OGR2OSM

A powerful tool for converting geodata to .osm format

SOTM-US 2012
What ogr2osm can do for you
How ogr2osm works
A case study of a data conversion
Why care about converting?
Why care? To avoid this
History

- Written in 2009 by Iván Sánchez Ortega
- Rewritten 2012 by Andrew Guertin for UVM buildings
- I now maintain it
Features

- Can read any ogr supported data source
  - .shp, .mdb, .gdb, sqlite, etc
- Reprojects if necessary – eliminates a step with many sources
- Works with multiple layer sources or shapefile directories
- Uses python translation functions that you write to convert source field values to OSM tags
  - This allows you to use complicated logic to get the tagging right
- Documentation
Installing

- Requires gdal with python bindings
  - Simply `sudo apt-get install python-gdal git` on Ubuntu
  - May require compiling gdal from source and third-party SDKs for some formats (.mdb, .gdb)
- Run `git clone --recursive https://github.com/pnorman/ogr2osm` to install
- Full instructions at https://github.com/pnorman/ogr2osm
Code flow

Read in data source
• Uses python ogr bindings to read the files

Process each layer
• Converts from ogr to osm tagging and objects

Merge nodes
• Merges duplicate nodes
• Adjustable threshold for distance

preOutputTransform()
• A user-defined filtering step, not commonly used

Output XML
• Write to a .osm file that can be opened in JOSM
Read in data source
• Uses python ogr bindings to read the files

Process each layer
• Converts from ogr to osm tagging and objects

Merge nodes
• Merges duplicate nodes
• Adjustable threshold for distance

preOutputTransform()
• A user-defined filtering step, not commonly used

Output XML
• Write to a .osm file that can be opened in JOSM

Code flow
Read in data source
• Uses python ogr bindings to read the files

Process each layer
• Converts from ogr to osm tagging and objects

Merge nodes
• Merges duplicate nodes
• Adjustable threshold for distance

preOutputTransform()
• A user-defined filtering step, not commonly used

Output XML
• Write to a .osm file that can be opened in JOSM
Layer processing

filterLayer()
- Allows layers to be dropped
- Allows for the creation of new fields
  - e.g. a field that indicates the layer of a feature for later

Reproject
- Projects the layer into EPSG:4326

filterFeature()
- Allows features to be removed

Reproject
- Projects the feature into EPSG:4326

Convert to OSM geometries
- Creates nodes and ways
- Only creates multipolygons if necessary

filterTags()
- Where all the magic occurs

filterFeaturePost()
- A user-defined filtering step, not commonly used
Layer processing

filterLayer()
- Allows layers to be dropped
- Allows for the creation of new fields
- e.g. a field that indicates the layer of a feature for later

Reproject
- Projects the layer into EPSG:4326

filterFeature()
- Allows features to be removed

Reproject
- Projects the feature into EPSG:4326

Convert to OSM geometries
- Creates nodes and ways
- Only creates multipolygons if necessary

filterTags()
- Where all the magic occurs

filterFeaturePost()
- A user-defined filtering step, not commonly used
Layer processing

- **filterLayer()**: Allows layers to be dropped, allows for the creation of new fields, e.g., a field that indicates the layer of a feature for later
- **Reproject**: Projects the layer into EPSG:4326
- **filterFeature()**: Allows features to be removed
- **Reproject**: Projects the feature into EPSG:4326
- **Convert to OSM geometries**: Creates nodes and ways, only creates multipolygons if necessary
- **filterTags()**: Where all the magic occurs
- **filterFeaturePost()**: A user-defined filtering step, not commonly used
Layer processing

- **filterLayer()**
  - Allows layers to be dropped
  - Allows for the creation of new fields
  - e.g. a field that indicates the layer of a feature for later

- **Reproject**
  - Projects the layer into EPSG:4326

- **filterFeature()**
  - Allows features to be removed

- **Convert to OSM geometries**
  - Creates nodes and ways
  - Only creates multipolygons if necessary

