ckanext-spatial Documentation

Release 0.1

Open Knowledge Foundation

May 12, 2018

Contents

This extension contains plugins that add geospatial capabilities to CKAN.

You should have a CKAN instance installed before adding these plugins. Head to the CKAN documentation for information on how to set up CKAN.

The extension adds a spatial field to the default CKAN dataset schema, using PostGIS as the backend. This allows to perform spatial queries and display the dataset extent on the frontend. It also provides harvesters to import geospatial metadata into CKAN from other sources, as well as commands to support the OGC CSW standard via pycsw.

Contents:

CHAPTER 1

Installation and Setup

Check the Troubleshooting section if you get errors at any stage.

Install PostGIS and system packages

Warning: If you are looking for the geospatial preview plugins to render (eg GeoJSON or WMS services), these are now located in ckanext-geoview. They have a much simpler installation, so you can skip all the following steps if you just want the previews.

Note: The package names and paths shown are the defaults on Ubuntu installs. Adjust the package names and the paths if you are using a different platform.

All commands assume an existing CKAN database named `ckan_default`.

Ubuntu 14.04 (PostgreSQL 9.3 and PostGIS 2.1)

1. Install PostGIS:

   ```
   sudo apt-get install postgresql-9.3-postgis-2.1
   ```

2. Run the following commands. The first one will create the necessary tables and functions in the database, and the second will populate the spatial reference table:

   ```
   sudo -u postgres psql -d ckan_default -f /usr/share/postgresql/9.3/contrib/postgis-2.1/postgis.s
   sudo -u postgres psql -d ckan_default -f /usr/share/postgresql/9.3/contrib/postgis-2.1/spatial_r
   ```

3. Change the owner of spatial tables to the CKAN user to avoid errors later on:

   ```
   sudo -u postgres psql -d ckan_default -c 'ALTER VIEW geometry_columns OWNER TO ckan_default;'
   sudo -u postgres psql -d ckan_default -c 'ALTER TABLE spatial_ref_sys OWNER TO ckan_default;'
   ```

4. Execute the following command to see if PostGIS was properly installed:

   ```
   sudo -u postgres psql -d ckan_default -c "SELECT postgis_full_version()"
   ```

   You should get something like:

```
                                                                    postgis_full_version
-------------------------------------------------------------------------------------------------------
 POSTGIS="2.1.2 r12389" GEOS="3.4.2-CAPI-1.8.2 r3921" PROJ="Rel. 4.8.0, 6 March 2012" GDAL="GDAL
(1 row)
```

5. Install some other packages needed by the extension dependencies:

```
sudo apt-get install python-dev libxml2-dev libxslt1-dev libgeos-c1
```

Ubuntu 12.04 (PostgreSQL 9.1 and PostGIS 1.5)

Note: You can also install PostGIS 2.x on Ubuntu 12.04 using the packages on the UbuntuGIS repository. Check the documentation there for details.

---

1. Install PostGIS:

```
sudo apt-get install postgresql-9.1-postgis
```

2. Run the following commands. The first one will create the necessary tables and functions in the database, and the second will populate the spatial reference table:

```
sudo -u postgres psql -d ckan_default -f /usr/share/postgresql/9.1/contrib/postgis-1.5/postgis.s
sudo -u postgres psql -d ckan_default -f /usr/share/postgresql/9.1/contrib/postgis-1.5/spatial_r
```

Note: If using PostgreSQL 8.x, run the following command to enable the necessary language:

```
sudo -u postgres createlang plpgsql ckan_default
```

---

3. Change the owner to spatial tables to the CKAN user to avoid errors later on:

```
sudo -u postgres psql -d ckan_default -c 'ALTER TABLE geometry_columns OWNER TO ckan_default;'
sudo -u postgres psql -d ckan_default -c 'ALTER TABLE spatial_ref_sys OWNER TO ckan_default;'
```

4. Execute the following command to see if PostGIS was properly installed:

```
sudo -u postgres psql -d ckan_default -c "SELECT postgis_full_version()"
```

You should get something like:

```
                                  postgis_full_version
-----------------------------------------------------------------------------------------------
POSTGIS="1.5.2" GEOS="3.2.2-CAPI-1.6.2" PROJ="Rel. 4.7.1, 23 September 2009" LIBXML="2.7.7" USE_
(1 row)
```

5. Install some other packages needed by the extension dependencies:

```
sudo apt-get install python-dev libxml2-dev libxslt1-dev libgeos-c1
```

Install the extension

1. Activate your CKAN virtual environment, for example:

```
. /usr/lib/ckan/default/bin/activate
```

2. Install the ckanext-spatial Python package into your virtual environment:

```
pip install -e "git+https://github.com/ckan/ckanext-spatial.git#egg=ckanext-spatial"
```

3. Install the rest of Python modules required by the extension:

   ```
   pip install -r /usr/lib/ckan/default/src/ckanext-spatial/pip-requirements.txt
   ```

4. Restart CKAN. For example if you've deployed CKAN with Apache on Ubuntu:

   ```
   sudo service apache2 reload
   ```

To use the *Spatial Harvesters*, you will need to install and configure the harvester extension: ckanext-harvest. Follow the install instructions on its documentation for details on how to set it up.

Configuration

Once PostGIS is installed and configured in the database, the extension needs to create a table to store the datasets extent, called `package_extent`.

This will happen automatically the next CKAN is restarted after adding the plugins on the configuration ini file (eg when restarting Apache).

If for some reason you need to explicitly create the table beforehand, you can do it with the following command (with the virtualenv activated):

```
(pyenv) $ paster --plugin=ckanext-spatial spatial initdb [srid] --config=mysite.ini
```

You can define the SRID of the geometry column. Default is 4326. If you are not familiar with projections, we recommend to use the default value. To know more about PostGIS tables, see `postgis-manual`

Each plugin can be enabled by adding its name to the `ckan.plugins` in the CKAN ini file. For example:

```
ckan.plugins = spatial_metadata spatial_query
```

When enabling the spatial metadata, you can define the projection in which extents are stored in the database with the following option. Use the EPSG code as an integer (e.g 4326, 4258, 27700, etc). It defaults to 4326:

```
ckan.spatial.srid = 4326
```

As with any configuration change, for it to take effect you need to restart CKAN. For example if you've deployed CKAN with Apache on Ubuntu:

```
sudo service apache2 reload
```

Troubleshooting

Here are some common problems you may find when installing or using the extension:

When upgrading the extension to a newer version

This version of ckanext-spatial requires geoalchemy2

```
File "/home/adria/dev/pyenvs/spatial/src/ckanext-spatial/ckanext/spatial/plugin.py", line 39, in <mod
    check_geoalchemy_requirement()
File "/home/adria/dev/pyenvs/spatial/src/ckanext-spatial/ckanext/spatial/plugin.py", line 37, in chec
    raise ImportError(msg.format('geoalchemy'))
ImportError: This version of ckanext-spatial requires geoalchemy2. Please install it by running `pip
For more details see the "Troubleshooting" section of the install documentation
```

Starting from CKAN 2.3, the spatial requires GeoAlchemy2 instead of GeoAlchemy, as this is incompatible with the SQLAlchemy version that CKAN core uses. GeoAlchemy2 will get installed on a new deployment, but if you are upgrading an existing ckanext-spatial install you'll need to install it manually. With the virtualenv CKAN is installed on activated, run:

```
pip install GeoAlchemy2
```

Restart the server for the changes to take effect.

