Syntool Portal Config

Table of Contents

        - LINE
        - ELLIPSE
        - MUSHROOM
        - WKT
    - ShapeArrow
    - ShapeStyleMap
    - ShapeStyle
    - Layer
    - TimeRange
    - Timespan
    - MarkerSymbols
    - MarkerSymbol
    - Product
    - ProductType
        - LAYER
        - TIMELAYER
        - NCWMS
        - IMAGE
        - ZXY
        - ARROWS
        - BARBS
        - STREAMLINES
        - TRAJECTORIES
        - GEOJSON
        - MOORED

        - USER_SHAPES
        - SERVER_SHAPES
    - ProductOutlines
    - Group
    - Color
    - Colormap
    - ColormapTick
    - DataFields
    - DataField
    - RenderFields
    - Marker
    - MarkerFn
    - MarkerStyle
    - MarkerStateStyle
    - ImageMarkerStyle
    - PolygonMarkerStyle
    - GranuleInfo
    - GranuleInfoGroup
    - GranuleInfoResource
    - GranuleInfoMetadata
    - ProductGranuleInfoConfig
    - GranuleInfoConfigFn
    - GranuleInfoConfig
    - GranuleInfoGroupsConfig
    - GranuleInfoUnknownGroupConfig

The following document is split into two sections. The first one is dedicated to the different functions available on Syntool web portal and the second one describes the configuration file.

Introduction

Timeline

The timeline consists of two concepts:

- the date and time (aka datetime),
- the date range of a certain width centered around the datetime (aka timespan).

The currently selected datetime is referred to as the "current datetime". And the currently selected timespan is referred to as the "current timespan".

The user can change both the current datetime and the current timespan.

An animation is just an automatic increment/decrement of the current datetime by a fixed step (controlled by the animation speed) every fixed delay.

Dataset (aka Granule)

A temporally and spatially referenced piece of data.

It can be a trajectory (e.g. drifters), an image (e.g. SAR), or a vector matrix (barbs, arrows, or streamlines) (e.g. surface current).

It must have:

- spatial attributes: a bounding box (or point) and an optional outlining polygon (aka outline).
- temporal attributes: a start date and an end date (a date range).

The dataset will be added to the map either:

- when its date range contains the current datetime, or
- when its product's `mustBeCurrent` is `false` and its date range intersects the current timespan.

Product (aka Collection)

A collection of datasets that share the same configuration.

Syntool Web

In this section, the different components and functionalities of Syntool web portals are described.

Products selection panel (aka Catalog)

All available products are listed in the left panel.
The products may be grouped based on tags.
The user can filter the list with a search query to be matched against the product's label and tags.
This panel allows the user to add or remove them from the Display data dialog.

Display data dialog

This dialog is accessible by clicking on the "Display data" button in the top bar.

This dialog allows the user to:

- to toggle a product on/off
- see the colormaps of each product (when applicable)
- reorder the products (a low product on the list is displayed below a higher one)
- change the transparency of a product
- filter the datasets of a product using a pattern (matched against the dataset's ID)

Note: When a product is toggled off, it is ignored entirely.

Timeline

The timeline is located in the bottom of the page.
It's initially collapsed (the detailed view is hidden).

The timeline is composed of three parts:

1. the Timeline bar

2. the coverage timeline
3. the detailed timeline

Timeline bar

Item	Description
Animation controls	Allow to start and stop the animation, and control its speed.
Timespan selection	Allow to change the current timespan by picking one of the pre-defined ones.
Find nearest button	Allow to find and go to the nearest datetime that contains at least one dataset.
Collocation toggle	Toggles the collocation feature which changes the behavior of the coverage timeline. Will be red with active. For more info see the coverage timeline.
Number of datasets in the map	Is the number of datasets in the map, visible or not (see pagination).
Pagination controls	Allow to change the current page, and shows the number of datasets currently visible. For more info see pagination.
Dataset selection info	Shows the number of selected datasets, or when only one is selected its ID and the ID of its product. (Only visible when one or more datasets are selected)
The current date and time	Shows the current current datetime. (Only visible when no dataset is selected)
Coordinates	Shows the coordinates of the mouse pointer (or the map center) in degrees longitude and latitude.

Coverage timeline

Composed of three rows:

1. Contains all the possible years.
2. Contains all the months of the selected year.
3. Contains all the days of the selected month of the selected year.

Each cell (year/month/day) of the calendar can be:

- Grey: none of the selected products have any datasets in that year/month/day.
- White: at least one of the selected products have at least one dataset in that year/month/day.
- Yellow: that year/month contains at datasets from all of the selected products. But not on the same day.
- Red: at least one day of that year/month contains datasets from all of the selected products.

Note: The yellow and red states are only used if the Collocation is active (the collocation toggle in the timeline bar is red).

Detailed timeline

Hidden by default, can be shown by clicking on the arrow pointing upward just above the timeline and centered horizontally.

Its full width represents the current timespan.
The vertical line in the center represents the current datetime.

The dots and horizontal lines represent datasets.
A horizontal line represent the datetime range of a dataset.
A horizontal line become a dot when it's too small.
A horizontal line is considered too small when its length on screen is less then half the width of a dot (by default when less then 5 pixels).

The dots and horizontal lines can be:

- Grey: if their datasets are not in the map.
- Blue: if their datasets are in the map.
- Yellow: if their datasets are selected.

The dots and horizontal lines are clickable (if their datasets are in the map).
When clicked its respective dataset is selected in the map.

They also have a tooltip which contains the ID of their respective dataset.

Pagination

Syntool is rendered unresponsive or unusable when the number of datasets in the map gets to big.
To solve this the datasets are split into pages, and shown one page of datasets at a time.

Smart pagination

The `maxDatasetsWeight` is the rendering bugdet.

The `datasetWeight` of a product is the average cost of rendering one of its datasets.

The `minPageSize` of a product is the minimum number of its datasets to be rendered (even if we go over-budget).

The pagination system tries to maximize the number of rendered datasets. While distributing as uniformly as possible the bugdet on all selected products and respecting the `minPageSize` of all products.

The `priority` of a product comes in play in the case that the budget can't be distributed uniformly (the number of rendered datasets must be an integer). In which case the products with the highest priorities (biggest `priority`) will have their datasets rendered.

Examples

1. ▶ Normal

2. ▶ Priority
3. ▶ Weight
4. ▶ `minPageSize`
5. ▶ `minPageSize` with going over-budget

Messages

A message is like a notification, it conveys some information that the user should see.

Hotspots

A hotspot is a bookmark of the current configuration (datetime, timespan, map viewport and the selected products with their respective transparency levels and order), with an associated name or title and an optional description link (a link to an external web page that describes it, like an article).

Hotspots can be created, viewed, shared, and deleted from the "Hotspots" popup (accessible by clicking on the "Hotspots" button in the top bar).