- **filterTags()**
  - Where all the magic occurs

- **filterFeaturePost()**
  - A user-defined filtering step, not commonly used
Layer processing

- **filterLayer()**
  - Allows layers to be dropped
  - Allows for the creation of new fields
  - e.g. a field that indicates the layer of a feature for later

- **Reproject**
  - Projects the layer into EPSG:4326

- **filterFeature()**
  - Allows features to be removed

- **Convert to OSM geometries**
  - Creates nodes and ways
  - Only creates multipolygons if necessary

- **filterTags()**
  - Where all the magic occurs

- **filterFeaturePost()**
  - A user-defined filtering step, not commonly used
Layer processing

- **filterLayer()**
  - Allows layers to be dropped
  - Allows for the creation of new fields
  - e.g. a field that indicates the layer of a feature for later

- **Reproject**
  - Projects the layer into EPSG:4326

- **filterFeature()**
  - Allows features to be removed

- **Convert to OSM geometries**
  - Creates nodes and ways
  - Only creates multipolygons if necessary

- **filterTags()**
  - Where all the magic occurs

- **filterFeaturePost()**
  - A user-defined filtering step, not commonly used
Layer processing

- **filterLayer()**
  - Allows layers to be dropped
  - Allows for the creation of new fields
  - e.g. a field that indicates the layer of a feature for later

- **Reproject**
  - Projects the layer into EPSG:4326

- **filterFeature()**
  - Allows features to be removed

- **Convert to OSM geometries**
  - Creates nodes and ways
  - Only creates multipolygons if necessary

- **filterTags()**
  - Where all the magic occurs

- **filterFeaturePost()**
  - A user-defined filtering step, not commonly used
Layer processing

- **filterLayer()**
  - Allows layers to be dropped
  - Allows for the creation of new fields
  - e.g. a field that indicates the layer of a feature for later

- **Reproject**
  - Projects the layer into EPSG:4326

- **filterFeature()**
  - Allows features to be removed

- **Convert to OSM geometries**
  - Creates nodes and ways
  - Only creates multipolygons if necessary

- **filterTags()**
  - Where all the magic occurs

- **filterFeaturePost()**
  - A user-defined filtering step, not commonly used
Layer processing

filterLayer()
- Allows layers to be dropped
- Allows for the creation of new fields
  - e.g. a field that indicates the layer of a feature for later

Reproject
- Projects the layer into EPSG:4326

filterFeature()
- Allows features to be removed

Convert to OSM geometries
- Creates nodes and ways
- Only creates multipolygons if necessary

filterTags()
- Where all the magic occurs

filterFeaturePost()
- A user-defined filtering step, not commonly used
Code flow

Read in data source
• Uses python ogr bindings to read the files

Process each layer
• Converts from ogr to osm tagging and objects

Merge nodes
• Merges duplicate nodes
• Adjustable threshold for distance

preOutputTransform()
• A user-defined filtering step, not commonly used

Output XML
• Write to a .osm file that can be opened in JOSM
Code flow

Read in data source
• Uses python ogr bindings to read the files

Process each layer
• Converts from ogr to osm tagging and objects

Merge nodes
• Merges duplicate nodes
• Adjustable threshold for distance

preOutputTransform()
• A user-defined filtering step, not commonly used

Output XML
• Write to a .osm file that can be opened in JOSM
**Code flow**

**Read in data source**
- Uses python ogr bindings to read the files

**Process each layer**
- Converts from ogr to osm tagging and objects

**Merge nodes**
- Merges duplicate nodes
- Adjustable threshold for distance

**preOutputTransform()**
- A user-defined filtering step, not commonly used

**Output XML**
- Write to a .osm file that can be opened in JOSM
Surrey case study

- Shapefile fields similar to other government GIS sources
- Fields or values periodically change with no notice
- 58 layers in 7 zip files
  - Not counting orthos and LIDAR-derived contours
- 153 MB compressed, 1.7 GB uncompressed
- Covers 187 km²
- Too much data to write conversions for without a method
Reduce the amount of data

