Syntool → Develop and customize

Developer guide

OceanDataLab | syntool@oceandatalab.com | 2018-10-12

Introduction

The objective of this document is to provide examples and hints on how to add new products in a Syntool portal and how to activate some non-obvious features offered by the application.

Please note that this guide uses some paths and URLs from the installation manual.

The last sections of this guide cover a large number of unrelated subjects, so you should use the search functionality of your PDF reader to find keywords if you are looking for specific information.

Conversion

The converter reads raw files in various formats, extracts the data that must be displayed in the Syntool portal and saves it along with some metadata in a GeoTIFF file.

The generated GeoTIFF **must** contain the following global metadata:

`product_name`	a string that identifies the product (e.g. `"GlobCurrent_L4_geostrophic_nrt"`)
`name`	a string which identifies the granule (**must** be unique within the product). Usually built using the name of the input file, for example: `"20170414000000-GLOBCURRENT-L4-CURgeo_0m-ALT_OI_NRT-v03.0-fv01.0"`
`datetime`	a string which provides the central datetime of the temporal coverage for the granule. **Must** be provided with the YYYY-mm-ddTHH:MM:SS format (e.g. `"2017-04-14T00:00:00"`)
`time_range`	two strings separated by a single space that give the span of the temporal coverage around the central datetime. Each string must be composed of an integer value with a time unit (possible units: d | h | m | s | ms, respectively days | hours | minutes | seconds | milliseconds). For example: `"-12h" "+12h"`

Each band of the GeoTIFF must also provide two metadata (self-describing):

- `units`
- `description`

Ad-hoc script

A simple way to start is to write a Python (version 2.7) script which contains:

- a method which takes an input path and an output path
- a main "method" so that the script can be called directly

You can use the following as a template:

```python
# -*- encoding: utf-8 -*-
import logging

logger = logging.getLogger(__name__)


def convert_data(input_path, output_path):
    """Convert input data into a GeoTIFF file containing the metadata
    expected by syntool-ingestor"""

    logger.debug('Converting {} into {}'.format(input_path, output_path))


if '__main__' == __name__:
    import sys

    # Setup logging
    main_logger = logging.getLogger()
    main_logger.handlers = []
    handler = logging.StreamHandler()
    handler.setLevel(logging.DEBUG)
    main_logger.addHandler(handler)
    main_logger.setLevel(logging.DEBUG)

    input_path = sys.argv[1]
    output_path = sys.argv[2]

    convert_data(input_path, output_path)
```

The `syntool_converter` Python package provides some helper methods to create the GeoTIFF files while making sure that the metadata are in the expected format.

For a more detailed description of the conversion process, you can refer to the comments in the https://ftp.odl.bzh/odl/syntool/reader_skeleton.py file.

The `syntool_converter` Python package also contains a large number of readers that can be used as a base for your own readers.

Integration with syntool-converter

Once the ad-hoc script generates GeoTIFF files and you are satisfied with the result, you can either use the script "as is" or package it so that other users may benefit from it.

`syntool-converter` includes a rudimentary mechanism to add readers without actually modifying the `syntool_converter` Python package: it uses a configuration file to define the available readers and reads the path of this configuration file in the `SYNTOOL_CONVERTER_CONFIG` environment variable.

The configuration file is written in JSON format and contains a single dictionary.

- The keys of the dictionary are the keywords you pass to the `-t` option of `syntool-converter` to select the reader.
- The values are lists of two elements: the first element is the path of the Python module which contains the reader (the argument you would pass to the `import` statement in Python) and the second element is the name of the method which implements the conversion.