AttributeError: type object 'UserDefinedType' has no attribute 'Comparator'

```
File "/home/adria/dev/pyenvs/spatial/src/ckanext-spatial/ckanext/spatial/plugin.py", line 30, in chec
  import geoalchemy2
File "/home/adria/dev/pyenvs/spatial/local/lib/python2.7/site-packages/geoalchemy2/__init__.py", line
  from .types import (  # NOQA
File "/home/adria/dev/pyenvs/spatial/local/lib/python2.7/site-packages/geoalchemy2/types.py", line 15
  from .comparator import BaseComparator, Comparator
File "/home/adria/dev/pyenvs/spatial/local/lib/python2.7/site-packages/geoalchemy2/comparator.py", li
  class BaseComparator(UserDefinedType.Comparator):
AttributeError: type object 'UserDefinedType' has no attribute 'Comparator'
```

You are trying to run the extension against CKAN 2.3, but the requirements for CKAN haven't been updated (GeoAlchemy2 is crashing against SQLAlchemy 0.7.x). Upgrade the CKAN requirements as described in the upgrade documentation.

ckan.plugins.core.PluginNotFoundException: geojson_view

```
File "/home/pyenvs/spatial/src/ckan/ckan/plugins/core.py", line 149, in load
  service = _get_service(plugin)
File "/home/pyenvs/spatial/src/ckan/ckan/plugins/core.py", line 256, in _get_service
  raise PluginNotFoundException(plugin_name)
  ckan.plugins.core.PluginNotFoundException: geojson_view
```

Your CKAN instance is using the `geojson_view` (or `geojson_preview`) plugin. This plugin has been moved from ckanext-spatial to ckanext-geoview. Please install ckanext-geoview following the instructions on the README.

TemplateNotFound: Template dataviewer/geojson.html cannot be found

```
File '/home/pyenvs/spatial/src/ckan/ckan/lib/base.py', line 129 in render_template
  template_path, template_type = render_.template_info(template_name)
File '/home/pyenvs/spatial/src/ckan/ckan/lib/render.py', line 51 in template_info
  raise TemplateNotFound('Template %s cannot be found' % template_name)
TemplateNotFound: Template dataviewer/geojson.html cannot be found
```

See the issue above for details. Install ckanext-geoview and additionally run the following on the ckanext-spatial directory with your virtualenv activated:

```
python setup.py develop
```

ImportError: No module named nongeos_plugin

```
File "/home/pyenvs/spatial/src/ckan/ckan/plugins/core.py", line 255, in _get_service
  return plugin.load()(name=plugin_name)
File "/home/pyenvs/spatial/local/lib/python2.7/site-packages/pkg_resources.py", line 2147, in load
  ['__name__'])
ImportError: No module named nongeos_plugin
```

See the issue above for details. Install ckanext-geoview and additionally run the following on the ckanext-spatial directory with your virtualenv activated:

```
python setup.py develop
```

Plugin class 'GeoJSONPreview' does not implement an interface

```
File "/home/pyenvs/spatial/src/ckanext-spatial/ckanext/spatial/nongeos_plugin.py", line 175, in <modu
  class GeoJSONPreview(GeoJSONView):
File "/home/pyenvs/spatial/local/lib/python2.7/site-packages/pyutilib/component/core/core.py", line
  return PluginMeta.__new__(cls, name, bases, d)
File "/home/pyenvs/spatial/local/lib/python2.7/site-packages/pyutilib/component/core/core.py", line
  raise PluginError("Plugin class %r does not implement an interface, and it has already been defined
  pyutilib.component.core.core.PluginError: Plugin class 'GeoJSONPreview' does not implement an inter
```

You have correctly installed ckanext-geoview but the ckanext-spatial source code is outdated, with references to the view plugins previously part of this extension. Pull the latest version of the code and re-register the extension. With the virtualenv CKAN is installed on activated, run:

```
git pull
python setup.py develop
```

When initializing the spatial tables

No function matches the given name and argument types

```
LINE 1: SELECT AddGeometryColumn('package_extent','the_geom', E'4326...
               ^
HINT:  No function matches the given name and argument types. You might need to add explicit type cas
 "SELECT AddGeometryColumn('package_extent','the_geom', %s, 'GEOMETRY', 2)" ('4326',)
```

PostGIS was not installed correctly. Please check the "Setting up PostGIS" section.

permission denied for relation spatial_ref_sys

```
sqlalchemy.exc.ProgrammingError: (ProgrammingError) permission denied for relation spatial_ref_sys
```

The user accessing the ckan database needs to be owner (or have permissions) of the geometry_columns and spatial_ref_sys tables.

When migrating to an existing PostGIS database

If you are loading a database dump to an existing PostGIS database, you may find errors like

```
ERROR:  type "spheroid" already exists
```

This means that the PostGIS functions are installed, but you may need to create the necessary tables anyway. You can force psql to ignore these errors and continue the transaction with the ON_ERROR_ROLLBACK=on:

```
sudo -u postgres psql -d ckan_default -f /usr/share/postgresql/8.4/contrib/postgis-1.5/postgis.sql -v
```

You will still need to populate the spatial_ref_sys table and change the tables permissions. Refer to the previous section for details on how to do it.

When performing a spatial query

SQL expression, column, or mapped entity expected - got '<class 'ckanext.spatial.model.PackageExtent'>

```
InvalidRequestError: SQL expression, column, or mapped entity expected - got '<class 'ckanext.spatial
```

The spatial model has not been loaded. You probably forgot to add the `spatial_metadata` plugin to your ini configuration file.

Operation on two geometries with different SRIDs

```
InternalError: (InternalError) Operation on two geometries with different SRIDs
```

The spatial reference system of the database geometry column and the one used by CKAN differ. Remember, if you are using a different spatial reference system from the default one (WGS 84 lat/lon, EPSG:4326), you must define it in the configuration file as follows:

```
ckan.spatial.srid = 4258
```

When running the spatial harvesters

```
File "xmlschema.pxi", line 102, in lxml.etree.XMLSchema.__init__ (src/lxml/lxml.etree.c:154475)
lxml.etree.XMLSchemaParseError: local list type: A type, derived by list or union, must have the simp
```

The XSD validation used by the spatial harvesters requires libxml2 ersion 2.9.

With CKAN you would probably have installed an older version from your distribution. (e.g. with `sudo apt-get install libxml2-dev`). You need to find the SO files for the old version:

```
$ find /usr -name "libxml2.so"
```

For example, it may show it here: `/usr/lib/x86_64-linux-gnu/libxml2.so`. The directory of the SO file is used as a parameter to the `configure` next on.

