JOSM Separate Data Store

Documentation

2013-03-01, Version 2

by Frederik Ramm <ramm@geofabrik.de>

v2. additions Tim Waters <tim@geothings.net>

Table of Contents

1 Background

HOT are working with OSM data in Indonesia with the aim of mapping the whole country for disaster risk reduction. Most of the information acquired is suitable for inclusion in OSM ("public information"), but some items, while useful for the purpose of disaster risk reduction, may not be for public dissemination e.g. to protect the privacy of residents or for similar reasons ("private information").

HOT are therefore looking to pair OSM with a second, HOT-operated data store where such private information would be stored. This requires modifications to the editing software (JOSM) as well as the implementation of a suitable database back-end.

The current main use cases revolve around information anchored to buildings, i. e. buildings will be mapped in OSM with basic structural information, and additional information about the building will be recorded separately.

2 Solution

Geofabrik has developed a plug-in for the JOSM editor that can, after public data has been loaded from the OpenStreetMap server, query a second, HOT-operated server, sending the OSM IDs of all loaded objects to that server. The server replies with all private information recorded against these objects.

The plugin mixes public and private information so that all editing steps, presets, etc. are exactly the same as in normal JOSM operation; to the user, public and private information look exactly the same and are handled the same, except that private information uses a special, configurable prefix, e.g. “hot:...”.

On upload, the plugin again separates data into public and private streams, uploading public data to OpenStreetMap, and private data to the HOT server.

The server itself has a web interface that allows the addition of data to existing OSM objects as a lighter alternative to using the JOSM editor; the web interface does not however allow modifications in OSM data.

3 Server Basics

3.1 Architecture

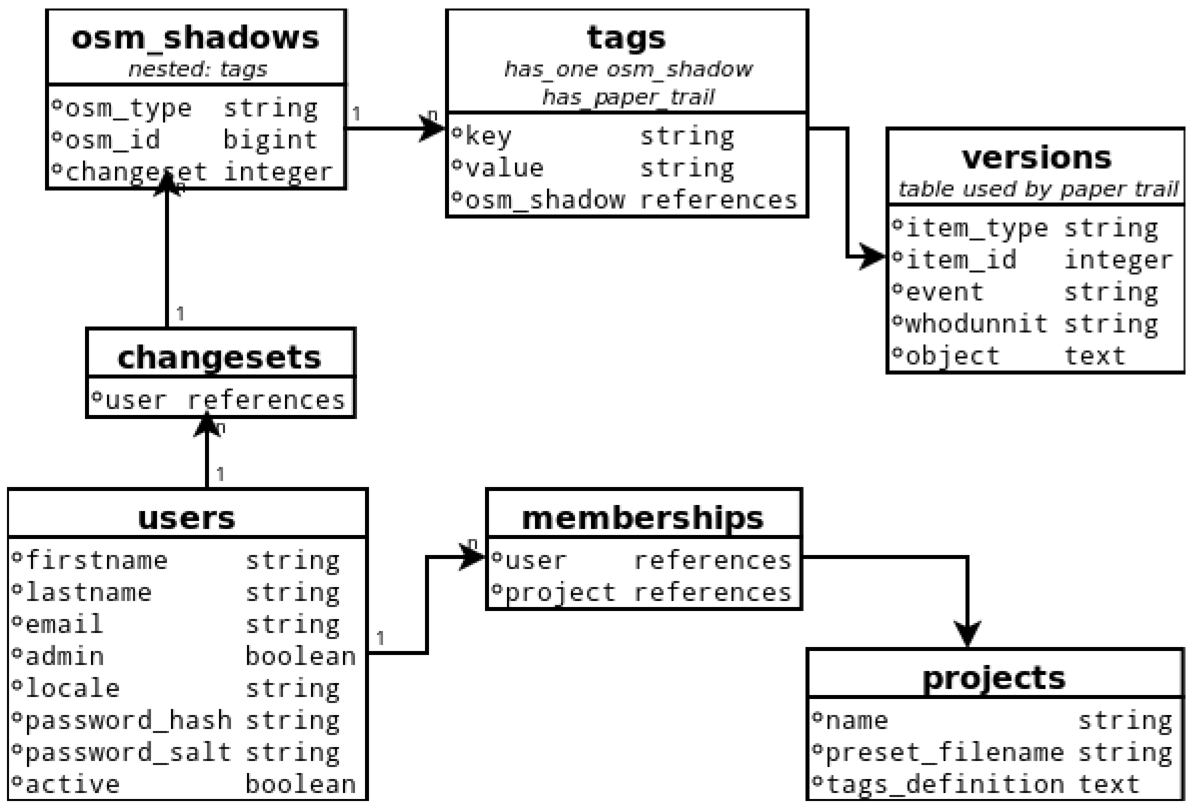
The server is built as a standard Ruby on Rails application with a REST-like interface. Data is stored in a PostgreSQL database, and the server can be run in any typical Rails deployment mode. We have set up one instance of the server at datastore.hotosm.org which uses the “Passenger” Apache module for deployment.

3.2 Data Model

Data on the private server is keyed against OSM object type and ID, not OSM version number. There is a risk that such links are severed by somebody deleting or re-adding an object in OSM but it has been decided to ignore that risk for now, and deal with possible problems later.

The data model is centered around the class “osm_shadow” which serves as a link to, or placeholder for, an existing object in the OpenStreetMap database. Every osm_shadow object has an osm_type/osm_id attribute pair which precisely identify one object in the OpenStreetMap domain. (It is possible for the object in OpenStreetMap to be deleted or, rarely, re-purposed so that the link would be dangling. It has been decided to ignore that potential risk for now.). More than one osm_shadow object can have the same osm_type/osm_id attribute pair. In this way you can have multiple osm_shadows for each object, each with their own collection of tags.

An “osm_shadow” can have any number of tags; just like in OSM, such tags are simple key-value combinations of arbitrary strings.



The “tags” model tracks changes made - they contain every edit made, and they can be used to find out who made which changes, to view each change and the ability to revert any changes made. This functionality utilises the paper_trail ruby gem library.

For recording time and user information, osm_shadow objects are linked to a timestamped “changeset” object which in turn links to an “user” object. A “changeset” encapsulates one or a series of changed made by one user in one session. In contrast to OSM, where changesets have to be explicitly opened, the SDS server will automatically generate a changeset to go along with a web session or a JOSM upload.