For example, with this JSON configuration file:

```
{ "keyword": ["path.to.python.module", "name_of_conversion_method"],
  ...
}
```

Calling `syntool-converter -t keyword -o output_path -i input_path -opt foo=bar` will result in invoking a Python code similar to:

```
import path.to.python.module
path.to.python.module.name_of_conversion_method(input_path, output_path,
                                                foo=bar)
```

Note that the Python module for the reader must be in the `PYTHONPATH` otherwise it will not be possible to import it. For an ad-hoc script which has not be packaged yet, it means that you will probably have to edit `PYTHONPATH` when calling `syntool-converter` or it will not work:

```
PYTHONPATH=. \  # include current directory in PYTHONPATH
SYNTOOL_CONVERTER_CONFIG=config.json \  # override configuration file
syntool-converter -t myreader -o converted -i input_file -opt opt1=valueA
```

Note

The list of additional options that can be passed to the reader method with -opt is currently hardcoded in the syntool-converter script. If you need to add an option which is not supported at the moment, you will have to modify the code of syntool-converter or use the ad-hoc script directly.

Ingestion

The ingestor reads the GeoTIFF files generated by `syntool-converter`, transforms them into a web-compatible representation (a pyramid of PNG tiles, a single PNG, a GeoJSON file, etc...) and extracts the metadata required to feed the database.

> ***Note***
>
> It can be difficult to get ingestion settings right from the get-go, so it is better to start with a single GeoTIFF input file, then tweak options until the syntool-ingestor command produces a result and follow the instructions to export data to MySQL and configure the web portal so you can actually see the result of the ingestion process.

Configuration

In order to extract the data from the GeoTIFF and produce the web-compatible representation, `syntool-ingestor` requires some hints about the GeoTIFF file:

- Format: for the current version of `syntool-ingestor` you should always use `geotiff` as input format (it is the default value). It might accept other formats (NetCDF) in the future.
- Projection (pixels): although some information about the projection are already available in the metadata of the GeoTIFF file, it is recommended to pass it directly to `syntool-ingestor` on the command line as an EPSG code.
- Projection (angles): for GeoTIFF which contain vector fields, you **must** provide the EPSG code for the projection wherein the angles are valid.

  If the EPSG code is the same for angles and for the Syntool portal, the angle values are not transformed during the ingestion process.

  Otherwise `syntool-ingestor` will try to convert the angles so that they remain meaningful in the output projection (only works if the projection for angles is EPSG 4326).

The user also needs to describe the expected output of the ingestion:

- Output format: `syntool-ingestor` uses plugins to generate the web-compatible representations.

  These plugins are identified by a keyword that **must** be specified when invoking `syntool-ingestor` on the command line.

  The most common plugins are:

  - `raster` generates a single PNG image from a 2D image
  - `rastertiles` generates a pyramid of PNG tiles from a 2D image
  - `trajectorytiles` generates a pyramid of PNG tiles from a 1D image (it rasterizes 1D data, like an altimeter trace for example)
  - `vectorfield` generates a PNG image wherein the color channels of each pixel describe the modulus and angle of the vectors (and a third optional parameter)

- Output settings: arguments passed to the formatting plugins. Applicable values depend on the plugin selected with the output format keyword.

The output settings listed below are the most used options:

- Generic settings

Option	Description
`with-shape`	either `yes` or `no` (defaults to `yes` if not specified or if another value is provided). `with-shape=no` means that Syntool should consider that the spatial coverage of the ingested data is equal or includes the entire viewport of the portal. In this case no spatial filtering will be applied on this granule and it will not be selectable by clicking on the map (to avoid selecting them when trying to navigate on the map).
`shape-tolerance`	the shape (contour) of the granule is stored in the database and is used to filter search results on a spatial criterium (only granules whose shape intersects the viewport of the portal will be returned). The web portal sends many search requests, so this filtering operation must be as fast as possible to maintain good performance. Syntool generates the shape from the GCPs of the granule and the result can contain hundreds of vertices. Computing intersections with such complex shapes hurts the performance of the database, so the shape-tolerance option can be used to simplify the shape by replacing as many vertices as possible by a single one while keeping the accuracy of the result within the tolerance range (expressed in meters).

