## CONTENTS

1 **Installation and Setup**  
1.1 Install the extension  .................................................. 3  
1.2 Configuration ............................................................. 4  
1.3 Legacy PostGIS support .................................................. 4  
1.4 Troubleshooting ........................................................... 4

2 **Spatial Search**  
2.1 Setup ................................................................. 7  
2.2 Geo-Indexing your datasets .............................................. 7  
2.3 Spatial Search Widget ................................................... 11  
2.4 Dataset Extent Map ...................................................... 12

3 **Spatial Harvesters**  
3.1 Overview and Configuration ............................................ 15  
3.2 Customizing the harvesters ............................................. 16  
3.3 Writing custom validators .............................................. 17  
3.4 Harvest Metadata API ................................................... 19  
3.5 Legacy harvesters ....................................................... 20

4 **CSW support**  
4.1 ckan-pycsw ........................................................... 21

5 **Previews for Spatial Formats**  

6 **Common base layers for Map Widgets**  
6.1 Configuring the base layer ............................................. 27  
6.2 For developers .......................................................... 29
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 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
ONE

INSTALLATION AND SETUP

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

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: Starting from ckanext-spatial 2.0.0 PostGIS is no longer required to use the extension, and its use has been deprecated. If for some reason you still need to use the old PostGIS backend see Legacy PostGIS support.

All commands assume an existing CKAN database named ckan_default.

1.1 Install the extension

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.

1. Install some packages needed by the extension dependencies:

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

2. Activate your CKAN virtual environment, for example:

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

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

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

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

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

5. 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.

### 1.2 Configuration

Add the following plugins to the ckan.plugins directive in the CKAN ini file:

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

### 1.3 Legacy PostGIS support

Starting from ckanext-spatial 2.0.0 PostGIS is no longer required to use the extension, and its use has been deprecated. If for some reason you are using the deprecated postgis search backend or you still need to store the dataset extents in the PostGIS enabled package_extent table you can use the following steps, but note that this support will removed in the future so update your workflows accordingly.

1. Add the ckan.spatial.use_postgis=true config option to your ini file.
2. Create the package_extent table with the following command (with the virtualenv activated):

   ```bash
   (pyenv) $ ckan --config=mysite.ini spatial initdb [srid]
   ```

   On CKAN 2.8 and below use:

   ```bash
   (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.

   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:

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

### 1.4 Troubleshooting

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

#### 1.4.1 When upgrading the extension to a newer version

**ckan.plugins.core.PluginNotFoundException: geojson_view**

```python
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 <module>
    class GeoJSONPreview(GeoJSONView):
File '/home/pyenvs/spatial/local/lib/python2.7/site-packages/pyutilib/component/core/core.py', line 732 in __new__
    return PluginMeta.__new__(cls, name, bases, d)
File '/home/pyenvs/spatial/local/lib/python2.7/site-packages/pyutilib/component/core/core.py', line 659 in __new__
    raise PluginError('Plugin class %r does not implement an interface, and it has already been defined in environment %r', (str(name), PluginGlobals.env().name))
pyutilib.component.core.core.PluginError: Plugin class 'GeoJSONPreview' does not implement an interface, and it has already been defined in environment '
pca'
```

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:

```
1.4. Troubleshooting
```
1.4.2 When running the spatial harvesters

The XSD validation used by the spatial harvesters requires libxml2 version 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 2.9.0
    compiled with: Threads Tree Output Push Reader Patterns Writer SAXv1 FTP HTTP DTDValid
    HTML Legacy C14N Catalog XPath XPointer XInclude Iconv ISO8859X Unicode Regexps
    Automata Expr Schemas Schematron Modules Debug Zlib
```
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.:

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

The `ext_bbox` parameter must be provided in the form `ext_bbox={minx},{miny},{maxx},{maxy}`

### 2.1 Setup

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

```
ckan.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-bbox
```

### 2.2 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 the location information (a geometry), indexed in Solr. You can provide this information in two ways.

