion method), 27
get_pixel() (tmxlib.tile.TileLikeObject method), 17
get_pixel() (tmxlib.tileset.TilesetTile method), 21
get_tiles() (tmxlib.map.Map method), 13
gid (tmxlib.tile.TileLikeObject attribute), 17
gid() (tmxlib.tileset.TilesetTile method), 21

H

height (EllipseObject attribute), 24
height (Map attribute), 12
height (tmxlib.helpers.SizeMixin attribute), 29
height (tmxlib.helpers.TileMixin attribute), 30
hflip() (tmxlib.tile.TileLikeObject method), 18

I

Image (class in tmxlib.image_base), 26
image (tmxlib.draw.DrawImageCommand attribute), 28
image (tmxlib.layer.ImageLayer attribute), 16
image (tmxlib.tile.TileLikeObject attribute), 17
image (tmxlib.tileset.ImageTileset attribute), 20
image (tmxlib.tileset.TilesetTile attribute), 21
image_classes (tmxlib.image.tmxlib.image attribute), 25
ImageBase (class in tmxlib.image_base), 25
ImageLayer (class in tmxlib.layer), 16
ImageRegion (class in tmxlib.image_base), 26
ImageTileset (class in tmxlib.tileset), 20
index (tmxlib.layer.Layer attribute), 14
index() (tmxlib.layer.LayerList method), 32
index() (tmxlib.tileset.TilesetList method), 33
insert() (tmxlib.helpers.NamedElementList method), 28

insert() (tmxlib.layer.LayerList method), 32
insert() (tmxlib.layer.ObjectLayer method), 31
insert() (tmxlib.tileset.TilesetList method), 33
insert_after() (tmxlib.helpers.NamedElementList method), 28
insert_after() (tmxlib.layer.LayerList method), 32
insert_after() (tmxlib.layer.ObjectLayer method), 31
insert_after() (tmxlib.tileset.TilesetList method), 33

L

Layer (class in tmxlib.layer), 14
layer (MapObject attribute), 22
layer (tmxlib.tile.MapTile attribute), 18
LayerElementMixin (class in tmxlib.helpers), 30
LayerList (class in tmxlib.layer), 16
layers (Map attribute), 12
load() (Map class method), 13
load() (tmxlib.tileset.ImageTileset class method), 34
load() (tmxlib.tileset.Tileset class method), 19
load_image() (tmxlib.image_base.Image method), 26

M

Map (class in tmxlib.map), 12
map (MapObject attribute), 22
map (tmxlib.layer.Layer attribute), 14
map (tmxlib.tile.TileLikeObject attribute), 18
MapObject (class in tmxlib.mapobject), 22
MapTile (class in tmxlib.tile), 18
margin (tmxlib.tileset.ImageTileset attribute), 20
modification_context() (tmxlib.helpers.NamedElementList method), 28
modification_context() (tmxlib.layer.LayerList method), 32
modification_context() (tmxlib.tileset.TilesetList method), 22
move() (tmxlib.helpers.NamedElementList method), 28
move() (tmxlib.layer.LayerList method), 32
move() (tmxlib.layer.ObjectLayer method), 31
move() (tmxlib.tileset.TilesetList method), 33

N

name (MapObject attribute), 22
name (tmxlib.layer.Layer attribute), 14
name (tmxlib.terrain.Terrain attribute), 24
name (tmxlib.tileset.ImageTileset attribute), 20
name (tmxlib.tileset.Tileset attribute), 19
NamedElementList (class in tmxlib.helpers), 28
number (tmxlib.tile.TileLikeObject attribute), 17
number (tmxlib.tileset.TilesetTile attribute), 21

O

ObjectLayer (class in tmxlib.layer), 15
objtype (MapObject attribute), 22

opacity (tmxlib.layer.Layer attribute), 14
 open() (in module tmxlib.image), 25
 open() (Map class method), 13
 open() (tmxlib.tileset.ImageTileset class method), 34
 open() (tmxlib.tileset.Tileset class method), 19
 orientation (Map attribute), 12