Shapes (aka User shapes)

Shapes are special datasets. They may have an author and a label (which can hold any text value). They are filtered based on their label not their ID.

Shapes can be created by the user. The user created ones are always in the product of type ProductType.USER_SHAPES. A shape is a geometry (e.g. point, line, polygon, ...) or a text label positioned geographically. A shape can be imported and exported. The exported shapes are in the same format accepted in the config (see Shape).

Color Palettes

A color palette is a sequence of colors in a specific order over a range of values. With three special colors:

- The underflow color used to represent values smaller than the minimum.
- The overflow color used to represent values bigger than the maximum.
- The null color used to represent `NaN` values.

Supported formats:

- RGB
  - A list of new line separated triplets of red, green and blue.
  - The values of each triplet are separated by white space.
  - The values of each triplet are whole numbers between `0` and `255`.
  - The resulting color palette is made up of segments equal in length and who's color corresponds to its respective triplet.
  - The underflow color is the first color.
  - The overflow color is the last color.
  - The null color is transparent.
- PNG
  - A PNG image of which only the first row of pixels is used.
  - The resulting color palette is made up of segments equal in length and who's color corresponds to its respective pixel.
  - The underflow color is the first color.
  - The overflow color is the last color.
  - The null color is transparent.
- CPT (Color Palette Table)
  - CPT Reference
  - Only RGB color model is supported.
  - Only Regular CPTs are supported.
  - If no underflow color (**B**) is specified, the first color will be used.

- If no overflow color (**F**) is specified, the last color will be used.
- If no null color (**N**) is specified, transparent will be used.

Configuration file

A portal must be configured using is a JavaScript file having the form of a commonjs module that exports an object with the fields described in the configuration fields below.

The file will have the form:

```
// any code here...
module.exports = {
  // fields here...
};
```

The different fields of the configuration file are going to be explained in the following section. The example given below may be a good starting point for the creation of a new configuration file.

Example

▶ Show example

Asynchronous configuration file

A portal configuration can also be loaded asynchronously by exporting a Promise.

Example

`config.js`

```
module.exports = ODL.importScript('url/of/config.js', '__config');
```

`url/of/config.js`

```
window.__config = (function() {
  'use strict';

  // any code here...
  return {
    // fields here...
  };
})();
```

Configuration Fields

Field	Type	Default	Description
`version`	`string`	**Required**	**MUST** be `'1.3.x'` where `x` can be any positive integer.
Services			
~~`serviceHost`~~	`URL`	*Deprecated*	*Deprecated: Use `servers` instead.* The URL to the server running a Syntool services server (or a compatible one).

Field	Type	Default	Description
~~`dataServers`~~	`URL[]`	*Deprecated*	*Deprecated: Use `servers` instead.* A list of URLs to use for requesting data (tiles, geoJSON, ...). All URLs **MUST** be equivalent, a list is given in order to circumvent the browser's limitation on the number of simultaneous requests to one domain.
`servers`	`Server[]`	**Required***	A list of servers to be used to find and load the datasets. * If `serviceHost` and `dataServers` are defined, this field can be omited.
`shortenerForURL`	`URLShortenerCreator`	**Optional**	If provided, will be called to know how to shorten a given URL. Used, for example, when user clicks on the "Share" button.
Map			
`projection`	`string`	**Required**	A string identifying the Well Known Identifier for the projection. Specifies the projection of the map.
`maxExtent`	`number[4]`	**Required**	The maximum extent of the map defined as `[left, bottom, right, top]` all in the projection `projection`.
`restrictedExtent`	`number[4]`	`maxExtent`	The maximum navigable extent of the map defined as `[left, bottom, right, top]` all in the projection `projection`.
`zoomOffset`	`integer >= 0`	`0`	Some map providers (like Bing maps) skip some zoom levels (e.g. Bing maps considers the zoom level `0` as level `1`), if it's the case you can use this field to set the offset (e.g. set it to `1`).

Field	Type	Default	Description
`defaultZoom`	`integer >= 0`	**Optional**	If provided, will be used to set the zoom level of the map by default (if the `extent` or `zoom` are not set in the URL of the page).
`baseLayers`	`Layer[]`	**Required**	A list of layers. It **MUST** contain at least one layer. One of these layers (by default the first one) will be used as the background layer of the map. These layers will be displayed in a list which and be toggled using the globe button in the top bar of Syntool allowing the user to switch between them.
`additionalLayers`	`Layer[]`	`[]`	A list of layers that are always visible and placed above or below all products. For more info see Additional Layers.
Timeline			
`currentDate`	`Date`	**Optional**	The default current datetime of the portal. If not given it will be set to the client's local date and time and then a find nearest will be executed. (example for `01/06/2006 12:00` use `new Date(Date.UTC(2006, 5, 1, 12))`, for more info see MDN's Date.UTC)
`animationSpeed`	`number > 0`	20 hours / s	The base speed of the animations in *miliseconds of virtual time* per *second of real time*.
`timeRanges`	`TimeRange[]`	*see below*	A list of predefined timespans that the users can select.

Field	Type	Default	Description
`yearsRange`	`integer[2] > 1969`	*see below*	Timeline years range from January 1st to January 1st (which means the upper bound will not appear in the timeline)
Misc			
`storageId`	`string`	**Required**	Portals that have the same `storageId` and the same domain will share the same local storage (local hotspots and user shapes).
`changelogURL`	`URL`	**Optional**	The URL of a GitHub Flavored Markdown file containing the change log.
`maxDatasetsWeight`	`number >= 0`	`100`	The maximum cumulated weight of datasets rendered at the same time. For more info see Pagination.
`markerSymbols`	`MarkerSymbols`	`{}`	A hash map of polygons to be referenced in the product's `graphicName`.
~~`showGroups`~~	`boolean`	*Deprecated*	*Deprecated: Use `groupProductsBy` instead.* If set to `true`, the products will be grouped in the products panel, if possible.
`groupProductsBy`	`false` or `string`	`false`	If set to a `string`, will be used to choose the tags to group the products by. Otherwise the products will not be grouped.
`defaultGroupLabel`	`string`	**Optional**	If provided, will be used as label to the default group.

Field	Type	Default	Description
`collapsibleGroups`	`boolean`	`false`	If set to `true`, clicking on a group label will show/hide its contents in the products panel. Only applicable when grouping products. (See `showGroups` and `groupProductsBy`)
`sortGroups`	`function`	**Optional**	If provided, gets called before displaying the groups in the products list in order to sort them. For more info see `sortGroups` below.
Predefined Data			
`messages`	`Message[]`	`[]`	A list of predefined ordered messages. For more info see Messages.
`hotspots`	`Hotspot[]`	`[]`	A list of predefined ordered hotspots. For more info see Hotspots.
`drawableShapes`	`DrawableShape[]`	`[]`	A list of predefined drawable shapes that the users can use to draw user shapes. (Some shapes are defined in Syntool and can't be removed like: polygon, polyline, and point).
`products`	`Product[]`	`[]`	An ordered list of available products.
~~`groups`~~	`Group[]`	*Deprecated*	*Deprecated: Use product's `tags` and `groupProductsBy` instead.* An ordered list of groups.

