OGR2OSM

A powerful tool for converting geodata to .osm format

SOTM-US 2012

About

- What ogr2osm can do for you
- How ogr2osm works
- A case study of a data conversion
- Why care about converting?

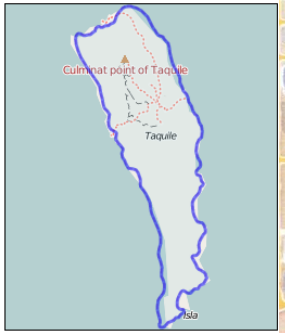
Why care? To avoid this

Way: 141322228

« 141322227 | 141322229 »

Edited at: Thu, 15 Dec 2011 10:50:49 +0000
Edited by: ovruni
Version: 1
In changeset: 10122385
Comment: Modificando fronteras distritales de Puno
Tags:
AREA_MINAM = 1394.26
DCTO = LEY
FECHA = 09/04/1965
IDDIST = 210103
IDDPTO = 21
IDPROV = 2101
LEY = 15489
NOMBDEP = PUNO
NOMBDIST = AMANTANI
NOMBPROV = PUNO
NOM_CAP = AMANTANI
OBJECTID = 176
SHAPE_AREA = 0.00116706227
SHAPE_AR_1 = 0.00116706194
SHAPE_LENG = 0.23321000322
SHAPE_LE_1 = 0.23321042956

