JOSM Separate Data Store

Documentation

2012-03-14, Version 1

by Frederik Ramm <ramm@geofabrik.de>

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.

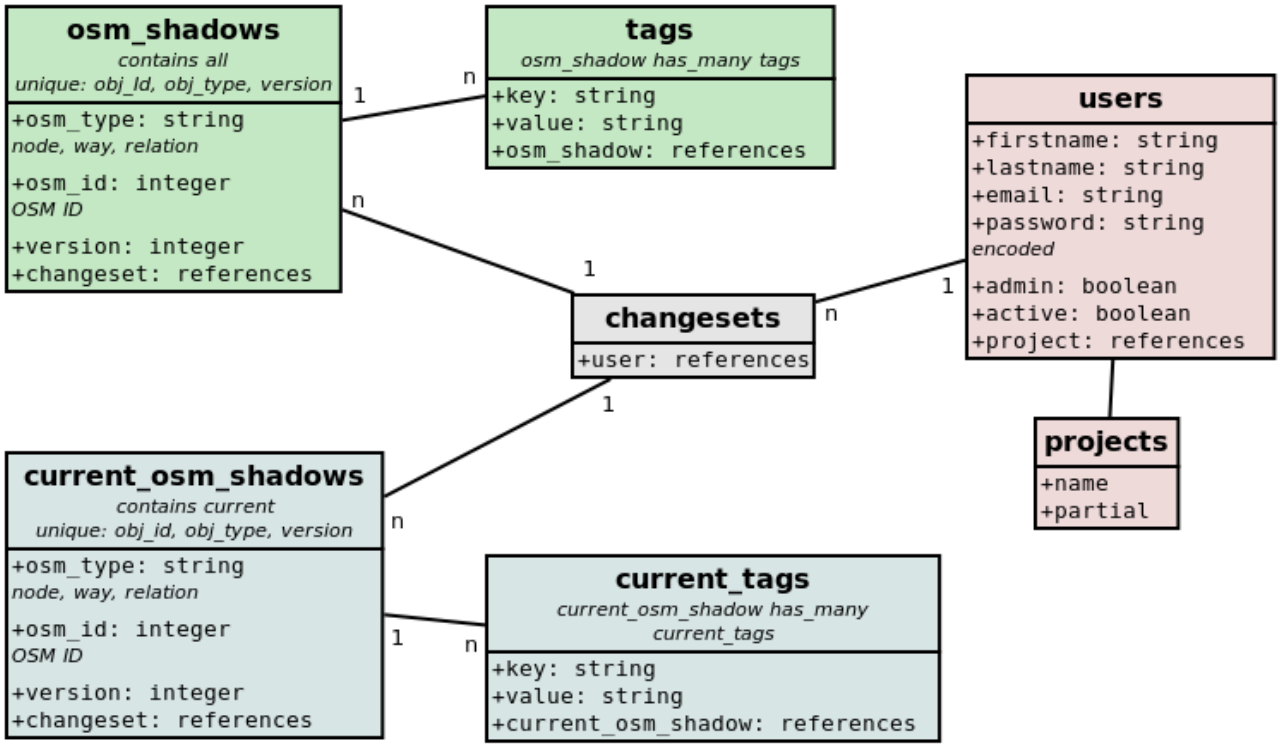
3.2 Data Model

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.)

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



There are other parallels to the OpenStreetMap data model. The SDS database has two parallel sets of “osm_shadows” and “tags” tables, one with the prefix “current_” and one without. The “current” tables always only carry the latest version of everything; if a new set of tags is uploaded for an object, then the previous set of tags is purged and the new set is is installed. The plain “osm_shadows” and “tags” tables, on the other hand, have a full history of everything; they contain every edit made, and they can be used to find out who made what change when. There is however not yet any API or web interface to access that historic information; it is stored but never retrieved.

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. This, however, only affects the default presentation in the web interface and not the functionality of the system.

The API used to modify data in this database (see chapter 5) is not a true REST API because you can never create or delete an “osm_shadow” object; the system behaves as if empty osm_shadow objects

were present for every single OSM object, and instead of deleting one, you would simply upload a new version with no tags.