There is also a “projects” table that lists the names of projects that this SDS instance is used for. The idea is that different projects might use different sets of extra tags, and have different kinds of web input forms; a user recorded as being with project A might see a different representation of things than a user who is with project B. A project has a definition of the type of tags it accepts (stored as JSON currently). An admin can assign users to projects – a user will only see those tags that belong to the projects the user is part of.

3.3 Deploying and Running

There are two options for deploying the application, depending of level of knowledge of Ruby and Rails applications or server admin know how.

3.3.1 Manual deployment

To deploy the SDS server on a new machine, you will first have to install Ruby 1.8 and Rails 3.1.1, as well as PostgreSQL:

```
apt-get install ruby1.8 rubygems1.8 postgresql-9.0
gem install rails 3.1.1
```

Create a PostgreSQL user and optionally give it a password:

```
createuser hot_josm
```

Modify PostgreSQL to allow logins by adding this line to /etc/postgresql/9.0/main/pg_hba.conf

```
local    all  all  trust
```

Check out, or unpack, the SDS rails repository into a suitable directory, and install the necessary gems.

```
bundle install
```

then create the database and set up the schema:

```
RAILS_ENV=production rake db:create
RAILS_ENV=production rake db:migrate
```

You will want to manually add an admin user to bootstrap,

```
psql hot_josm_production -c "insert into users (firstname,lastname,email,
plain_password,active,admin) values
('my','name','my@email.com','secret',true,true);"
```

Or, if the console is not directly accessible, create an admin user using a rake task:

```
db:create_admin[Admin,Adminson,admin@example.com,changemeplease] `
```

Deploy the Rails application using your preferred means (a Capistrano script is included) , e.g. through Apache mod_passenger. For a Passenger setup, you will have to install the libapache2-mod-passenger package and point your DocumentRoot to the webinterface/public directory of the checked out application. You will also have to add the line

```
RailsEnv production
```

to either the host configuration or to your /etc/apache2/mods_enabled/passenger.load file.

Instead of properly deploying the application, you can also simply run

```
rails server
```

to test the application on http://localhost:3000.

3.3.2 Automatic Server provisioning and Deployment

The quickest way to get the system working is to use the automatic server provisioning and deployment scripts. This assumes you have access to a dedicated bare bones server where you want to just install the SDS application to. It needs a clean Debian based / Ubuntu (10.04, 11.10, 12.04) server

These instructions allow you fully remotely provision a new server with all the libraries, database, and webserver. It will also deploy the SDS application to this remote server and start it for you. It will install:

Postgresql database
Ruby + all the gems and associated libraries
Nginx webserver
Unicorn (to run the Rails app on the server)

If you are familiar with deploying rails applications, or already have a server set up to run a rails application, then you can still use the usual Capistrano deploy calls - you may need to do some configuring of these, however.

3.3.3 Requirements

There are not that many requirements for deploying it from your local machine to a clean new server.

Git, Ruby 1.8.7, rubygems, capistrano, bundler:

For example, here are commands to install this on an Ubuntu LTS 12-04 desktop,

```
sudo apt-get install rubygems
sudo gem install capistrano –version=2.13.5
sudo gem install bundler
```

On the server, it should be a new empty server, running a new Debian based system. Ideally an Ubuntu server. The firewall should be configured to allow port 80 etc. It should have a user which is configured to have sudo access, and is able to ssh in. You should know the username and password of this user. You will be prompted by the scripts for the password at times.

3.3.4 Configuration Steps

Locally, you should get the source

```
git clone git://github.com/hotosm/sds-server.git
```

in config/deploy.rb

change the line that says

```
server "192.168.15.142"
```

and change it to the IP address, or the domain of your new server (e.g. server “sds.geothings.net”)

Find the line that says

```
set :user, "tim"
```

and change it to the username for the user you set up on your new server

Further optional configurations can be found in the documentation directory

3.3.5 Deploy

1. Install

```
cap deploy:install
```

This installs postgres, unicorn, nginx, ruby and all the libraries to the server.

2. Setup

During this step you will be prompted for a new postgresql user's password.

```
cap deploy:setup
```

3. Deploy

This sets up the database, links for capistrano, sets up directories, etc

```
cap deploy:cold
```

4. Create Admin User

This deploys the actual application onto the server

```
cap deploy:create_admin
```

This creates the first admin user, with the email of admin@example.com and the password of changemeplease. This account should be logged in and the password changed to something more secure and memorable.

3.4 User Management

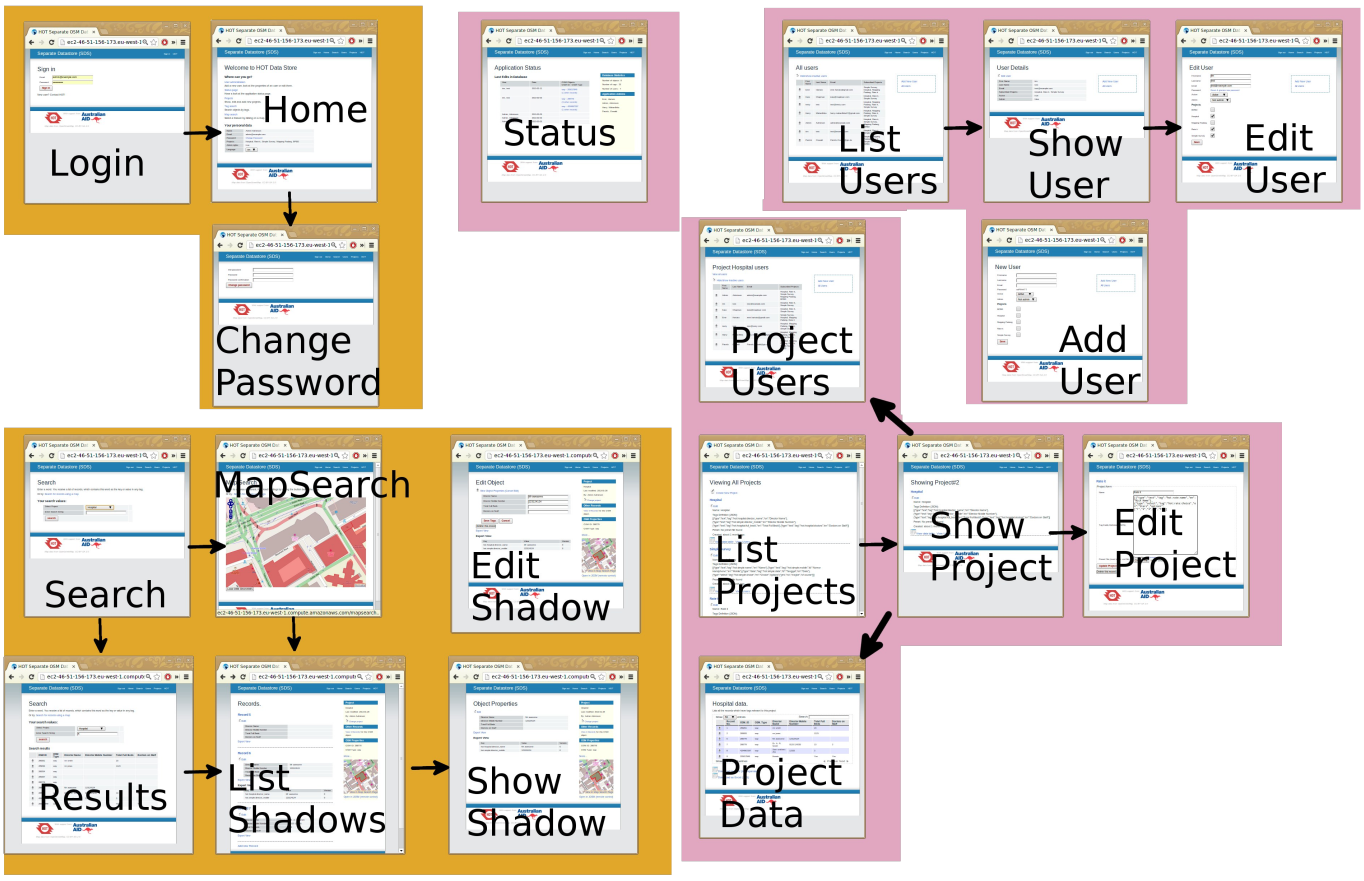
To do anything in this application, users need an account. Only the admin can create accounts. We have deliberately not chosen any of the standard Rails methods of creating accounts (where users are automatically sent confirmation emails and have a “password forgotten” button etc.). When a user is created a unique password is stored and shown to the admin user. A user can then log in using this password and can change the password into something more memorable. Passwords are stored securely with salt and hash in the database.

