Geodatenservice,
Kartenwerkstatt
Gis-Schule
Dr.-Ing. Claas Leiner

Wilhelmshöher Allee 304 E
34131 Kassel
Tel: 0561/56013445,

Mail: claas.leiner@gkg-kassel.de
Web: http://www.gkg-kassel.de

SKRIPT ZUM WORKSHOP

Inhalte aus OSM-Daten mit ogr2ogr für QGIS bereitstellen

OSM-Daten beschaffen

Rohdaten im PBF-Format gibt es unter:
https://download.geofabrik.de/

Auch Shapes werden angeboten. Die Shapes enthalten jedoch nur eine Auswahl.

Für Übungen eine gute Größe:
https://download.geofabrik.de/europe/luxembourg-latest.osm.pbf

Rohdaten für bestimmte Gebeite befinden sich auch hier:
https://download.bbbike.org/osm/

BITT laden Sie sich herunter:

https://download.bbbike.org/osm/bbbike/Kassel/Kassel.osm.pbf

Informationen über OSM-Inhalte

Die Attributtierung (taging) der OSM-Daten wird hier beschrieben.

https://wiki.openstreetmap.org/wiki/DE:Map_Features

Dort werden die Schlüssel (KEYS) und Werte (VALUES) erläutert, welche die Inhalte der OSM-Daten beschreiben und für Auswahlen verwendet werden können.

Tags, Attribute und die Datei osmconf.ini

OSM-Tags, bestehent aus **KEY** (Schlüssel) und **VALUE** (Wert) und beschreiben die Inhalte der OSM-Daten. Aus den unzähligen **TAGS** müssen wir diejenigen in QGIS-Attribute umsetzen, die wir benötigen.

Welche TAGS QGIS als eigene Spalten in der Attributtabelle darstellt, wird in der Datei **osmconf.ini** festgelegt.Sämtliche TAGS, die in dieser Datei nicht erwähnt werden, landen im QGIS in der spalte OTHER_TAGS.

Da QGIS OSM-Daten mit OGR ließt, erfolgt auch das direkte Lesen von PBF-Dateien mit den Vorgabenj aus der OSMCONF.INI.

Auf einem Linux-System (Ubuntu) findet sich die Datei hier:
file:///usr/share/gdal/osmconf.ini

Unter Windows bei QGIS 3.16 an diesem Ort:

file://C:\Program Files\QGIS 3.16\share\gdal\osmconf.ini

Es ist möglich mit ogr2ogr direkt auf eines bestimmte osmconf.ini zuzugreifen:

Folgende Einträge in der osmconf.ini sind für die Attributierung wichtig.

Welche Inhalte werden als Flächen dargestellt

`closed_ways_are_polygons=`

Hinter dem **=** findet sich eine Auflistung von TAGS, die erzwingen, dass ein geschlossener Linienzug (WAY) als Fläche dargestellt wird.

Standard:

`closed_ways_are_polygons=``aeroway,amenity,boundary,building,craft,geological,historic,landuse,leisure,military,natural,office,place,shop,sport,tourism,highway=platform,public_transport=platform`

Nicht aufgeführt sind z.B die tags:

`waterway,riverbank,power`

Was dazu führt, dass beim Import mit ogr2ogr oder dem direkten Laden ins QGIS keine flächenhafte Darstellung von Fließgewässerflächen umgesetzt wird. Auch für Umspannwerke gibt es keine Darstellung von Flächen, wie sie in der OSM-Webkarte umgesetzt ist.

Werden `waterway,riverbank,power` der Liste angehängt , werden die entsprechenden Flächen erzeugt.

Für welche Inhalte gibt es eine eigene Attributtspalte

Werden die OSM-Daten ohne weitere Abfrage mit *ogr2ogr* importiert, entstehen im Ziel vier Layer:

- ✗ **POINTS** (Sämtliche NODES, die nicht zu einem WAY verbunden sind also Punkte)
- ✗ **LINES** (Sämtliche WAYS die als einfache Linienobjekte vorliegen, also Verkehrswege, Leitungen Fließgewässer)
- ✗ **MULTIPOLYGONS** (Sämtliche WAYS, die als Flächen dargestellt werden, also Gebäude, Waldflächen etc.
- ✗ **MULTILINESTRINGS** (Linien, die zu weiteren Objekten zusammengefasst wurden, z.B. Buslinien oder Wanderrouten)