P

parent (tmxlib.image_base.ImageRegion attribute), 26
 pixel_height (EllipseObject attribute), 24
 pixel_height (Map attribute), 12
 pixel_pos (MapObject attribute), 22
 pixel_size (EllipseObject attribute), 23
 pixel_size (Map attribute), 12
 pixel_size (MapObject attribute), 22
 pixel_size (RectangleObject attribute), 23
 pixel_size (tmxlib.tileset.TilesetTile attribute), 21
 pixel_width (EllipseObject attribute), 24
 pixel_width (Map attribute), 13
 pixel_x (MapObject attribute), 23
 pixel_x (tmxlib.helpers.TileMixin attribute), 30
 pixel_y (MapObject attribute), 23
 pixel_y (tmxlib.helpers.TileMixin attribute), 30
 points (PolygonObject attribute), 24
 PolygonObject (class in tmxlib.mapobject), 24
 PolylineObject (class in tmxlib.mapobject), 24
 pop() (tmxlib.layer.LayerList method), 32
 pop() (tmxlib.layer.ObjectLayer method), 31
 pop() (tmxlib.tileset.TilesetList method), 33
 pos (MapObject attribute), 22
 pos (tmxlib.draw.DrawImageCommand attribute), 28
 pos (tmxlib.tile.MapTile attribute), 18
 preferred_image_class (tmxlib.image.tmxlib.image attribute), 25
 probability (tmxlib.tileset.TilesetTile attribute), 21
 properties (Map attribute), 12
 properties (MapObject attribute), 22
 properties (tmxlib.layer.Layer attribute), 14
 properties (tmxlib.tile.MapTile attribute), 18
 properties (tmxlib.tileset.Tileset attribute), 19
 properties (tmxlib.tileset.TilesetTile attribute), 21
 Property (class in tmxlib.helpers), 30

R

RectangleObject (class in tmxlib.mapobject), 23
 remove() (tmxlib.layer.LayerList method), 32
 remove() (tmxlib.layer.ObjectLayer method), 31
 remove() (tmxlib.tileset.TilesetList method), 33
 retrieved_value() (tmxlib.helpers.NamedElementList method), 29
 retrieved_value() (tmxlib.layer.LayerList method), 33
 retrieved_value() (tmxlib.layer.ObjectLayer method), 31
 retrieved_value() (tmxlib.tileset.TilesetList method), 34
 reverse() (tmxlib.layer.LayerList method), 32

reverse() (tmxlib.layer.ObjectLayer method), 31
 reverse() (tmxlib.tileset.TilesetList method), 33
 rotate() (tmxlib.tile.TileLikeObject method), 18
 row_count (tmxlib.tileset.ImageTileset attribute), 20

S

save() (Map method), 13
 save() (tmxlib.tileset.ImageTileset method), 34
 save() (tmxlib.tileset.Tileset method), 19
 set_value_at() (tmxlib.layer.TileLayer method), 15
 size (EllipseObject attribute), 23
 size (Map attribute), 12
 size (MapObject attribute), 22
 size (RectangleObject attribute), 23
 size (tmxlib.canvas.Canvas attribute), 27
 size (tmxlib.image_base.Image attribute), 26
 size (tmxlib.image_base.ImageRegion attribute), 26
 size (tmxlib.tile.TileLikeObject attribute), 18
 SizeMixin (class in tmxlib.helpers), 29
 source (tmxlib.image_base.Image attribute), 26
 source (tmxlib.tileset.ImageTileset attribute), 20
 source (tmxlib.tileset.Tileset attribute), 19
 spacing (tmxlib.tileset.ImageTileset attribute), 20
 stored_value() (tmxlib.helpers.NamedElementList method), 29
 stored_value() (tmxlib.layer.LayerList method), 33
 stored_value() (tmxlib.layer.ObjectLayer method), 31
 stored_value() (tmxlib.tileset.TilesetList method), 34