- ogr2osm will happily turn out a gigabyte .osm but good luck opening it
- Use ogr2ogr -spat to trim the input files down
  - Converting from some formats to shapefiles will truncate field names
    - Can use .gdb when coming from a format with long field names and layers
  - -spat wants coordinates in layer coordinate system
    - Use gdaltransform to turn latitude/longitude into desired coordinates
Use the layer translation (-t layer) and see what layers should be dropped

Most multi-layer sources have layers that should not be imported

In the case of the Surrey data filtering is done in the script that downloads the data

```python
def filterLayer(layer):
    layername = layer.GetName()
    if layername in ('WBD_HU2', 'WBD_HU4', 'WBD_HU6'):
        return

    if layername not in ('NHDArea', 'NHDAreaEventFC'):
        print 'Unknown layer ' + layer.GetName()

    field = ogr.FieldDefn('_LAYER', ogr.OFTString)
    field.SetWidth(len(layername))
    layer.CreateField(field)

    for j in range(layer.GetFeatureCount())�:
        ogrfeature = layer.GetNextFeature()
        ogrfeature.SetField('_LAYER', layername)
        layer.SetFeature(ogrfeature)

    layer.ResetReading()
    return layer
```
Writing a good filterTags(attrs)

- When testing you want unknown fields to be kept
- Delete items from attrs as you convert them to OSM tags
- Delete fields which shouldn’t be converted to an OSM tag

```python
def filterTags(attrs):
    if not attrs: return
    tags = {}

    if '__LAYER' in attrs and attrs['__LAYER'] == 'wtrHydrantsSHP':
        # Delete the warranty date
        if 'WARR_DATE' in attrs: del attrs['WARR_DATE']

    if 'HYDRANT_NO' in attrs:
        tags['ref'] = attrs['HYDRANT_NO'].strip()
        del attrs['HYDRANT_NO']
    elif '__LAYER' in attrs and attrs['__LAYER'] == 'trnRoadCentrelinesSHP':
        # ... More logic ...

    for k,v in attrs.iteritems():
        if v.strip() != '' and not k in tags:
            tags[k]=v

    return tags
```
What not to include

- Duplications of geodata
  - SHAPE_AREA, SHAPE_LENGTH, latitude and longitude
- Unnecessary meta-data
  - e.g. username of the last person in the GIS department to edit the object
  - A single object ID can be useful but generally isn’t
- A good translation will often drop more than it includes
Identify the main field

- Convert to .osm with no translation
- View statistics about tags
- Easiest way is to open in JOSM, select -untagged, select the tags, paste into a text editor
- Need to look at a large area for this

<table>
<thead>
<tr>
<th>COMMENTS</th>
<th>LC_COST</th>
<th>RIGHTTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONDDATE</td>
<td>LEFTFROM</td>
<td>ROADCODE</td>
</tr>
<tr>
<td>CONDTN</td>
<td>LEFTTO</td>
<td>ROAD_NAME</td>
</tr>
<tr>
<td>DATECLOSED</td>
<td>LEGACYID</td>
<td>ROW_WIDTH</td>
</tr>
<tr>
<td>DATECONST</td>
<td>LOCATION</td>
<td>SNW_RTEZON</td>
</tr>
<tr>
<td>DESIGNTN</td>
<td>MATERIAL</td>
<td>SPEED</td>
</tr>
<tr>
<td>DISR_ROUTE</td>
<td>MRN</td>
<td>STATUS</td>
</tr>
<tr>
<td>FAC_ID</td>
<td>NO_LANE</td>
<td>STR_ROUTE</td>
</tr>
<tr>
<td>GCNAME</td>
<td>OWNER</td>
<td>TRK_ROUTE</td>
</tr>
<tr>
<td>GCPREDIR</td>
<td>PAV_DATE</td>
<td>WTR_DATE</td>
</tr>
<tr>
<td>GCROADS</td>
<td>PROJ_NO</td>
<td>WTR_PRIOR</td>
</tr>
<tr>
<td>GCSUFDIR</td>
<td>RC_TYPE</td>
<td>WTR_VEHCL</td>
</tr>
<tr>
<td>GCTYPE</td>
<td>RC_TYPE2</td>
<td>YR</td>
</tr>
<tr>
<td>GIS_ES</td>
<td>RD_CLASS</td>
<td>YR</td>
</tr>
<tr>
<td>GREENWAY</td>
<td>RIGHTFROM</td>
<td>YTD_COST</td>
</tr>
</tbody>
</table>

