

## Chapter 1

# OpenStreetMap Integration

OpenStreetMap is a collaborative project to create maps (<http://openstreetmap.org>) which offers a nice API to access and download map tiles.

Onil Goubier (onil.goubier@gmail.com), Thierry Goubier (thierry.goubier@cea.fr), and Sergio Maass (sergio.maass@gmail.com) have contributed to integration of OpenStreetMap in Roassal. These contributions are under the MIT License.

The class RTOSM stands for OpenStreetMap and is a shape that downloads and renders tiles. Consider the following example:

```
v := RTView new.  
map := RTOSM new.  
e := map element.  
v add: e.  
v @ RTDraggableView.  
RTEmptyViewContext setOnView: v.  
v
```

Figure 1.1 is obtained by simply creating an element from the shape RTOSM and adding it to a view. The map can be scrolled (thanks to `v @ RTDraggableView`). The Pharo inspector offers zooming in and out. When visualizing OpenStreet map tiles, you may want to disable the automatic camera positioning. This is achieved with `RTEmptyViewContext setOnView: v`.

### 1.1 Moving the camera to particular locations

The camera represents the location where the user sees the Roassal output, and is located at the center of the window that renders the scene. The camera may be located at any particular geographical location. Consider the

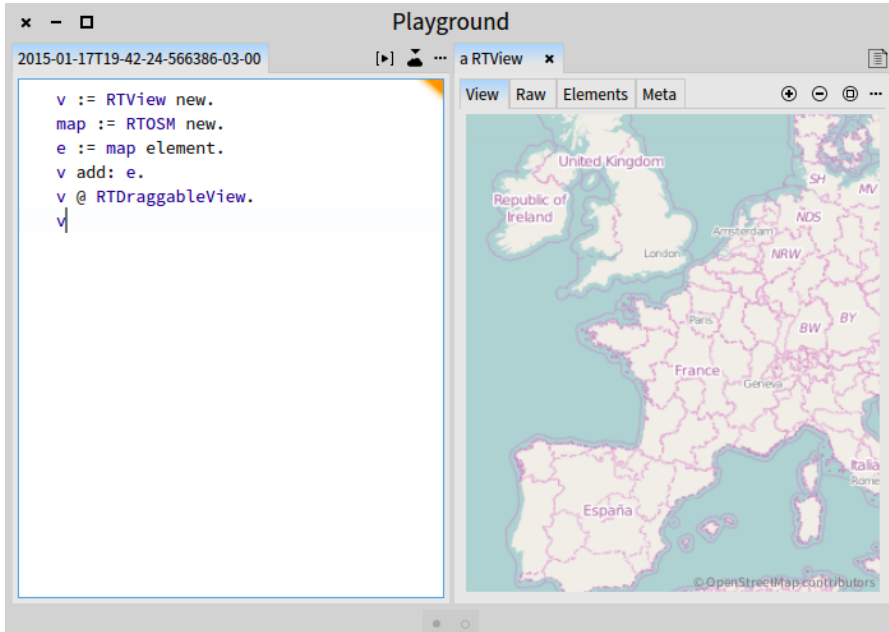


Figure 1.1: Simple example of using RTOSM.

following example that points to Paris:

```

v := RTView new.
map := RTOSM new.
e := map element.

v add: e.

paris := 48.8567 @ 2.3508.

v @ RTDraggableView.

v canvas camera translateTo: (map latLonToRoassal: paris).
v canvas camera noInitializationWhenOpen.
RTEmptyViewContext setOnView: v.
v

```

The variable `paris` contains a point latitude @ longitude. This coordinate is easily obtained by googling for it or is available from Wikipedia.

Consider the following variation (Figure 1.3):

```

v := RTView new.