T

Terrain (class in tmxlib.terrain), 24
 terrain_indices (tmxlib.tileset.TilesetTile attribute), 21
 TerrainList (class in tmxlib.terrain), 24
 terrains (tmxlib.tileset.Tileset attribute), 19
 terrains (tmxlib.tileset.TilesetTile attribute), 21
 tile (tmxlib.terrain.Terrain attribute), 24
 tile_height (Map attribute), 12
 tile_height (tmxlib.helpers.TileMixin attribute), 30
 tile_height (tmxlib.tileset.Tileset attribute), 19
 tile_image() (tmxlib.ImageTileset method), 34
 tile_image() (tmxlib.tileset.ImageTileset method), 21
 tile_image() (tmxlib.tileset.Tileset method), 20
 tile_offset (tmxlib.tileset.Tileset attribute), 19
 tile_offset_x (tmxlib.tileset.Tileset attribute), 19
 tile_offset_y (tmxlib.tileset.Tileset attribute), 19
 tile_size (Map attribute), 12
 tile_size (tmxlib.tileset.ImageTileset attribute), 20
 tile_to_image_coordinates() (tmxlib.tile.TileLikeObject method), 17
 tile_width (Map attribute), 12
 tile_width (tmxlib.helpers.TileMixin attribute), 30
 tile_width (tmxlib.tileset.Tileset attribute), 19
 TileLayer (class in tmxlib.layer), 14
 TileLikeObject (class in tmxlib.tile), 16

TileMixin (class in tmxlib.helpers), 30
Tilset (class in tmxlib.tilset), 18
tilset (tmxlib.tile.TileLikeObject attribute), 17
tilset (tmxlib.tilset.TilsetTile attribute), 21
tilset_tile (tmxlib.tile.TileLikeObject attribute), 17
TilsetList (class in tmxlib.tilset), 21
TilsetNotInMapError, 28
tilsets (Map attribute), 12
TilsetTile (class in tmxlib.tilset), 21
tmxlib (module), 11
tmxlib.canvas (module), 27
tmxlib.draw (module), 27
tmxlib.helpers (module), 28
tmxlib.image (module), 25
tmxlib.image_base (module), 25
tmxlib.layer (module), 14
tmxlib.terrain (module), 24
tmxlib.tile (module), 16
tmxlib.tilset (module), 18
to_dict() (tmxlib.ImageLayer method), 32
to_dict() (tmxlib.layer.ImageLayer method), 16
to_dict() (tmxlib.layer.Layer method), 14
to_dict() (tmxlib.layer.ObjectLayer method), 16, 31
to_dict() (tmxlib.layer.TileLayer method), 15
to_dict() (tmxlib.map.Map method), 13
to_dict() (tmxlib.mapobject.EllipseObject method), 35
to_dict() (tmxlib.mapobject.MapObject method), 23
to_dict() (tmxlib.mapobject.RectangleObject method), 35
to_dict() (tmxlib.TileLayer method), 30
to_dict() (tmxlib.tilset.ImageTilset method), 34
to_dict() (tmxlib.tilset.Tilset method), 19
to_image() (tmxlib.canvas.Canvas method), 27
top_left (tmxlib.image_base.ImageRegion attribute), 26
trans (tmxlib.image_base.Image attribute), 26
type (MapObject attribute), 22
type (tmxlib.layer.Layer attribute), 14

U

unpacked_properties() (in module tmxlib.helpers), 29
UsedTilsetError, 28

V

value (RectangleObject attribute), 23
value (tmxlib.tile.TileLikeObject attribute), 17
value_at() (tmxlib.layer.TileLayer method), 15
vflip() (tmxlib.tile.TileLikeObject method), 18
visible (tmxlib.layer.Layer attribute), 14

W

width (EllipseObject attribute), 24
width (Map attribute), 12
width (tmxlib.helpers.SizeMixin attribute), 29
width (tmxlib.helpers.TileMixin attribute), 30

X

x (MapObject attribute), 22
x (tmxlib.helpers.TileMixin attribute), 30

Y

y (MapObject attribute), 22
y (tmxlib.helpers.TileMixin attribute), 30