Die OSMCONF.INI legt für jeden Geometrietyp, die in Attributspalten umgesetzten tags fest. **Nur tags die explizit aufgeführt sind, können beim Import mit ogr2ogr über SQL abgefragt werden.!**

Punkte

```
[points]
# keys to report as OGR fields
attributes=name,barrier,highway,ref,address,is_in,place,man_made
```

Sollen z.B. Bushaltestellen bezüglich ihrer Barrierefreiheit und Überdachungs ausgewertet werden, benötigen wird nicht nur die Spalte HIGHWAY (`highway = 'bus_stop'`) sonder auch die TAGS `wheelchair`, `shelter` und `public_transport`, die wir der Liste anfügen

Möchten wir Windanlagen erfassen und dabei Informationen zur Höhe und zur Leistungs auswerten, sollten auch folgende tags der Liste zugefügt werden:

`power,generator,voltage,generator:source,generator:method,generator:output:electricity,height:hub,height`

Anschließend sieht die Liste folgendermaßen aus:

```
attributes=name,barrier,highway,ref,address,is_in,place,man_made,wheelchair
,shelter,public_transport,power,generator,voltage,generator:source,generato
r:method,generator:output:electricity,height:hub,height
```

Linien

```
[lines]
# keys to report as OGR fields
attributes=name,highway,waterway,aerialway,barrier,man_made
```

Mit dieser Standrddarstellung bekommen wir keine Spalten-Information zum Thema Eisenbahn.

Mit der folgenden Ergänzung erhalten wir Informationen zum Eisenbahnnetz und zu der Frage ob Straßen oder Eisenbahnen über Tunnel und Brücken verlaufen.

```
attributes=name,highway,waterway,aerialway,barrier,man_made,railway,tunnel,
bridge
```

Flächen

```
[multipolygons]
# keys to report as OGR fields
attributes=name,type,aeroway,amenity,admin_level,barrier,boundary,building,
craft,geological,historic,land_area,landuse,leisure,man_made,military,natur
al,office,place,shop,sport,tourism
```

Sollen auch auch eine flächenhafte Darstellung von großen Flüssen und PLZ-Bereichen umgesetzt werden, könnte folgende Ergänzung durchgeführt werden:

```
attributes=name,type,aeroway,amenity,admin_level,barrier,boundary,building,
craft,geological,historic,land_area,landuse,leisure,man_made,military,natur
al,office,place,shop,sport,tourism,waterway,water,postal_code
```

Wenn die Original **osmconf.ini** unverändert gelassen werden soll, kann die geänderte osmconf.ini auch direkt mit ogr2ogr angesprochen werden.

Import mit ogr2ogr nach Spatialite oder Postgis

Die grundlegende Vorgehensweise ist in der Datei OGR2OGR_KOMMANDOS.SH dokumentierte. In der Datei *sql_kurs_FOSSGIS.sql* gibt es weitere Informationen zur Anwendung von SQL. Wir können in die ogr2ogr-Kommandos SQL-Abfragen einbauen, um für QGIS exakt die gewünschten aus dem OSM-Datenbestand verfügbar zu machen.

Infos

- ogr2ogr ist ein Kommandozeilenprogramm zum konvertieren von Vektorgeodaten. ogr2ogr bietet SQL-Unterstützung und ist auch zum umprojizieren von Vektordaten geeigent. Einige Toolboxwerkzeuge im qgis (rubrik GDAL / OGR) generieren ogr2ogr-Aufrufe.
- Sehr geeignet zum Import von OSM-Rohdaten und NAS-XML in PostGis und SpatiaLite-Datenbanken, insbesondere bei großen Datenmengen.
- Über kleine Shellskripte lassen sich ganze Verzeichnisse mit Shapes umprojizieren oder in eine Datenbank schreiben.
- Mit integrierten SQL-Abfragen lassen sich schon beim Import Inhalte auswählen.
- Und vielens mehr.