Notes

- All server IDs **MUST** be unique (no two servers can have the same `id`).

- All product IDs **MUST** be unique (no two products can have the same `id`).
- The `products` list **MUST** contain exactly one product with the type USER_SHAPES.
- The product with the type USER_SHAPES will be used for the user's local shapes.
- All message IDs **MUST** be unique (no two messages can have the same `id`).
- If `servers` is used, all products **MUST** be handled by at least one server (except the ones of type USER_SHAPES, SERVER_SHAPES or LAYER which are handled internally only).
- If `groups` are defined, All products **MUST** be in at least one group.

`timeRanges` default value

```
[
  ['6-Hour', '6h'],
  ['Daily', '1d', true],
  ['3-Day', '3d'],
  ['Weekly', '1w'],
  ['Bi-weekly', '2w'],
]
```

`yearsRange` default value

The default value for the `yearsRange` property is dynamically generated so that the timeline displays the last 15 years (+ current year).

`sortGroups`

Type: `(key: string) -> (group1: string|null, group2: string|null) -> number`

This function receives the tag key used to group the products and must return a function that compares the passed tag values or `null` for the default group.

The comparator function should return:

- a **negative** number to indicate the first argument (ie. `group1`) should come **before** the second (ie. `group2`)
- a **positive** number to indicate the first argument (ie. `group1`) should come **after** the second (ie. `group2`)
- **zero** (`0`) to indicate the first argument (ie. `group1`) and the second (ie. `group2`) are considered **equal** and thus their order remains unchanged.

Additional Layers

Additional layers are normal OpenLayers layers with an optional additional custom option `syntoolIsOverlay`. `syntoolIsOverlay` defaults to `false` and when set to `true` the layer is displayed above all products. Otherwise it's displayed under all products.

The order of the layer specifies the order in which they get rendered:

- The first element of the list is closest to the base layer and displayed below all other layers of the same group (with the same value `syntoolIsOverlay`).
- The last element of the list is farthest from the base layer and displayed above all other layers of the same group (with the same value `syntoolIsOverlay`).

Types

Server

Type: `object`

Field	Type	Default	Description
`id`	`string`	**Required**	The identifier of the server. Can be any string not starting with "_" (underscore).

Field	Type	Default	Description
`Class`	`ServerClass`	`SyntoolAPIServer`	The API server class.
...	...	...	*The configuration fields of the `Class`.*

ServerClass

Type: Any subclass of APIServer (or of it's subclasses, etc...)

LocalAPIServer and RemoteAPIServer are the only subclasses defined in Syntool.
SyntoolAPIServer is the only subclass of RemoteAPIServer defined in Syntool.

APIServer

Type: Abstract class

This is the parent of all the APIServer classes usable in Syntool.

TODO

LocalAPIServer

Type: Abstract subclass of `APIServer`

TODO

Example

The actual implaimentation of `LayerAPIServer` which handles all LAYER products.

```
import {LocalAPIServer} from '/js/syntool.js';

export class LayerAPIServer extends LocalAPIServer {
  /*Public*/
  handlesAllProductsOfType(type) {
    return type === 'LAYER';
  }

  /*Protected*/
  _getAllInfosForProductWithId(productId) {
    var product = this.api.syntool.productsStore.getProduct(productId);

    // TODO: cache these info objects
    return [
      {
        productId: productId,
        datasetId: productId + '-layer_dataset',
        name     : product.layer.title,
        start    : product.validFrom,
        end      : product.validTill,
      },
    ];
  }
}
```

RemoteAPIServer

Type: Abstract subclass of `APIServer`

TODO

Configuration fields

Field	Type	Default	Description
`serviceURL`	`URL`	**Required**	The URL of the Syntool services server (or a compatible one).
`dataURLs`	`URL[]`	**Optional**	A list of URLs to use for requesting data (tiles, geoJSON, ...). All URLs **MUST** be equivalent, a list is given in order to circumvent the browser's limitation on the number of simultaneous requests to one domain.
`productIdMap`	`ProductIdMap`	**Required**	See ProductIdMap for more info.

Note

`dataURLs` is **Required** if at least one of the products handled by the server will fetch data (tiles, geoJSON, ...) from it.

SyntoolAPIServer

Type: Subclass of `RemoteAPIServer`

Configuration fields

Same as RemoteAPIServer plus:

Field	Type	Default	Description
`datasetsResponseLimit`	`integer > 0`	`5000`	The maximum number of datasets returned from the `/datasets` endpoint. Configurable on the server with `results_limit`.

A note on relative GranuleInfoResource `url`s

All relative GranuleInfoResource `url`s will be resolved using their dataset's `uri` field, which is `${serviceURL}/data/ingested/${product.id}/${dataset.name}/`, except:

- for `IMAGE` datasets: `${serviceURL}/data/ingested/${product.id}/${dataset.name}/imageLayer.png`
- for `ZXY` datasets: `${serviceURL}/data/ingested/${product.id}/${dataset.name}/tiles.zxy/`
- for `ARROWS`, `BARBS` and `STREAMLINES` datasets: `${serviceURL}/data/ingested/${product.id}/${dataset.name}/vectorFieldLayer.png`

Example

```
{
  id: 'example',
  serviceURL: 'https://syntoolws.example.com',
  dataURLs: [
    'https://syntooldata1.example.com',
    'https://syntooldata2.example.com',
    'https://syntooldata3.example.com',
    'https://syntooldata4.example.com',
    'https://syntooldata5.example.com',
    'https://syntooldata6.example.com',
    'https://syntooldata7.example.com',
    'https://syntooldata8.example.com',
    'https://syntooldata9.example.com',
    'https://syntooldata10.example.com',
  ],
  productIdMap: {
    '3857_ARGO_Deep_NATL1000': '3857_ARGO_Deep_NATL1000',
  },
}
```

ProductIdMap

Type: `{[id: string]: string|string[]}`

The key (`id`) is the product's ID used in this config referencing `id` of Product.
The value is the product's ID known by the server, or a list product IDs known by the server to be merged together client-side and presented to the user as one product.

Notes

- The key and value can be the same.
- All product IDs MUST be IDs of existing products in the config.

Example

```
{
  'ARGO_Deep_NATL1000': '3857_ARGO_Deep_NATL1000',
  '3857_SAR_roughness': [
    '3857_Sentinel-1A_SAR_roughness',
    '3857_Sentinel-1B_SAR_roughness',
  ],
}
```

URLShortenerCreator

Type: `(url: URL) -> URLShortener`

This function will be called in order to shorten a URL, and must return a URLShortener.