Nodes:
1546977627
1546977638
1546977664
1546977688
1546977708



View area on larger map
Edit area
View way on larger map
Edit way

History

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

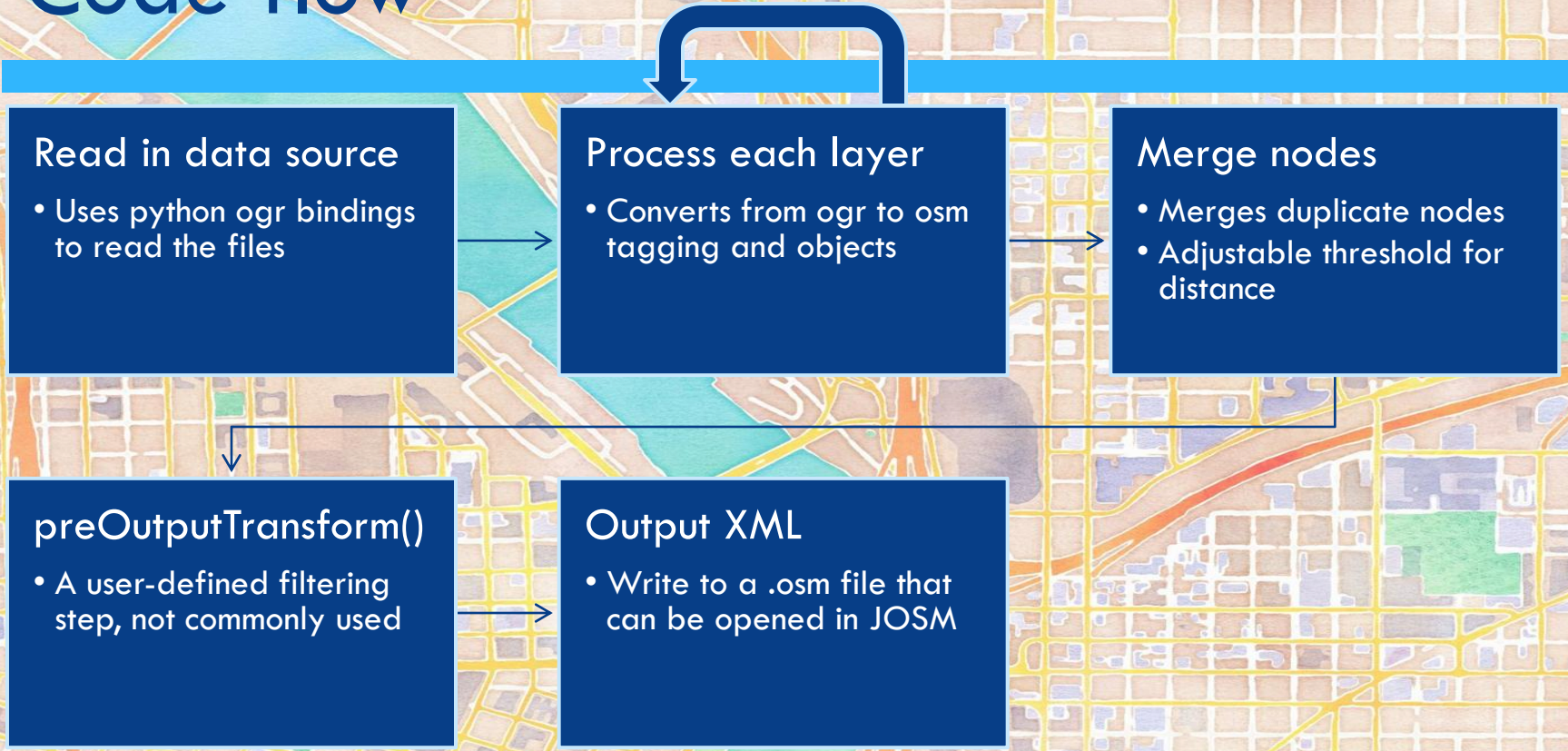
Features

- 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

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

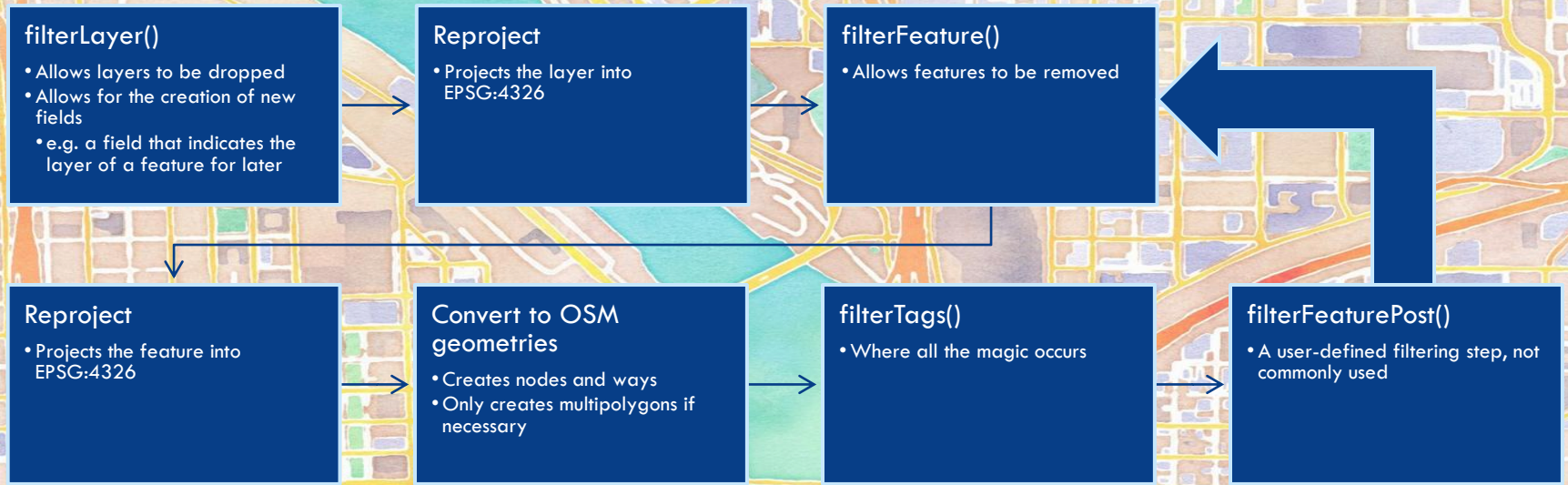
preOutputTransform()