Download the libxml2 source:

```
$ cd ~
$ wget ftp://xmlsoft.org/libxml2/libxml2-2.9.0.tar.gz
```

Unzip it:

```
$ tar zxvf libxml2-2.9.0.tar.gz
$ cd libxml2-2.9.0/
```

Configure with the SO directory you found before:

```
$ ./configure --libdir=/usr/lib/x86_64-linux-gnu
```

Now make it and install it:

```
$ make
$ sudo make install
```

Now check the install by running xmllint:

```
$ xmllint --version
xmllint: using libxml version 20900
 compiled with: Threads Tree Output Push Reader Patterns Writer SAXv1 FTP HTTP DTDValid HTML Legacy C
```

CHAPTER 2

Spatial Search

The spatial extension allows to index datasets with spatial information so they can be filtered via a spatial search query. This includes both via the web interface (see the Spatial Search Widget) or via the action API, e.g.:

```
POST http://localhost:5000/api/action/package_search
    { "q": "Pollution",
      "facet": "true",
      "facet.field": "country",
      "extras": {
          "ext_bbox": "-7.535093,49.208494,3.890688,57.372349" }
    }
```

Changed in version 2.0.1: Starting from this version the spatial filter it is also supported on GET requests:

http://localhost:5000/api/action/package_search?q=Pollution&ext_bbox=-7.535093,49.208494,3.890688,57.372349

Setup

To enable the spatial search you need to add the `spatial_query` plugin to your ini file. This plugin requires the `spatial_metadata` plugin, eg:

```
ckan.plugins = [other plugins] spatial_metadata spatial_query
```

To define which backend to use for the spatial search use the following configuration option (see Choosing a backend for the spatial search):

```
ckanext.spatial.search_backend = solr
```

Geo-Indexing your datasets

Regardless of the backend that you are using, in order to make a dataset searchable by location, it must have a special extra, with its key named 'spatial'. The value must be a valid GeoJSON geometry, for example:

```
{
  "type":"Polygon",
  "coordinates":[[[2.05827, 49.8625],[2.05827, 55.7447], [-6.41736, 55.7447], [-6.41736, 49.8625], [2
}
```

or:

```
{
  "type": "Point",
  "coordinates": [-3.145,53.078]
}
```

Every time a dataset is created, updated or deleted, the extension will synchronize the information stored in the extra with the geometry table.

If you already have datasets when you enable Spatial Search then you'll need to reindex them:

paster –plugin=ckan search-index rebuild –config=/etc/ckan/default/development.ini

Choosing a backend for the spatial search

There are different backends supported for the spatial search, it is important to understand their differences and the necessary setup required when choosing which one to use.

The following table summarizes the different spatial search backends:

Backend	Solr Versions	Supported geometries	Sorting and relevance	Performance with large number of datasets
`solr`	>= 3.1	Bounding Box	Yes, spatial sorting combined with other query parameters	Good
`solr-spatial-field`	>= 4.x	Bounding Box, Point and Polygon [1]	Not implemented	Good
`postgis`	>= 1.3	Bounding Box	Partial, only spatial sorting supported [2]	Poor

[1] Requires JTS

[2] Needs `ckanext.spatial.use_postgis_sorting` set to True

We recommend to use the `solr` backend whenever possible. Here are more details about the available options:

- **`solr` (Recommended)** This option uses normal Solr fields to index the relevant bits of information about the geometry and uses an algorithm function to sort results by relevance, keeping any other non-spatial filtering. It only supports bounding boxes both for the geometries to be indexed and the input query shape. It requires EDisMax query parser, so it will only work on versions of Solr greater than 3.1 (We recommend using Solr 4.x).

  You will need to add the following fields to your Solr schema file to enable it:

  ```
  <fields>
      <!-- ... -->
      <field name="bbox_area" type="float" indexed="true" stored="true" />
      <field name="maxx" type="float" indexed="true" stored="true" />
      <field name="maxy" type="float" indexed="true" stored="true" />
      <field name="minx" type="float" indexed="true" stored="true" />
      <field name="miny" type="float" indexed="true" stored="true" />
  </fields>
  ```

  The solr schema file is typically located at: (..)/src/ckan/ckan/config/solr/schema.xml

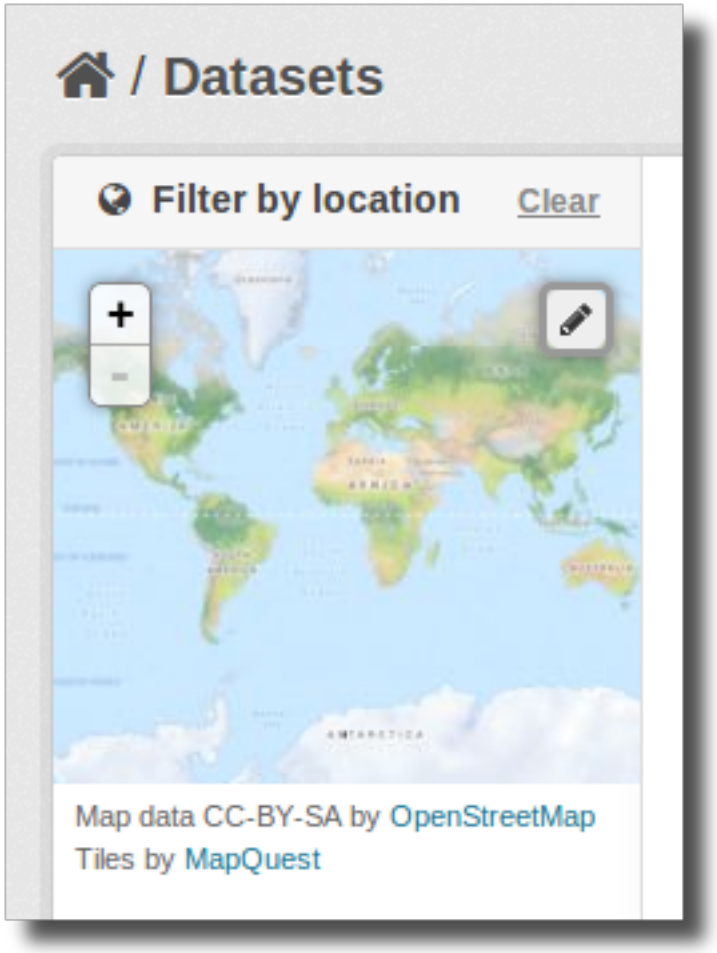
- **`solr-spatial-field`** This option uses the spatial field introduced in Solr 4, which allows to index points, rectangles and more complex geometries (complex geometries will require JTS, check the documentation). Sorting has not yet been implemented, users willing to do so will need to modify the query using the `before_search` extension point.

You will need to add the following field type and field to your Solr schema file to enable it (Check the Solr documentation for more information on the different parameters, note that you don't need `spatialContextFactory` if you are not using JTS):

```
<types>
    <!-- ... -->
    <fieldType name="location_rpt" class="solr.SpatialRecursivePrefixTreeFieldType"
        spatialContextFactory="com.spatial4j.core.context.jts.JtsSpatialContextFactory"
        autoIndex="true"
        distErrPct="0.025"
        maxDistErr="0.000009"
        distanceUnits="degrees" />
</types>
<fields>
    <!-- ... -->
    <field name="spatial_geom"  type="location_rpt" indexed="true" stored="true" multiValued
</fields>
```

- **`postgis`** This is the original implementation of the spatial search. It does not require any change in the Solr schema and can run on Solr 1.x, but it is not as efficient as the previous ones. Basically the bounding box based query is performed in PostGIS first, and the ids of the matched datasets are added as a filter to the Solr request. This, apart from being much less efficient, can led to issues on Solr due to size of the requests (See Solr configuration issues on legacy PostGIS backend). There is support for a spatial ranking on this backend (setting `ckanext.spatial.use_postgis_sorting` to True on the ini file), but it can not be combined with any other filtering.