```

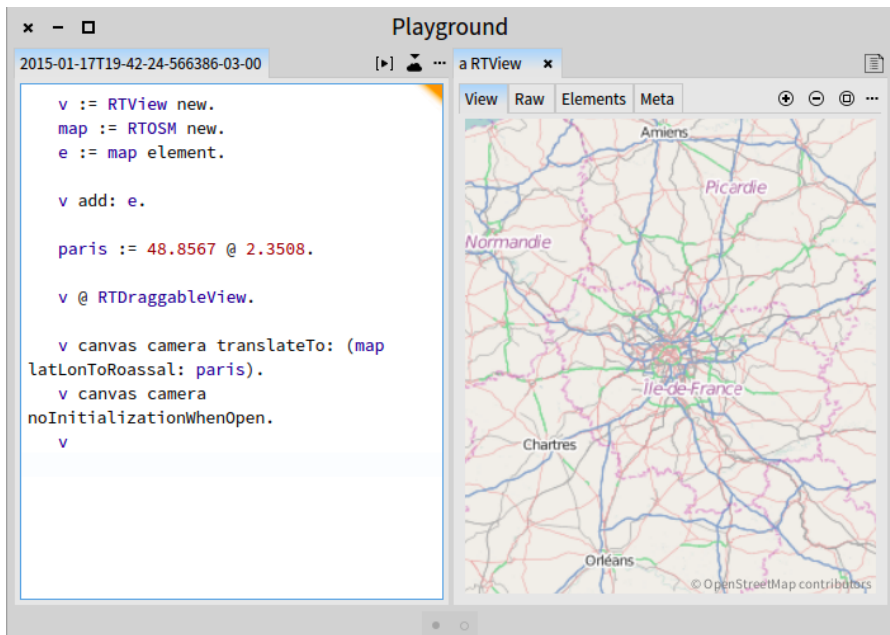


Figure 1.2: Pointing the camera to Paris.

```

map := RTOSM new.
e := map element.

```

```

v add: e.

```

```

v @ RTDraggableView.

```

```

movingCamera := [ :locationLatLong |
  v canvas camera translateTo:
    (map latLonToRoassal: locationLatLong).
  v signalUpdate ].

```

"Adding a menu"

```

mb := RTMenuBuilder new.

```

```

mb view: v.

```

```

mb menu: 'Paris' callback: [ movingCamera value: 48.8567 @ 2.3508 ].

```

```

mb menu: 'London' callback: [ movingCamera value: 51.50722 @ -0.12750 ].

```

```

mb menu: 'NewYork' callback: [ movingCamera value: 40.7127 @ -74.0059 ].

```

```

mb build.

```

```

RTEmptyViewContext setOnView: v.

```

```

v

```

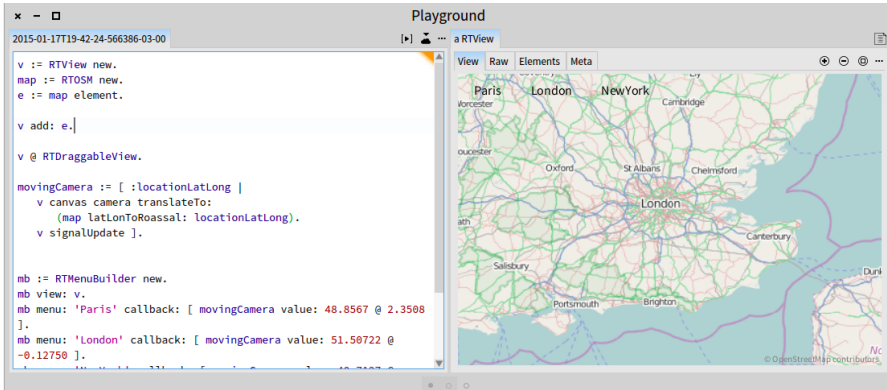


Figure 1.3: Adding a menu with locations.

The script above defines the block called `movingCamera` which takes as argument a geographical location. The blocks move the camera accordingly and signal a refresh. A menu is then defined to let the user click on each city name to move the camera.

## 1.2 Decorating the map

Roassal elements may be added to a view, over a map. The method `RTOSM>>latLonToRoassal:` is used to translate geographical coordinates into Roassal space. Consider the following example that shows seismic activity on Earth (Figure 1.4):

```

csv := (ZnEasy get: 'http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary
/2.5_month.csv') contents.
tab := RTTabTable new
  input: csv
  usingDelimiter: $,.
tab removeFirstRow.
tab replaceEmptyValuesWith: '0' inColumns: #(2 3 5).
tab convertColumnsAsFloat: #(2 3 5).

v := RTView new.
map := RTOSM new.
e := map element.
v add: e.
tab values
  do: [ :row |
    e := (RTEllipse new
      size: (2 raisedTo: row fifth) * 10;

```



Figure 1.4: Seismic activity on Earth.

```

color: (Color red alpha: 0.3)) elementOn: row fifth.
e @ RTPopup @ RTHighlightable.
e translateTo: (map latLonToRoassal: row second @ row third).
v add: e ].