Parameter	Type	Description
`url`	`URL`	The URL that needs shortening.

Note: This function might be called multiple times, or just once and its return value cached.

URLShortener

Type: `object`

Field	Type	Default	Description
`url`	`URL`	**Required**	The URL to send the HTTP request to.
`method`	`string`	`'GET'`	The request method (example `'GET'`, `'POST'`).
`data`	`object`	**Optional**	The data to be sent with the request. If provided, data will be encoded and added to the URL for `'HEAD'` and `'GET'` requests, or sent as the request's form data body for all other request methods.
`responseType`	`ResponseType`	`'text'`	How the response should be interpreted. For more info see MDN's XMLHttpRequest.responseType.
`callback`	`ShortenerCallback`	**Optional**	If provided, will be called with the received response as its only argument and should return the shortened URL or an `Error` object. Otherwise the response will be treated as the shortened URL.
`errorback`	`ShortenerErrorback`	**Optional**	If provided, will be called with the error that occurred as its only argument and should return the shortened URL or an `Error` object. Otherwise the error will be handled as is.

Example

```
function createBitlyShortener(url) {
  // For more info see http://dev.bitly.com/authentication.html#apikey
  var login = '<your bitly login>';
  var apiKey = '<your bitly api-key>';
```

```
  return {
    method: 'GET',
    url: (
      window.location.protocol === 'http:'
      ? 'http://api.bit.ly/v3/shorten'
      : 'https://api-ssl.bit.ly/v3/shorten'
    ),
    data: {longUrl: url, apiKey: apiKey, login: login},
    responseType: 'json',
    callback: function(response) {
      if (response.status_code === 200) {
        return response.data.url;
      } else {
        return new Error(response.status_txt);
      }
    },
  };
}
```

ResponseType

Type: `string`
Possible values:

- `'arraybuffer'`
- `'blob'`
- `'json'`
- `'text'`

For more info see MDN's XMLHttpRequest.responseType.

ShortenerCallback

Type: `(response) -> URL | Error`

ShortenerErrorback

Type: `(error: Error) -> URL | Error`

Message

Type: `object`

Field	Type	Default	Description
`id`	`string`	**Required**	The ID of the message used to mainly to remember if it was shown before.
`type`	`MessageType`	**Required**	The type of the message, used to style it. For more info see MessageType.
`text`	`string`	**Required**	The body of the message.
`showOnce`	`boolean`	`false`	When set to `true`, once the user closes the message it will not be shown again.

Examples

```
{
  id: 'odyssea reprocessing from 01/07/2016 to 03/02/2017',
  type: 'warning',
  text: 'All Odyssea products are currently being reprocessed from July 2016 until the 3<sup>rd</sup> February
  showOnce: true,
}
```

```
{
  id: 'test_info',
  type: 'info',
  text: 'This is an info message.',
}
```

```
{
  id: 'test_warning',
  type: 'warning',
  text: 'This is a warning message.',
}
```

```
{
  id: 'test_error',
  type: 'error',
  text: 'This is an error message.',
}
```

```
{
  id: 'test_long',
  type: 'info',
  text: 'This is a looooooooooooooooooong looooooooooooooooong looooooooooong long long looooooooooong long long
}
```

```
{
  id: 'test_multi_lines',
  type: 'info',
  text: 'This is a message<br>on multiple<br>lines.',
}
```

```
{
  id: 'test_multi_spaces',
  type: 'info',
  text: 'This is a message       with    lots of     spaces.',
}
```

MessageType

Type: `string`
Possible values:

- `'info'`
- `'warning'`
- `'error'`

Hotspot

Type: `string` or `object`

`string` (hotspot permalink)

Can be found by `Right-Click`ing a hotspot item (from the "Hotspots" panel) and choosing "Copy link address" (or equivalent).

Notes

- Not to be confused with a share permalink (or a live URL).
- Not to be confused with a short link.

Example

```
'?
name=Chlorophyll%20A%20and%20Globcurrent%20%40%20Agulhas%20region&date=1412426592000&timespan=1d%3B1w&products=
```

`object`

Field	Type	Default	Description
`name`	`string`	**Required**	The name shown in the "Hotspots" panel
`description`	`URL`	**Optional**	A URL of a description page
`date`	`Date` or `integer >= 0`	**Optional**	If provided, the date of the hotspot or the date in milliseconds since 1 January 1970 UTC
`extent`	`number[4]`	**Optional**	The extent of the viewport defined as `[left, bottom, right, top]` all in the projection `projection`.
`timespan`	`Timespan`	**Optional**	The timespan used
`zoom`	`integer >= 0`	**Optional**	The zoom level
`products`	`string[]`	**Optional**	A list of product ids
`opacity`	`number[]`	**Optional**	A list of product opacities
`stackLevel`	`number[]`	**Optional**	A list of product stack levels
`pattern`	`string[]`	**Optional**	A list of product filtering patterns

Example

```
{
  name: 'Chlorophyll A and Globcurrent @ Agulhas region',
  date: 1412426592000, // 2014-10-04T12:43:12.000Z
  extent: [768344.98869256, -5048359.949897, 4280779.311964, -3243223.0901656],
  timespan: '1d',
  zoom: 3,
  products: [
    '900913_Chlorophyll_a_concentration_VIIRS',
    '900913_GlobCurrent_L4_geostrophic_streamline',
  ],
  opacity: [100, 50],
  stackLevel: [50, 90],
}
```

DrawableShape

Type: `object`

Field	Type	Default	Description
`title`	`string`	**Required**	The tooltip of the toolbar button that allows the user to draw that shape.
`cssClassNames`	`string`	**Optional**	The CSS class names added to the toolbar button that allows the user to draw that shape. Useful for customizing the icon.
`wkt`	`string`	**Required**	The geometry of the drawable shape formatted as WKT in the projection `projection`.

For each `DrawableShape` a button is added in the user shapes toolbar. For more info see Shapes.