- Raster settings (keyword: `raster`)

`resampling`	resampling option passed to GDAL when it warps the data (use either `near`, `bilinear` or `cubic`)
`warp-size`	Size (in pixels) of the output PNG file. Can be either a single integer value (square image), or the width and height separated by the x character (e.g. `warp-size=1440x720`)

- Raster tiles settings (keyword: `rastertiles`)

`resampling`	resampling option passed to GDAL when it warps the data (use either `near`, `bilinear` or `cubic`).
`min-zoom`	`syntool-ingestor` will only produce tiles starting from this zoom level (the top of the tiles pyramid is truncated).
`max-zoom`	`syntool-ingestor` will only produce tiles up to this zoom level (the base of the pyramid is truncated).
`tile-resampling`	resampling method passed to the tool which produces tiles. Valid values are `nearest` and `antialias`.

• Trajectory tiles settings (keyword: `trajectorytiles`)

`resampling`	resampling option passed to GDAL when it warps the data (use either `near`, `bilinear` or `cubic`).
`min-zoom`	`syntool-ingestor` will only produce tiles starting from this zoom level (the top of the tiles pyramid is truncated).
`max-zoom`	`syntool-ingestor` will only produce tiles up to this zoom level (the base of the pyramid is truncated).
`linewidth-meter`	when rasterizing the 1D data, `syntool-ingestor` will try to give it a width close to the value (in meters) passed to this option.
`min-linewidth-pixel`	when rasterizing the 1D data, `syntool-ingestor` will give it a width that is greater or equal to the value (in pixels) passed to this option.

• Vector field settings (keyword: `vectorfield`)

`resampling`	resampling option passed to GDAL when it warps the data (use either `near`, `bilinear` or `cubic`)
`resolution`	resolution (in "units of output projection" per pixel) of the output file. The value can be a single numerical value (same horizontal and vertical resolutions) or the resolutions along the horizontal axis and along the vertical axis separated by the x character (e.g. `resolution=12500x25000`)
`modulus-band`	Index of the GeoTIFF band (GDAL starts indexing at 1) where the modulus of the vectors are stored
`scale-min`	Minimal value for the modulus (values below this threshold are truncated)
`scale-max`	Maximal value for the modulus (values above this threshold are truncated)
`angle-band`	Index of the GeoTIFF band (GDAL starts indexing at 1) where the angle of the vectors are stored
`color-band`	in case there is another parameter attached to the vector, index of the GeoTIFF band (GDAL starts indexing at 1) where the parameter is stored
`color-min`	Minimal value for the additional parameter (values below this threshold are truncated)
`color-max`	Maximal value for the additional parameter (values above this threshold are truncated)

Warning

The max-zoom option with a relative value (prefixed by + or -) does not seem to work very well with stereographic projections. You should start by using an absolute value, check that the ingestor produces the expected output (i.e. the other options are all ok) and then switch to a relative max-zoom value.

Processing the GeoTIFF files

The syntax to invoke the `syntool-ingestor` command is as follows:

```
syntool-ingestor --config 3857.cfg \
                 --input-format IN_FMT \
                 --input-options IN_KEY_N=IN_VALUE_N ... \
                 --output-format OUT_FMT \
                 --output-options OUT_KEY_N=OUT_VALUE_N ... \
                 --output-dir INGESTED_DIR \
                 GEOTIFF_PATH
```

The `3857.cfg` file passed to the `--config` defines the projection to use for the output, the extent associated with this projection (in projected coordinates) and the viewport of the web portal (in projected coordinates).

For reference, here is the content of the `3857.cfg` file used for most Syntool portals:

```
[portal]
projection = 3857
viewport = -20037508.34 -20037508.34 20037508.34 20037508.34
extent = -20037508.34 -20037508.34 20037508.34 20037508.34
```

Here is an example showing how the `syntool-ingestor` can be invoked to process a GeoTIFF file for the Sentinel-1 SAR roughness product:

```
syntool-ingestor --config 3857.cfg \
                 --input-format geotiff \
                 --input-options projection=4326 \
                 --output-format rastertiles \
                 --output-options resampling=cubic \
                                  min-zoom=3 \
                                  max-zoom=+1 \
                                  shape-tolerance=1000 \
                 --output-dir /mnt/syntool/ingested \
                 "${input_geotiff_path}"
```

Once the processing is complete, you can check the output in the `/mnt/syntool/ingested` directory.