```

```

v canvas camera scale: 0.02; noInitializationWhenOpen.

```

```

v @ RTDraggableView.
RTEmptyViewContext setOnView: v.
v

```

The script begins by fetching data from the Earthquake Hazards Program server. The fetched CSV file has to be slightly processed to replace double commas (,) entries by a zero (,0) and to convert some columns into float numbers. Column 2 of the file corresponds to the latitude of the event; Column 3 to the longitude; Column 5 to the seism intensity.

For each table row an ellipse is created with an exponential size. Each produced element has a popup and is highlightable. The ellipse is then translated to its calculated position in the Roassal space.

A scale of 0.02 is used to give an overview of the map. Thanks to the `noInitializationWhenOpen` setting the map keeps the 0.02 scale value when opened. Without this setting, the value of 1 is used. The view is made draggable.

### 1.3 Getting country location

Some data are frequently given for a particular country. Visualizing data-related country involves a translation from the country name to the geographical location. Consider the data offered by the United Nations High Commissioner for Refugees (UNHCR, <http://popstats.unhcr.org>).

Here is an excerpt of a file obtained from the UNHCR server:

```
Year,Country/territory of residence>Total population
2013,Afghanistan,985191
2013,Albania,7747
2013,Algeria,95921
...
```

Visually representing the number of refugees on an OpenStreetMap requires translating the word Afghanistan and all other country names into points as latitude @ longitude. Since this is a frequent need, we provide facilities for this. The expression `RTOSM downloadCountries` uses `restcountries.eu` to extract the list of countries and their position. The expression returns a list of dictionaries. Since the result is faithfully structured as the HTTP request, extracting data is a bit cumbersome. Here is an example of getting the geolocation of Afghanistan:

```
countries := RTOSM downloadCountries.
d := (countries detect: [ :dd | (dd at: 'name') = 'Afghanistan' ]) at: 'latlng'.
```

Consider the following script (Figure 1.5):

```
"Data extracted from http://popstats.unhcr.org/PSQ_POC.aspx"
countries := RTOSM downloadCountries.
tab := RTTabTable new input: RTOSM peopleForUNHCR usingDelimiter: $,.
4 timesRepeat: [ tab removeFirstRow ].
tab
  convertColumn: 3
  to: [ :value |
    value = '*'
      ifTrue: [ 0 ]
      ifFalse: [ value asNumber ] ].
v := RTView new.
map := RTOSM new.
e := map element.
v add: e.
```

```
"We are interested in countries that have at least one refugee and we want the
countries in reverse order according to the number of refugees they have. This
is helpful in the case of element overlapping"
interestingRows := (tab values reject: [ :row | row third < 1 ])
sorted: [ :a :b | a third > b third ].
```

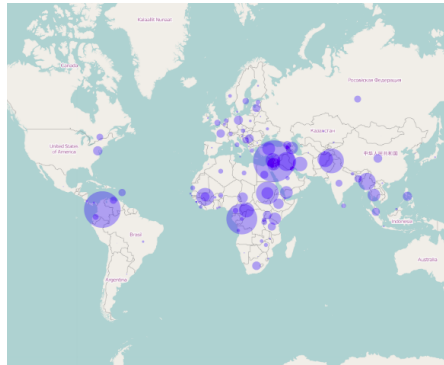


Figure 1.5: Refugees monitored by the UNHCR.

```
interestingRows do: [ :row |
  "Trying to get the country corresponding to the row"
  theCountry := countries
    detect: [ :each | '*' , (each at: 'name') , '*' match: row second ]
    ifNone: [ nil ].
  theCountry ifNotNil: [
    | lat lon dict |
    latLng := (theCountry at: 'latlng').
    lat := latLng first.
    lon := latLng second.
    e := (RTEllipse new
      size: (row third / Float pi) sqrt * 2;
      color: (Color blue alpha: 0.3))
    elementOn:
      {(row second).
      (row third)}.
    e @ RTPopup.
    e translateTo: (map latLonToRoassal: lat @ lon).
    v add: e ] ].

v @ RTDraggableView @ RTZoomableView.
RTEmptyViewContext setOnView: v.
v canvas camera focusOnCenterScaled.
v
```

The method `peopleForUNHCR` contains the data fetched from the UNHCR server.