Example

```
{
  title: 'Right Arrow',
  cssClassNames: '-RightArrow',
  wkt: 'MULTILINESTRING((0 0, 100 0),(90 10, 100 0, 90 -10, 90 10))',
}
```

Shape

Type: `object`

Field	Type	Default	Description
General			
`id`	`string`	**Optional**	The ID of the shape used to generate the corresponding dataset's ID. If not given, the index of the shape in its product's `shapes` array will be used.
`type`	`ShapeType`	**Required**	The type of the shape. For more info see ShapeType.
`text`	`string`	**Optional**	The label displayed with the shape. Also used for filtering.
`start`	`Date` or `number`	**Optional**	If provided, the shape will be hidden before that date. A number is considered as milliseconds since 1 January 1970 00:00:00 UTC.
`end`	`Date` or `number`	**Optional**	If provided, the shape will be hidden starting from that date. A number is considered as milliseconds since 1 January 1970 00:00:00 UTC.
`color`	`Color`	**Optional**	If provided, will be used to color the shape.
Arrow Shape			*Applies to ARROW shapes only*

Field	Type	Default	Description
`points`	`number[2][2]`	**Required**	A couple of coordinates pairs defined as `[lon, lat]` both in EPSG:4326. The first pair marks the start (the tail) of the arrow. The second marks the end (the head) of the arrow.
`arrowSize`	`number > 0`	`15000`	The size in **meters** of the arrow head.
Text Shape			*Applies to TEXT shapes only*
`location`	`number[2]`	**Required**	The coordinates of the shape defined as `[lon, lat]` both in EPSG:4326.
Line Shape			*Applies to LINE shapes only*
`points`	`number[2][]`	**Required**	A list of **at least two** coordinate pairs defined as `[lon, lat]` both in EPSG:4326.
`arrow`	`ShapeArrow`	**Optional**	The description of an optional arrow head to rendered on the line. For more info see ShapeArrow.
Ellipse Shape			*Applies to ELLIPSE shapes only*
`center`	`number[2]`	**Required**	The coordinates of the center of the ellipse defined as `[lon, lat]` both in EPSG:4326.
`rx`	`number > 0`	**Required**	The radius of the ellipse on the x-axis in **meters**.
`ry`	`number > 0`	`rx`	The radius of the ellipse on the y-axis in **meters**.
`angle`	`number ∈ [-180, 180]`	`0`	The amount of rotation applied to the ellipse in degrees (counter-clockwise).

Field	Type	Default	Description
`arrow`	`ShapeArrow`	**Optional**	The description of an optional arrow head to rendered on the line. For more info see ShapeArrow.
Mushroom Shape			*Applies to MUSHROOM shapes only*
`baseWidth`	`number >= 0`	**Required**	The distance between the two vertical edges of the base of the mushroom in **meters**.
`baseHeight`	`number >= 0`	**Required**	The length of base of the mushroom in **meters**.
`baseAngle`	`number ∈ [-45, 45]`	`0`	The amount of rotation applied to the base of the mushroom in degrees (counter-clockwise).
`leftLobeCenter`	`number[2]`	**Required**	The coordinates of the center of the left lobe defined as `[lon, lat]` both in EPSG:4326.
`leftLobeRx`	`number >= 0`	**Required**	The radius of the left lobe ellipse on the x-axis in **meters**.
`leftLobeRy`	`number >= 0`	**Required**	The radius of the left lobe ellipse on the y-axis in **meters**.
`rightLobeCenter`	`number[2]`	**Required**	The coordinates of the center of the right lobe defined as `[lon, lat]` both in EPSG:4326.
`rightLobeRx`	`number >= 0`	**Required**	The radius of the right lobe ellipse on the x-axis in **meters**.
`rightLobeRy`	`number >= 0`	**Required**	The radius of the right lobe ellipse on the y-axis in **meters**.

Field	Type	Default	Description
Generic Shape			*Applies to WKT shapes only*
`author`	`string`	**Optional**	The author associated to the shape.
`wkt`	`string`	**Required**	The geometry of the shape formatted as WKT in EPSG:4326.

Notes

- All TEXT shapes require the `text` field to contain a none-empty string.
- All ARROW shapes require the `points` field to contain exactly 2 points.

For more info see Shapes.

See ShapeType for examples.

ShapeType

Type: `string`
Possible values:

- `'TEXT'`
- `'ARROW'`
- `'LINE'`
- `'ELLIPSE'`
- `'MUSHROOM'`
- `'WKT'`

TEXT

Example

```
{
  type: 'TEXT',
  text: 'Text goes here',
  start: 1459814400000,
  end: 1459900800000,
  location: [133.23063184654524, -21.083556353366014]
}
```

ARROW

Example

```
{
  type: 'ARROW',
  text: '',
  start: 1459814400000,
  end: 1459900800000,
  points: [[28.762538978264597,-33.09396148741701],[27.872646400139594,-33.745014639370424]]
}
```

LINE

Example

```
{
  type: 'LINE',
  text: '',
  start: 1459814400000,
  end: 1459900800000,
  points: [[132.48356160920454,-26.78394450491424],[136.1749678592054,-26.469667679940457]],
```

```
  arrow: {
    directionFlipped: false,
    sideFlipped: false,
    ratio: 0.5,
    density: 0.5,
    offset: 0.5
  }
}
```

ELLIPSE

Example

```
{
  type: 'ELLIPSE',
  text: 'C',
  start: 1459814400000,
  end: 1459900800000,
  center: [134.02164747155183,-23.945171626522214],
  rx: 244598.49047851562,
  ry: 244598.49047851562,
  angle: 0,
  arrow: {
    directionFlipped: false,
    sideFlipped: false,
    ratio: 0.5,
    density: 0.2,
    offset: 0.125
  }
}
```

MUSHROOM

Example

```
{
  type: 'MUSHROOM',
  text: '',
  start: 1486944000000,
  end: 1487203200000,
  baseWidth: 955.4628534317018,
  baseHeight: 1997.5447529548906,
  baseAngle: 0.6778513460412228,
  leftLobeCenter: [-4.814060593358906, 48.0702570920605],
  leftLobeRx: 761.6419180129138,
  leftLobeRy: 645.4213335569948,
  rightLobeCenter: [-4.8014029583976665, 48.07037335406251],
  rightLobeRx: 739.6453667184011,
  rightLobeRy: 664.7914310097694
}
```

WKT

Example

```
{
  type: 'WKT',
  text: 'text',
  start: 1149163200000,
  end: 1149163200000,
  author: 'anonymous',
  wkt: 'POLYGON((77.51953122494571 59.26588064106204,40.95703122494571 44.339565266015946,64.68749997494598 21
}
```

ShapeArrow

Type: `object`

Field	Type	Default	Description
`directionFlipped`	`boolean`	`false`	If `true` the arrow will point from the end of the line segment to its start, instead of start to end. In the case of an ellipse `false` means counter-clockwise and `true` means clockwise.
`sideFlipped`	`boolean`	`false`	If `true` the colors of the arrow head are swapped.
`ratio`	`number ∈ [0, 1]`	`0.5`	The ratio of the proportion of the left triangle and the right.
`density`	`number ∈ [0, 1]`	*see below*	The density of arrow heads on the shape. `0` means one arrow head, and `1` means arrow head on each line segment.
`offset`	`number ∈ [0, 1]`	*see below*	The offset of the first arrow head. `0` means on the first line segment, and `1` means on the last.
`size`	`number > 0`	`15000`	The size in **meters** of the arrow head.