Spatial Search Widget



The extension provides a snippet to add a map widget to the search form, which allows filtering results by an area of interest.

To add the map widget to the sidebar of the search page, add the following block to the dataset search page template (`myproj/ckanext/myproj/templates/package/search.html`). If your custom theme is simply extending the CKAN default theme, you will need to add `{% ckan_extends %}` to the start of your custom search.html, then continue with this:

```
{% block secondary_content %}

  {% snippet "spatial/snippets/spatial_query.html" %}

{% endblock %}
```

By default the map widget will show the whole world. If you want to set up a different default extent, you can pass an extra `default_extent` to the snippet, either with a pair of coordinates like this:

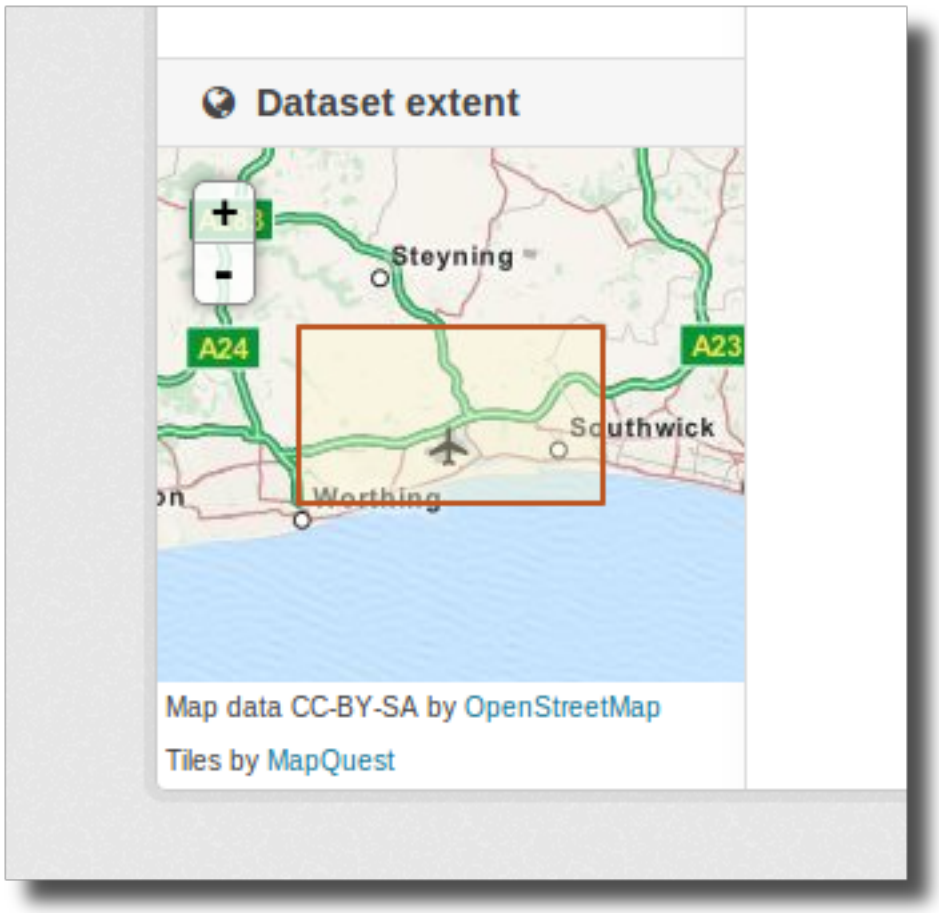
```
{% snippet "spatial/snippets/spatial_query.html", default_extent="[[15.62,
    -139.21], [64.92, -61.87]]" %}
```

or with a GeoJSON object describing a bounding box (note the escaped quotes):

```
{% snippet "spatial/snippets/spatial_query.html", default_extent="{ \"type\":
    \"Polygon\", \"coordinates\": [[[74.89, 29.39],[74.89, 38.45], [60.50,
    38.45], [60.50, 29.39], [74.89, 29.39]]]}" %}
```

You need to load the `spatial_metadata` and `spatial_query` plugins to use this snippet.

Dataset Extent Map



Using the snippets provided, if datasets contain a `spatial` extra like the one described in the previous section, a map will be shown on the dataset details page.

There are snippets already created to load the map on the left sidebar or in the main body of the dataset details page, but these can be easily modified to suit your project needs

To add a map to the sidebar, add the following block to the dataset page template (eg `ckanext-myproj/ckanext/myproj/templates/package/read_base.html`). If your custom theme is simply extending the CKAN default theme, you will need to add `{% ckan_extends %}` to the start of your custom read.html, then continue with this:

```
{% block secondary_content %}
  {{ super() }}

  {% set dataset_extent = h.get_pkg_dict_extra(c.pkg_dict, 'spatial', '') %}
  {% if dataset_extent %}
```

```
    {% snippet "spatial/snippets/dataset_map_sidebar.html", extent=dataset_extent %}
  {% endif %}

{% endblock %}
```

For adding the map to the main body, add this to the main dataset page template (eg `ckanext-myproj/ckanext/myproj/templates/package/read.html`):

```
{% block primary_content_inner %}

  {{ super() }}

  {% set dataset_extent = h.get_pkg_dict_extra(c.pkg_dict, 'spatial', '') %}
  {% if dataset_extent %}
    {% snippet "spatial/snippets/dataset_map.html", extent=dataset_extent %}
  {% endif %}

{% endblock %}
```

You need to load the `spatial_metadata` plugin to use these snippets.

Legacy Search

Solr configuration issues on legacy PostGIS backend

Warning: If you find any of the issues described in this section it is strongly recommended that you consider switching to one of the Solr based backends which are much more efficient. These notes are just kept for informative purposes.

If using Spatial Query functionality then there is an additional SOLR/Lucene setting that should be used to set the limit on number of datasets searchable with a spatial value.

The setting is `maxBooleanClauses` in the solrconfig.xml and the value is the number of datasets spatially searchable. The default is `1024` and this could be increased to say `16384`. For a SOLR single core this will probably be at */etc/solr/conf/solrconfig.xml*. For a multiple core set-up, there will me several solrconfig.xml files a couple of levels below */etc/solr*. For that case, *all* of the cores' *solrconfig.xml* should have this setting at the new value.

Example:

```
<maxBooleanClauses>16384</maxBooleanClauses>
```

This setting is needed because PostGIS spatial query results are fed into SOLR using a Boolean expression, and the parser for that has a limit. So if your spatial area contains more than the limit (of which the default is 1024) then you will get this error:

```
Dataset search error: ('SOLR returned an error running query...
```

and in the SOLR logs you see:

```
too many boolean clauses ...  Caused by:
org.apache.lucene.search.BooleanQuery$TooManyClauses: maxClauseCount is set to
1024
```

Legacy API

The extension adds the following call to the CKAN search API, which returns datasets with an extent that intersects with the bounding box provided:

```
/api/2/search/dataset/geo?bbox={minx,miny,maxx,maxy}[&crs={srid}]
```

If the bounding box coordinates are not in the same projection as the one defined in the database, a CRS must be provided, in one of the following forms:

- *urn:ogc:def:crs:EPSG::4326*
- EPSG:4326
- 4326

CHAPTER 3

Spatial Harvesters

Overview and Configuration

The spatial extension provides some harvesters for importing ISO19139-based metadata into CKAN, as well as providing a base class for writing new ones. The harvesters use the interface provided by ckanext-harvest, so you will need to install and set it up first.