A user can belong to, or have access to projects, specified by an admin. The admin user can specify these projects when editing the user. The users for a project can be viewed also. The data and tags that are shown to the user are limited to those tags that are specified in each project that the user is part of. In this way you can have projects with data and their own users, and the data that is in these projects will not been seen, or edited, or download by users not in the project.

4 Web Interface

The web interface is only accessible for logged-in users; users without an account cannot go past the login page. Authentication is stored in a session cookie.

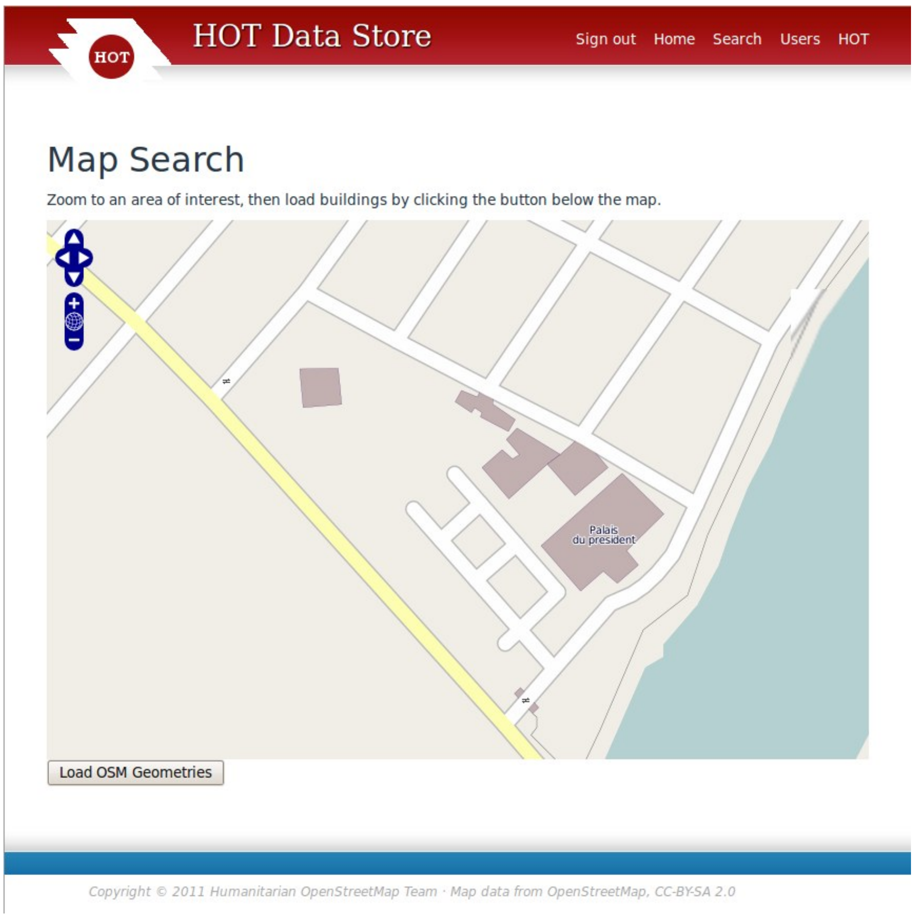
After logging in successfully, users are presented with a “landing page” that allows them to go to two different kinds of search form – the map search, and the tag search. Users will also see a link to change their password here. Users will see a list of the project they are part of here. Users can change the locale, the language that the website users. Users with admin privileges can also access the user management page where users can be created or blocked. Admins can also access the projects pages to create or edit projects and assign users to them.