Absetzen der Kommandos:

- ✗ Linux: In der voreingestellten Kommandozeile (bash)
- ✗ Windows: In die OsGeo4w-Shell (Aus der QGIS-Pprogrammgruppe zu öffnen)
- ✗ MacOsX: In der voreingestellten Kommandozeile

Liste der auf dem eigenen Rechner unterstützten Formate ogrinfo --formats

Webquellen mit wichtigen Informationen

- ✗ Hauptseite Ogr2ogr mit allen Parametern:
  https://www.gdal.org/ogr2ogr.html
- ✗ Übersicht über unterstützte Formate und formatspezifische Optionen:
  (-lco LayerCreation Options und -dco DatabaseCreation Options)
  https://www.gdal.org/ogr_formats.html
- ✗ SQLITE-SQL-Dialekt:
  https://www.gdal.org/ogr_sql_sqlite.html
- ✗ OGR-SQL-Dialekt:
  https://www.gdal.org/ogr_sql.html
- ✗ Weitere Konfigurationsoptionen:
  https://trac.osgeo.org/gdal/wiki/ConfigOptions

ogr2ogr-Basisbefehl für Export nach Shapefile

```
ogr2ogr -f "ESRI Shapefile" -s_srs EPSG:4326 -t_srs EPSG:25832 osmshapes kassel.pbf
```

- ✗ **Zieldatenformat**: `-f "ESRI Shapefile"` (hier Shapefile)
- ✗ **Quell-Koordinatensystem**: `-s_srs EPSG:4326` (Geographisch WGS 84)
- ✗ **Ziel-Koordinatensystem:** `-t_srs EPSG:25832` (ETRS89/UTM32N)
- ✗ **Quelldatei**: `osmquelle` (hier kassel.pbf
- ✗ **Ziel**: `Shapeausgebeverzeichnis` (hier osmshapes)
  (dort werden Shapefiles für Polygone,
  Linien und Punkte geschrieben)

OSM zu Shapefile mit weiteren Optionen

```
ogr2ogr -f "ESRI Shapefile" -s_srs EPSG:4326 -t_srs EPSG:25832 -lco ENCODING=UTF-8
-lco SPATIAL_INDEX=YES -skipfailures osmshapes kassel.pbf
```

- ✗ `-lco ENCODING=UTF-8` Zeichencodierung für Shapes setzen, im Beispiel utf-8
- ✗ `-lco SPATIAL_INDEX=YES` Räumlichen Index erzeugen für perfomanteres arbeiten mit großen Shapes
- ✗ `-skipfailures` Kein Abbruch bei Verarbeitungsfehlern

PBF-Datei nach SpatiaLite oder Geopackage importieren und dabei umprojizieren

```
ogr2ogr -f "SQLite" -dsco SPATIALITE=yes  -lco SRID=25832  -gt unlimited -s_srs
EPSG:4326 -t_srs EPSG:25832 -skipfailures ziel.sqlite osmquelle.osm/pbf

ogr2ogr -f "GPKG"   -gt unlimited -s_srs EPSG:4326 -t_srs EPSG:25832 -skipfailures
ziel.gpkg osmquelle.osm/pbf
```

- ✗ `-f "SQLite"` Zieldatenformat:(hier sqlite) oder `-f "GPKG"`
- ✗ `-dsco SPATIALITE=yes` Es wird eine SpatiaLite-Datenbank mit allen räumlichen Funktionen und keine nackte SQLite-Datei erstellt
- ✗ `-lco SRID=25832` Beschreibung des Koordinatensystems, ansonsten wird es beim Laden der Layer in qgis jedes mal abgefragt
- ✗ `-s_srs EPSG:4326` Quell-Koordinatensystem: (Geographisch WGS 84)
- ✗ `-t_srs EPSG:25832` Ziel-Koordinatensystem: (ETRS89/UTM32N)
- ✗ `-gt unlimited` (Anzahl der Zeilen, die mit jeder Transaction gesetzt wird) unlimited = in einer Transaktion
- ✗ `-skipfailures` Schreiben bricht bei Fehlern nicht ab, sondern wird fortgesetzt.
- ✗ `osmquelle:` Quelldatei:
- ✗ `ziel.sqlite` Ziel (In der SQLIE-Datei werden verschiedene Layer angelegt - bei OSM Multipolygons, line, Points)