- A user-defined filtering step, not commonly used

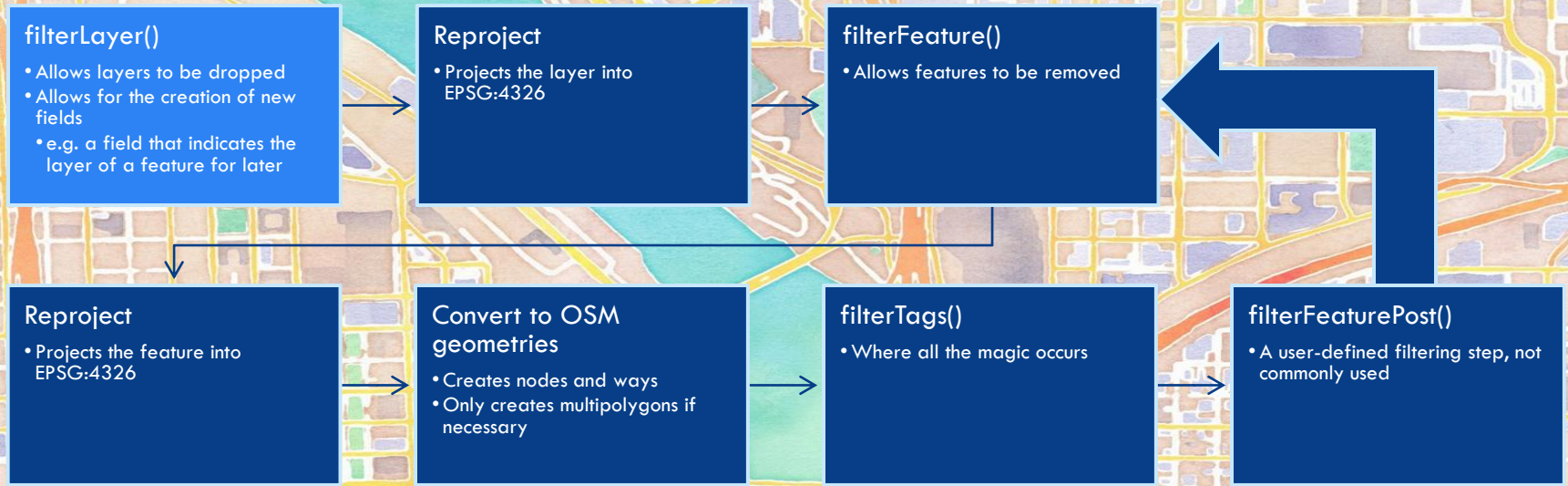
Output XML

- Write to a .osm file that can be opened in JOSM

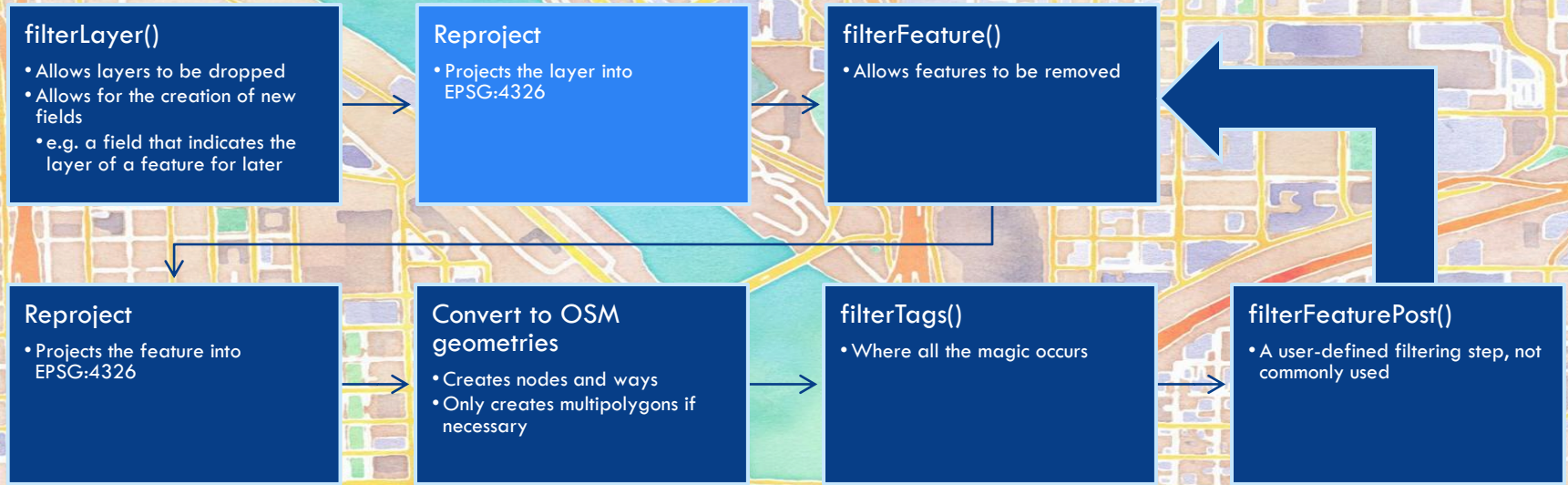
Layer processing



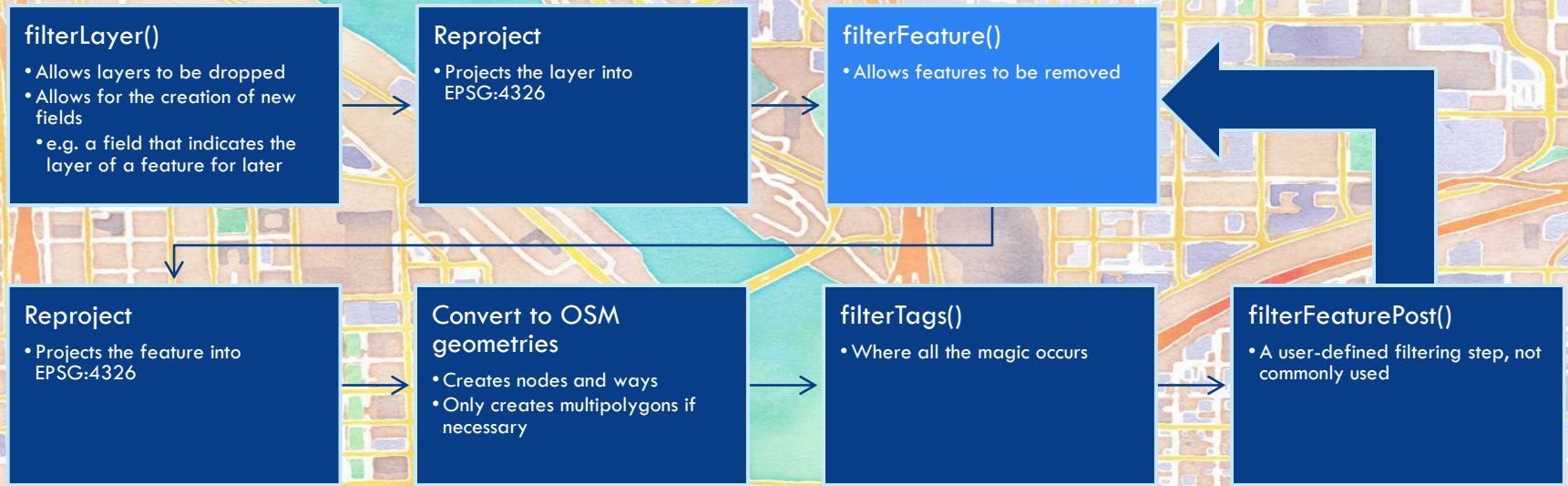
Layer processing



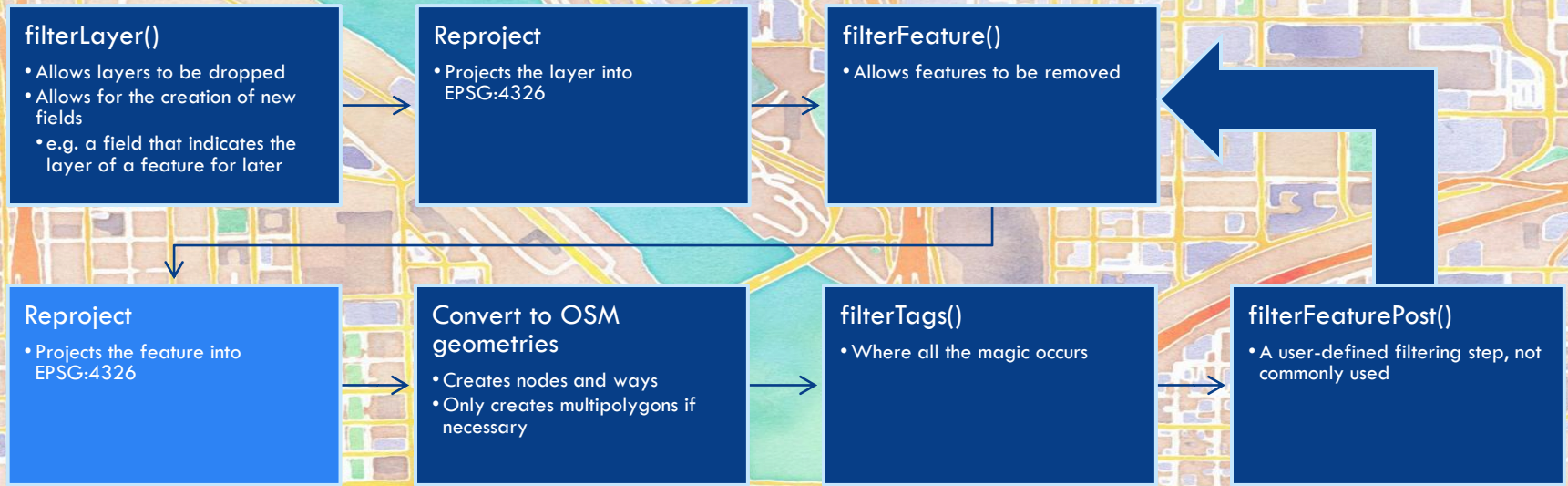
Layer processing



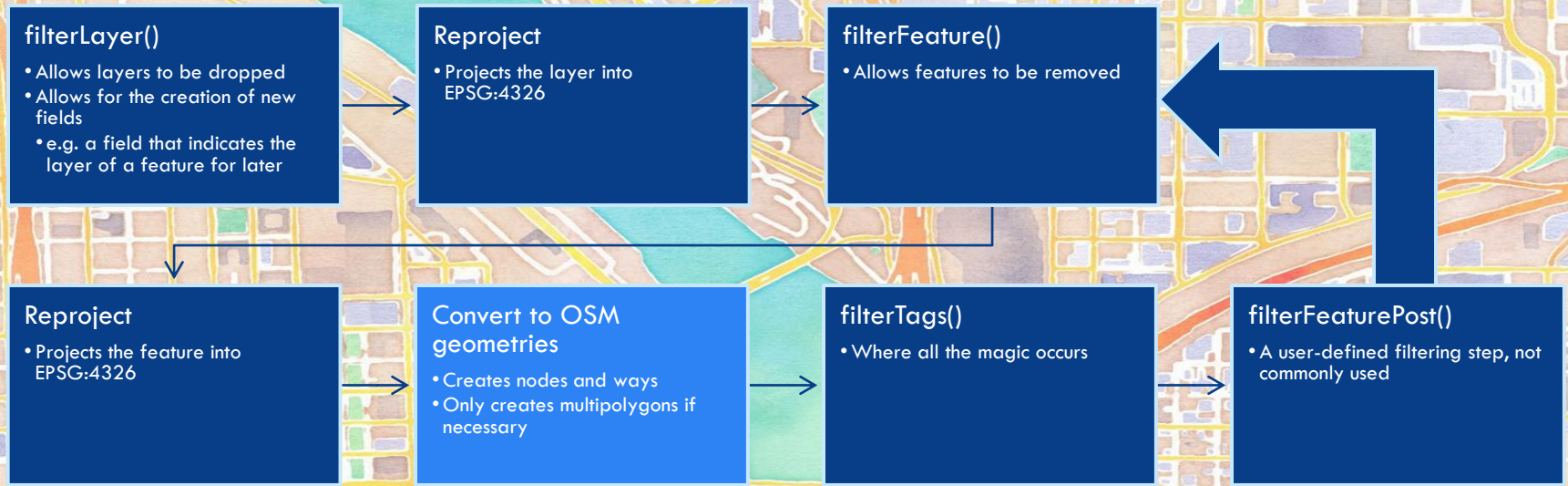
Layer processing



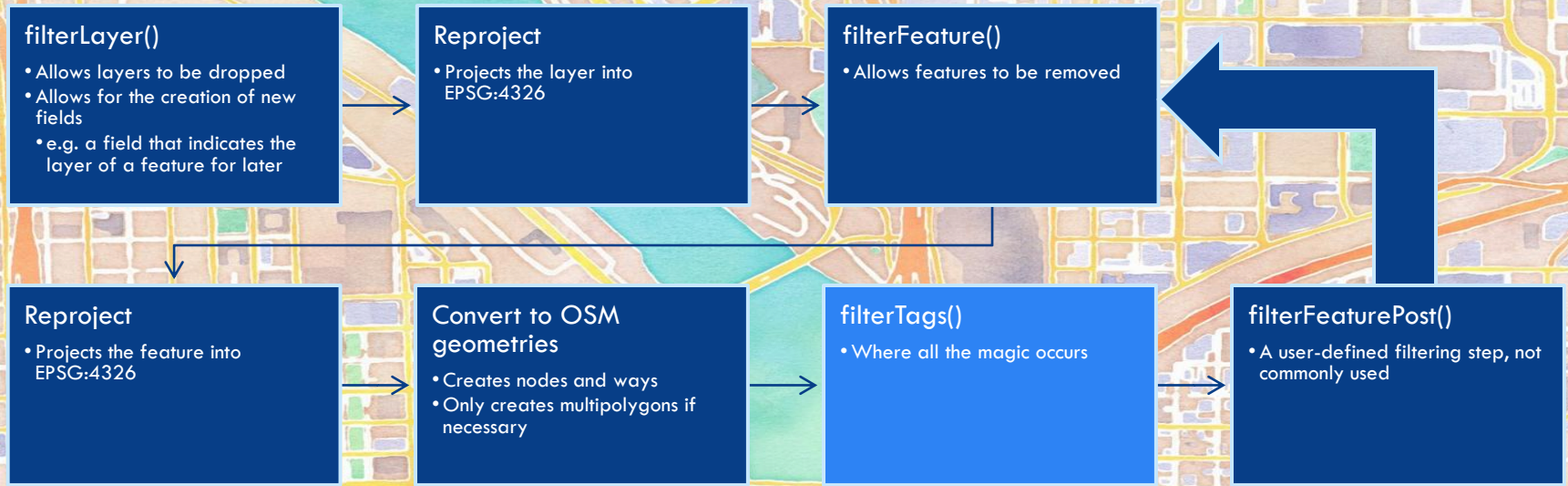
Layer processing



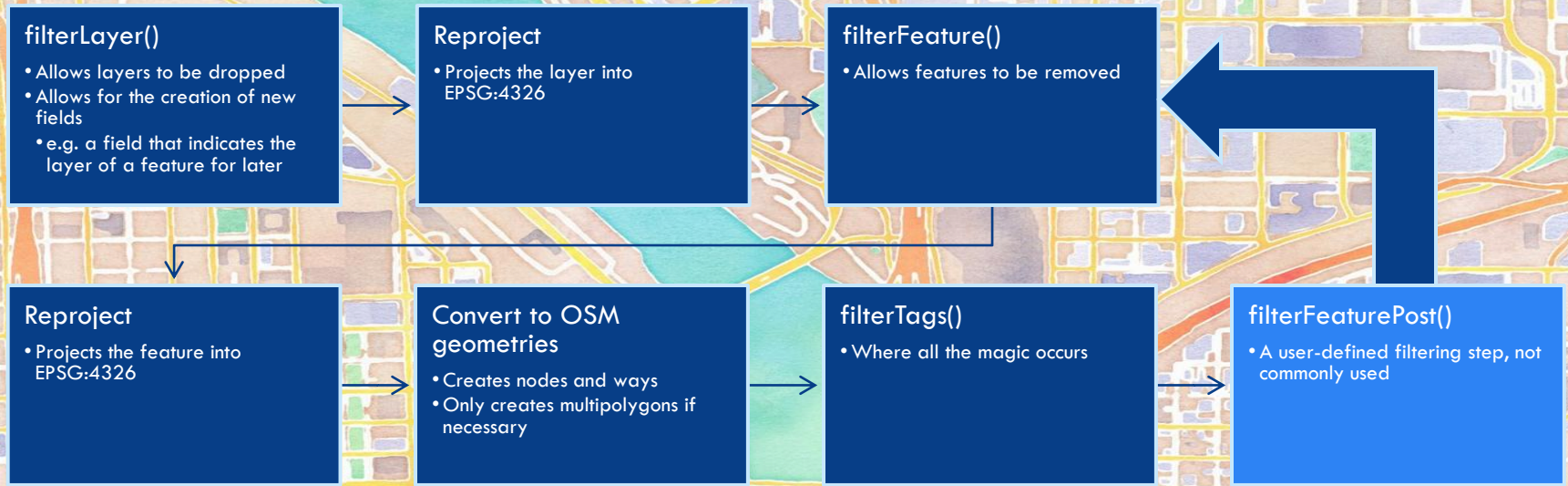