Once ckanext-harvest is installed, you can add the following plugins to your ini file to enable the different harvesters (If you are upgrading from a previous version to CKAN 2.0 see legacy_harvesters):

- `csw_harvester` - CSW server
- `waf_harvester` - WAF (Web Accessible Folder): An online accessible index page with links to metadata documents
- `doc_harvester` - A single online accessible metadata document.

Have a look at the ckanext-harvest documentation if you want to have an overview of how the CKAN harvesters work, but basically there are three separate stages:

- gather_stage - Aggregates all the remote identifiers for a particular source (eg identifiers for a CSW server, files for a WAF).
- fetch_stage - Fetches all the remote documents and stores them on the database.
- import_stage - Performs all the processing for transforming the remote content into a CKAN dataset: validates the document, parses it, converts it to a CKAN dataset dict and saves it in the database.

The extension provides different XSD and schematron based validators, and you can also write your own (see Writing custom validators). You can specify which validators to use for the remote documents with the following configuration option:

```
ckan.spatial.validator.profiles = iso19193eden
```

By default, the import stage will stop if the validation of the harvested document fails. This can be modified setting the `ckanext.spatial.harvest.continue_on_validation_errors` to True. The setting can also be applied at the source level setting to True the `continue_on_validation_errors` key on the source configuration object.

By default the harvesting actions (eg creating or updating datasets) will be performed by the internal site admin user. This is the recommended setting, but if necessary, it can be overridden with the `ckanext.spatial.harvest.user_name` config option, eg to support the old hardcoded 'harvest' user:

```
ckanext.spatial.harvest.user_name = harvest
```

When a document has not been updated remotely, the previous harvest object is replaced by the current one rather than keeping it, to avoid cluttering the `harvest_object` table. This means that the `harvest_object_id` reference on the linked dataset needs to be updated, by reindexing it. This will happen by default, but if you want to turn it off (eg if you are doing separate reindexing) it can be turn off with the following option:

```
ckanext.spatial.harvest.reindex_unchanged = False
```

You can configure the single harvesters using a JSON object in the configuration form field. The currently supported configuration options are:

- `default_tags`: A list of tags that will be added to all harvested datasets. Tags don't need to previously exist. This field takes a list of strings.
- `default_extras`: A dictionary of key value pairs that will be added to extras of the harvested datasets.
- `override_extras`: Assign default extras even if they already exist in the remote dataset. Default is False (only non existing extras are added).
- `clean_tags`: By default, tags are not stripped of accent characters, spaces and capital letters for display. If this option is set to True, accent characters will be replaced by their ascii equivalents, capital letters replaced by lower-case ones, and spaces replaced with dashes. Setting this option to False gives the same effect as leaving it unset.
- `validator_profiles`: A list of string that specifies a list of validators that will be applied to the current harvester, overriding the global ones defined by the 'ckan.spatial.validator.profiles' option.

Customizing the harvesters

The default harvesters provided in this extension can be extended from extensions implementing the `ISpatialHarvester` interface.

Probably the most useful extension point is `get_package_dict`, which allows to tweak the dataset fields before creating or updating it:

```
import ckan.plugins as p
from ckanext.spatial.interfaces import ISpatialHarvester

class MyPlugin(p.SingletonPlugin):

    p.implements(ISpatialHarvester, inherit=True)

    def get_package_dict(self, context, data_dict):

        # Check the reference below to see all that's included on data_dict

        package_dict = data_dict['package_dict']
        iso_values = data_dict['iso_values']

        package_dict['extras'].append(
            {'key': 'topic-category', 'value': iso_values.get('topic-category')}
        )

        package_dict['extras'].append(
            {'key': 'my-custom-extra', 'value': 'my-custom-value'}
        )

        return package_dict
```

`get_validators` allows to register custom validation classes that can be applied to the harvested documents. Check the Writing custom validators section to know more about how to write your custom validators:

```
import ckan.plugins as p
from ckanext.spatial.interfaces import ISpatialHarvester
from ckanext.spatial.validation.validation import BaseValidator

class MyPlugin(p.SingletonPlugin):

    p.implements(ISpatialHarvester, inherit=True)

    def get_validators(self):
        return [MyValidator]


class MyValidator(BaseValidator):

    name = 'my-validator'

    title= 'My very own validator'

    @classmethod
    def is_valid(cls, xml):

        return True, []
```

`transform_to_iso` allows to hook into transformation mechanisms to transform other formats into ISO1939, the only one directly supported by the spatial harvesters.

Here is the full reference for the provided extension points:

class `ckanext.spatial.interfaces.`**`ISpatialHarvester`**

`get_package_dict`(*context*, *data_dict*)

Allows to modify the dataset dict that will be created or updated

This is the dict that the harvesters will pass to the *package_create* or *package_update* actions. Extensions can modify it to suit their needs, adding or removing filds, modifying the default ones, etc.

This method should always return a package_dict. Note that, although unlikely in a particular instance, this method could be implemented by more than one plugin.

If a dict is not returned by this function, the import stage will be cancelled.

Parameters

- **context** (*dict*) – Contains a reference to the model, eg to perform DB queries, and the user name used for authorization.
- **data_dict** (*dict*) – Available data. Contains four keys:
  - ***package_dict*** The default package_dict generated by the harvester. Modify this or create a brand new one.
  - ***iso_values*** The parsed ISO XML document values. These contain more fields that are not added by default to the `package_dict`.
  - ***xml_tree*** The full XML etree object. If some values not present in `iso_values` are needed, these can be extracted via xpath.

- ***harvest_object*** A `HarvestObject` domain object which contains a reference to the original metadata document (`harvest_object.content`) and the harvest source (`harvest_object.source`).

Returns A dataset dict ready to be used by `package_create` or `package_update`

Return type dict

`get_validators`()
Allows to register custom Validators that can be applied to harvested metadata documents.

Validators are classes that implement the `is_valid` method. Check the Writing custom validators section in the docs to know more about writing custom validators.

Returns A list of Validator classes

Return type list

`transform_to_iso`(*original_document*, *original_format*, *harvest_object*)
Transforms an XML document to ISO 19139

This method will be only called from the import stage if the harvest_object content is null and original_document and original_format harvest object extras exist (eg if an FGDC document was harvested).

In that case, this method should do the necessary to provide an ISO 1939 like document, otherwise the import process will stop.

Parameters

- **original_document** (*string*) – Original XML document
- **original_format** (*string*) – Original format (eg 'fgdc')
- **harvest_object** (*HarvestObject*) – HarvestObject domain object (with access to job and source objects)

Returns An ISO 19139 document or None if the transformation was not successful

Return type string

If you need to further customize the default behaviour of the harvesters, you can either extend `CswHarvester`, `WAFfHarverster` or the main `SpatialHarvester` class., for instance to override the whole `import_stage` if the default logic does not suit your needs.