It must contain a directory whose name is built from the EPSG code of the output projection and the name of the product (as it was spelled in the `product_name` metadata of the GeoTIFF file). For example: `3857_GlobCurrent_L4_geostrophic_nrt`.

Note that some formatting plugins append a suffix to the product name. The `vectorfield` product adds `_vectorfield` after the product name, which results in the creation of an output directory named `3857_GlobCurrent_L4_geostrophic_nrt_vectorfield`.

Inside this directory you will find a subdirectory which has the same name as the granule (the `name` metadata of the GeoTIFF file). And inside this subdirectory, a file named metadata.json which contains the extracted metadata and either a single PNG file (for `raster` and `vectorfield` plugins) or a subdirectory named `tiles.zxy` (for `rastertiles` and `trajectorytiles` plugins).

```
ingested
|- 3857_GlobCurrent_L4_geostrophic_nrt
|   |- 20170414000000-GLOBCURRENT-L4-CURgeo_0m-ALT_OI_NRT-v03.0-fv01.0
|     |- metadata.json
|     |- tiles.zxy
|        |- ...
.        .
.        .
.        .
|- 3857_GlobCurrent_L4_geostrophic_nrt_vectorfield
   |- 20170414000000-GLOBCURRENT-L4-CURgeo_0m-ALT_OI_NRT-v03.0-fv01.0
      |- metadata.json
      |- vectorFieldLayer.png
```

Saving the configuration

Once you are satisfied with the result, copy the command-line options (`--input-format`, `--input-options`, `--output-format`, `--output-options`) you used in a text file (called `MyProduct.cfg` for example).

If you plan to ingest data automatically, it might be a good idea to put this file in a version control system so you can keep track of the changes made to the ingestion options.

This file can also be used to reduce the size of the syntool-ingestor command, as the `--[in|out]put-[format|options]` arguments can be replaced by `--options-file MyProduct.cfg`:

```
cat > 3857_SAR_roughness.cfg <<EOF
--input-format geotiff \
--input-options projection=4326 \
--output-format rastertiles \
--output-options resampling=cubic min-zoom=3 max-zoom=+1 shape-tolerance=1000
EOF

# Now the first command can be written like this:
syntool-ingestor --config 3857.cfg \
                 --options-file 3857_SAR_roughness.cfg \
                 --output-dir /mnt/syntool/ingested \
                 "${input_geotiff_path}"
```

Export to MySQL

Generating SQL statements

Syntool provides the `syntool-meta2sql` command to produce SQL statements from the information contained in the metadata.json files (produced by `syntool-ingestor` in the previous section).

The syntax of the command is as follows:

```
syntool-meta2sql OUTPUT.sql -- METADATA_PATH
```

Producing one SQL file for each metadata.json is not very efficient if you process data frequently.

Instead you can generate an SQL file which includes the content of several metadata.json **as long as they all belong to the same product** using the following syntax:

```
syntool-meta2sql --chunk_size=100 OUTPUT.sql -- METADATA1 METADATA2 ...
```

or if you already have a text file which contains a list of paths for metadata.json files:

```
cat LISTING.txt | syntool-meta2sql --chunk_size=100 OUTPUT.sql --
```

A more practical example using the `find` command:

```
find "${PRODUCT_DIR}" -mindepth 2 -maxdepth 2 -name "metadata.json" \
  | syntool-meta2sql --chunk_size=100 OUTPUT.sql --
```

Always use the `--chunk_size` option when generating a single SQL file from several metadata.json: when it has several input files, syntool-meta2sql generates bulk INSERT statements which can get so big that they saturate the statement buffer of the MySQL server. The `--chunk_size` limits the number of rows inserted by a single statement to avoid this issue (100 seems to be a good value in most cases).