3.3 Deploying and Running

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, 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,
password,active,admin) values ('my','name','my@email.com','secret',true,true);"
```

Deploy the Rails application using your preferred means, 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.

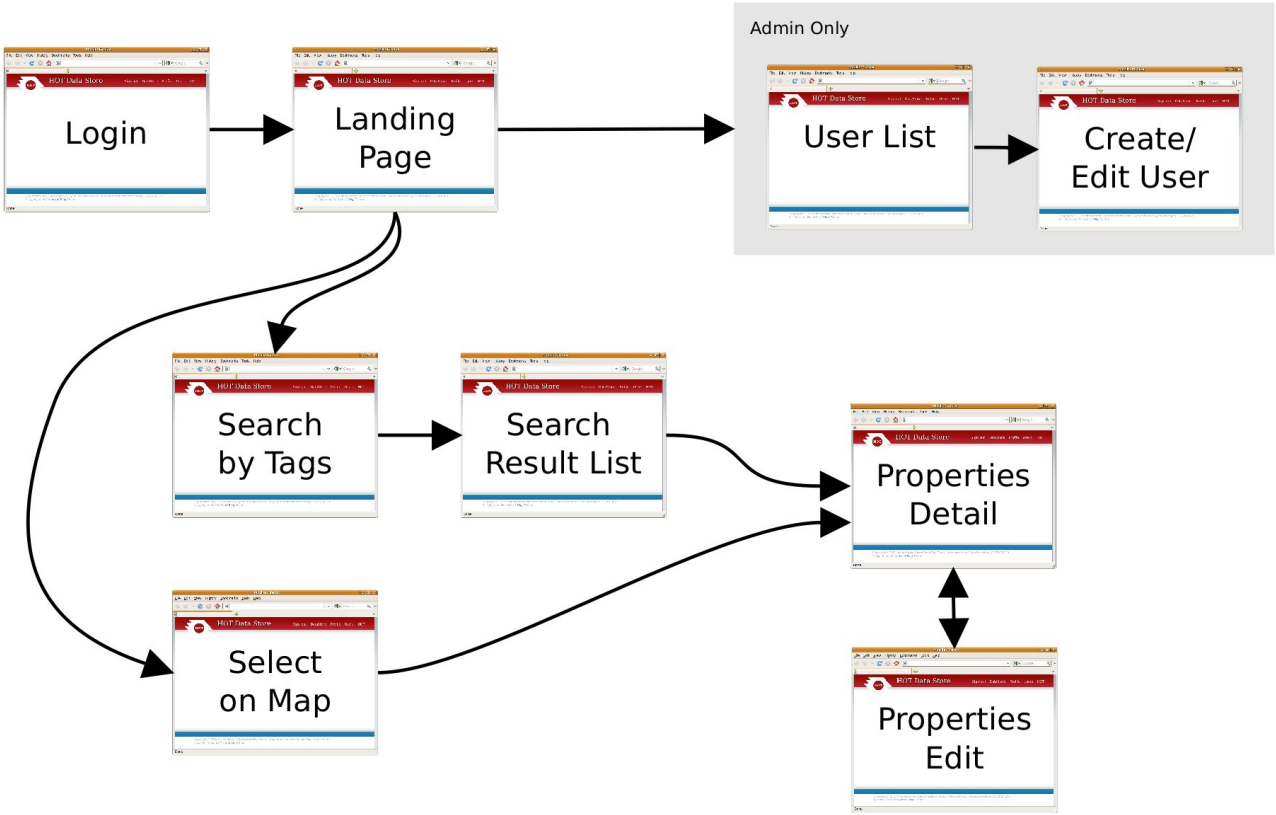
3.4 User Management

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.), and we have deliberately chosen to store the password in plain text form and allow the administrator to set and read user passwords.

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.

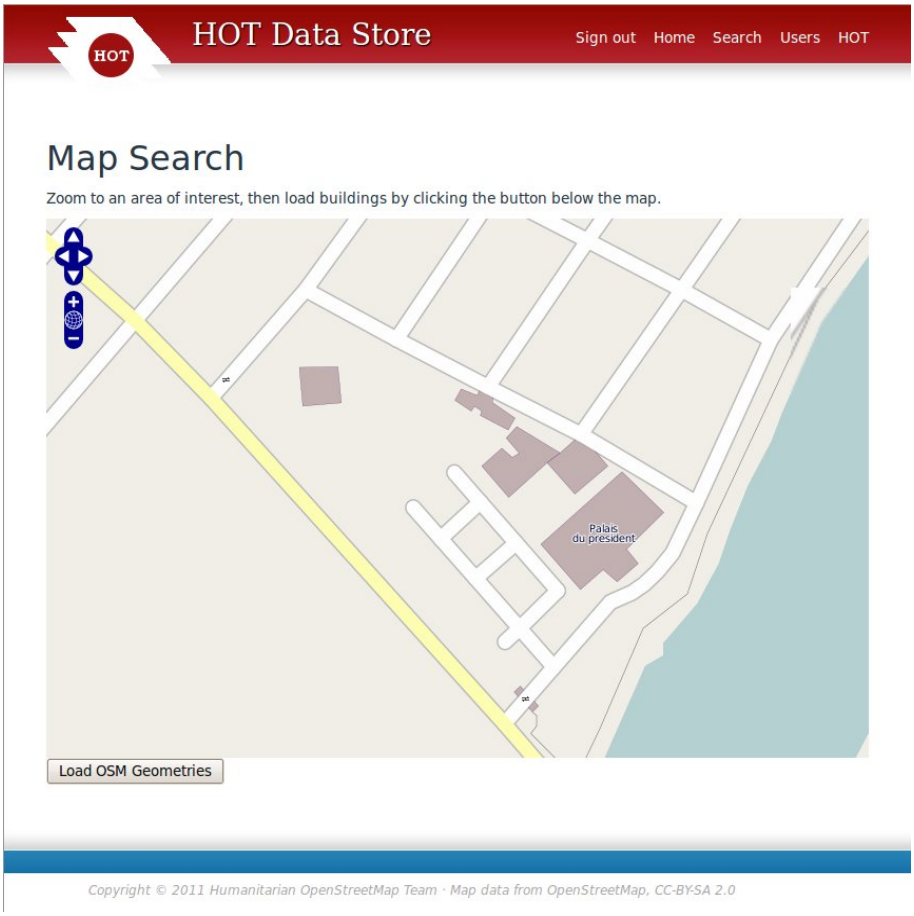
After logging in successfully, users are presented with a “landing page” that allows them to go to two different kinds of search form – the map search, and the tag search. Users with admin privileges can also access the user management page where users can be created or blocked.