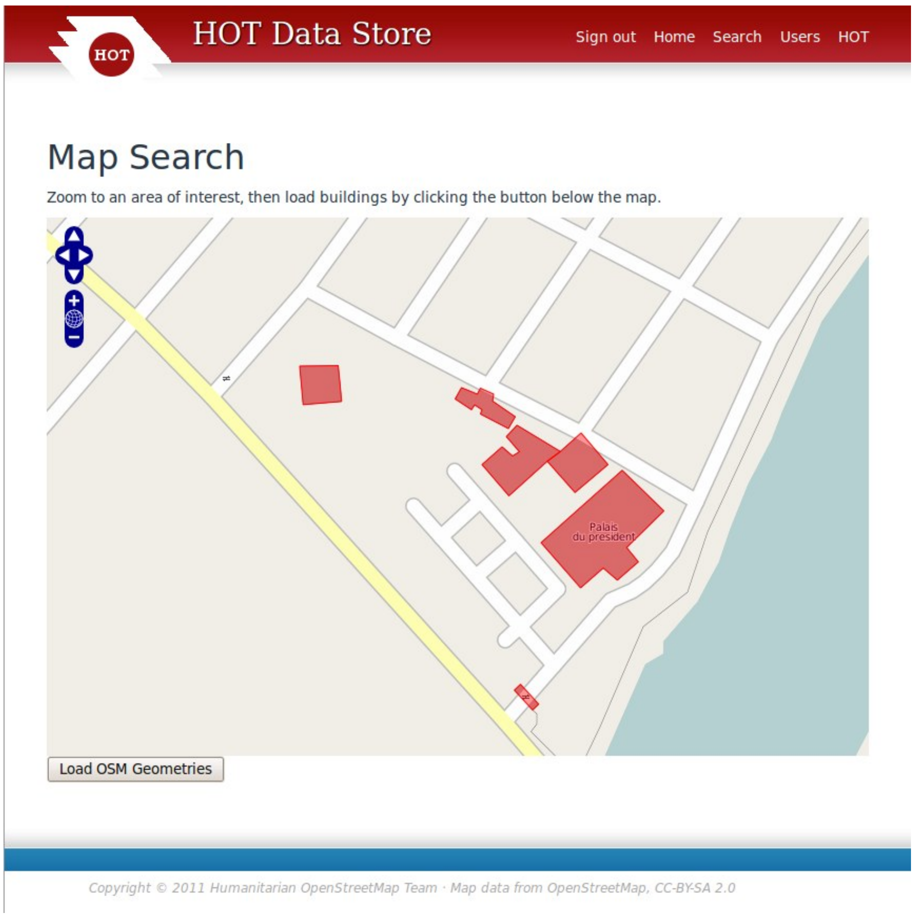
Web Interface Screenflow.

4.1 Map Search

The map search displays a standard map of OpenStreetMap tiles. If you are zoomed in far enough, you can click on the “Load OSM Geometries” button to query the OSM server (or a local mirror, see chapter 17) for all objects in the selected area.


HOT
HOT Data Store
Sign out
Home
Search
Users
HOT
Map Search
Zoom to an area of interest, then load buildings by clicking the button below the map.
Palais du president
Load OSM Geometries
Copyright © 2011 Humanitarian OpenStreetMap Team · Map data from OpenStreetMap, CC-BY-SA 2.0

The display will be greyed out for a while and then come back with all buildings in the area marked with a red outline. Vector data loaded in this manner will remain loaded even when you zoom and pan further; it will only be replaced if you click “Load OSM Geometries” for a second time.


HOT
HOT Data Store
Sign out
Home
Search
Users
HOT
Map Search
Zoom to an area of interest, then load buildings by clicking the button below the map.
Palais du president
Load OSM Geometries
Copyright © 2011 Humanitarian OpenStreetMap Team · Map data from OpenStreetMap, CC-BY-SA 2.0

In this resulting view, clicking on one of the red building outlines will lead to the multiple object detail view - where all records assigned to that OSM object are shown (section 4.3). A permalink is added to the map on the bottom right of the map page, so users can bookmark the pages. Permalinks to this page also work, and are used in this application by the side bar on the details page.

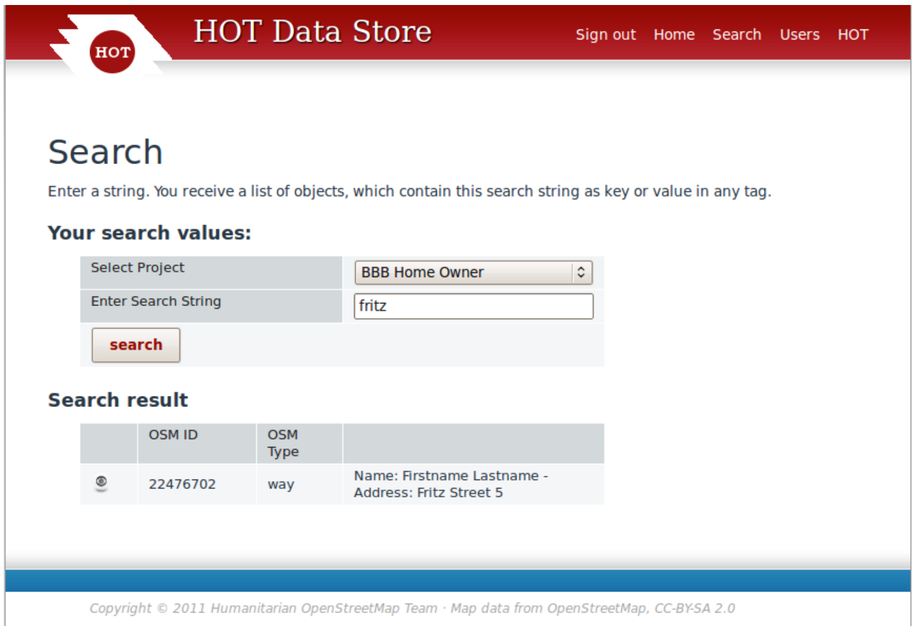
4.2 Tag search

Instead of identifying an object on the map, it is also possible to search for objects by entering a keyword on the tag search form. Additionally, this requires that the user select one of several “projects” which then controls the way in which the search results are presented (see section 4.4); the selection of a project does not influence which tags or objects are searched. A user would only be able to change to a project that the user is part of. The results also will only be shown according to the users projects.

The search will find any object where the current version has at least one tag that contains the search word (case insensitive) in either the key (tag name) or the value.

Note: The search only looks at properties stored in the SDS database itself; it does not have access to tags that are stored on OpenStreetMap.

In the search result, clicking on the "eye" symbol next to a row in the results table will lead to the single object detail view (section 4.3).



Search

Enter a string. You receive a list of objects, which contain this search string as key or value in any tag.