Layer processing



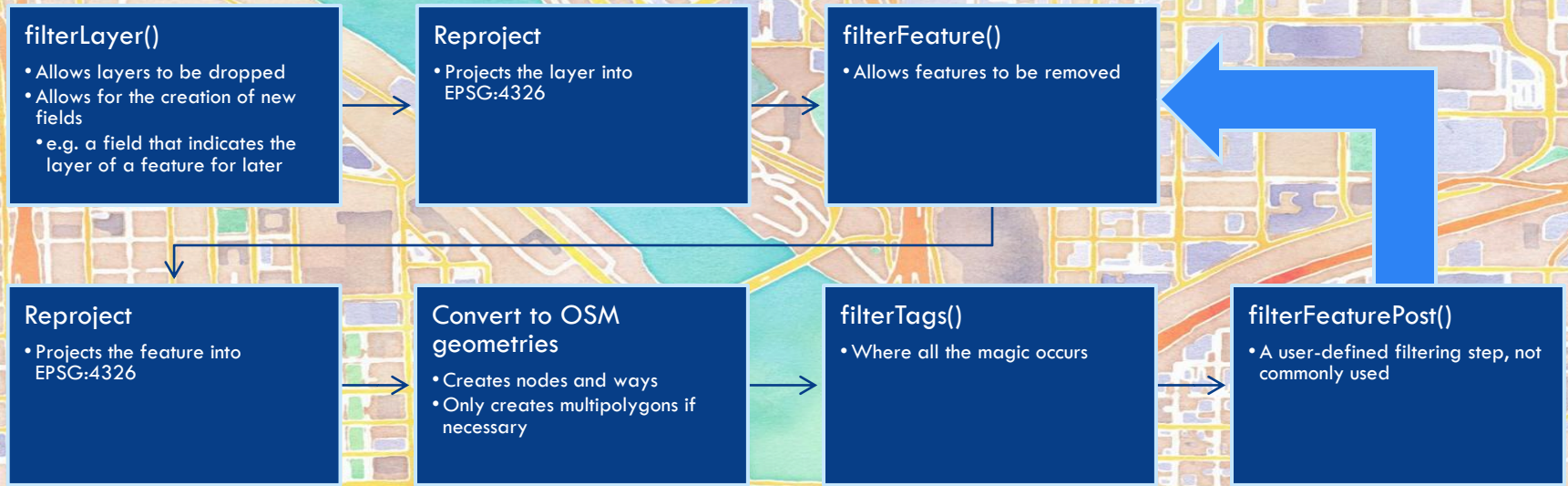
Layer processing



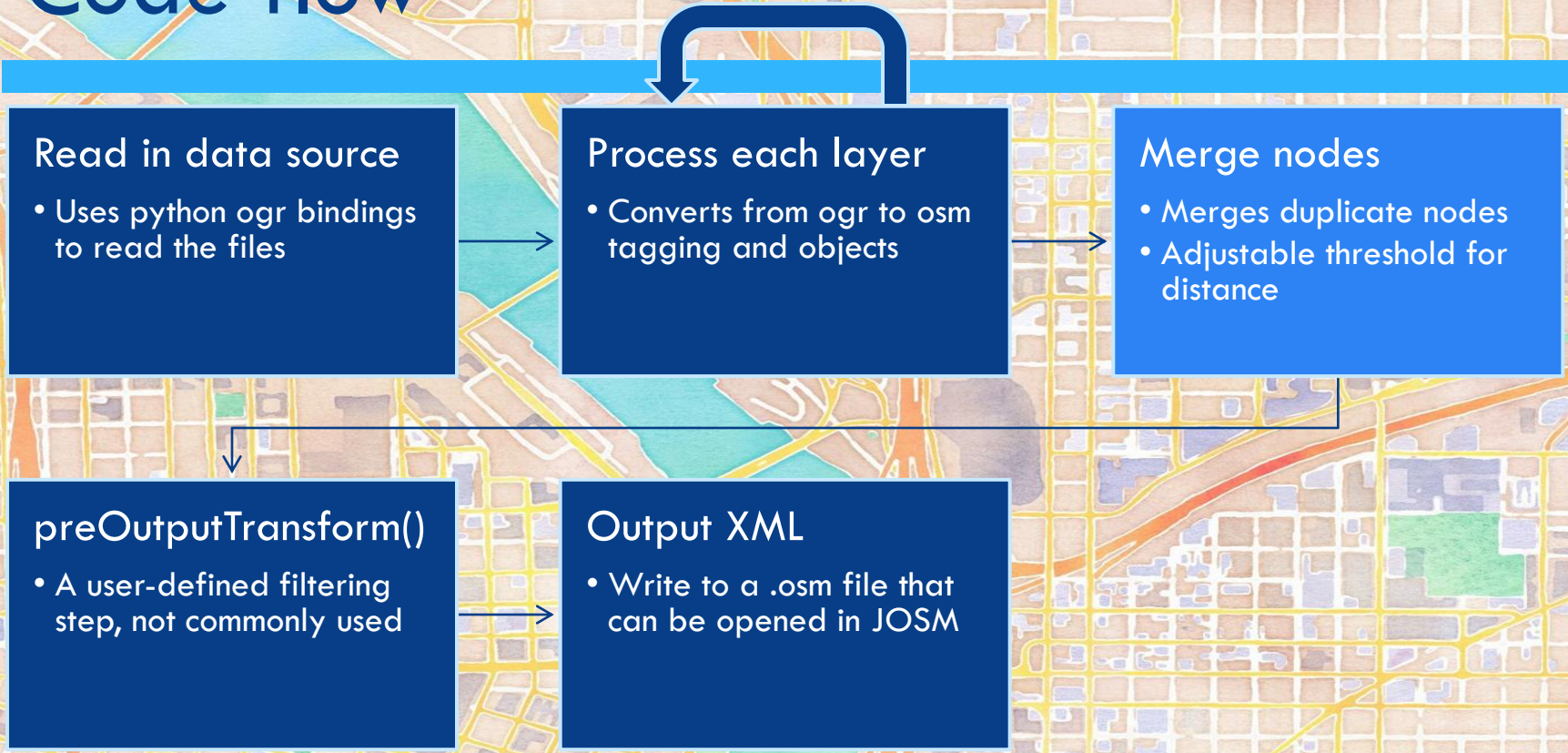
Layer processing



Layer processing



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 km^2
- 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

Drop layers

- 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

```
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

```
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

