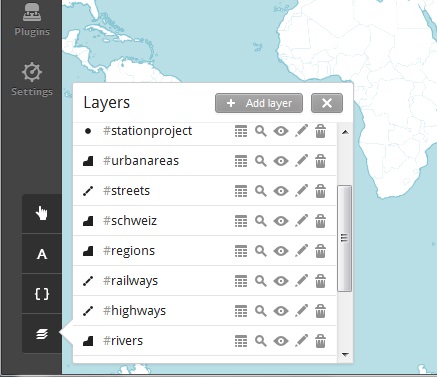
In this document is showed how to use TileMill to create a base layer with real information about Switzerland. The first steps about how to create the project and start the map are the same as in the initial example.

It will be useful to take a look to TileMill CartoCSS reference. It is accessible by clicking the {} button, and there is [an online CartoCSS reference by MapBox](http://mapbox.com/carto/).

Once we have the map, we add the layers. As it is explained in the initial example, we click in the “Add Layer” button and we select the datasource path. We could change the id that appears by default and we can add a class, to write the css.



Once we have added the datasource of the layer, we can choose if we want a style file to be generated by default. In this case, the style will be the necessary to show the data of the layer in the map. To do that, we press the “Save & Style” button.

If we prefer to generate our own style, we just press the “Save” button.

There are different kinds of layers, which will we treated differently when we write the necessary css. Once we have added the layers, we can know the type of every layer. In this case, #stationproject will be an example of point; #urbanareas will be an example of polygon; and #streets will be an example of line.

A point is a single 'spot' in space. It has no dimension, i.e. no length, width, or height. Points are typically defined by a set of coordinates, also known as a coordinate tuple.

A line is built of points. A sequence of points will form a line, or linestring.

A polygon is similar to a line, except that its start and end points are the same. In effect, polygons are closed loops. Polygons form an area, and can have loops cut out of them.

* linestring: the only css code we need is the color in which we want to show it
* ::outline { line-color: #800080; }
* polygon: the css code we need is the color in which we want to show the border and the inside
* ::outline { line-color: #00CED1; }
* ::fill { polygon-fill: #00CED1; }
* point: the css code we need is the color in which we want to show
* marker-fill: #FFFF00;

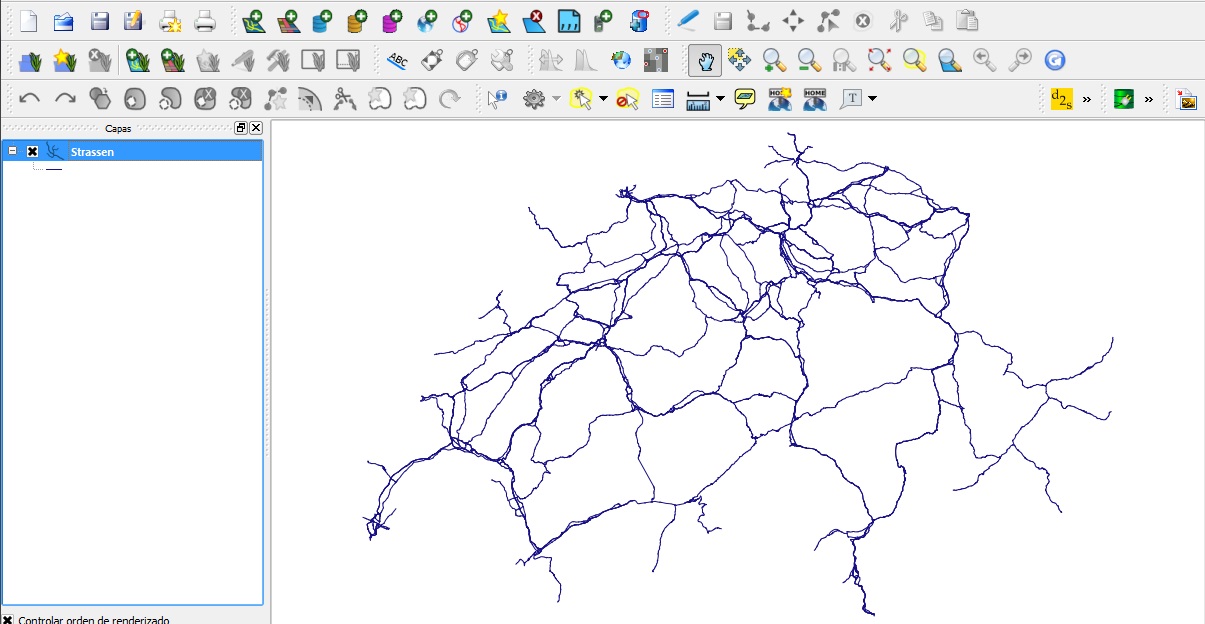
point: and we can change the size of the point

* marker-width: 3;

We can also add a layer using a SQLite databases as geographic datasource. SQLite files can be edited with any SQLite client, including free GIS tools like [Quantum GIS](http://www.qgis.org/).