Finally, it is possible to pass the output of syntool-meta2sql directly to MySQL by specifying `-` (standard output) as the output path:

```
find "${PRODUCT_DIR}" -mindepth 2 -maxdepth 2 -name "metadata.json" \
  | syntool-meta2sql --chunk_size=100 - -- \
  | mysql syntool
```

MySQL optimization

As no assumption can be made about the strategies enforced by the database administrator about security and optimization, the SQL statements generated by syntool-meta2sql only create tables and rows.

Creating a composite index based on the two fields that define the time range (`begin_datetime` and `end_datetime`) can improve performance by several orders of magnitude for tables that contain a lot of entries (often the case with in-situ or L1/L2 satellite products).

If this is in line with the database administrator strategy, you should always create this index after inserting the first granule of your new product in MySQL.

```
CREATE INDEX timerange ON `product_3857_MyProduct`(begin_datetime, end_datetime);
```

Web portal

Editing the configuration

As any error in the configuration of the portal will prevent all users from using it, it is recommended to make a copy of `/srv/http/syntool/index.html` before changing anything.

```
cd /srv/http/syntool/
cp index.html index_backup.html
```

Now go to the directory where you saved the Syntool portal generator (`~/syntool` if you followed the installation guide) and create a copy of the sample portal configuration:

```
cd ~/syntool
cp -r developer/portal developer/custom
```

In the `developer/custom` directory, you will find several HTML files which can be modified to change the titlebar of the portal or include new elements in the header and the footer of the web page.

This directory also contains a `config.json` file where most of the portal configuration resides. Open this file with your favorite text editor (`vim`) and look for the definition of an object called `module.exports`.

One of the properties of this object is named `products` and contains of list of JavaScript objects that describe the products available in the portal.

For example, a raster tiles product:

```
{
  label: 'GlobCurrent geostrophic NRT',
  id: '3857_GlobCurrent_L4_geostrophic_nrt',
  type: 'ZXY',
  mapMinZoom: 3,
  mapMaxZoom: 12,
  selected: true,
  mustBeCurrent: false,
  colorbar: 'images/colorbars/3857_GlobCurrent_L4_total_hs_colorbar.png',
  opacity: 0.8,
  stackLevel: 100,
  tileOrigin: tileOrigin,
  tileSize: tileSize,
},
```

or for a vectorfield product:

```
{
  label: 'Geostrophic surface current streamlines (Globcurrent)',
  id: '3857_GlobCurrent_L4_geostrophic_vectorfield',
  type: 'STREAMLINES',
  selected: true,
  mustBeCurrent: true,
  opacity: 0.60,
  stackLevel: 120,
  fields: {
    modulus: {
      channel: 0,
      min: 0,
      max: 0,
    },
    angle: {
      channel: 1,
      min: 0,
      max: 360,
    },
  },
  palette: {
    field: 'modulus',
    uniform: '#FFFFFF',
  },
},
```

Copy one of these product configurations (the one which matches best with your new product) at the end of the `products` list.

Change the `label` and replace the value of the `id` field with the actual identifier for your product (the name of the directory located directly under `/mnt/syntool/ingested`, where the ingestion results are stored).

Once your are done with the edition of this file, regenerate the portal with the following commands:

```
./developer/generate.sh \
  --portal ./developer/custom /srv/http/syntool/index.html "Custom portal"
```

You can check that everything is working fine by opening the following URL in your web browser: http://555.666.777.888/

The changes should be visible immediately, but some web browsers implement a very aggressive caching strategy so if the content of the portal does not mirror the changes you made in the configuration, try to clear the browser cache and reload the page.

Please refer to the portal configuration document for an in-depth description of the available settings and options.

Adding a colorbar

The colorbars displayed in Syntool portals are just PNG images, you can generate them manually as long as you respect the width constraint (the image **must** have a width of 320 pixels).