The ckanext-geodatagov extension contains live examples on how to extend the default spatial harvesters and create new ones for other spatial services like ArcGIS REST APIs.

Writing custom validators

Validator classes extend the `BaseValidator` class:

Helper classes are provided for XSD and schematron based validation, and completely custom logic can be also implemented. Here are some examples of the most common types:

- XSD based validators:

```
class ISO19139NGDCSchema(XsdValidator):
    '''
    XSD based validation for ISO 19139 documents.

    Uses XSD schema from the NOAA National Geophysical Data Center:
```

```
    http://ngdc.noaa.gov/metadata/published/xsd/

    '''
    name = 'iso19139ngdc'
    title = 'ISO19139 XSD Schema (NGDC)'

    @classmethod
    def is_valid(cls, xml):
        xsd_path = 'xml/iso19139ngdc'

        xsd_filepath = os.path.join(os.path.dirname(__file__),
                                    xsd_path, 'schema.xsd')
        return cls._is_valid(xml, xsd_filepath, 'NGDC Schema (schema.xsd)')
```

- Schematron validators:

```
class Gemini2Schematron(SchematronValidator):
    name = 'gemini2'
    title = 'GEMINI 2.1 Schematron 1.2'

    @classmethod
    def get_schematrons(cls):
        with resource_stream("ckanext.spatial",
                             "validation/xml/gemini2/gemini2-schematron-20110906-v1.2.sch") as s
            return [cls.schematron(schema)]
```

- Custom validators:

```
class MinimalFGDCValidator(BaseValidator):

    name = 'fgdc_minimal'
    title = 'FGDC Minimal Validation'

    _elements = [
        ('Identification Citation Title', '/metadata/idinfo/citation/citeinfo/title'),
        ('Identification Citation Originator', '/metadata/idinfo/citation/citeinfo/origin'),
        ('Identification Citation Publication Date', '/metadata/idinfo/citation/citeinfo/pubdate
        ('Identification Description Abstract', '/metadata/idinfo/descript/abstract'),
        ('Identification Spatial Domain West Bounding Coordinate', '/metadata/idinfo/spdom/bound
        ('Identification Spatial Domain East Bounding Coordinate', '/metadata/idinfo/spdom/bound
        ('Identification Spatial Domain North Bounding Coordinate', '/metadata/idinfo/spdom/boun
        ('Identification Spatial Domain South Bounding Coordinate', '/metadata/idinfo/spdom/boun
        ('Metadata Reference Information Contact Address Type', '/metadata/metainfo/metc/cntinfo
        ('Metadata Reference Information Contact Address State', '/metadata/metainfo/metc/cntinf
        ]

    @classmethod
    def is_valid(cls, xml):

        errors = []

        for title, xpath in cls._elements:
            element = xml.xpath(xpath)
            if len(element) == 0 or not element[0].text:
                errors.append(('Element not found: {0}'.format(title), None))
        if len(errors):
            return False, errors

        return True, []
```

The validation.py file included in the ckanext-spatial extension contains more examples of the different types.

Remember that after registering your own validators you must specify them on the following configuration option:

```
ckan.spatial.validator.profiles = iso19193eden,my-validator
```

Harvest Metadata API

This plugin allows to access the actual harvested document via API requests. It is enabled with the following plugin:

```
ckan.plugins = spatial_harvest_metadata_api
```

(It was previously known as `inspire_api`)

To view the harvest objects (containing the harvested metadata) in the web interface, these controller locations are added:

- raw XML document: /harvest/object/{id}
- HTML representation: /harvest/object/{id}/html

Note: The old URLs are now deprecated and redirect to the previously mentioned:

- /api/2/rest/harvestobject/<id>/xml
- /api/2/rest/harvestobject/<id>/html

For those harvest objects that have an original document (which was transformed to ISO), this can be accessed via:

- raw XML document: /harvest/object/{id}/original
- HTML representation: /harvest/object/{id}/html/original

The HTML representation is created via an XSLT transformation. The extension provides an XSLT file that should work on ISO 19139 based documents, but if you want to use your own on your extension, you can override it using the following configuration options:

```
ckanext.spatial.harvest.xslt_html_content = ckanext.myext:templates/xslt/custom.xslt
ckanext.spatial.harvest.xslt_html_content_original = ckanext.myext:templates/xslt/custom2.xslt
```

If your project does not transform different metadata types you can ignore the second option.

Legacy harvesters

Prior to CKAN 2.0, the spatial harvesters available on this extension were based on the GEMINI2 format, an ISO19139 profile used by the UK Location Programme, and the logic for creating or updating datasets and the resulting fields were somehow adapted to the needs for this particular project. The harvesters were still generic enough and should work fine with other ISO19139 based sources, but extra care has been put to make the new harvesters more generic and robust, so these ones should only be used on existing instances:

- `gemini_csw_harvester`
- `gemini_waf_harvester`
- `gemini_doc_harvester`

If you are using these harvesters please consider upgrading to the new versions described on the previous section.

CHAPTER 4

CSW support

The extension provides the support for the CSW standard, a specification from the Open Geospatial Consortium for exposing geospatial catalogues over the web.

This support consists of:

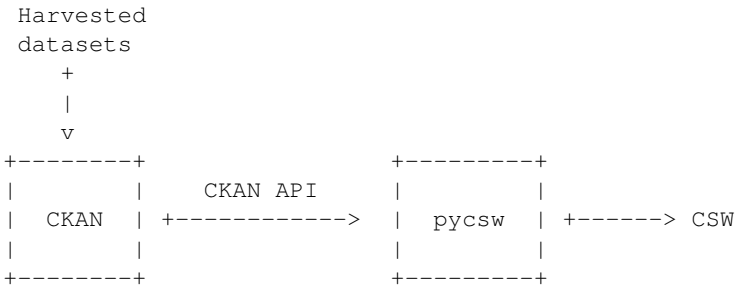
- Ability to import records from CSW servers with the CSW harvester. See *Spatial Harvesters* for more details.
- Integration with pycsw to provide a fully compliant CSW interface for harvested records. This integration is described in the following sections.

ckan-pycsw

The spatial extension offers the `ckan-pycsw` command, which allows to expose the spatial datasets harvested from other sources in a CSW interface. This is powered by pycsw, which fully implements the OGC CSW specification.

How it works

The current implementation is based on CKAN and pycsw being loosely integrated via the CKAN API. pycsw will be generally installed in the same server as CKAN (although it can also be run on a separate one), and the synchronization command will be run regularly to keep the records on the pycsw repository up to date. This is done using the CKAN API to get all the datasets identifiers (more precisely the ones from datasets that have been harvested) and then deciding which ones need to be created, updated or deleted on the pycsw repository. For those that need to be created or updated, the original harvested spatial document (ie ISO 19139) is requested from CKAN, and it is then imported using pycsw internal functions:



Remember, only datasets that were harvested with the *Spatial Harvesters* can currently be exposed via pycsw.

All necessary tasks are done with the `ckan-pycsw` command. To get more details of its usage, run the following:

```
cd /usr/lib/ckan/default/src/ckanext-spatial
paster ckan-pycsw --help
```

Setup

1. Install pycsw. There are several options for this, depending on your server setup, check the pycsw documentation.

   Note: CKAN integration requires least pycsw version 1.8.0. In general, use the latest stable version.

