Custom OpenStreetMap Export Platform

User & Technical Documentation

<frederik.ramm@geofabrik.de>

2012-04-03

Table of Contents

1 Background

OpenStreetMap is used as a versatile tool for recording various bits of information relevant for humanitarian work. HOT has established, or is about to establish, tool chains that facilitate entering such data into OSM, as well as creating specialist maps from OSM data.

The weak link in the process is getting data out of OSM for further processing. Currently, you can either request raw data from the OSM server directly which is size-limited and requires post-processing e.g. into shape files; or you can download ready-made extracts from providers such as Geofabrik but these are limited to fixed areas, formats, and data fields.

This offer is about creating a special download service that creates up-to-data extracts in various formats on demand, using data fields requested by the user.

The service must be able to cover various areas of HOT involvement (called AOIs in the following), but not the whole planet. Export jobs will typically cover a larger area than allowed by the existing OpenStreetMap "export" function but will not be on the scale of whole countries. Initially, the service will be rolled out for the AOI of Haiti only.

2 Using the Service

Generating custom extracts is easy. Simply point your browser to http://hot-exports.geofabrik.de and be welcomed by the start page which gives you a few instructions.

2.1 Generating New Export Jobs
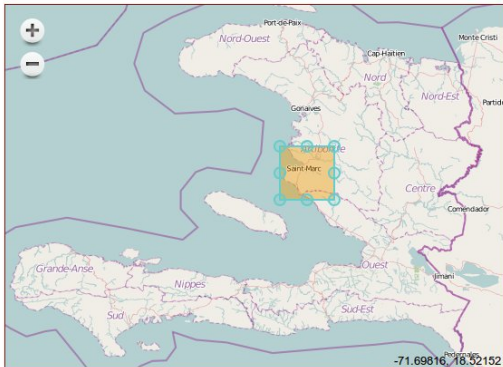
Click on "new job" to create a new export job:
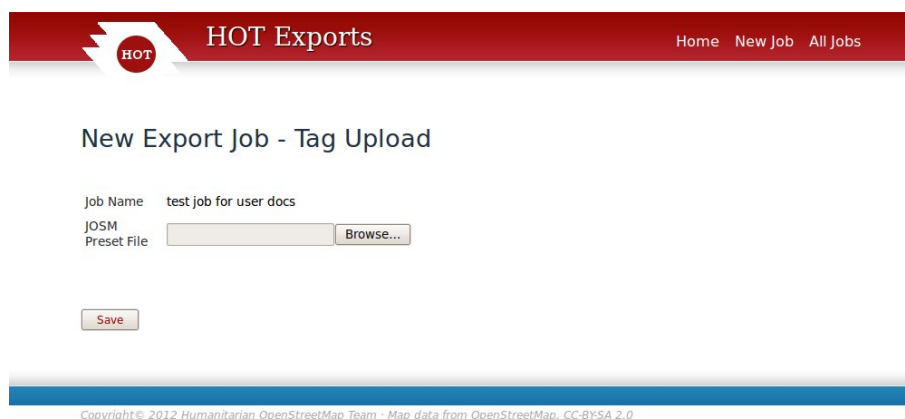
Enter a job name and description. There are two ways to select an area on the map; either simply

zoom to the area of interest and then click "Save" (the whole map area will be used), or click on "select smaller area" and draw a rectangle on the map.

You can also enter coordinates for the bounding box manually.
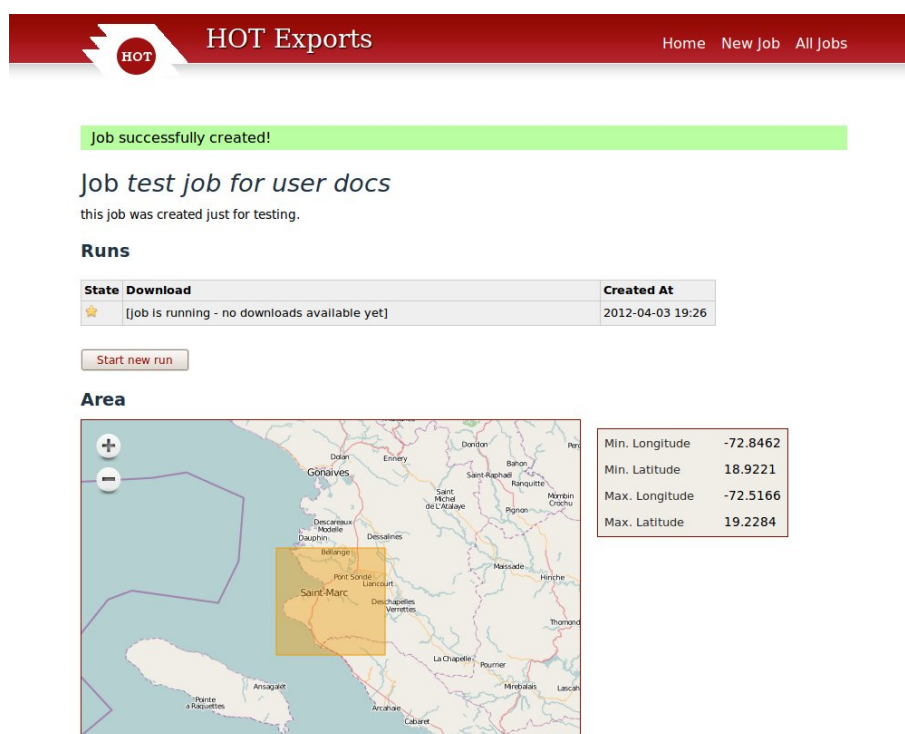
In the next wizard step, you can upload a JOSM preset file that describes the tags you want exported:

If you don't upload a document, a set of standard tags will be exported.

After hitting "save", your job is now in the system and waiting to be processed.

As long as the status is "new" or "running", this page will automatically reload every so often. The usual run time for a new job is just 20 or 30 seconds, but if multiple jobs are started at the same time things might take a bit longer. When the job has completed, the detail page will change and present a list of downloadable files:

Click on any of the files to download them to your computer.

2.2 Re-Running Existing Jobs

At any time, you can request a re-run of an existing job. This will not change any of the parameters, but it will create a new set of export files.

Simply click on "all jobs" in the navigation bar, and select the job you want from the list. (Enter a keyword in the search box to shorten the list if desired.)

Click on the job to go to its detail page, and then click on "start new run". This re-runs the job, and after a while you will be able to download the newly created data files.

2.3 File Formats

The application currently supports the following output formats:

- ESRI Shapefile: A zip archive containing four shape files (points, lines, polygons, roads) together with a matching "crosswalk" text file that describes the original names of shortened columns.

- Spatiallite file: A SpatialLite database file.

- PostGIS dump: A SQL file suitable for loading into a PostGIS database.

- SQLite: A SQLite database, with geometries written as WKT in text columns.

- OSM source file: The OpenStreetMap source data that formed the basis of this export, in PBF format, suitable for processing with Osmosis, osm2pgsql, or mkgmap. This file does not have tag filtering applied, i.e. it has all data from OSM for the area and not just what was selected for export.

3 Architecture

The application contains of three separate components. One is a Ruby on Rails application for user interaction. It allows the user to create jobs and view the results. The second is an independent, Osmosis-based OSM data updater that makes sure we always have current OSM data on disk. The third is a request processor that takes requests from the database and OSM data from the updater and creates the desired extracts.

The following illustrations shows the components working together:
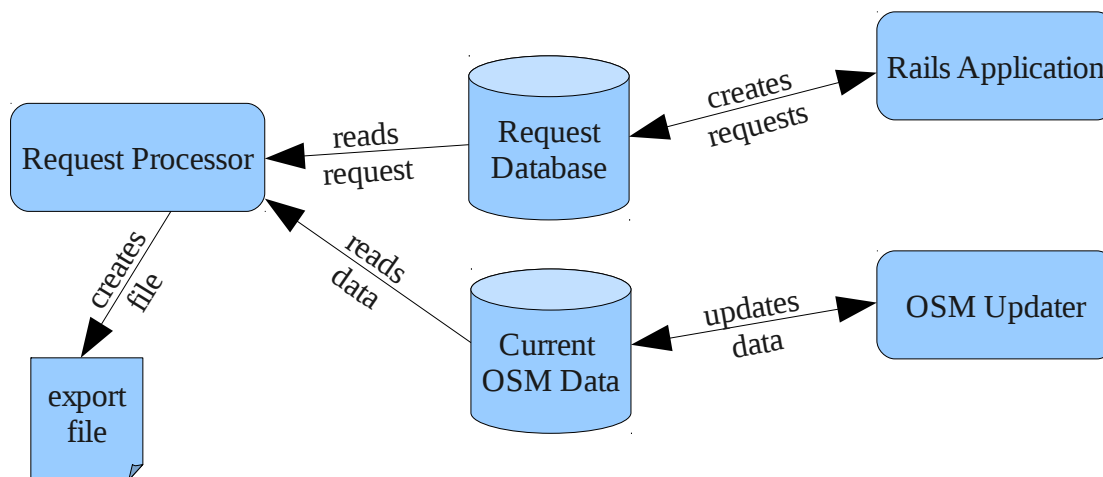
3.1 Rails Application

The rails application is a simple, standard Rails 3 application that revolves mainly around the classes "Job" and "Run". A "Job" has a name, a description, and a series of tags for exporting. In addition, a "Job" usually has one or more "Runs", which have a status and an associated list of output files.

When a job is created, a run is also started automatically, but users can request additional runs for the same job later. Runs newly created have the status "new" and this status is never changed by the application; only the external request processor changes that.

The application contains an XML parser for JOSM preset files and parses them into the database when they are uploaded.