Default values by type

Type	`density`	`offset`
`LINE`	`0.5`	`0.5`

Type	density	offset
ELLIPSE	0.2	0.125

Example

```
{
  directionFlipped: false,
  sideFlipped: false,
  ratio: 0.5,
  density: 0.2,
  offset: 0.125
}
```

ShapeStyleMap

Type: `object`

Field	Type	Default	Description
ALL	ShapeStyle	**Optional**	The style applied to all shapes.
TEXT	ShapeStyle	**Optional**	The style applied to TEXT shapes. Overrides the "ALL" style.
ARROW	ShapeStyle	**Optional**	The style applied to ARROW shapes. Overrides the "ALL" style.
LINE	ShapeStyle	**Optional**	The style applied to LINE shapes. Overrides the "ALL" style.
ELLIPSE	ShapeStyle	**Optional**	The style applied to ELLIPSE shapes. Overrides the "ALL" style.
WKT	ShapeStyle	**Optional**	The style applied to WKT shapes. Overrides the "ALL" style.

Example

```
{
  ALL: {
    strokeColor: '#00EE00',
    fillOpacity: 0,
    fontColor: '#00EE00',
    fontSize : 16,
  },
  WKT: {
    strokeColor: '#AAAAEE',
    fillOpacity: 0.4,
    strokeDashstyle: 'dash',
    fontColor: '#AAAAEE',
    fontSize : 12,
  },
}
```

ShapeStyle

Type: `object`

Field	Type	Default	Description
`pointRadius`	`number > 0`	`10`	The radius in pixels of the point disk marker when rendered on the screen.
`fillColor`	`SVGPaint`	`'#AAAAAA'`	See MDN's SVG Paint
`fillOpacity`	`number ∈ [0, 1]`	`0.4`	The opacity of the fill color. `0.0` (fully transparent) to `1.0` (fully opaque).
`strokeColor`	`SVGPaint`	`'#AAAAEE'`	See MDN's SVG Paint

Field	Type	Default	Description
`strokeOpacity`	`number ∈ [0, 1]`	`1`	The opacity of the stroke color. `0.0` (fully transparent) to `1.0` (fully opaque).
`strokeWidth`	`string` or `number`	`2`	The width of the outline on the polygon. If a value of `0` is used the outline will never be drawn. MDN's SVG stroke-width
`strokeDashstyle`	`string`	`'solid'`	Controls the pattern of dashes and gaps used to stroke the polygon. Can be one of (`'dot'`, `'dash'`, `'dashdot'`, `'longdash'`, `'longdashdot'`, `'solid'`) or a list of white space separated `<length>`s and `<percentage>`s that specify the lengths of alternating dashes and gaps. If an odd number of values is provided, then the list of values is repeated to yield an even number of values. Thus, `'5 3 2'` is equivalent to `'5 3 2 5 3 2'`.
`fontColor`	`SVGPaint`	`'#AAAAAA'`	See MDN's SVG Paint
`fontSize`	`number >= 0`	`12`	The text size in pixels when rendered on the screen.

Example

```
{
  strokeColor: '#00EE00',
  fillOpacity: 0,
}
```

Layer

Type: `OpenLayers.Layer`

For more info see OpenLayers 2's official documentation.

Example

```
new OpenLayers.Layer.Google('Google Satellite', {
  type: google.maps.MapTypeId.SATELLITE,
  numZoomLevels: 20,
  maxExtent: new OpenLayers.Bounds(maxExtent),
}),
```

```
new OpenLayers.Layer.Google('Google Streets', {
  type: google.maps.MapTypeId.ROADMAP,
  numZoomLevels: 22,
  maxExtent: new OpenLayers.Bounds(maxExtent),
}),
```

TimeRange

Type: `[string, Timespan, boolean]` or `[string, Timespan]`

The elements of the tuple are as follows:

1. The label of the time range.
2. The width of the time range.
3. `true` if the time range should be selected by default, otherwise `false`.

For more info see Timeline.

Examples

```
['6-Hour', '6h']
```

```
['Daily', '1d', true]
```

Timespan

Type: `string`

Syntax

```
<Timespan> ::= <number> <unit>
<number> ::= <a positive decimal number>
<unit> ::= "s" # second
         | "M" # minute
         | "h" # hour
         | "d" # day
         | "w" # week
         | "m" # month
         | "y" # year
         | "D" # decade
         | "c" # century
```

Examples

- `'6h'` 6 hours timespan
- `'1m'` 1 month timespan

MarkerSymbols

Type: `{[name: string]: MarkerSymbol}`

Example

```
{
    left_arrow_tip: [-8,4, -8,-4, 0,0, -8,4],
}
```

MarkerSymbol

Type: `number[]`

A list of coordinate pairs (x and y) (much like the MDN's SVG polygon points) which will form a polygon **anchored on** `(0, 0)`.

Example

```
[-8,4, -8,-4, 0,0, -8,4]
```

gives the following SVG polygon:

```
<polygon points="-8,4 -8,-4 0,0 -8,4"></polygon>
```

Product

Type: `object`

Field Field	Type Type	Default Default	Description Description
General			
`id`	`string`	**Required**	The ID used to reference the product in the portal.
`label`	`string`	**Required**	The label of the product (used in the products side panel among other places), it **supports HTML**.
`type`	`ProductType`	**Required**	The type of the product's datasets and how they are rendered. For more info see ProductType.
`description`	`string`	**Optional**	The description of the product (shown as a tooltip in products side panel)
`infoURL`	`URL`	**Optional**	The URL of a GitHub Flavored Markdown file containing information about this product.
`tags`	`string[]`	`[]`	A list of tags associated to the product. Used for grouping and searching in the products list. May be displayed to the user. Each tag may be of the form `${key}:${value}` in which case it can be used for grouping and only the `value` will be used for searching.
`selected`	`boolean`	`false`	If `true` the product is selected by

Field	Type	Default	Description
			default.
`colormaps`	`Colormap[]`	**Optional**	The description of the colormaps associated with this product to be displayed in the Display data dialog.
`unit`	`string`	**Optional**	The unit of the values given in the colormaps. Will be displayed in the Display data dialog. For log-scale units prefix the unit with `log`.
`granuleInfoConfig`	`ProductGranuleInfoConfig`	**Optional**	If provided specifies which granule info groups, resources and metadata fields should be available to the user in the interface, and how they should be presented. Otherwise none will be available to the user in the interface.
Timeline			
`mustBeCurrent`	`boolean`	**Required**	If `true` the product's datasets are considered valid (and thus added to the map) only when the current date/time is inside their respective definition range. Otherwise (if `false`) they are considered valid if their respective definition range intersects the current timespan.