   The following instructions assume that you have installed CKAN via a package install and should be run as root, but the steps are the same if you are setting it up in another location:

   ```
   cd /usr/lib/ckan/default/src
   source ../bin/activate

   # From now on the virtualenv should be activated

   git clone https://github.com/geopython/pycsw.git
   cd pycsw
   # always use the latest stable version
   git checkout 1.10.4
   pip install -e .
   python setup.py build
   python setup.py install
   ```

2. Create a database for pycsw. In theory you can use the same database that CKAN is using, but if you want to keep them separated, use the following command to create a new one (we'll use the same default user though):

   ```
   sudo -u postgres createdb -O ckan_default pycsw -E utf-8
   ```

   It is strongly recommended that you install PostGIS in the pycsw databaset, so its spatial functions are used. See the *Install PostGIS and system packages* section for details.

3. Configure pycsw. An example configuration file is included on the source:

   ```
   cp default-sample.cfg default.cfg
   ```

   To keep things tidy we will create a symlink to this file on the CKAN configuration directory:

   ```
   ln -s /usr/lib/ckan/default/src/pycsw/default.cfg /etc/ckan/default/pycsw.cfg
   ```

   Open the file with your favourite editor. The main settings you should tweak are `server.home` and `repository.database`:

   ```
   [server]
   home=/usr/lib/ckan/default/src/pycsw
   ...
   [repository]
   database=postgresql://ckan_default:pass@localhost/pycsw
   ```

   The rest of the options are described here.

4. Setup the pycsw table. This is done with the `ckan-pycsw` paster command (Remember to have the virtualenv activated when running it):

```
cd /usr/lib/ckan/default/src/ckanext-spatial
paster ckan-pycsw setup -p /etc/ckan/default/pycsw.cfg
```

At this point you should be ready to run pycsw with the wsgi script that it includes:

```
cd /usr/lib/ckan/default/src/pycsw
python csw.wsgi
```

This will run pycsw at http://localhost:8000. Visiting the following URL should return you the Capabilities file:

http://localhost:8000/?service=CSW&version=2.0.2&request=GetCapabilities

5. Load the CKAN datasets into pycsw. Again, we will use the `ckan-pycsw` command for this:

```
cd /usr/lib/ckan/default/src/ckanext-spatial
paster ckan-pycsw load -p /etc/ckan/default/pycsw.cfg
```

When the loading is finished, check that results are returned when visiting this link:

http://localhost:8000/?request=GetRecords&service=CSW&version=2.0.2&resultType=results&outputSchema=http://www.isotc

The `numberOfRecordsMatched` should match the number of harvested datasets in CKAN (minus import errors). If you run the command again new or udpated datasets will be synchronized and deleted datasets from CKAN will be removed from pycsw as well.

Setting Service Metadata Keywords

The CSW standard allows for administrators to set CSW service metadata. These values can be set in the pycsw configuration `metadata:main` section. If you would like the CSW service metadata keywords to be reflective of the CKAN tags, run the following convenience command:

```
paster ckan-pycsw set_keywords -p /etc/ckan/default/pycsw.cfg
```

Note that you must have privileges to write to the pycsw configuration file.

Running it on production site

On a production site you probably want to run the load command regularly to keep CKAN and pycsw in sync, and serve pycsw with Apache + mod_wsgi like CKAN.

- To run the load command regularly you can set up a cron job. Type `crontab -e` and copy the following lines:

```
# m h  dom mon dow   command
0 *  *   *   *     /usr/lib/ckan/default/bin/paster --plugin=ckanext-spatial ckan-pycsw load -p
```

This particular example will run the load command every hour. You can of course modify this periodicity, for instance reducing it for huge instances. This Wikipedia page has a good overview of the crontab syntax.

- To run pycsw under Apache check the pycsw installation documentation or follow these quick steps (they assume the paths used in previous steps):
  - Edit `/etc/apache2/sites-available/ckan_default` and add the following line just before the existing `WSGIScriptAlias` directive:

```
WSGIScriptAlias /csw /usr/lib/ckan/default/src/pycsw/csw.wsgi
```

  - Edit the `/usr/lib/ckan/default/src/pycsw/csw.wsgi` file and add these two lines just after the imports on the top of the file:

```
activate_this = os.path.join('/usr/lib/ckan/default/bin/activate_this.py')
execfile(activate_this, {"__file__":activate_this})
```

We need these to activate the virtualenv where we installed pycsw into.

- Restart Apache:

```
service apache2 restart
```

pycsw should be now accessible at http://localhost/csw

CHAPTER 5

Previews for Spatial Formats

Note: The view plugins for rendering spatial formats have been moved to ckanext-geoview, which contains view plugins based on OpenLayers and Leaflet to display several geospatial files and services in CKAN.

CHAPTER 6

Common base layers for Map Widgets

To provide a consistent look and feel and avoiding code duplication, the map widgets (at least the ones based on Leaflet) can use a common function to create the map. The base layer that the map will use can be configured via configuration options.

Configuring the base layer

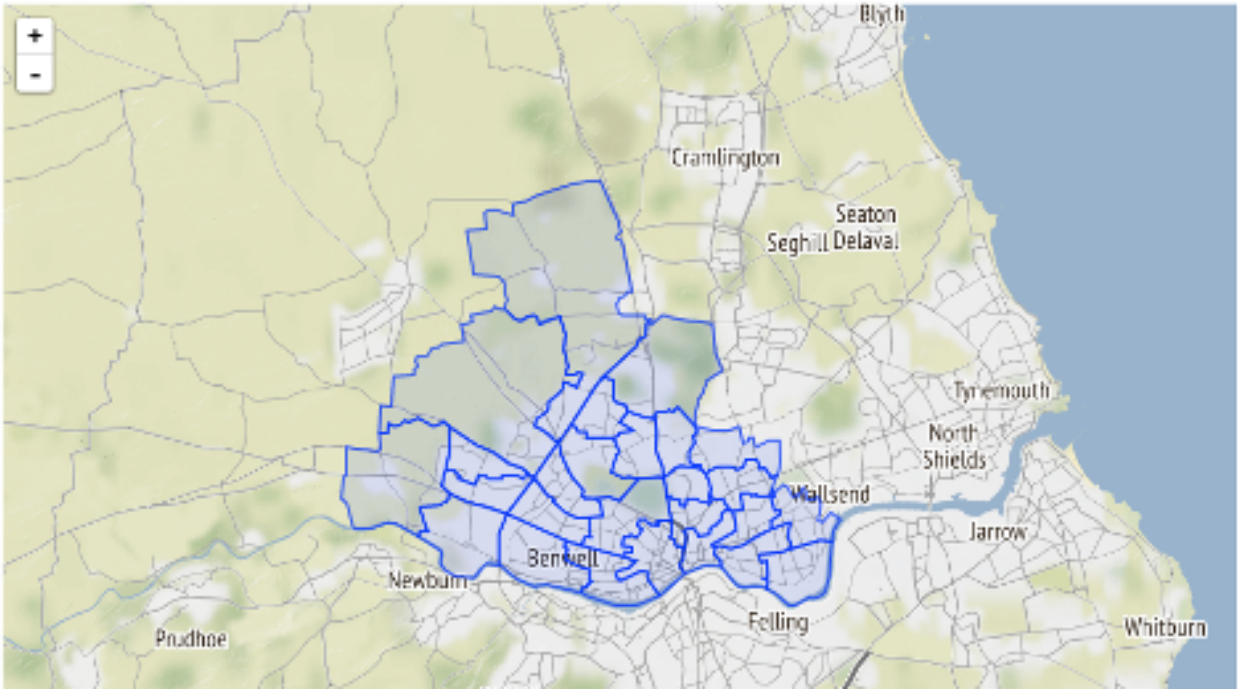
The main configuration option to manage the base layer used is `ckanext.spatial.common_map.type`. Depending on the map type additional options may be required. The spatial extension provides default settings for popular tiles providers based on OpenStreetMap, but you can use any tileset that follows the XYZ convention.