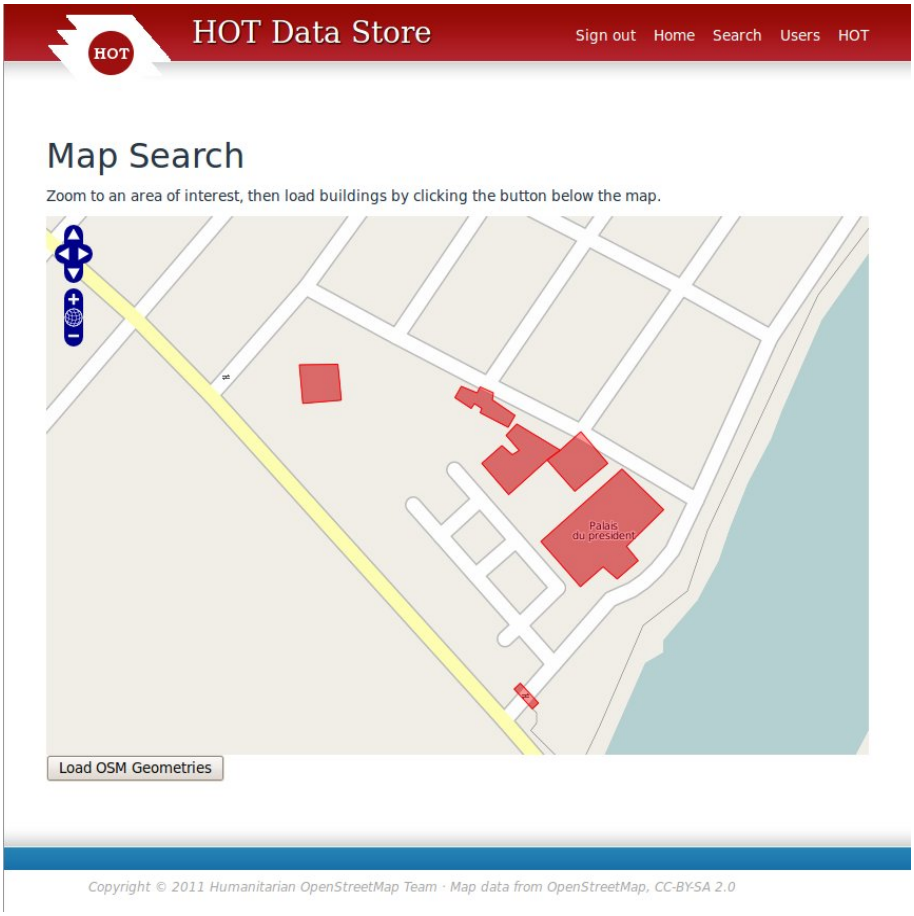
Web Interface Screenflow.

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 12) for all objects in the selected area.



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.



In this resulting view, clicking on one of the red building outlines will lead to the detail view (section 4.3).

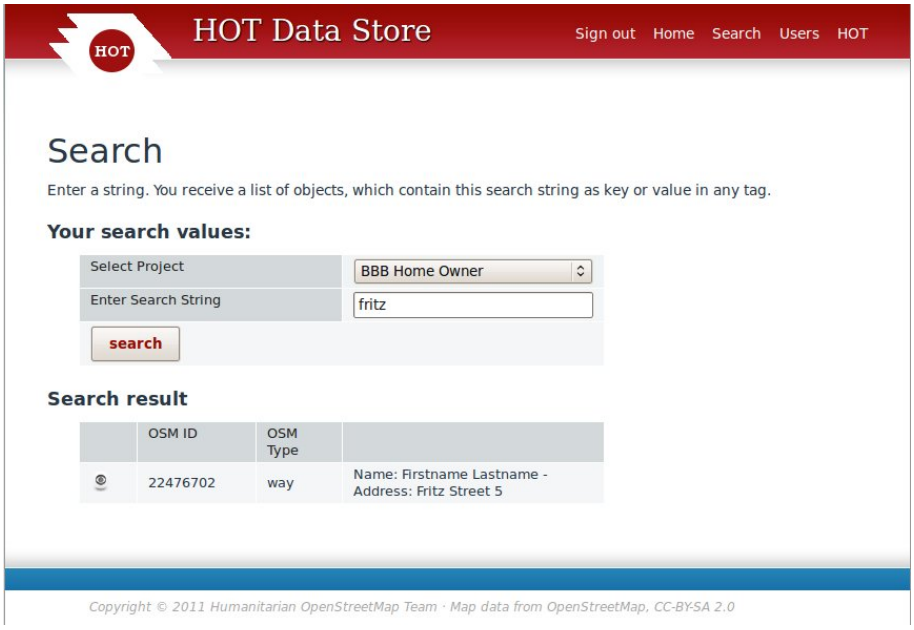
4.2 Tag search

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.

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 detail view (section 4.3).

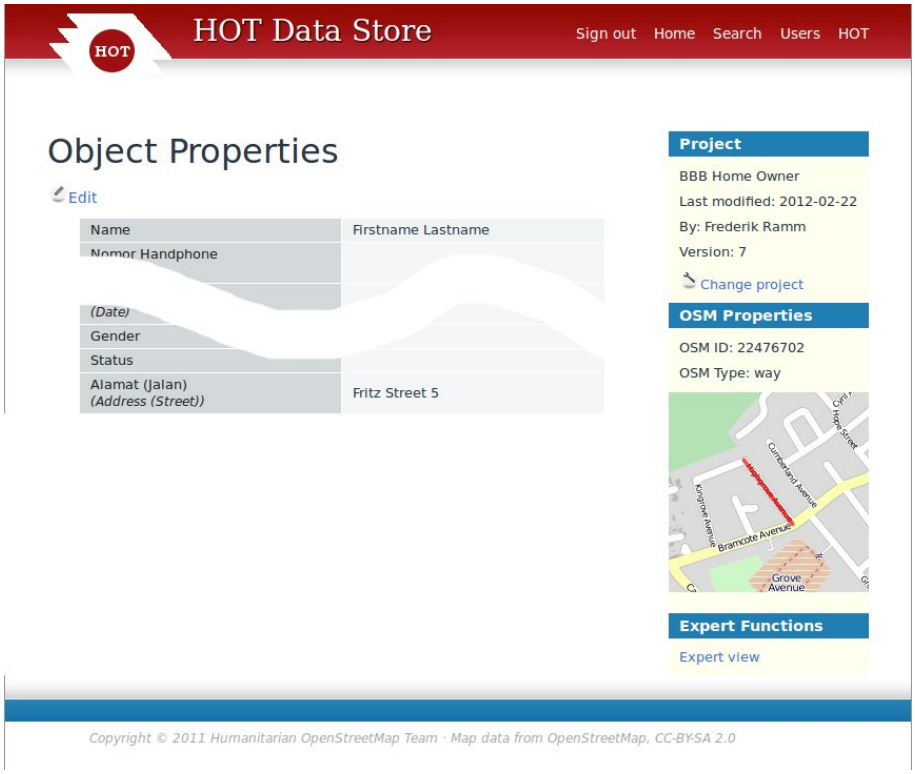


4.3 Viewing and Editing Object Properties

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.

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.



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 object's version number will automatically be increased, and the change will be recorded against a new “changeset” that is associated with your browser session.

4.4 Per-Project Partials

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 has a default “project”, 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 “partials” the web site code will load to display details about a record.

Three aspects of the API are controlled by such project-based “partials”: The displaying of search results after the tag search, the details view, and the edit screen. Partials/Projects may also be used to offer support for different languages – you could have two “projects” that are technically the same, but present their labels in different languages.

Chapter 6 has more on how to create new projects/partials.

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, thenumber 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).

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.

The server makes absolutely no assumptions about the tags, it doesn't verify them in any way.

It is not possible to delete an osm_shadow object, but a version with no tags can be uploaded which

has the same effect:

```
<osm_shadow osm_id="124" osm_type="way" />
```

6 Extending the Server

The server can be extended with new projects and matching "partials" to change the representation of objects to the user.

Three steps are required to introduce a new project:

6.1 Add new project to database

You can add the project to the database using the PostgreSQL command line, or you could write a Rails migration that does it. The easiest way is probably by using the Rails console from the Unix shell:

```
$ rails console
Loading development environment (Rails 3.1.1)

irb(main):001:0> p = Project.new(:name => "test project", :partial => "test_project")
=> #<Project id: nil, name: "test project", partial: "test_project", created_at: nil,
updated_at: nil>
irb(main):001:0> p.save!
   (0.1ms)  BEGIN
  SQL (27.7ms)  INSERT INTO "projects" ("created_at", "name", "partial", "updated_at") VALUES
($1, $2, $3, $4) RETURNING "id"  [["created_at", Wed, 22 Feb 2012 17:58:23 UTC +00:00],
["name", "test project"], ["partial", "test_project"], ["updated_at", Wed, 22 Feb 2012
17:58:23 UTC +00:00]]
   (11.3ms)  COMMIT
=> true
```

As the project name, specify the name that should be shown to the user; as the "partial", specify the name of the code snippets you will create in the following three steps.

6.2 Add partial for search

In the app/views/search directory, create a file named _partial.erb, where "partial" is the name you used when creating the project. In our example: "_test_project.erb".

In that snippet, you have to provide Ruby code that iterates over a list of search results (of class osm_shadow) given in the @results variable, and formats them for display in a four-column table. Column 1 is the detail link, column 2 and 3 the OSM object type and ID, and column 4 is a description that identifies the record to the user.

The following snippet would be suitable to display the content of the "hot:bbb:street_address" and "hot:bbb:name" tags for every match found:

```
<% @result.each do |shadow| %>
<% desc = []
   shadow.tags.each do |t|
       if t.key == "hot:bbb:name"
           desc.push "Name: #{t.value}"
```

```
        elsif t.key == "hot:bbb:street_address"
            desc.push "Address: #{t.value}"
        end
    end

%>
    <tr>
    <td><a href="<%= show_shadow_path(shadow.osm_type, shadow.osm_id) %>" title="View
Object"><img src="<%= asset_path 'snow/view.png' %>" width="20"></a></td>
    <td><%= shadow.osm_id %></td>
    <td><%= shadow.osm_type %></td>
    <td><%= desc.join(" - ") %></td>
    </tr>
<% end %>
```

6.3 Add partial for view/edit

In the app/views/osm_shadows directory, there is another partial with the same name as the previous one; this time, it is used for controlling how the detail view and the edit form are laid out. The basic structure of this partial is as follows:

```
<% if (params['action'] == "show") then %>
    ... code to display object properties ...
<% end %>
<% if (params['action'] == "new") or (params['action'] == "edit") then %>
<%= form_tag({:controller => "osm_shadows", :action => "create", :method => "post"}, :id =>
"osm_shadows_form") do %>
    ... code to display an input form ...
<% end %>
<% end %>
```

This partial receives all information about the object to be shown in the variables @osm_shadow and @taghash, the latter being a key-value map of all tags specified for this object.

The implementation created by Geofabrik is fairly generic and has a block of input field definitions at the top, so that it can easily be copied and adapted for other purposes.

7 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.

7.1 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.

7.2 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
    uri = 'http://api.openstreetmap.org/api/0.6/' + 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.

8 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.

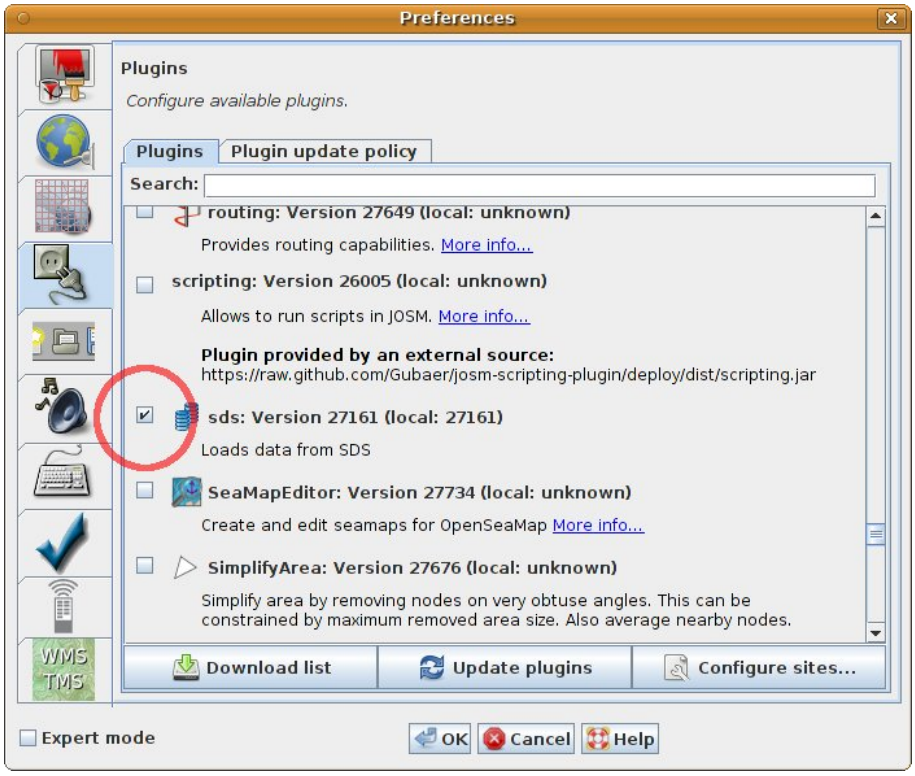
8.1 Installing and Configuring the Plugin

The plugin is not yet published in the general JOSM plugin repository (but see 8.4). 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
```

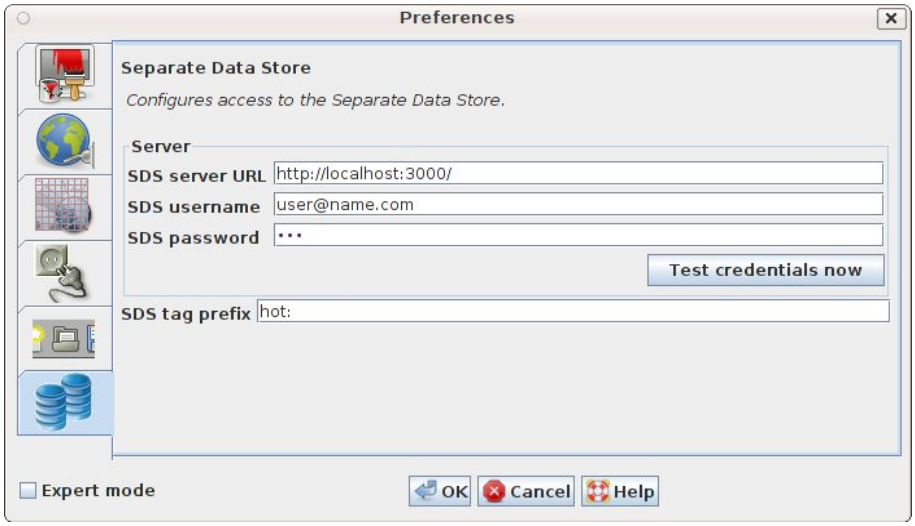
(If you haven't installed any plugins yet, you might have to create the directory first.)

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.



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

You can now confiugure the plugin either through the normal preferences panel or the direct "preferences" link from the SDS menu.



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.

8.2 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!

8.3 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.

8.3.1 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.

8.3.2 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.

8.3.3 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.

8.4 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.