

LevelSVG Manual

Installing Inkscape

Warning: The SVG parser **ONLY** supports SVG files generated by Inkscape. You should NOT use any other SVG editor to generate the SVG files.

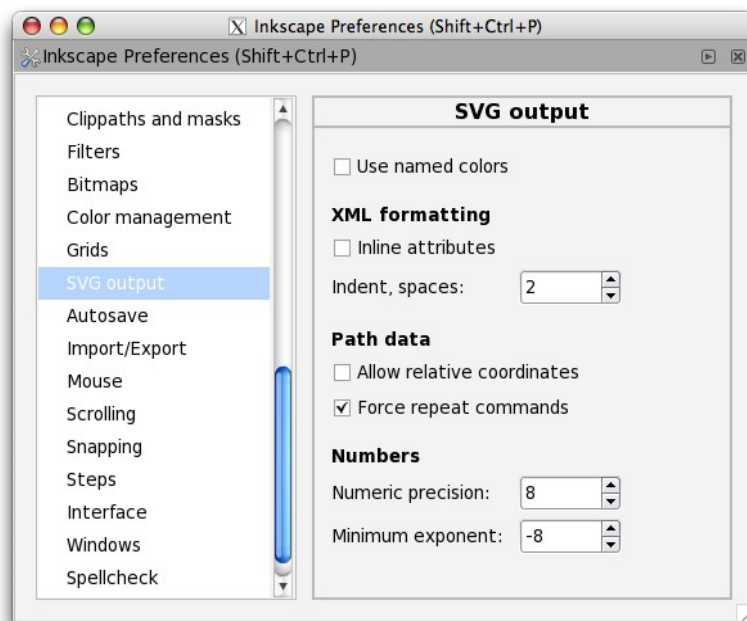
Download:

- <https://sourceforge.net/projects/inkscape/files/inkscape/0.47pre3/>

Version 0.46 has some incompatibilities with the fontcache in OS/X, so try to use the 0.47 version.

Preferences:

1. Open Inkscape
2. Open Inkscape preferences (File → Inkscape Preferences)
3. Set the *SVG output* preferences
4. Path data
 5. Allow relative coordinates **MUST** be DISABLED
 6. Force repeat commands **SHOULD** be ENABLED for debugging purposes



Using Inkscape

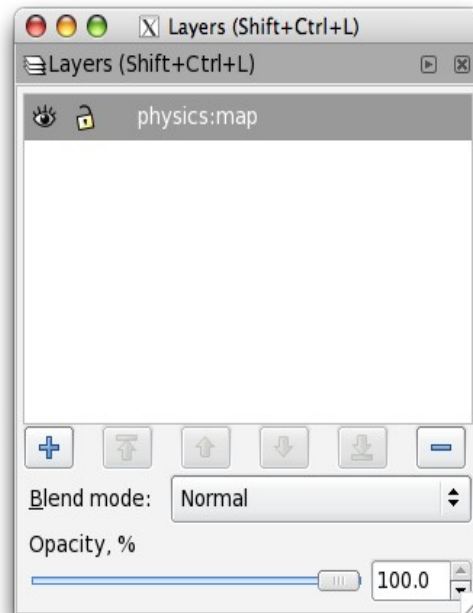
Layers

In your SVG file you can have as many layers as you want, but only the layers that begins with the

physics: name will be parsed.

Example of layers:

- physics:static (notice that “static” is a helper name)
- physics:dynamic (“dynamic” is also a helper name)
- physics:enemies (“enemies”, another helper name)
- map (this layer won't be parsed)



Supported SVG objects



Supported Inkscape objects:

- *Rectangles*: It will create a solid box2d rectangle (1 fixture). They support: *restitution*, *density*, *friction* and *isSensor*. Take into account that *rounded* boxes are not supported.
- *3D Boxes*: They are supported, but they are not very useful for 2d games.
- *Ellipses*: It will create a solid box2d circle (1 fixture). If the X-radius is different from the Y-

radius, it will create a circle of radius: $(rx+ry)/2$. They support: *restitution*, *density*, *friction* and *isSensor*:

- *Stars*: It will create a body with multiple edges (multiple fixtures). Since these objects are not “solid” objects, *density* is not supported.
- *Spirals*: It will create a body with multiple edges (multiple fixtures). Since these objects are not “solid” objects, *density* is not supported.
- *Pencil*: Limited support. **Warning**: Only useful to create straight lines. If it is used as a 'freehand' pencil, it will create objects that will make box2d crash.
- *Pen*: It will create a body with multiple edges (multiple fixtures). Since these objects are not “solid” objects, *density* is not supported.
- *Calligraphy*: Limited support. **Warning**: Only works with very simple objects, otherwise it will create objects that will make box2d crash.
- *Text*: Not supported

Additional features:

- Each of the supported objects can be transformed (translated, rotated and/or scaled)
- Each of the supported objects can be grouped with other supported objects.
- Each of the supported objects can be joined (union) with other supported objects, thus, creating just one box2d object.

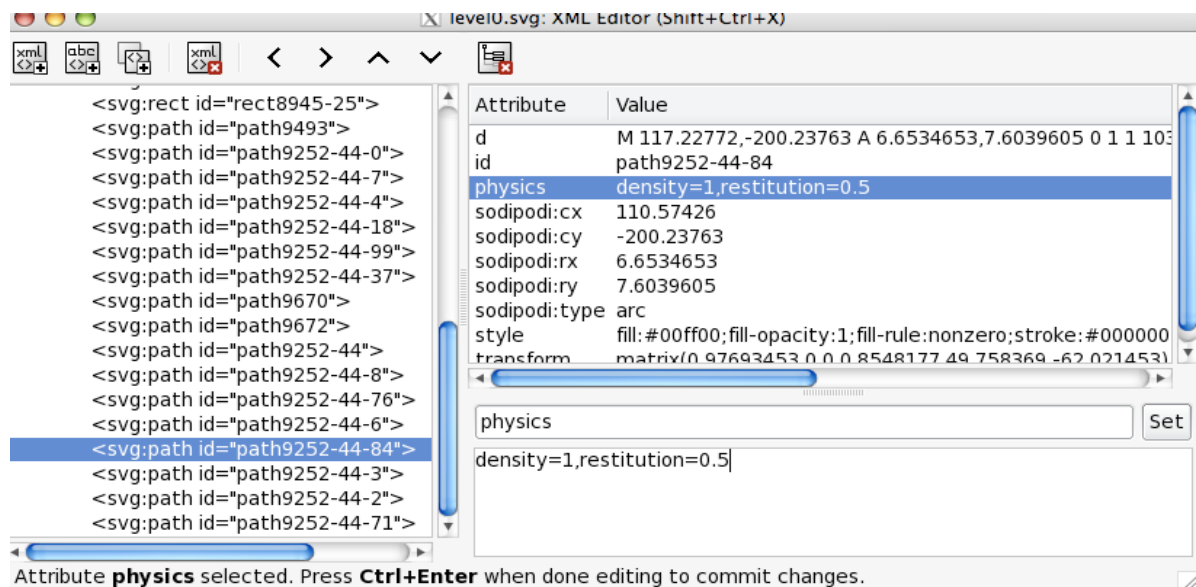
Integration with physics

For each SVG object a `b2Body` will be created. You can't create compound shapes in Inkscape, but you can do it from the “game_data” callback. See: *Integration with your game*.

Supported physics properties in the created SVG objects:

- *restitution*: float number. The restitution (elasticity) usually in the range [0,1].
- *density*: float number. If density is not present or if density is 0, the body will be treated as an static body. Otherwise it will be treated as a dynamic body. Usually in kg/m^2 .
- *friction*: float number. The friction coefficient, usually in the range [0,1].
- *isSensor*: expects a boolean (0 or 1). Only supported on “solid” objects (circles and rectangles)

To add any of these properties, you need to open the XML editor (Shift + Control + X).



