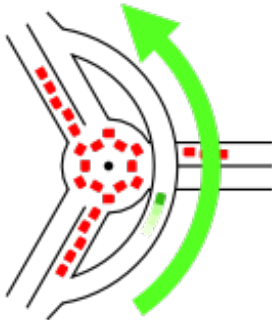


Die Nutzung von Programmierschnittstellen

Jan-Philipp Kolb

22 Februar 2017

The Overpass API



Overpass API

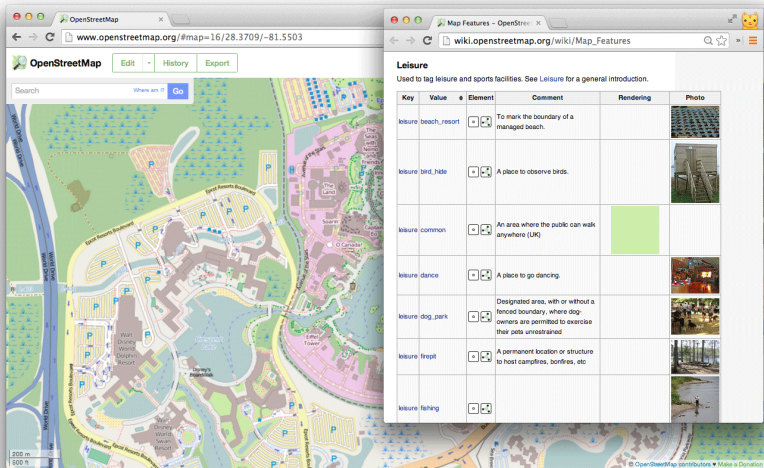
Figure 1: Logo Overpass API

The Overpass API is a read-only API that serves up custom selected parts of the OSM map data.

(http://wiki.openstreetmap.org/wiki/Overpass_API)

Wichtige Information

http://wiki.openstreetmap.org/wiki/Map_Features



The image shows two overlapping browser windows. The background window displays the OpenStreetMap interface with a map of Walt Disney World, including labels for 'Walt Disney World Dolphin Resort' and 'Walt Disney World Swan Resort'. The foreground window shows the 'Map Features' wiki page, which contains a table of leisure-related tags.

Leisure
Used to tag leisure and sports facilities. See [Leisure](#) for a general introduction.

Key	Value	Element	Comment	Rendering	Photo
leisure	beach_resort		To mark the boundary of a managed beach.		
leisure	bird_hide		A place to observe birds.		
leisure	common		An area where the public can walk anywhere (UK)		
leisure	dance		A place to go dancing.		
leisure	dog_park		Designated area, with or without a fenced boundary, where dog-owners are permitted to exercise their pets unrestrained		
leisure	firepit		A permanent location or structure to host campfires, bonfires, etc		
leisure	fishing				

Figure 2: osm map features

Jan-Philipp Kolb

Die Nutzung von Programmierschnittstellen

Beispiel: Nutzung der Overpass API

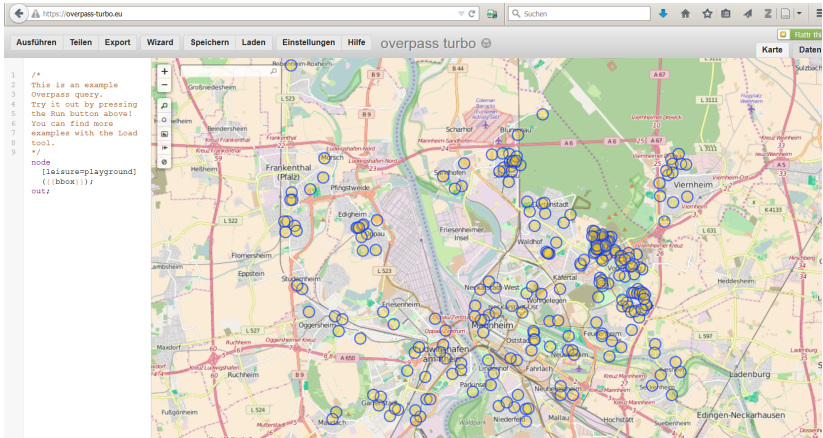


Figure 3: Spielplätze Mannheim

Export der Rohdaten

The screenshot shows the Overpass Turbo web application interface. The main map displays a region with several blue circular markers. On the left, a code editor contains the following query:

```
1 /*  
2 This is an example  
3 Overpass query.  
4 Try it out by pressing  
5 the Run button above!  
6 You can find more  
7 examples with the Load  
8 tool.  
9 */  
10 node  
11 [leisure=playground]  
12 {{bbox}};  
13 out;
```

Below the code editor, the 'Exportieren' (Export) menu is open, showing options for 'Daten' (Data):

- Als [GeoJSON](#)
- Als [GPX](#)
- als [KML](#)
- [Rohdaten](#)

Under 'Rohdaten', it says 'Rohdaten direkt von [Overpass API](#)' and 'In einen OSM-Editor laden: [IOSM](#), [Level0](#)'. There is also a button 'GeoJSON als [gist](#) speichern'.