A command named `syntool-colorbar` can also be used to produce the colorbar that will be displayed in the Syntool portal. It is part of the `syntool_converter` Python package installed on the processing machine.

This command has been developed to handle specific cases, so its interface may not be intuitive: some options are mutually exclusive, others **must** be used together, but the help message does not provide this information...

Common arguments:

Argument	Description
-o	the path of the output directory (will be created if it does not exist yet)
-p	used to generate the name of the output image

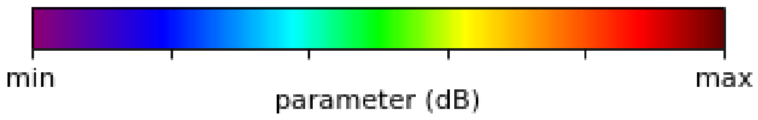
`-v`	the label displayed below the colormap
`-u`	the unit displayed between parenthesis after the label
`-m`	a keyword which identifies the colormap to include in the result (see --help to get the list of supported values)
`--min`	the lower bound for the numerical ticks
`--max`	the upper bound for the numerical ticks

Here are some examples showing how to generate a colorbar with `syntool-colorbar` for specific cases:

- When colors only have a meaning at the granule level (per-granule min/max instead of min/max shared by all the granules that belong to the product), use the `--relative` option:

```
syntool-colorbar -o /OUTPUT -p "per_granule" -v "parameter" -u "dB" \
                 -m rainbow --relative
```

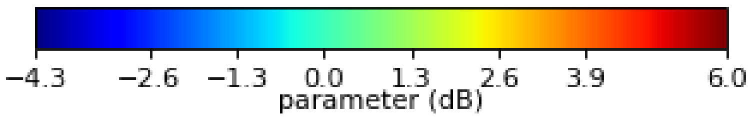
It generates the `/OUTPUT/per_granule_colorbar.png` file:



- When you need numerical ticks with a fixed step, but want to make sure that the min and max values are displayed, use `--bin_size`:

```
syntool-colorbar -o /OUTPUT -p "fixed_size_ticks" -v "parameter" -u "dB" \
                 -m matplotlib_jet --min -4.3 --max 6 --bin_size 1.3
```

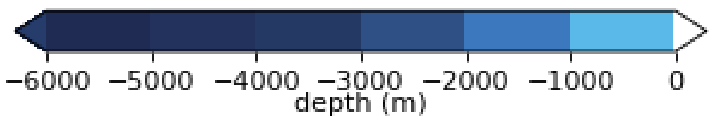
It generates the `/OUTPUT/fixed_size_ticks_colorbar.png` file:



- When you need a colorbar with discrete steps, use `--discrete`, `--nbins` and `--bounds`:

```
syntool-colorbar -o /OUTPUT -p discrete -v "depth" -u "m" \
                 -m ibcso --min -6000 --max 0 --nbins 7 --discrete \
                 --bounds -6000 -5000 -4000 -3000 -2000 -1000 0
```

It generates the `/OUTPUT/discrete_colorbar.png` file:



Once you have generated the PNG colorbar, copy it in the `/srv/http/syntool/images/colorbars/` directory and edit (or add) the `colorbar` property of the product in the portal configuration.

This property takes the URL of a PNG image, including relative URLs. As the root of the HTTP server installed on the web machine points to `/srv/http/syntool`, you can set the value of the `colorbar` to `/images/colorbars/discrete_colorbar.png` for example.