Your search values:

Select Project	BBB Home Owner
Enter Search String	fritz

Search result

	OSM ID	OSM Type	
	22476702	way	Name: Firstname Lastname - Address: Fritz Street 5

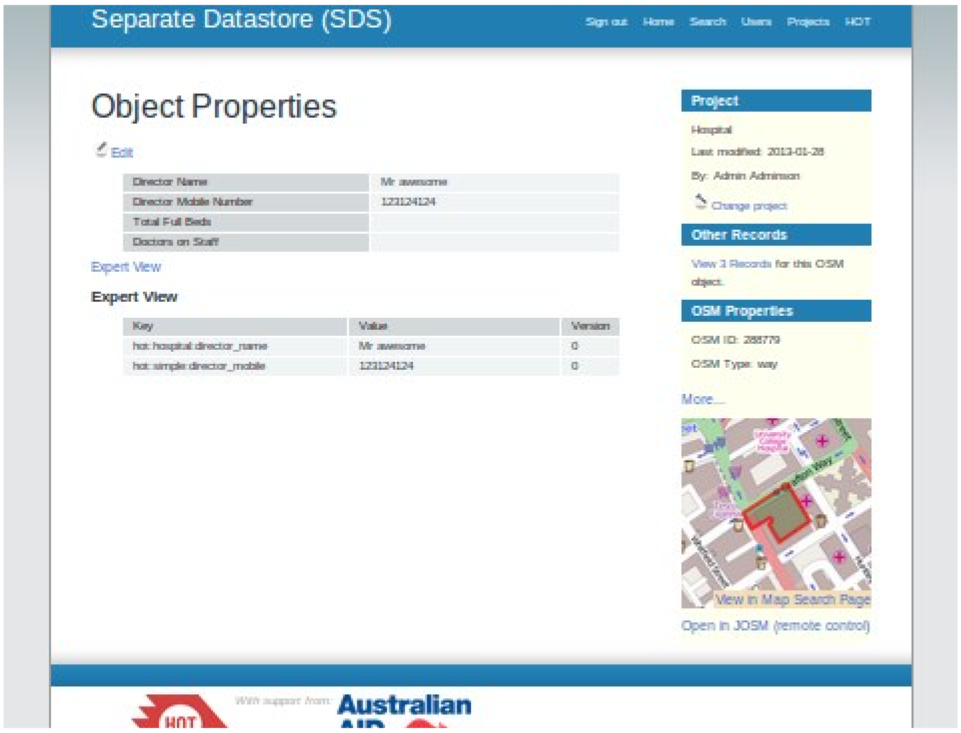
4.3 Viewing and Editing Object Properties

There is a way to view all the objects that are at the same OSM object (share the same osm_id and osm_type) and a way to view individual records.

Proceeding to the detail view from one of the search forms will take you a table that shows all the object's properties in a human-readable form, as well as a mini map with the object and it surroundings on OpenStreetMap.

The layout of this table is dependent on the "project" setting which can be changed through the "change project" link in the right sidebar. See section 4.4 for details. Available projects are determined by which ones the user belongs to. A link to view all records at the same location is also given in the right hand view. Tags from OSM database can also be optionally shown above the mini map, and a permalink to that map view is shown. If, for the currently selected project a JOSM preset file exists, a link will be given so that the user can download it. Additionally a Remote Control JOSM link is shown at the bottom of the right hand bar making it easier to load up the JOSM editor for that area.

There's also an "Expert view" link which will replace the human-readable display by one that lists the raw tag names instead of natural language labels. The expert view always lists all tags that are stored against the object, whereas the project views only show what is coded for that project. The list of tags in both the expert view and the main view are determined by the allowed tags in the users projects. The user will not be able to see tags that are not specified in their projects.



Object Properties

Edit

Director Name	Mr awesome
Director Mobile Number	123124124
Total Full Beds	
Doctors on Staff	

Expert View

Expert View

Key	Value	Version
hot:hospital:director_name	Mr awesome	0
hot:simple:director_mobile	123124124	0

Project

Hospital

Last modified: 2013-01-28

By: Admin Adminson

Change project

Other Records

View 3 Records for this OSM object.

OSM Properties

OSM ID: 288779

OSM Type: way

More...

View in Map Search Page

Open in JOSM (remote control)

Clicking on the “Edit” link above the table will make the fields editable. There is a “Save” button that you can use to save your changes. The tags that have been changed will have a new revision. Revisions can be shown in more detail from the expert section.

4.4 Projects

The SDS web interface supports the concept of a “project”, which is meant to describe one particular real-world campaign or project for which SDS is used. The storage of data is not affected by projects at all, but the representation of data is.

Every user needs to be assigned a project when creating the user and when viewing or editing a record, you can flip the “project” to be something else – the same record can be viewed through different projects. The current “project” controls which tags the web site code will load to display details about a record. Users that do not belong to a project will not see tags that are defined in the project, unless the tags are shared with a project that the user does belong to.

4.4.1 New Projects

Previously adding a new project required the use of a server admin and some knowledge of Ruby, to create a series of files and manipulate the database.

In this version of the application, the creation of new projects is easier, and can be done by an admin user on the web application pages.

A project must have a name defined, and it must have the JSON tag definition specified. It optionally can have a JOSM preset file added, for users to download.

4.4.2 Tags Definition

Currently the definition for the tags that are in a project are stored in the database in a field called “tags_definition” this is stored as text but needs to be valid JSON. The definition is a solution to the hard coded forms and data in the previous version of this applications “partials”.

It is recommended that the JSON be valid, and a utility such as http://jsonlint.com can be used to help with this. In the future, the specification of these tags could be simplified, through a form builder perhaps, or some JSON validation could be added.

The following gives an example of a definition in JSON format.

A tag must have a “type”, “tag” and “en”

“type” can be one of “text”, “date”, “select”, “url”

“en” is the English translation for name of the tag

Translations for the name of the tag are added by giving the locale as a key, For example