The 'Export - Rohdaten' dialog box is open, displaying the raw XML data:

```
<tag k="leisure" v="playground"/>  
</node>  
<node id="3552199716" lat="49.4940053" lon="8.4638522">  
<tag k="leisure" v="playground"/>  
</node>  
<node id="3699593805" lat="49.4697566" lon="8.4125343">  
<tag k="leisure" v="playground"/>  
</node>  
<node id="3763978570" lat="49.5829548" lon="8.3555268">  
<tag k="leisure" v="playground"/>  
</node>  
<node id="3802189892" lat="49.4927414" lon="8.4579843">  
<tag k="leisure" v="playground"/>  
<tag k="operator" v="Evangelische Hafenkirche"/>  
</node>  
</osm>
```

Below the XML data, there is a section 'Öffnen von export.osm' with the following options:

- Sie möchten folgende Datei öffnen:
• **export.osm**
Vom Typ: XML Document (28,6 KB)
Von: bibt:
- Wie soll Prefix mit dieser Datei verfahren?
☐ Öffnen mit [XML Editor \(Standard\)](#)
☒ Datei speichern
☐ Für Dateien dieses Typs immer diese Aktion ausführen

Buttons 'OK' and 'Abbrechen' are at the bottom.

Figure 4: Export Rohdaten

Import von der Overpass API zu R

```
Link1 <- "http://www.overpass-api.de/api/interpreter?  
data=[maxsize:1073741824][timeout:900];area[name=\""
```

```
library(XML)  
place <- "Mannheim"  
type_obj <- "node"  
object <- "leisure=playground"
```

```
InfoList <- xmlParse(paste(Link1,place,"\""];",  
type_obj,"(area) [",object,  
"] ;out; ",sep=""))
```

XML Output

```
<?xml version="1.0" encoding="UTF-8"?>
- <osm generator="Overpass API" version="0.6">
  <note>The data included in this document is from www.openstreetmap.org. The data is made available under ODbL.</note>
  <meta areas="2017-02-06T06:35:03Z" osm_base="2017-02-06T06:48:02Z"/>
  - <node lon="8.5028074" lat="49.5190994" id="30560755">
    <tag v="Potlatch 0.5d" k="created_by"/>
    <tag v="playground" k="leisure"/>
  </node>
  - <node lon="8.5393963" lat="49.4963345" id="76468450">
    <tag v="Potlatch 0.4a" k="created_by"/>
    <tag v="playground" k="leisure"/>
    <tag v="Rutsche, Schaukel, großer Sandkasten, Tischtennis" k="note"/>
  </node>
  - <node lon="8.5529589" lat="49.4967807" id="76468534">
    <tag v="playground" k="leisure"/>
  </node>
  - <node lon="8.5487501" lat="49.4923030" id="76468535">
    <tag v="playground" k="leisure"/>
  </node>
```

Figure 5: Spielplätze in Mannheim

Das Arbeiten mit XML Daten (xpath)

Die Liste der ID's mit dem Wert *playground*:

```
node_id <- xpathApply(InfoList,  
  "//tag[@v= 'playground']/parent::node/@ id")  
## node_id[[1]]
```

```
> node_id[[1]]  
      id  
"30560755"  
attr(,"class")  
[1] "XMLAttributeValue"  
> |
```

Figure 6: Erste node id

latitude und longitude bekommen

```
lat_x <- xpathApply(InfoList,  
  "//tag[@v= 'playground']/parent::node/@ lat")  
# lat_x[[1]];lat_x[[2]]
```

```
lat_x <- xpathApply(InfoList,  
  "//tag[@v= 'playground']/parent::node/@ lon")
```

```
lat  
"49.5190994"  
attr(,"class")  
[1] "XMLAttributeValue"  
lat  
"49.4963345"
```

Paket auf Github

```
library(devtools)
install_github("Japhilko/gosmd")
```

```
library(gosmd)
pg_MA <- get_osm_nodes(object="leisure=playground",
                        "Mannheim")
info <- extract_osm_nodes(OSM.Data=pg_MA,
                           value="playground")
```

Ausschnitt der Ergebnisse

	leisure	lat	lon	note
30560755	playground	49.51910	8.502807	NA
76468450	playground	49.49633	8.539396	Rutsche, Schaukel, groÃ
76468534	playground	49.49678	8.552959	NA
76468535	playground	49.49230	8.548750	NA
76468536	playground	49.50243	8.548140	Schaukel, Rutsche, Sand
76468558	playground	49.49759	8.542036	NA

- Tutorial zur Nutzung der Overpass API
- Vignette xml2