3.2 OSM Updater

The OSM Updater is a simple shell script that downloads an update file from the OpenStreetMap server and applies it to a series of locally held data extracts:

```
#!/bin/sh

NOW=`date +%s`

sleep 120 &

cd /home/hot/
exec >> log/update.log 2>&1

osmosis/bin/osmosis --rri var/osmosis-status-dir --simc --wxc var/diffs/$NOW.osc || exit 1

for i in var/extracts/*osm.pbf
do
        POLY=etc/polygons/`basename $i .osm.pbf`.poly
        if [ -f $POLY ]
        then
                osmosis/bin/osmosis --rxc var/diffs/$NOW.osc --read-pbf $i --ac --bp
file=$POLY clipIncompleteEntities=true --write-pbf $i.new && mv $i.new $i
        fi
done
```

The updater is built so that it simply processes all data extracts on disk by applying the downloaded update to them.

Crucially, Osmosis is asked to cut out the area polygon after the update is applied, else the data file would slowly accumulate edits all over the world.

The OSM updater is usually run once every 120 seconds in a loop, secured by a cron job.

3.3 Request Processor

The request processor consists of two parts. The core is called "custom_data_export" and is written in C++ based on the Osmium library. It performs the single task of taking an OSM data file and converting it into an SQLite output file that conforms roughly to osm2pgsql specifications. It can take a list of desired columns for exporting in an environment variable.

This core component is wrapped in a Perl script that does the following:

- fetch pending request from database, if any
- call Osmosis to cut out bounding box from current data file
- call custom_data_export to convert to SQLite file
- call ogr2ogr, a number of times, to generate various output files
- update the database with a list of created files and new status.

The request processor is usually run once every five seconds to check for new requests, also secured by a cron job.

4 Setup and Modifications

Information in this chapter pertains to future installations on different servers; on hot-export.geofabrik.de, these steps are already in place.

4.1 Prerequisites

The system requires the following external components:

- Apache web server with mod_passenger (or other suitable server)

- Rails, Ruby, and gems as per bundle

- Postgres database

- zip (for creating zipped shape files)

- Perl with libdbd-xbase-perl module

- gdal utility

- protobuf library

- Java runtime

4.2 Directory Setup

The shell/perl scripts assume that things are installed in /home/hot.

/home/hot/etc/plygons – .poly files for each area (currently Haiti only)

/home/hot/bin – scripts and custom_data_export binary

/home/hot/osmosis – Osmosis installation

/home/hot/var/extracts – place for raw OSM data (currently only haiti.osm.pbf)

/home/hot/var/osmosis-status-dir – place for Osmosis replication status

/home/hot/var/diffs, /home/hot/var/runs, /home/hot/log – writable empty directories

/home/hot/web – Rails application

4.3 Database and Web Server Setup

The system expects to have a database user named hot_export (password hot_export) who is allowed to create database. Then, the database can be created with rake db::create (for production mode) and db::migrate after that.

The web server needs to be pointed to the appropriate root directory for the application, and an Alias needs to be created so that users can download the files created by the backend:

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost

    DocumentRoot /home/hot/web/public
```

```
    <Directory />
          Options Indexes FollowSymLinks
          AllowOverride None
    </Directory>

    Alias /download /home/hot/var/runs
    ErrorLog /var/log/apache2/error.log
    LogLevel warn

    CustomLog /var/log/apache2/access.log combined

</VirtualHost>
```

4.4 OSM Updater and Request Processor Setup

The OSM updater must be primed with an initial data extract and matching Osmosis replication settings. This can be done by cutting out the desired polygon out of a full planet file or Geofabrik extract, and then initializing the Osmosis replication directory with --rrii and setting the state.txt file with the help of http://toolserver.org/~mazder/replicate-sequences/.

For both the updater and the request processor, a cron job needs to be created so that they are restarted if they should die or the machine should be rebooted:

```
# m h  dom mon dow   command
* * * * * /home/hot/bin/check_export.sh
* * * * * /home/hot/bin/check_update.sh
```

It is suggested to run the backend application under the user "hot". Make sure that the directory /home/hot/var/runs is readable by www-data though, so that downloads are possible.

4.5 Adding File Formats

Adding new output file formats is relatively straightforward in the request processor. Say you wanted to add a basic KML output; simply edit export_request_processor.pl and find the lines that say

```
    # call ogr2ogr to make Spatiallite file
    mysystem("ogr2ogr -overwrite -f 'SQLite' -dsco SPATIALLITE=YES $OUTPUT_PATH/
$rid/extract.spatiallite $OUTPUT_PATH/$rid/extract.sqlite");
    addfile($rid, "extract.spatiallite", "Spatiallite file");
```

The "mysystem" line runs an Unix utility, and the "addfile" line makes sure the resulting file is probed for its size and added to the output table in the database so that the user gets a download link for that later.

To add KML processing, add a block like this:

```
    # call ogr2ogr to make KML file
    mysystem("ogr2ogr -overwrite -f 'KML' -$OUTPUT_PATH/$rid/extract.kml $OUTPUT_PATH/
$rid/extract.sqlite");
    addfile($rid, "extract.kml", "KML File");
```

That's all.