Field	Type	Default	Description
`timelineValidity`	`number > 0`	**Optional**	If provided and `mustBeCurrent` is set to `false`, the product's datasets are considered valid if their respective definition range intersects a range of ± the value of this field in hours around the current date/time.
`permanent`	`boolean`	`false`	If `true` the product's datasets are excluded for the coverage and find nearest requests.
Pagination			For more info see Pagination.
`priority`	`number`	`0`	The priority of the product. Bigger values imply higher priority.
`datasetWeight`	`number >= 0`	`1`	The average rendering weight of any one of the product's datasets.
`minPageSize`	`number >= 0`	`1`	The minimum number of the product's datasets that will be rendered (if available) regardless of any pagination.
General Rendering			*Applies to all types*
~~`noOutline`~~	`boolean`	*Deprecated*	*Deprecated: Use `outlines` instead.* If `true` the outlines (in white around the data) of the product's datasets will be hidden.

Field	Type	Default	Description
`outlines`	`ProductOutlines`	*see below*	When to show the outlines (in white around the data) of the product's datasets. This takes precedence over the deprecated `noOutline` if both specified.
`mapMinZoom`	`integer >= 0`	**Optional**	If provided the product's datasets will be hidden for zoom level smaller (farther away) than the value of this field.
`mapMaxZoom`	`integer >= 0`	**Optional**	If provided the product's datasets will be hidden for zoom level bigger (nearer) than the value of this field.
`opacity`	`number ∈ [0, 1]`	`1`	The default opacity of the datasets. `0.0` (fully transparent) to `1.0` (fully opaque).
`stackLevel`	`integer ∈ [0, 1000]`	**Required**	Specifies the default z-order of the product's datasets. When datasets overlap, z-order determines which one covers the other. A dataset with a larger `stackLevel` generally covers an element with a lower one.
Tile Rendering (ZXY)			*Applies to ZXY products only*
`tileOrigin`	`number[2]`	**Required**	The coordinates of the tile (0, 0) defined as `[lon, lat]` both in the projection `projection`.

Field	Type	Default	Description
`tileSize`	`number[2]`	**Required**	The size of each tile defined as `[width, height]` both in pixels.
Dynamic Rendering			*Applies to some product types and with restrictions, see Product Dynamic Rendering Notes*
`dataFields`	`DataFields`	**Required**	Defines the fields (or channels) found in the datasets with their min and max. For more info see DataFields.
`renderFields`	`RenderFields`	**Required**	Defines the fields used for rendering the datasets. For more info see RenderFields.
`density`	`number ∈ ]0, 2]`	`1`	The density used when rendering the product's datasets.
Streamlines Rendering			*Applies to STREAMLINES products only*
`speed`	`number > 0`	`1`	The factor to multiply the modulus by when rendering the product's datasets. (Bigger values imply faster streamlines)
Moorings Rendering			*Applies to MOORED products only*
`marker`	`MarkerStyle`	**Optional**	Defines the pictogram displayed at the location of the dataset. Defaults to a buoy icon. For more info see MarkerStyle.

Field	Type	Default	Description
Trajectory Rendering			*Applies to TRAJECTORIES products only*
`marker`	`Marker`	**Optional**	The trajectory marker to render to indicate the position and direction at the closest (in the past) point of the trajectory to the current datetime. Defaults to a diamond shape. For more info see Marker.
`markerPostion`	`number ∈ [0, 1]`	**Optional**	If provided specifies a fixed position (independent of the current datetime) of the marker. `0` being the first point of the trajectory and `1` the last.
`startMarker`	`Marker`	**Optional**	The trajectory marker to render at the start of the trajectory. For more info see Marker.
`endMarker`	`Marker`	**Optional**	The trajectory marker to render at the end of the trajectory. For more info see Marker.
`withRangeMarkers`	`boolean`	`false`	If `true` brackets will rendered around the part of the trajectory that is in the current timespan.
`restrictMarkersToZoomRange`	`boolean`	`false`	If `true` markers will be hidden on zoom levels outside of the product's zoom

Field	Type	Default	Description
			range (`mapMinZoom` and `mapMaxZoom`).
Shapes			*Applies to USER_SHAPES and SERVER_SHAPES products only*
`style`	`ShapeStyleMap`	**Optional**	The shapes rendering style description. For more info see ShapeStyleMap.
Server Shapes			*Applies to SERVER_SHAPES products only*
`shapes`	`Shape[]`	**Required**	A list of predefined shapes. For more info see Shapes.
Layer			*Applies to LAYER products only*
`layer`	`Layer`	**Required**	Any OpenLayers layer to be used as the one and only dataset of the product.
`validFrom`	`Date`	**Optional**	If provided, the layer will be hidden before that date.
`validTill`	`Date`	**Optional**	If provided, the layer will be hidden starting from that date.
`updateLayer`	`function`	**Optional**	If provided, gets called once on startup and whenever the current date changes and passed the layer and the current date. It allows the product to update its layer when the current date changes.
Time Layer			*Applies to TIMELAYER products only*

Field	Type	Default	Description
`layer`	`Layer`	**Required**	Any OpenLayers layer to be used for all the datasets of the product.
`updateLayer`	`function`	**Required**	Gets called whenever any dataset of the product becomes visible and passed the layer and the current date. It allows the product to update its layer when the dataset shown changes.
ncWMS			*Applies to NCWMS products only*
`ncwmsURL`	`URL`	**Required**	The URL of the ncWMS.
`ncwmsParams`	`Object`	`{}`	The parameters to pass with every request to the ncWMS server.
`ncwmsQueryAdapter`	`function`	**Optional**	If provided, gets called when a dataset is created and each time it is added to the map. This function receives a dataset object and must return an object with `url` and `params` (dict containing modified query parameters) properties.

`outlines` default value

The default value for the `outlines` property is `'ALWAYS_SHOWN'`.
Except for TRAJECTORIES and GEOJSON products, the default value is `'HIDDEN_IN_ZOOM_RANGE'`.

Product Dynamic Rendering Notes

Required restrictions on `renderFields` field by type:

type	Restrictions
STREAMLINES	must include `angle`, `modulus` and `color` fields, and `color` must be a uniform color.
ARROWS	must include `angle`, `modulus` and `color` fields.
BARBS	must include `angle`, `modulus` and `color` fields.
TRAJECTORIES	must include `color` field.
GEOJSON	must include `color` field.