Weitere Optionen zur Erhöhung der Performance bei SQLite basierten Formaten

```
ogr2ogr --config OGR_SQLITE_SYNCHRONOUS OFF --config OGR_SQLITE_CACHE 8192 --config
OSM_MAX_TMPFILE_SIZE 4000 -progress -f "SQLite" -dsco SPATIALITE=yes  -lco
SRID=25832 -gt unlimited -s_srs EPSG:4326 -t_srs EPSG:25832 -skipfailures
kassel.sqlite kassel.pbf

ogr2ogr --config OGR_SQLITE_SYNCHRONOUS OFF --config OGR_SQLITE_CACHE 8192 --config
OSM_MAX_TMPFILE_SIZE 4000 -progress -f "GPKG"  -gt unlimited -s_srs EPSG:4326 -
t_srs EPSG:25832 -skipfailures  kassel.gpkg kassel.pbf
```

`-progress` Eine Fortschrittsmeldung wird angezeigt

`--config OGR_SQLITE_SYNCHRONOUS OFF` Ohne diese Option läuft es sehr langsam, mit sehr viel schneller theoretisch Risiko des Datenverlustes, beim Update einer Datenbank, wenn es beim Import zum Programmabsturz kommt.

`--config OGR_SQLITE_CACHE 8192` oder höher - Zwischenspeicher erhöht perfomance

`--config OSM_MAX_TMPFILE_SIZE 4000` Maximale Größe der im Arbeitspseicher gehaltenen temporären Datei: Möglichst hoch setzen

`2>> ogrmeldungen.txt` An den Befehl anhängen, um Meldungen in eine Datei zu schreiben

Auswahl nur der Gebäude beim Import

 # Befehl mehrzeilig durch \ Zeilenumbrüche eingefügt Im windows muss wieder alles in einer Zeile stehen

```
ogr2ogr --config OGR_SQLITE_SYNCHRONOUS OFF --config OGR_SQLITE_CACHE 8192 --config
OSM_MAX_TMPFILE_SIZE 4000 --config OSM_COMPRESS_NODES YES -progress \
-f "SQLite" -dsco SPATIALITE=yes  -lco SRID=25832 -nln gebaeude  -gt unlimited -
update -s_srs EPSG:4326 -t_srs EPSG:25832 -skipfailures kassel.sqlite kassel.pbf \
-sql "select name,building from multipolygons where building is not null"
```

`-update` Die Datei wird nicht überschrieben, es werden neue Layer rangelegt

`-append` Neue Zeilen an vorhandene Layer anhängen

`-overwrite` -overwrite (hier nicht verwendet) bewirkt ein überschreiben der Datei

`-nln gebaeude` Der Ergebnislayer heißt Gebäude

`-sql "select name,building from multipolygons where building is not null"`

SQL-Abfrage zur Auswahl der Gebäude

Eisenbahnlinien in einen Layer verkehr importieren

```
ogr2ogr --config OGR_SQLITE_SYNCHRONOUS OFF --config OGR_SQLITE_CACHE 8192
--config OSM_MAX_TMPFILE_SIZE 4000 --config OSM_COMPRESS_NODES YES  \
--config OSM_CONFIG_FILE osmconf.ini -progress \
-f "SQLite" -dsco SPATIALITE=yes  -lco SRID=25832 -nln verkehr  -gt
unlimited -update -s_srs EPSG:4326 -t_srs EPSG:25832 -skipfailures
kassel.sqlite kassel.pbf \
-sql "select name,railway from lines where railway is not null"
```

`--config OSM_CONFIG_FILE osmconf.ini` Eine benutzerdefinierte OSM-Datei zur Definition der Attributspalten Liegt im selben Verzeichnis und sorgt dafür, die Spalte railway anzulegen

`-update` Die Datenbank wird nicht überschrieben sondern geändert

`-nln verkehr` der resultierende Layer heißt verkehr

Import des Straßennetzes in den Layer verkehr

Straßennetz und Eisenbahn könnten auch in eine SQL-Abfrage geschrieben werden:

```
-sql "select name,highway,railway,bridge,tunnel,ref from lines where highway is not
null and railway is not null"
```