NOT INCLUDED IN ROADS TRANSLATION
NOT INCLUDED IN ANY TRANSLATION
MAIN FIELD
The main field

- A numeric field and a text field in this case
- Don’t trust field descriptions when writing OSM tagging
- Always verify!
  - Access Lane would be `highway=service` from the description but this would be wrong
  - Use imagery, surveys or other sources

<table>
<thead>
<tr>
<th>RC_TYPE</th>
<th>RC_TYPE2</th>
<th>Count</th>
<th>Tagging</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Road</td>
<td>11375</td>
<td><code>highway=?</code></td>
</tr>
<tr>
<td>1</td>
<td>Frontage Road</td>
<td>38</td>
<td><code>highway=residential</code></td>
</tr>
<tr>
<td>2</td>
<td>Highway Interchange</td>
<td>54</td>
<td><code>highway=motorway_link</code></td>
</tr>
<tr>
<td>3</td>
<td>Street Lane</td>
<td>20</td>
<td><code>highway=service</code></td>
</tr>
<tr>
<td>4</td>
<td>Access Lane</td>
<td>1442</td>
<td><code>highway=?</code></td>
</tr>
<tr>
<td>5</td>
<td>Railway</td>
<td>28</td>
<td><code>railway=rail</code></td>
</tr>
</tbody>
</table>
Looking at a value in more detail

- Should be carried out for each value, even if you think you’re sure on the tagging
- Look at all tags for just those matching the field value
- In this case search in JOSM for `RC_TYPE2="Road"`

<table>
<thead>
<tr>
<th>RD_CLASS</th>
<th>highway</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local</td>
<td>residential</td>
<td>8284</td>
</tr>
<tr>
<td>Major Collector</td>
<td>tertiary</td>
<td>1350</td>
</tr>
<tr>
<td>Arterial</td>
<td>primary secondary tertiary</td>
<td>1583</td>
</tr>
<tr>
<td>Provincial Highway</td>
<td>motorway primary</td>
<td>156</td>
</tr>
<tr>
<td>Translink</td>
<td>unclassified</td>
<td>1</td>
</tr>
</tbody>
</table>
Even more detail

- Gets very close to OSM tagging practice locally
- Loss of information with Arterial MRN=No and Major Collector both mapping to tertiary
- Does this matter in this case? No, road classifications require some judgment

<table>
<thead>
<tr>
<th>MRN</th>
<th>highway=</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>secondary</td>
<td>504</td>
</tr>
<tr>
<td>No</td>
<td>tertiary</td>
<td>1079</td>
</tr>
</tbody>
</table>
Dropping objects

- You may come across objects that you shouldn’t add to OSM
- In this case there are “paper roads” in the data
- Use filterFeature() to remove these

```python
def filterFeature(ogrfeature, fieldNames, reproject):
    if not ogrfeature:
        return

    index = ogrfeature.GetFieldIndex('STATUS')
    if index >= 0 and ogrfeature.GetField(index) in ('History', 'For Construction', 'Proposed'):
        return None

    return ogrfeature
```
Putting it all together

```python
def filterLayer(layer):
    layername = layer.GetName()

    field = ogr.FieldDefn('_LAYER', ogr.OFTString)
    field.SetWidth(len(layername))
    layer.CreateField(field)

    for j in range(layer.GetFeatureCount()):
        ogrfeature = layer.GetNextFeature()
        ogrfeature.SetField('_LAYER', layername)
        layer.SetFeature(ogrfeature)

    layer.ResetReading()
    return layer

def filterFeature(ogrfeature, fieldNames, reproject):
    if not ogrfeature:
        return

    index = ogrfeature.GetFieldIndex('STATUS')
    if index >= 0 and ogrfeature.GetField(index) in ('History', 'For Construction', 'Proposed'):
        return None

    return ogrfeature
```