Adding a description

There are two ways to provide a textual description for products in Syntool:

- with the `description` property: the text associated with this property will be displayed in a tooltip when the user leaves the mouse cursor over the item of the products list (left side of the screen) which corresponds to the product.
- with the `infoURL` property: this property takes the URL of a file which contains the description of the product in Markdown format. The Markdown file will be loaded, parsed and formatted by the web application and displayed in a popup when the user clicks on the product's information button.

  i

  For example:

  ```
  The Multi-Sensor Precipitation Estimate (MPE) product consists of the
  near-real-time rain rates in mm/hr for each Meteosat image in original
  pixel resolution.

  The algorithm is based on the combination of polar orbiter microwave
  measurements and images in the Meteosat IR channel by a so-called blending
  technique.

  The MPE is most suitable for convective precipitation.

  Applications and Users: Operational weather forecasting in areas with poor
  or no radar coverage, especially in Africa and Asia.

   * [EUMETSAT website](https://www.eumetsat.int/website/home/index.html)
  ```

Side notes

Locations and trajectories

The `trajectorytiles` plugin of the ingestor generates tiles by rasterizing the 1D data. But PNG tiles is not the only way to represent this type of data in Syntool, and it is not always possible to generate a meaningful raster (self-intersecting trajectories for example).

Syntool provides a way to build interactive features on the map by reading the coordinates from a GeoJSON file.

Creating such features is very costly in terms of performance, so it is only used when the product resolution is low and/or when the number of granules to display simultaneously on the screen is limited.

Syntool is also able to render pictograms at specific locations, to display the position of moored buoys for example. For this kind of product, no GeoJSON is required, the location is read directly from the shape stored in the database.

The recommended way to convert/ingest these kinds of data is to write a script which reads the raw data file and produces the metadata.json (and geojson.json for trajectories) files directly.

metadata.json files contain a dictionary with (at least) the following entries:

Key	Value
`product`	Name of the product (equivalent to the `product_name` metadata in GeoTIFF files)
`syntool_id`	Identifier of the product (usually the product name prefixed with the EPSG code and an underscore)
`dataset`	Identifier for the granule (equivalent to the `name` metadata in GeoTIFF files)
`begin_datetime`	Start of the temporal coverage in YYYY-mm-dd HH:MM:SS format
`end_datetime`	End of the temporal coverage in YYYY-mm-dd HH:MM:SS format
`output_type`	Set to 'LOCATION' or 'TRAJECTORY'
`min_zoom_level`	Only used for raster products, set it to 0
`max_zoom_level`	Only used for raster products, set it to 0
`resolutions`	Syntool does not support multiple resolutions for 1D data representations, set it to an empty list: `[]`
`bbox_str`	Bounding box of the granule, defined as a polygon in WKT format using coordinates in the output projection
`shape_str`	A geometry in WKT format using coordinates in the projection. See below for more details

- Locations
  - `shape_str` **must** contain a point in WKT format: `POINT(x y)`, with `x` and `y` expressed in the output projection
  - `bbox_str` **must** be a valid polygon, not a point, so it is necessary to add an artificial margin.

    For example:

```
wkt_polygon = 'POLYGON(({} {},{} {},{} {},{} {},{} {}))'
bbox_str = wkt_polygon.format(x - 0.1, y - 0.1,
                              x + 0.1, y - 0.1,
                              x + 0.1, y + 0.1,
                              x - 0.1, y + 0.1,
                              x - 0.1, y - 0.1)
```

- The portal configuration for this type of products should look like this:

```
{
  label: 'Moorings',
  id: '3857_moorings',
  type: 'MOORED',
  description: '',
  selected: false,
  mustBeCurrent: false,
  opacity: 1,
  stackLevel: 120,
  marker: {
            default: {
                       externalGraphic: 'resources/mooring.png',
                       graphicWidth: 28,
                       graphicHeight: 28
                     }
          },
},
```

  `marker` contains the description of the pictogram displayed at the location of the granule (URL and dimensions of the image to display).

- Trajectories
  - `shape_str`: as explained before, Syntool sends a lot of queries to the database to list the granules whose shape intersects the current viewport of the portal. As the actual shape of the trajectories will already be stored in the GeoJSON file, it is recommended to define this property using a shape which is easier to handle for the database.

    For example, the enveloppe of the shape could be used, as long as it is defined as a WKT polygon using coordinates in output projection. Even better, you should use the bounding box: in this case `bbox_str` and `shape_str` will have the same value.