Wir tun jetzt aber so, als hätten wir es vergessen, um das Anfügen von Zeilen zu testen:

```
ogr2ogr --config OGR_SQLITE_SYNCHRONOUS OFF --config OGR_SQLITE_CACHE 8192 \
--config OSM_MAX_TMPFILE_SIZE 4000 --config OSM_COMPRESS_NODES YES \
--config OSM_CONFIG_FILE osmconf.ini -progress \
-f "SQLite" -dsco SPATIALITE=yes  -lco SRID=25832 -nln verkehr -update -addfields
-gt unlimited -s_srs EPSG:4326 -t_srs EPSG:25832 -skipfailures \
kassel.sqlite kassel.pbf \
-sql "select name,highway,bridge,tunnel,ref from lines where highway is not null"
```

`-append` Weitere Zeilen an eine vorhandene Tabelle anhängen.

`-addfields` Wie append, es werden fehlende Spalten angelegt, wenn der zweite Datensatz über mehr Spalten verfügt diese Option wird angewendet

Räumlichen Auschnitt wählen

Es ist möglich beim Import einen räumlichen Ausschnit festzulegen

- ✗ `-spat  xmin ymin xmax ymax` Import der Daten nach räumlicher Auswahl 0hne zuschneiden,
- ✗ `-spat_srs srs_def` Optional kann das KBS der Koordinaten angegeben werden

Beispiel für den Kassel-Import

`-spat 9.43650534 51.3045991  9.4596809  51.3221616`

Nur das Umfeld des Bahnhof Wilhelmshöhe Importieren

```
ogr2ogr  --config OGR_SQLITE_SYNCHRONOUS off --config OGR_SQLITE_CACHE 8192
--config OSM_MAX_TMPFILE_SIZE 5000 --config OSM_COMPRESS_NODES YES \
 --config OSM_CONFIG_FILE osmconf.ini  -f "SQLite" -dsco SPATIALITE=yes \
-lco SRID=25832 -spat 9.43650534 51.3045991  9.4596809  51.3221616 \
-nln points -update -append -gt unlimited  -s_srs EPSG:4326 -t_srs
EPSG:25832 -skipfailures kassel.sqlite kassel.pbf \
 -sql "select name,addr_housenumber from points where addr_housenumber is
not NULL"
```

Anhang_ Struktur der OSM-Daten

Zum Datenformat

Die OSM-Daten werden in einem *XML-Dateiformat* ausgegeben,
dass heißt die Daten stehen als Text in einer definierten XML-Syntax zur Verfügung.

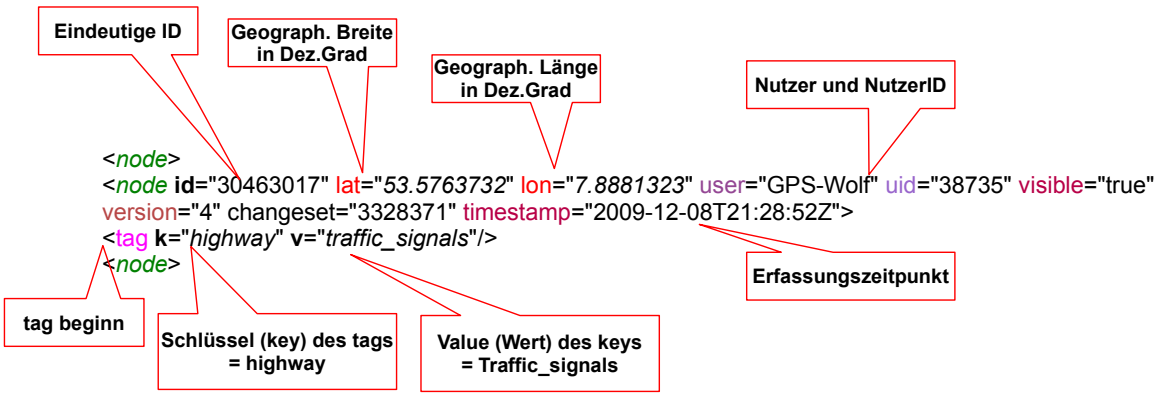
Die Syntax beschreibt GeoObjekte (*Nodes, Ways, Relationen*) und ihre Eigenschaften (tags).
Nodes sind einzelne Punkte, deren Position in der Welt über *geographische Koordinaten* definiert werden.
Sie verfügen über eine eindeutige **ID,** über Koordinaten
(lat = *Geog. Breite* und lon = *geog. Länge* jeweils in *Dezimalgrad z.B. lat="53.5763732" lon="7.8881323"*)
und können über Eigenschaften (tags) verfügen, wenn es sich um eigenständige *Inhalte*,
wie z.B. eine *Haltestelle* handelt.