“id” is the Indonesian translation for the tag

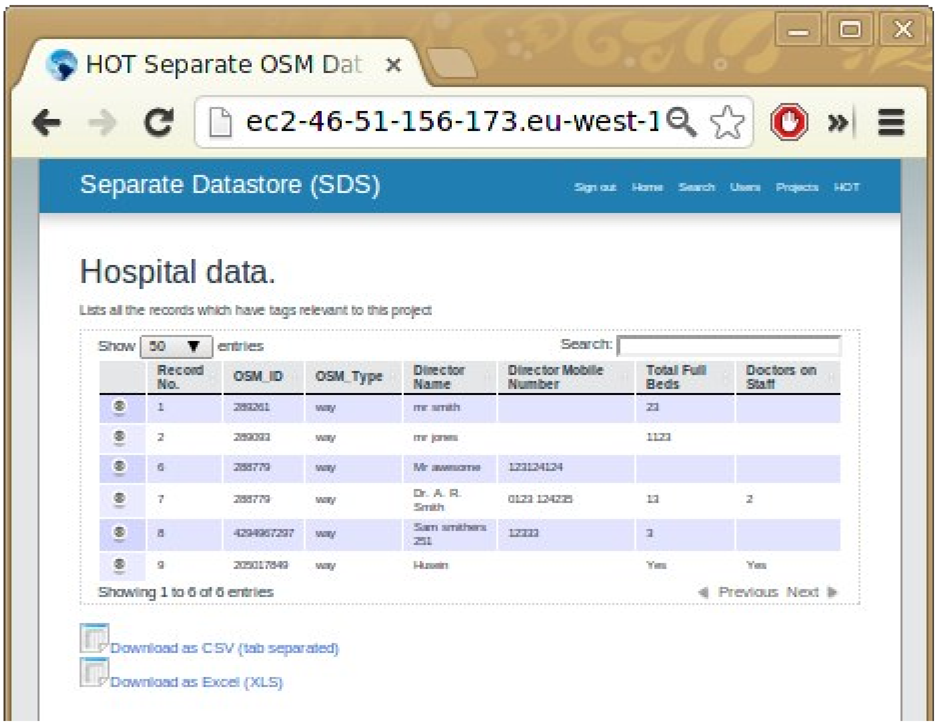
The following is an example of a tags definition for a project:

```
[
{       "type": "text",
        "tag": "hot:simple:name",
        "en": "Name"
   },
    {
        "type": "text",
        "tag": "hot:simple:mobile",
        "id": "Nomor Handphone",
        "en": "Mobile"
    },
    {
        "type": "date",
        "tag": "hot:simple:date",
        "id": "Tanggal",
        "en": "Date"
    },
    {
        "type": "select",
        "tag": "hot:simple:choice",
        "en": "Choice",
        "options": [
            "yes",
            "no",
            "maybe",
            "of course"
        ]
    }
]
```

The definition is an Array ([..])of objects (contained in the curly brackets{..}) . Each tag object has the following: tag, type, en. “tag” defines the specified tag that is wanted. For example "tag": "hot:simple:mobile" - “type“ defines the type of tag for the purposes of display and editing. For example a tag where "type": "date" is given will show a date selection widget in the form on the application, and a type of select will show a drop down selection box, composed of the entries in the “options“ box. “en” is the English name for the tag and “id” is the Indonesian name. You can add extra translations in this way. (See Section 4.7 for further information)

4.5 Project Data

An admin user can view all the data collected for a project. The data that is shown is based on the tags specified in the tags definition. The data table can be searched, has pagination, and the columns are sortable.



Hospital data.

Lists all the records which have tags relevant to this project

Show 50 entries Search:

	Record No.	OSM_ID	OSM_Type	Director Name	Director Mobile Number	Total Full Beds	Doctors on Staff
◉	1	289261	way	mr smith		23	
◉	2	289093	way	mr jones		1123	
◉	6	288779	way	Mr awesome	123124124		
◉	7	288779	way	Dr. A. R. Smith	0123 124235	13	2
◉	8	4294967297	way	Sam smithers 251	12333	3	
◉	9	205017849	way	Husein		Yes	Yes

Showing 1 to 6 of 6 entries ◀ Previous Next ▶

Download as CSV (tab separated)

Download as Excel (XLS)

Under the data table are two links to export this data. The data can be exported as tab separated (.csv) format, and in Microsoft Excel (.xls) format.

4.6 Internationalization and Translations

The application supports different languages in a few ways. The names of the tags specified in a project's tags definition can be translated (See section 4.4.2). The user can change the locale of the website from the front page where they will see a drop down select box listing the available languages. The chosen language is saved with the user, and is kept when they log in at a later date. In addition a server admin can specify the default language of the application. The locale files are located in config/locales and are in the form of YAML. To add support for another language requires knowledge of Ruby on Rails Internationalisation: http://guides.rubyonrails.org/i18n.html.

5 Server API

In addition to the web interface, where the Rails code talks to human operators, there's also a few API calls which are used by the JOSM plugin to talk to the server. In contrast to the normal user interface, the API requires HTTP “Basic” authentication with every API call.

Both API calls described here will return a HTTP code of “200 OK” on success, and a “500 Internal Server Error” otherwise. The most likely reason for a failure would be a database access problem.

5.1 The collectshadows API call

The “collectshadows” API call accepts three lists of object IDs – nodes, ways, relations – for input and returns an XML document detailing all tags stored against any of the objects given. Example:

```
wget --user myusername --password mypassword \
   'http://localhost:3000/collectshadows?
nodes=1,2,3&ways=22476702&relations=4,5,6'
```

This returns a document with anything stored about any of the objects, in this case only the way:

```
<?xml version="1.0" encoding="UTF-8"?>
<osm_sds>
  <osm_shadow osm_id="22476702" osm_type="way">
    <tag k="hot:bbb:name" v="Firstname Lastname"/>
    <tag k="hot:bbb:street_address" v="Fritz Street 5"/>
  </osm_shadow>
</osm_sds>
```

At most, the number of <osm_shadow> elements returned will be the number of objects requested in the API call. If none of the objects have tags associated with them, an empty <osm_sds> element will be returned.

This call can be executed either as a GET request or as a POST request (where the content type is application/x-www-form-urlencoded and the query string is passed in the message body).

The tags returned with the object are specific to the tags defined for a users projects. A user should not be able to get tags that do not belong to a project that they are part of.

Important Note: If multiple records have been added for an OSM Object – only the first, or oldest record will be retrieved in this call. It is proposed that the API and the JOSM plugin be developed further so that the user can, within JOSM, select which record to edit.

5.2 The createshadows API call

The “createshadows” API call creates new osm_shadow objects or updates existing ones for one or several objects. It is always sent as a POST request, with a document formatted just like the one returned in the “colletshadows” call:

```
<?xml version="1.0" encoding="UTF-8"?>
<osm_sds>
  <osm_shadow osm_id="123" osm_type="way">
    <tag k="hot:bbb:something" v="otherthing"/>
    <tag k="hot:bbb:this" v="that"/>
  </osm_shadow>
```

```
    <osm_shadow osm_id="124" osm_type="way">
      <tag k="hot:bbb:something" v="that"/>
      <tag k="hot:bbb:this" v="otherhing"/>
    </osm_shadow>
  </osm_sds>
```

If the object exists already on the SDS server, a new version will be created that contains only the tags uploaded (no merging with previous version). If the object does not yet exist, a first version will be created.

If multiple records exist for the same OSM object, only the first, or oldest record will be updated. It is envisaged that future development of the application and JSOM plugin will allow the user to select which records are edited.