COMMENTS	LC_COST	RIGHTTO
CONDDATE	LEFTFROM	ROADCODE
CONDTN	LEFTTO	ROAD_NAME
DATECLOSED	LEGACYID	ROW_WIDTH
DATECONST	LOCATION	SNW_RTEZON
DESIGNTN	MATERIAL	SPEED
DISR_ROUTE	MRN	STATUS
FAC_ID	NO_LANE	STR_ROUTE
GCNAME	OWNER	TRK_ROUTE
GCPREDIR	PAV_DATE	WAR_DATE
GCROADS	PROJ_NO	WTR_PRIOR
GCSUFDIR	RC_TYPE	WTR_VEHCL
GCTYPE	RC_TYPE2	YR
GIS_ES	RD_CLASS	YTD_COST
GREENWAY	RIGHTFROM	

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

RC_TYPE	RC_TYPE2	Count	Tagging
0	Road	11375	highway=?
1	Frontage Road	38	highway=residential
2	Highway Interchange	54	highway=motorway_link
3	Street Lane	20	highway=service
4	Access Lane	1442	highway=?
5	Railway	28	railway=rail

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"`

RD_CLASS	highway=	Count
Local	residential	8284
Major Collector	tertiary	1350
Arterial	primary secondary tertiary	1583
Provincial Highway	motorway primary	156
Translink	unclassified	1

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

MRN	highway=	Count
Yes	secondary	504
No	tertiary	1079

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

```
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


```
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

Putting it all together


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

  if '__LAYER' in attrs and attrs['__LAYER'] ==
      'trnRoadCentrelinesSHP':
    if 'COMMENTS' in attrs: del attrs['COMMENTS']
    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'
        del attrs['RD_CLASS']
      elif 'RD_CLASS' in attrs and attrs['RD_CLASS'] == 'Major Collector':
        tags['highway'] = 'tertiary'
        del attrs['RD_CLASS']
      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']
        else:
          l.error('trnRoadCentrelinesSHP RC_TYPE=0 logic fell through')
          tags['fixme'] = 'yes'
          tags['highway'] = 'road'
    elif 'RC_TYPE' in attrs and attrs['RC_TYPE'].strip() == '1':
      # 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