Ways fassen *Nodes* zu *Linien- oder Polygon-Objekten* (geschlossener Linienzug) zusammen.
Die *Nodes* werden über ihre **IDs** den *Ways* zugeordnet.
Jeder *Way* enthält seinerseits eine **ID** und kann über Sacheigenschaften (tags) verfügen.

Relationen beschreiben komplizierte *Objekte*, wie z.b. *Multipolygone* und *Polygone* mit Löchern bzw. Polygone in Polygonen. Eine *Relation* besteht aus verschiedenen *Ways*, die innerhalb der *Relation* eine bestimmte Rolle spielen. Soll z.B. ein Teich inmitten eines Waldes erfasst werden, so ist das Ufer des Teichs zugleich der Rand des Waldes. Dieser *geschlossene Linienzug* (*Way*) beschreibt in der *Relation Teich* das Ufer und in der *Relation Wald* die Innengrenze.

Nodes

Dieser Node mit tags beschreibt eine Ampelanlage

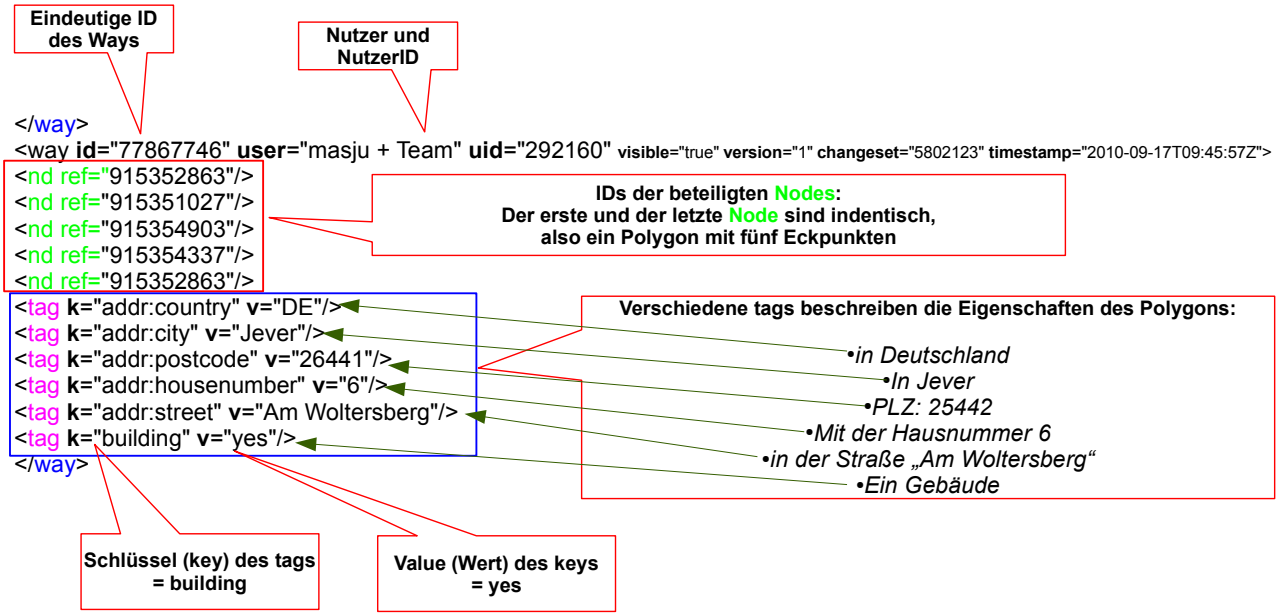


Node ohne tags

<node id="30463010" lat="53.5763732" lon="7.8881323" user="GPS-Wolf" uid="38735" visible="true" version="4"
changeset="3328371" timestamp="2009-12-08T21:28:52Z"/>

Ways

Dieser Way mit tags beschreibt ein Gebäude



Relationen

Diese Relationen beschreiben einen Wald mit zwei Wiesen im inneren