The server makes absolutely no assumptions about the tags, it doesn't verify them in any way. If a user tries to add tags that are not allowed, then these tags will not be saved. In other words, tags that are specified in projects for which the user is part of, that the user tries to save will not be saved.

An osm_shadow object can only be deleted via the web application.

6 OpenStreetMap Proxy/Mirror

The “map search” web interface and the “details” view both highlight objects on an OpenStreetMap background. This requires two kinds of access to the OpenStreetMap server.

6.1 OpenStreetMap map tiles

The map background is composed of standard OpenStreetMap map tiles, and shown in the browser using OpenLayers. A suitable version of OpenLayers is installed in the application's asset subsystem. The map background is loaded from tile.openstreetmap.org directly by the browser; the server neither produces nor caches map tiles.

6.2 OpenStreetMap geometries

The map search requires that the OpenLayers instance running on the browser loads geometries from the OpenStreetMap database. For this purpose it issues a “map” call against the OpenStreetMap API. Because of browser security policy, it is not possible for the browser to request this information from OSM directly. Instead, the API call is directed to the SDS application, where it is caught by the OsmapiController and forwarded to OSM proper (code shortened for illustration):

```
def proxy
    osm_api_url = APP_CONFIG[:osm_api_url] ||
"http://api.openstreetmap.org/api/0.6/"
    uri = APP_CONFIG[:osm_api_url] + params[:apirequest];
    if (request.query_string)
        uri = uri + "?" + request.query_string;
    end
    result = fetch(uri);
    render :text => result
end

def fetch(uri_str, limit = 10)
  response = Net::HTTP.get_response(URI(uri_str))
 case response
  when Net::HTTPSuccess then
    response.body
  else
    response.value
  end
end
```

The detail view also shows a mini-map and for this purpose loads either a “node”, “way”, or “relation” KML document from the OSM server.

It is possible to replace the direct OSM server access with access to a local OSM data proxy. For this, one would have to set up a PostgreSQL database and regularly update it with a data feed from OSM through the Osmosis utility, and then set up the OSM “rails port” that will provide the required API calls. The proxy controller above would then simply be changed to point to the local mirror instead of pointing to OSM directly.

The url for the OSM server can be configured by a server admin by editing the config/app_config.yml file (see the next section)

6.2.1 Configuration

A server admin can configure the URL for the OpenStreetMap API in the app_root/config/app_config.yml file The system will get the geometries using this defined URL.

Below is an example configuration with entries for the environment:

```
test:
    osm_api_url: 'http://api06.dev.openstreetmap.org/api/0.6/'
development:
    osm_api_url: 'http://api06.dev.openstreetmap.org/api/0.6/'
production:
    osm_api_url: 'http://api.openstreetmap.org/api/0.6/'bbb:this"
v="otherhing"/>
```

7 JOSM Plugin

The SDS JOSM plugin has been implemented as a standard JOSM plugin, in Java. It works with the current version of the JOSM core without any modification.

The plugin hooks into the data upload and data download routines in JOSM. On data download, it executes an extra query to the SDS server, fetching extra data and merging them with data downloaded from OSM. On data upload, it separates private and public data, making sure that only public data gets uploaded to JOSM, and private data gets uploaded to the private server. (If only private data has been changed, no uploading to OSM takes place.)

The plugin will merge data in a way that is transparent to other plugins or functions of the JOSM core; the private data will appear as “just another tag(s)”.

Standard JOSM conflict resolution is supported, but there is no additional conflict resolution for data on the SDS server, meaning that if two people edit data from the SDS server at the same time, the person uploading their change last will override the other person's change.

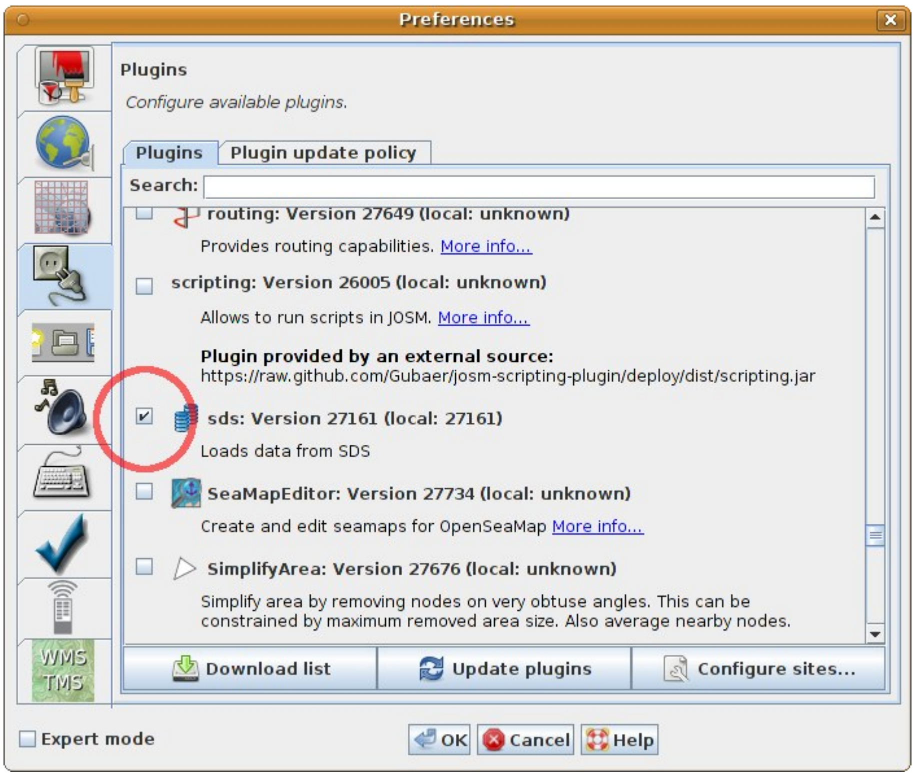
7.1 *Installing and Configuring the Plugin*

The plugin is not yet published in the general JOSM plugin repository (but see 7.4). To install the plugin, you have to copy the file sds.jar to the directory where JOSM keeps its plugins. Under Unix:

```
cp sds.jar ~/.josm/plugins
```

(If you haven't installed any plugins yet, you might have to create the directory first.)

Start JOSM and open the Edit → Preferences menu. In the “Plugins” tab, scroll down the the “sds” plugin and check the box next to it; then click “OK” and restart JOSM.



Plugins

Configure available plugins.

routing: Version 27649 (local: unknown)

Provides routing capabilities. More info...

scripting: Version 26005 (local: unknown)

Allows to run scripts in JOSM. More info...