#### 2.2.1 The spatial extra field

The easiest way to get your geometries indexed is to use an extra field named `spatial`. The value of this extra should 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],[2.05827,49.8625]]
}
```

or:
Every time a dataset is created, updated or deleted, the extension will index the information stored in the `spatial` in Solr, so it can be reflected on spatial searches.

If you already have datasets when you enable Spatial Search then you’ll need to **rebuild the search index**.

### 2.2.2 Custom indexing logic

You might not want to use the `spatial` extra field. Perhaps you don’t want to store the geometries in the dataset metadata but prefer to do so in a separate table, or you simply want to perform a different processing on the geometries before indexing.

In this case you need to implement the `before_dataset_index()` method of the `IPackageController` interface:

```python
def before_dataset_index(self, dataset_dict):
    # When using the default `solr-bbox` backend (based on bounding boxes), you need to
    # include the following fields in the returned dataset_dict:
    dataset_dict["minx"] = minx
    dataset_dict["maxx"] = maxx
    dataset_dict["miny"] = miny
    dataset_dict["maxy"] = maxy

    # When using the `solr-spatial-field` backend, you need to include the `spatial_geom`
    # field in the returned dataset_dict. This should be a valid geometry in WKT format.
    # Shapely can help you get the WKT representation of your geometry if you have it in GeoJSON:

    shape = shapely.geometry.shape(geometry)
    wkt = shape.wkt

    dataset_dict["spatial_geom"] = wkt

    # Don't forget to actually return the dict!
    return dataset_dict
```

Some things to keep in mind:

- Remember, you only need to provide one field, either `spatial_bbox` or `spatial_geom`, depending on the backend chosen.

- All indexed geometries should fall within the -180, -90, 180, 90 bounds. If you have polygons crossing the antimeridian (i.e. with longituded lower than -180 or bigger than 180) you’ll have to split them across the antimeridian.

- Check the default implementation of `before_dataset_index()` in `ckanext/spatial/plugins/__init__.py` for extra useful checks and validations.

- If you want to store the geometry in the `spatial` field but don’t want to apply the default automatic indexing logic applied by ckanext-spatial just remove the field from the dict (this won’t remove it from the dataset metadata, just...
from the indexed data):

```python
def before_dataset_search(self, dataset_dict):
    dataset_dict.pop("spatial", None)
    return dataset_dict
```

### 2.2.3 Choosing a backend for the spatial search

Ckanext-spatial uses Solr to power the spatial search. The current implementation is tested on Solr 8, which is the supported version, although it might work on previous Solr versions.

**Note:** The are official Docker images for Solr that have all the configuration needed to perform spatial searches (look for the ones with a `-spatial` suffix). This is the easiest way to get started but if you need to customize Solr yourself see below for the modifications needed.

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. To configure the search backend use the following configuration option:

```
ckanext.spatial.search_backend = solr-bbox | solr-spatial-field
```

The following table summarizes the different spatial search backends:

<table>
<thead>
<tr>
<th>Backend</th>
<th>Supported geometries indexed in Solr</th>
<th>Solr setup needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>solr-bbox (default)</td>
<td>Bounding Box, Polygon (extents only)</td>
<td>Custom fields</td>
</tr>
<tr>
<td>solr-spatial-field</td>
<td>Bounding Box, Point and Polygon</td>
<td>Custom field + JTS</td>
</tr>
</tbody>
</table>

**Note:** The default `solr-bbox` search backend was previously known as `solr`. Please update your configuration if using this version as it will be removed in the future.

The `solr-bbox` backend is probably a good starting point. Here are more details about the available options (again, you don’t need to modify Solr if you are using one of the spatially enabled official Docker images):

- **solr-bbox**
  
  This option always indexes just the extent of the provided geometries, whether if it’s an actual bounding box or not. It supports spatial sorting of the returned results (based on the closeness of their bounding box to the query bounding box). It uses standard Solr float fields so you just need to add the following to your Solr schema:

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

- **solr-spatial-field**
  
  This option uses the RPT Solr field, which allows to index points, rectangles and more complex geometries.

2.2. Geo-Indexing your datasets
like polygons. This requires the install of the JTS library. See the linked Solr documentation for details on this. Note that it does not support spatial sorting of the returned results. You will need to add the following field type and field to your Solr schema file to enable it