You have to add the `physics` attribute to the elements that you want to edit.

The value is list of key-values separated with comas. Example of possible values:

`density=0.3, friction=0.5, restitution=0.9`

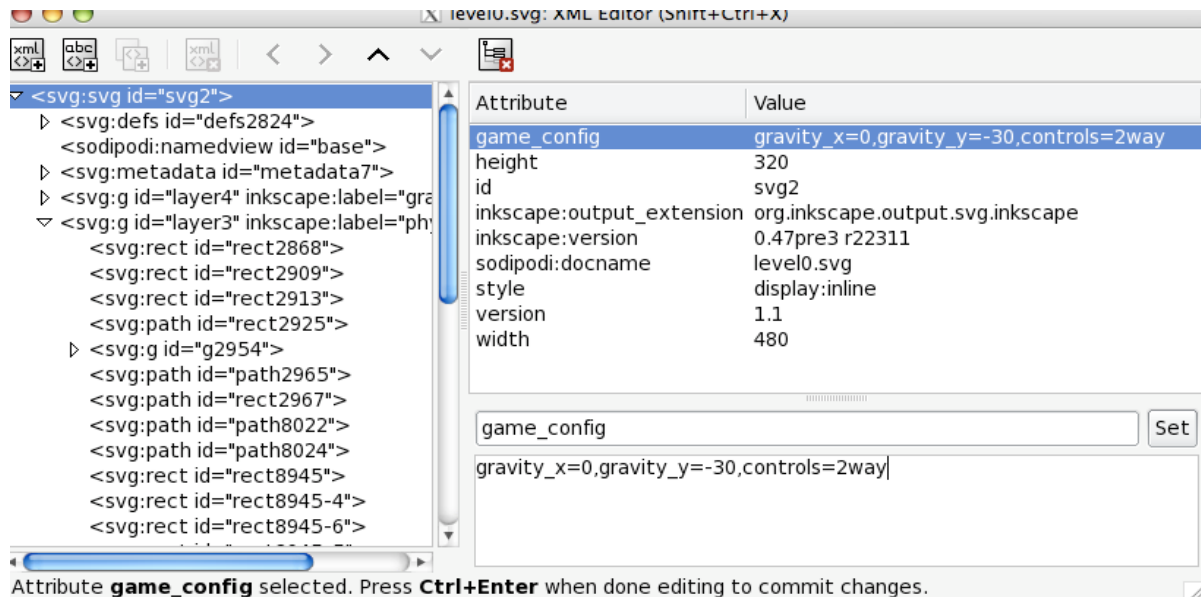
The order is not important.

Level configuration

You can also configure some level variables like:

- *gravity_x*: a *float* for the horizontal gravity (default is 0)
- *gravity_y*: a *float* for the vertical gravity (default is -10)
- *controls*:
 - *2way*: Supports only horizontal movement of the hero
 - *4way*: Supports both horizontal and vertical movement of the hero (default)

To do so, you need to edit the 1st SVG element, in particular you need to edit the `<svg:svg>` element, and you need to add the `game_config` attribute to it:



Integration with your game

You can also link elements (box2d bodies) with cocos2d objects using the `game_data` attribute.

At present, the only supported key is `object` but you can extend it easily to match your needs

Supported objects:

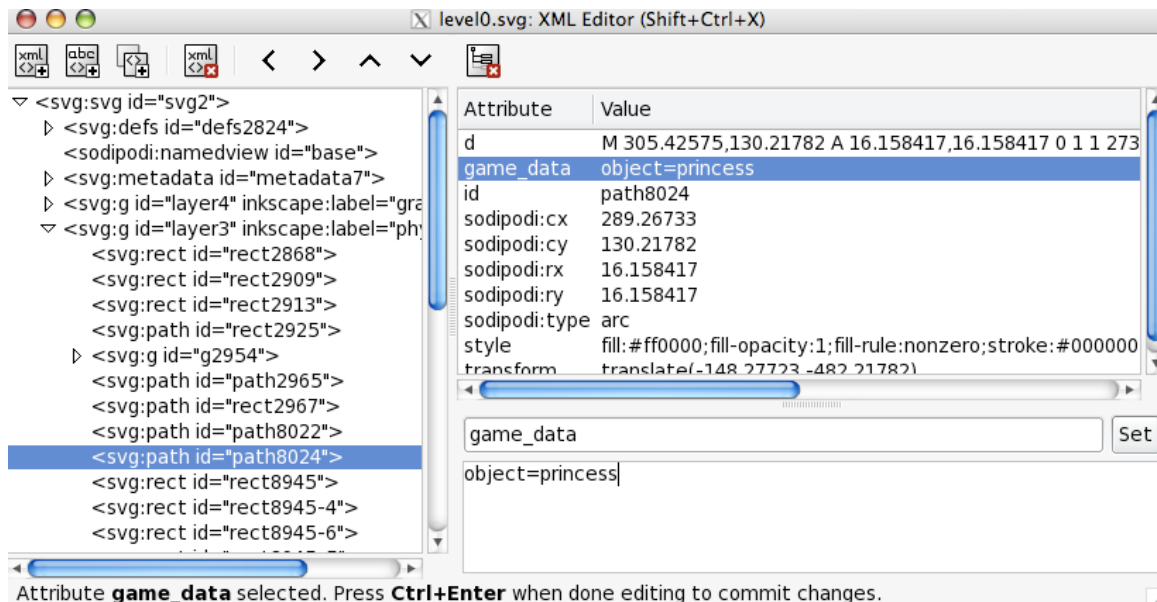
- `object=hero`: The sprite that will be controlled by gamer. Only one “hero” can be created per level.
- `object=princess`: The goal. The 'hero' needs to touch this sprite in order to win the level. There can be as many “princess” as you want.
- `object=enemy`: If the 'hero' touches any of these objects, he will die.
- `object=hole`: Another kind of enemy. If the 'hero' touches any of these objects, he will die.
- `object=platform1`: A One sided platform, AKA Super Mario platforms. In order to jump, the “hero” must be on top of these objects.
- `object=ground`: Standard “ground”. In order to jump, the “hero” must be on top of these objects.
- `object=spinner`: An 'spinner' object. It is recommended that you create a SVG circle element to define the *spinner*, since the *spinner* will be as big as the circle. The *spinner* is a complex object. It consists of: 1 dynamic body with 2 shapes, 1 static body and 1 revolute joint.

You can easily add you own custom objects by editing the `GameSceneObjects.mm` file. In order to add your own callback, just add a `createObjectName` where **Name** is the name of your object.

Example, when `object=enemy` is detected, the `createObjectEnemy` will be executed (`GameSceneObject.mm`).

I recommend to code complex physics objects using the following steps:

- Create complex box2d objects in [Mekanim](#) or any other complex box2d editor. Coding complex box2d objects *by hand* (writing all the code) is also possible.
- In Inkscape create a simple object with more or less the dimensions of the complex object
- Add the *game_data* attribute to that object
- And on the *game_data* callback update it with the complex object.



For example, take a look at the 'spinner' object. It is a complex object that was coded by 'hand' but in Inkscape we define the position and the size of it.

Touching the objects

All physics objects can be dragged if the *isTouchable* variable of the `BodyUserData` is *true*. By default it is *false*.

In order to set it to true, you have to edit the `GameSceneObjects.mm` file.

For example, the `createObjectEnemy` method, contains this code:

```

-(void) createObjectEnemy:(b2Body*)body
{
    // ...

    BodyUserData *data = &m_bodyUserData[ m_bodyUserDataCount++ ];
    data->type = kBodyTypeEnemy;
    data->isTouchable = YES;

    // ...
}

```

The *enemy* and the *spinner* objects are the ones that can be dragged by default.