- Code presented is a simplification and does not deal with all fields
- Filter features and layers
def filterTags(attrs):
    if not attrs: return
    tags = {}
    if '__LAYER' in attrs and attrs['__LAYER'] == 'trnRoadCentrelinesSHP':
        if 'COMMENTS' in attrs: del attrs['COMMENTS']
    if '__LAYER' in attrs:
        if 'DATECLOSED' in attrs: del attrs['DATECLOSED']
        # Lots more to delete
    if 'NO_LANE' in attrs:
        tags['lanes'] = attrs['NO_LANE'].strip()
        del attrs['NO_LANE']
    if 'RC_TYPE' in attrs and attrs['RC_TYPE'].strip() == '0':
        # Normal roads
        del attrs['RC_TYPE']
    if 'RC_TYPE2' in attrs:
        del attrs['RC_TYPE2']
    if 'RD_CLASS' in attrs and attrs['RD_CLASS'] == 'Local':
        tags['highway'] = 'residential'
    elif 'RD_CLASS' in attrs and attrs['RD_CLASS'] == 'Major Collector':
        tags['highway'] = 'tertiary'
    elif 'RD_CLASS' in attrs and attrs['RD_CLASS'] == 'Arterial':
        if 'ROAD_NAME' in attrs and attrs['ROAD_NAME'] in ('King George Blvd', 'Fraser Hwy'):
            tags['highway'] = 'primary'
        else:
            if 'MRN' in attrs and attrs['MRN'] == 'Yes':
                tags['highway'] = 'secondary'
            else:
                tags['highway'] = 'tertiary'
        del attrs['RD_CLASS']
    elif 'RD_CLASS' in attrs and attrs['RD_CLASS'] == 'Provincial Highway':
        # Special-case motorways
        if 'ROAD_NAME' in attrs and attrs['ROAD_NAME'] in ('No 1 Hwy', 'No 99 Hwy'):
            tags['highway'] = 'motorway'
        else:
            tags['highway'] = 'primary'
        del attrs['RD_CLASS']
    elif 'RD_CLASS' in attrs and attrs['RD_CLASS'] == 'Translink':
        tags['highway'] = 'unclassified'
        del attrs['RD_CLASS']
    elif 'RD_CLASS' in attrs and attrs['RD_CLASS'] == 'Local' and attrs['RD_CLASS'] == 'Major Collector':
        tags['highway'] = 'tertiary'
    elif 'RD_CLASS' in attrs and attrs['RD_CLASS'] == 'Arterial' and attrs['RD_CLASS'] == 'Major Collector':
        if 'ROAD_NAME' in attrs and attrs['ROAD_NAME'] in ('King George Blvd', 'Fraser Hwy'):
            tags['highway'] = 'primary'
        else:
            if 'MRN' in attrs and attrs['MRN'] == 'Yes':
                tags['highway'] = 'secondary'
            else:
                tags['highway'] = 'tertiary'
        del attrs['RD_CLASS']
    if 'COMMENTS' in attrs:
        del attrs['COMMENTS']
    if 'DATECLOSED' in attrs:
        del attrs['DATECLOSED']
    # More logic
    elif '__LAYER' in attrs and attrs['__LAYER'] == 'trnTrafficSignalsSHP':
        # More logic
        for k,v in attrs.iteritems():
            if v.strip() == '' and not k in tags:
                tags[k] = v
    return tags
Any questions?
Credits

- Background by Stamen Design under CC BY 3.0
- Surrey Data © 2012 City of Surrey under PDDL 1.0