Note: All tile providers have Terms of Use and will most likely require proper attribution. Make sure to read and understand the terms and add the relevant attribution before using them on your CKAN instance.

Stamen Terrain

The Terrain tiles are provided by Stamen, and are based on data by OpenStreetMap. This is the default base layer used by the map widgets, and you don't need to add any configuration option to use them. If you want to define it explicitly though, use the following setting:

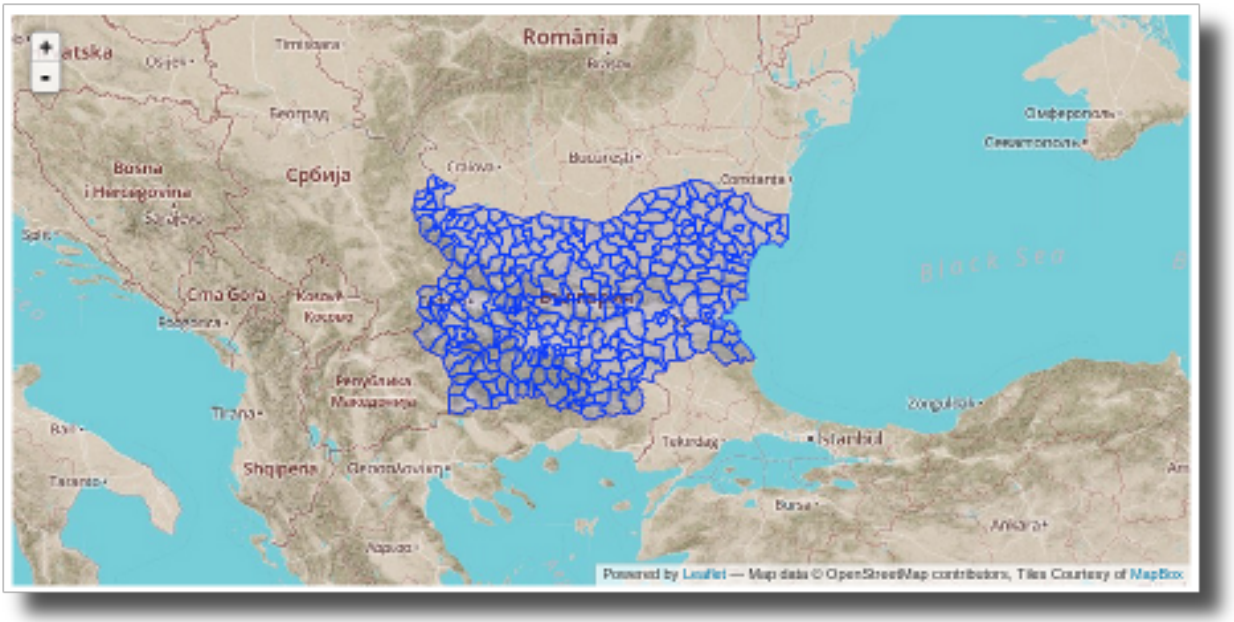
```
ckanext.spatial.common_map.type = stamen
```



MapBox

MapBox allows to define your custom maps based on OpenStreetMap data, using their online editor or the more advanced MapBox Studio desktop tool. You will need to provide a map id with the `[account].[handle]` form and and access token (Check here for more details):

```
ckanext.spatial.common_map.type = mapbox
ckanext.spatial.common_map.mapbox.map_id = youraccount.map-xxxxxxxx
ckanext.spatial.common_map.mapbox.access_token = pk.ey...
```



Custom

You can use any tileset that follows the XYZ convention using the `custom` type:

```
ckanext.spatial.common_map.type = custom
```

You will need to define the tileset URL using `ckanext.spatial.common_map.custom.url`. This follows the Leaflet URL template format (ie {s} for subdomains if any, {z} for zoom and {x} {y} for tile coordinates). Additionally you can use `ckanext.spatial.common_map.subdomains` and `ckanext.spatial.common_map.attribution` if needed (these two will also work for Stamen and MapBox layers if you want to tweak the defaults.

This is a complete example that uses Stamen's famous watercolor maps:

```
ckanext.spatial.common_map.type = custom
ckanext.spatial.common_map.custom.url = http://tile.stamen.com/watercolor/{z}/{x}/{y}.jpg
ckanext.spatial.common_map.attribution = Map tiles by <a href="http://stamen.com">Stamen Design</a>,
```

This is a example using TMS:

```
ckanext.spatial.common_map.type = custom
ckanext.spatial.common_map.custom.url = /url/to/your/tms/{z}/{x}/{y}.png
ckanext.spatial.common_map.tms = true
```

Note: For custom base layers you need to manually modify the attribution link on the templates for widgets on the sidebar, like the spatial query and dataset map widgets.

For developers

To pass the base map configuration options to the relevant Javascript module that will initialize the map widget, use the `h.get_common_map_config()` helper function. This is available when loading the `spatial_metadata` plugin. If you don't want to require this plugin, create a new helper function that points to it to avoid duplicating the names, which CKAN won't allow (see for instance how the GeoJSON preview plugin does it).

The function will return a dictionary with all configuration options that relate to the common base layer (that's all that start with `ckanext.spatial.common_map.`)

You will need to dump the dict as JSON on the `data-module-map_config` attribute (see for instance the `dataset_map_base.html` and `spatial_query.html` snippets):

```
{% set map_config = h.get_common_map_config() %}
<div class="dataset-map" data-module="spatial-query" ... data-module-map_config="{{ h.dump_json(map_c
  <div id="dataset-map-container"></div>
</div>
<div id="dataset-map-attribution">
  {% snippet "spatial/snippets/map_attribution.html", map_config=map_config %}
</div>
```

Once at the Javascript module level, all Leaflet based map widgets should use the `ckan.commonLeafletMap` constructor to initialize the map. It accepts the following parameters:

- `container`: HTML element or id of the map container
- `mapConfig`: (Optional) CKAN config related to the common base layer

- `leafletMapOptions`: (Optional) Options to pass to the Leaflet Map constructor
- `leafletBaseLayerOptions`: (Optional) Options to pass to the Leaflet TileLayer constructor

Most of the times you will want to do something like this for a sidebar map:

```
var map = ckan.commonLeafletMap('dataset-map-container', this.options.map_config, {attributionControl
```

And this for a primary content map:

```
var map = ckan.commonLeafletMap('map', this.options.map_config);
```

G

I

T