Plugin provided by an external source:
https://raw.github.com/Gubaer/josm-scripting-plugin/deploy/dist/scripting.jar

[✓] sds: Version 27161 (local: 27161)

Loads data from SDS

SeaMapEditor: Version 27734 (local: unknown)

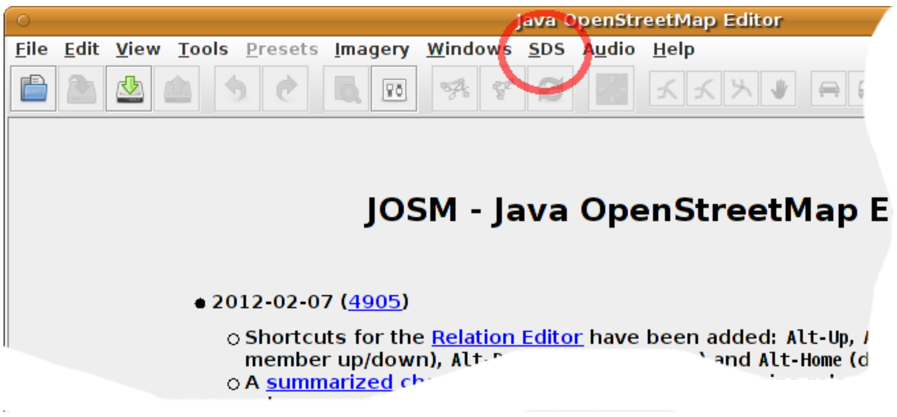
Create and edit seamaps for OpenSeaMap More info...

SimplifyArea: Version 27676 (local: unknown)

Simplify area by removing nodes on very obtuse angles. This can be constrained by maximum removed area size. Also average nearby nodes.

Expert mode

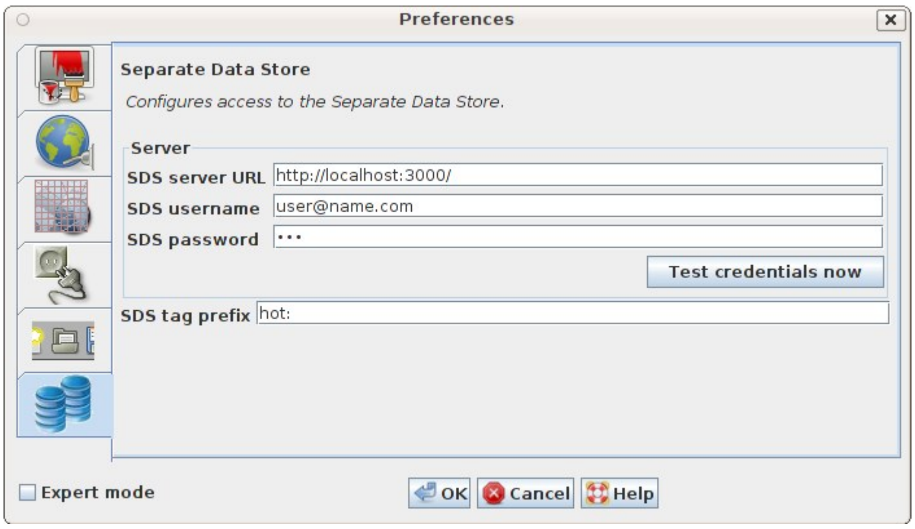
If you have successfully enabled the plugin, then you will see an extra “SDS” menu in the menu bar.



JOSM - Java OpenStreetMap E

- 2012-02-07 (4905)
  - Shortcuts for the Relation Editor have been added: Alt-Up, member up/down), Alt- and Alt-Home (d
  - A summarized ch

You can now configure the plugin either through the normal preferences panel or the direct “preferences” link from the SDS menu.



SDS server URL: http://localhost:3000/
SDS username: user@name.com
SDS password: •••
SDS tag prefix: hot:

You will have to enter the URL where the SDS server is running, as well as your credentials on the server. (If you don't enter credentials here, they will be requested in a pop-up box when the server is accesses.)

You also have to specify a tag prefix. All tags beginning with this prefix will be sent to the SDS server instead of the the OSM server.

7.2 Using the Plugin

When the plugin is installed and configured, it will automatically plug itself into the various form of server downloads that JOSM supports. Whenever anything is downloaded from the OSM server, the plugin will make a "collectshadows" API call against the SDS server (see section 5.1). If additional tags are reported for objects then these will be seamlessly added to the objects, and the user can work with them normally.

JOSM rendering rules and filters can be applied to these tags just like to all other tags.

When you upload data to OSM, all tags with the specified prefix will be separated out of the standard OSM upload stream, and will be sent to the SDS server instead.

If you save data to disk using the normal JOSM save mechanism, then any SDS tags will not be saved with the rest of the file. This is to protect against accidental uploading by someone who later loads the file and does not have the plugin activated. If you want to save SDS tags to disk, you have to use the "save" option from the SDS menu, resulting in a special SDS file. You can later load such an SDS file again, provided you have loaded the relevant OSM objects first!

7.3 Plugin Architecture

The plugin taps into JOSM's standard control flow in three places: Before uploading, after uploading, and after downloading. In addition, it has the standard “preference panel” capability that most other plugins share.

7.3.1 After Downloading

The “after downloading” hook is handled by the ReadPostprocessor class and plugs into JOSM's OsmReader class. This interface was newly created for this plugin. Any data read by JOSM – either from the OSM API or from a file – is passed to all registered implementations of OsmServerReadPostprocessor.

The SDS plugin uses this opportunity to determine the object IDs and query the SDS server for tags to add to these objects. It also stores the original versions of all objects for later comparison.

7.3.2 Before Uploading

The “before uploading” hook existed in JOSM before (called UploadHook). It is used by an instance of the DetermineSdsModificationsUploadHook class to determine if any SDS tags are present in the data to be uploaded, and if yes, it removes these from the upload stream and enqueues them for later upload to the SDS server. This requires comparing uploaded versions of objects to previously stored old versions in order to detect tag removals.

7.3.3 After Uploading

The “after uploading” hook as been newly created for this plugin. It is called WritePostprocessor and executed by the OsmServerWriter class whenever data has been sent to the OSM server. This implementation of the Write Postprocessor will send data to the SDS API.

7.4 Plugin Publication

The plugin will be published on the OSM SVN repository and it will be available through the normal JOSM plugin download channels. The SDS server code will also be suitably published so that third parties can make use of the plugin together with the server code.