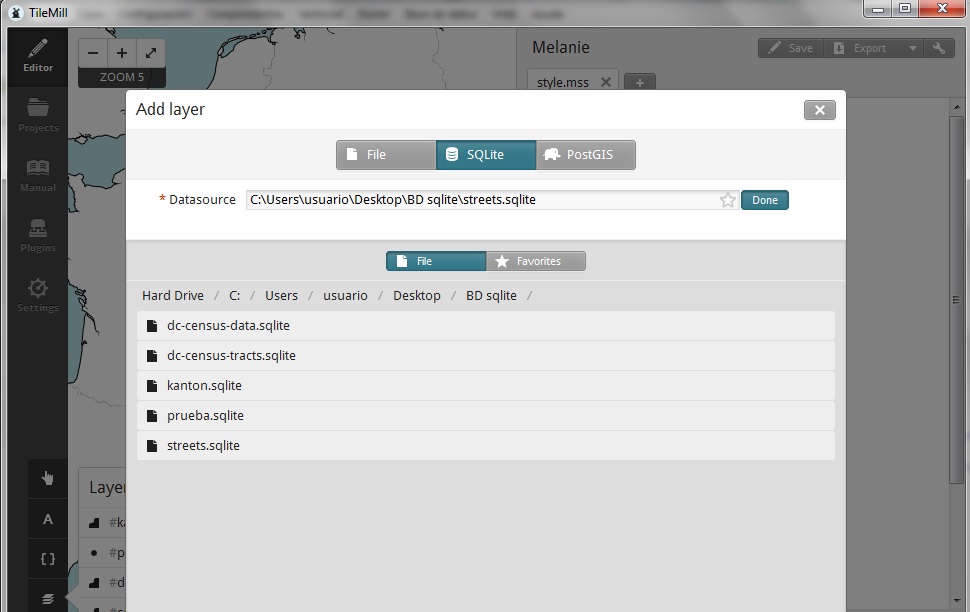
SQLite files are an ideal alternative to shapefiles because they consist of just one file, making them very easy to share, and just like shapefiles, SQLite databases can store geographic features along with non-geographic attributes about those features.

To create the SQLite database using Quantum GIS we need a shapefile with our data. For example, to create a database with the streets in Switzerland we need a folder containing the .dbf, .prj, .shp, .xml and .shx files about the streets in Switzerland. Once we have our shapefile, we just need to add the folder in Quantum GIS, just drag it to the Layers window.



Right-click on our layer named Strassen within the Layers window and click “Save As…” . Select SQLite for the Format, and enter the  name. You can browse to select the directory to save the file.

Once in TileMill, we click “Add layer” and we change the add layer type to SQLite. In datasource we have to add the .sqlite file we have just created.



For the Table or subquery field, enter (SELECT \* from streets). This is a query to select the data from table streets within your SQLite database. This field acts as a subquery so the information must be entered in a subquery fashion.

Select the Spatial Reference System (SRS) for your feature. This will be the projection your data is. TileMill can often autodetect this value. But it can also be obtained from [here](http://spatialreference.org/).

At this point, you can already click the “Save & Style” to add your layer with the default CartoCSS settings and preview the result.

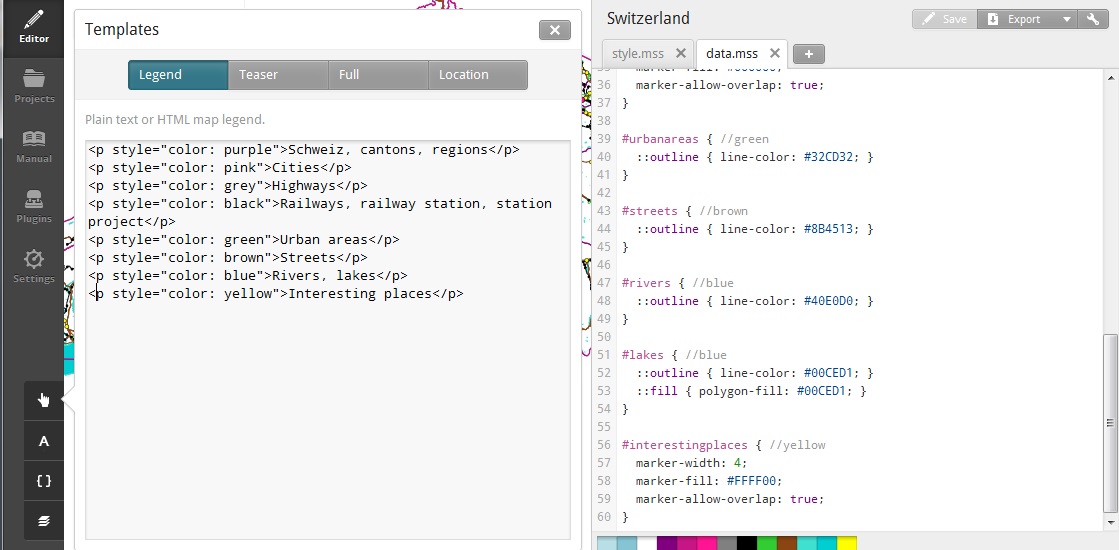
The real power of having SQLite in TileMill is the ability to join two or more SQLite databases together. This feature allows you supplement your geographic data with data from other sources and use it in your stylesheets and tooltips.

When joining multiple SQLite files, you will have one database that contains feature geometries and additional database can add more attributes about those features. To complete the join, the databases need to share a common key or ID.

We can edit the previous layer to add a join query to another database. In order to do that, in the “Attach DB” field we write where our second .sqlite database is stored. And into the “Table or subquery” field we could put something like this:

(SELECT \* FROM streets JOIN cities on streets.id = cities.id && cities.id=7)

When we have finished writing the css code for every layer, we could add a legend to show some extra information about a layer. For example, to know in which color we are showing each information.



We could also add a Teaser: a piece of information that will appear when we hover with the mouse in a point. Or a Full: a piece of information that will appear when we click with the mouse in a point. Here we can show normal text, or specific information about the particular point.

The final step will be to export as mbtiles. Remember not to choose to many zoom levels or the size of the map will be too big. You can also restrict the part of the world that is going to be shown to Switzerland and a little bit around it.