    This method reduces the accuracy of the intersection tests, but improves the performances of the webservice tremendously. So it is a trade-off, improving the user experience at the cost of the precision in the intersection tests.

  - Generating the GeoJSON file (which **must** be called `geojson.json`) can be quite difficult:
    - you have to split the geometries around the International Date Line (at least for Web Mercator projection)
    - if you have a list of point with measurements associated with each point, you might want to apply colors on your trajectory representation based on one of the measurements.

      Syntool can only define one color per segment, so you will have to decide how the points at each extremity of one segment will affect its color. One relatively simple way to handle this is to make the assumption that each point contributes equally: you can then split each segment into two and assign the color for each sub-segment based on the measurement of the closest point.

This is obviously not accurate as the assumption regarding the radius of validity for the measurement will be false in most case, but this is a necessary trade-off to avoid performance issue in the web browser.

You can take a look at the `geojson_trajectory.py` plugin from the `syntool_ingestor` Python package to get an example of implementation (it should be located in `syntool_ingestor/share/plugins/formatters/`)

- The portal configuration for this kind of product should look like this (for a GeoJSON file which contains measurements in a property named `salinity`):

```
{
  label: 'Surface drifters',
  id: '3857_surface_drifters',
  type: 'TRAJECTORIES',
  selected: false,
  colorbar: "images/colorbars/salinity_colorbar.png",
  mustBeCurrent: false,
  datasetWeight: 30,
  opacity: 1,
  stackLevel: 140,
  fields: {
    salinity: {
      channel: 'salinity',
      min: 32,
      max: 38,
    },
  },
  palette: {
    field: 'salinity',
    colormap: 'images/palettes/GMT_jet_saturated.cpt',
  },
},
```

Additional content (information, images, links)

External resources are displayed in the contextual menu which appears on the right side of the screen when users select a granule.

This contextual menu is built dynamically:

1. the application sends a request to the webservice to list the external resources associated with the selected granule
2. the webservice looks inside the directory which contains the granule data (i.e. the directory where the metadata.json file is stored)
3. the webservice will load and parse all the `*.ini` files contained in the `features` subdirectory (if it exists).
4. the webservice sends a response with the content of the files it managed to parse

In order to be accepted by the webservice, the `ini` files **must** at least:

- define a `global` section (which can remain empty)
- define at least one of the following sections: `metadata`, `images`, `links`

Each (key, value) couple under the `metadata` section will be displayed directly in the contextual menu

For the `images` section, each (key, value) couple will generate a link in the contextual menu. Clicking on the link will display the picture directly in the application, in a dialog box.

The key will be used as label for the link and the value will be split (using $ as the delimiter) to obtain:

- the description of the image (shown in a tooltip when the mouse cursor remains still over the link for a few seconds)
- the URL of the image
- the loading mode of the image: if it equals 1, the image is immediately loaded and displayed when the user selects the granule. Otherwise nothing happends until the user clicks on the link. Note that there can be only one image with autoload = 1.

For the `links section`, each (key, value) couple will generate a link in the contextual menu. Clicking on the link will open the URL of the remote resource in the current tab (use right click > "open in a new tab" if you don't want to leave Syntool).

The key will be used as label for the link and the value will be split (using $ as the delimiter) to obtain:

- the description of the remote resource (shown in a tooltip when the mouse cursor remains still over the link for a few seconds)
- the URL of the remote resource

An example:

```ini
[global]

[metadata]
key1 = value1
key2 = value2

[images]
labelImage = imageDescription$https://server.tld/imageURL.png$1

[links]
labelLink = linkDescription$https://server.tld/linkURL
```

The majority of the plots displayed in Syntool (profiles, timeseries) are pre-generated images associated with a granule using this mechanism.

Questions, feature requests and bugs

Please register and post your questions, feature requests and bug reports on the Syntool forum: https://forum.oceandatalab.com/forum-5.html