```xml
<types>
  <!-- ... -->
  <fieldType name="location_rpt" class="solr.SpatialRecursivePrefixTreeFieldType"
    spatialContextFactory="JTS"
    autoIndex="true"
    validationRule="repairBuffer0"
    distErrPct="0.025"
    maxDistErr="0.001"
    distanceUnits="kilometers" />
</types>

<fields>
  <!-- ... -->
  <field name="spatial_geom" type="location_rpt" indexed="true" multiValued="true" />
</fields>
```

By default, the solr-spatial-field backend uses the following query. This can be customized by setting the ckanext.spatial.solr_query configuration option, but note that all placeholders must be included:

```
"{{!field f=spatial_geom}}Intersects(ENVELOPE({minx}, {maxx}, {maxy}, {miny}))"
```

**Note:** The old postgis search backend is deprecated and will be removed in future versions of the extension. You should migrate to one of the other backends instead but if you need to keep using it for a while see Legacy PostGIS support.
2.3 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:

```html
{% 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:

```html
{% 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):

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

### 2.4 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 (e.g. 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:

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

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

(continues on next page)
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.
3.1 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:

```plaintext
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:

```plaintext
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.

### 3.2 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:

```python
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:

```python
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:

If you need to further customize the default behaviour of the harvesters, you can either extend CswHarvester, WAfSHarvester 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.

## 3.3 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:

  ```python
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/
    """

    (continues on next page)
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 schema:
            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/bounding/westbc'),
        ('Identification Spatial Domain East Bounding Coordinate', '/metadata/idinfo/spdom/bounding/eastbc'),
        ('Identification Spatial Domain North Bounding Coordinate', '/metadata/idinfo/spdom/bounding/northbc'),
        ('Identification Spatial Domain South Bounding Coordinate', '/metadata/idinfo/spdom/bounding/southbc'),
        ('Metadata Reference Information Contact Address Type', '/metadata/metainfo/metc/cntinfo/cntaddr/addrtype'),
    ]
@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

### 3.4 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:
If your project does not transform different metadata types you can ignore the second option.

### 3.5 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.
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.

### 4.1 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.

#### 4.1.1 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:

```plaintext
Harvested datasets
  +
  | v
  +--------+ +---------+ +---------+ +---------+
  |      |    |    |          |
  | CKAN  | CKAN API | pycsw | cs
  |       |        |      |    +------> CSW
+--------+ +-------+
```

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:

```bash
21
```
### 4.1.2 Setup

1. Install pycsw. There are several options for this, depending on your server setup, check the [pycsw documentation](#).

   **Note:** CKAN integration requires at 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 database, so its spatial functions are used.

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` script (Remember to have the virtualenv activated when running it):

```
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
python bin/ckan_pycsw.py load -p /etc/ckan/default/pycsw.cfg
```

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


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

### 4.1.3 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:

```
python ckan_pycsw.py set_keywords -p /etc/ckan/default/pycsw.cfg
```

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

### 4.1.4 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 * * * * /var/lib/ckan/default/bin/python /var/lib/ckan/default/src/ckanext-spatial/bin/ckan_pycsw.py load -p /etc/ckan/default/pycsw.cfg
```

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](https://en.wikipedia.org/wiki/Cron) 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:

```python
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:

```bash
service apache2 restart
```

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

6.1 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.

6.1.1 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
```
6.1.2 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 an access token (Check here for more details):

```python
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...
```
6.1.3 Custom

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

```plaintext
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:

```plaintext
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>, under <a href="http://creativecommons.org/licenses/by/3.0">CC BY 3.0</a>. Data by <a href="http://openstreetmap.org">OpenStreetMap</a>, under <a href="http://creativecommons.org/licenses/by-sa/3.0">CC BY SA</a>.
```

This is an example using TMS:

```plaintext
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.

6.2 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):

```plaintext
{% set map_config = h.get_common_map_config() %}
<div class="dataset-map" data-module="spatial-query" ... data-module-map_config="{{ h.dump_json(map_config) }}"></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:

6.2. For developers
• 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:

```javascript
var map = ckan.commonLeafletMap('dataset-map-container', this.options.map_config,
   {attributionControl: false});
```

And this for a primary content map:

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