The `density` field is restricted to the following types:

- ARROWS
- BARBS
- STREAMLINES

Example

```
{
  /* General */
  label: 'Drifters 1000m North Atlantic speed (ANDRO)',
  id: '3857_ARGO_Deep_NATL1000',
  tags: [
    'Group:In Situ',
    'Data type:In-Situ',
    'Level of processing:In-Situ',
    'Sensor:In-Situ',
    'Platform / Mission:In-Situ',
```

```
    'Depth:1000m',
    'Display type:Arrows',
  ],
  type: 'TRAJECTORIES',
  colormaps: [{
    palette: 'images/palettes/yellow_red.cpt',
    ticks: makeDeltaTicks(0, 0.35, 0.05),
  }],
  unit: 'm/s',

  /* Timeline */
  mustBeCurrent: false,

  /* Pagination */
  datasetWeight: 0.1,

  /* General Rendering */
  mapMinZoom: 2,
  stackLevel: 90,

  /* Dynamic Rendering */
  dataFields: {
    speed: {
      channel: 'speed',
      min: 0,
      max: 0.35,
    },
  },
  renderFields: {
    color: 'speed',
  },

  /* Trajectory Rendering */
  marker: argoMarker, // argoMarker is a function defined elsewhere in the config file.
  markerPostion: 1,
}
```

ProductType

Type: `string`
Possible values:

- `'LAYER'`
- `'TIMELAYER'`
- `'NCWMS'`
- `'IMAGE'`
- `'ZXY'`
- `'ARROWS'`
- `'BARBS'`
- `'STREAMLINES'`
- `'TRAJECTORIES'`
- `'GEOJSON'`
- `'MOORED'`
- `'USER_SHAPES'`
- `'SERVER_SHAPES'`

LAYER

Used for third-party/external layers that are not known by the Syntool services server, thus are neither included in the coverage nor the find nearest requests.

Internally, products with this type have a single and unique dataset created automatically and are valid all the time unless they are limited by `validFrom` and `validTill` (if specified).

Useful for adding Google Earth Engine, SHOM, or Meteosat layers to a portal.

TIMELAYER

Used for third-party/external layers that are known by the Syntool services server, but their data is loaded via an external WMS service, for example.

Useful when no two datasets of the product overlap in time.

NCWMS

Used for ncWMS based layers like CMEMS ones.

IMAGE

Used for non-tiled raster datasets. Allows unconstrained zoom.

ZXY

Used for tiled raster datasets.

ARROWS

Used for arrow vector fields.

BARBS

Used for barbs vector fields.

STREAMLINES

Used for streamline vector fields.

TRAJECTORIES

Used for trajectories with the notion of time. Allows to have a marker.

GEOJSON

Used for trajectories without the notion of time (like synoptic datasets).

MOORED

Used for moored buoys.

USER_SHAPES

See Shapes section.

SERVER_SHAPES

See Shapes section.

ProductOutlines

Type: `string`
Possible values:

- `'ALWAYS_SHOWN'`
- `'HIDDEN_IN_ZOOM_RANGE'`
- `'ALWAYS_HIDDEN'`

Group *(Deprecated: Use product's `tags` and `groupProductsBy` instead)*

Type: `object`

Field	Type	Default	Description
`label`	`string`	`null`	The label of the group (used in the products side panel if given)
`products`	`string[]`	**Required**	An ordered list of IDs of the products in that group

Notes

- All product IDs MUST be IDs of existing products in the config.
- All product IDs MUST be unique (a product can be in no group or in one group and only once in that group).

Deprecation Notes

- When `label` is not `null`, will add a `Group:${label}` tag to each one of `products`, if that product doesn't have a `Group` tag already.

Color

Type: `string`

See MDN's CSS Color, note that only the `rgb()`, `#RRGGBB`, `#RGB`, and `rgba()` format are supported and percentages are not supported.

- `#RRGGBB` with `RR`, `GG` and `BB` being 2-digit hexadecimal numbers.
- `#RGB` with `R`, `G` and `B` being 1-digit hexadecimal numbers.
- `rgb(red, green, blue)` with `red`, `green` and `blue` being integers between `0` and `255`.
- `rgba(red, green, blue, alpha)` with `red`, `green` and `blue` being integers between `0` and `255` and `alpha` between `0` and `1`.

Examples

- `#FF0000` is red
- `#F00` is red
- `rgb(255, 0, 0)` is red
- `rgba(255, 0, 0, 0.5)` is red with 50% opacity.

Colormap

Type: `object`

Field	Type	Default	Description
`label`	`string`	**Optional**	The label, if any, to be shown with the colormap. Useful in case of multiple colormaps for the same product.
`colorPalette`	`URL`	**Required**	The URL of the color palette to be used. For more info see the Color Palettes section.
`ticks`	`ColormapTick[]`	**Required**	The list of tick positions and values or labels to be placed on the colormap.

Helper JS functions

The following helper functions are for creating a colormap tick list in certain patterns:

- `makeValuesTicks`
- `makeDeltaTicks`
- `makeMinMaxTicks`

Example

```
{
  // use the "yellow" to "red" color palette
  colorPalette: 'images/palettes/yellow_red.cpt',

  // with ticks `0` at the left, `0.35` at the right,
  // and intermediate ticks for every `0.05` increments
  ticks: makeDeltaTicks(0, 0.35, 0.05),
}
```

ColormapTick

Type: `[number ∈ [0, 1], number | string]`

The elements of the tuple are as follows:

1. The relative position of the tick on the colormap (`0` being the minimum and `1` the maximum).
2. The value at the position of the tick as `number` or the label as `string`.

Note: Values of type `number` may be used in the future to calculate the value-under-cursor.

Examples

- `[0, -1]` places at tick at the minimum with a value of `-1`
- `[0.5, 2]` places at tick at the middle with a value of `2`
- `[1, 5]` places at tick at the maximum with a value of `5`
- `[0, 'min']` places at tick at the minimum labeled "min"
- `[0.5, '']` places at tick at the maximum without any label
- `[1, 'max']` places at tick at the maximum labeled "max"

DataFields

Type: `{[name: string]: DataField}`

Example

```
{
  modulus: {
    channel: 0,
    min: 0,
    max: 10,
  },
  temp: {
    min: 3.0,
    max: 12.5,
  },
}
```

DataField

Type: `object`

Field	Type	Default	Description
`channel`	`string` or `number`	**Optional**	The dataset's channel represented by this field. Defaults to the name of this field.
`min`	`number`	**Required**	The minimum value.
`max`	`number`	**Required**	The maximum value.

RenderFields

Type: `object`

Field	Type	Default	Description
`modulus`	`string` or `number`	**Optional**	The name of the field defined in `dataFields` of the same product. Or the modulus to apply uniformly.
`angle`	`string` or `number`	**Optional**	The name of the field defined in `dataFields` of the same product. Or the angle to apply uniformly.
`color`	`string` or `Color`	**Required**	The name of the field defined in `dataFields` of the same product. Or the color to apply uniformly.

See Product Dynamic Rendering Notes for type specific restrictions.

Example

```
{
  modulus: 'speed',
  color: 'temp',
}
```

Marker

Type: `MarkerStyle` or `MarkerFn`

Specifies the style in which the marker should be rendered.

MarkerFn

Type: `(color: Color, dataset: Dataset, features: OpenLayers.Feature[]) -> MarkerStyle`

If a function is provided, it will be called in order to get the marker style for the given color, and must return a MarkerStyle.

Parameter	Type	Description
`color`	`Color`	The color of the marker should have.
`dataset`	`Dataset`	The dataset in which the marker is used.
`features`	`OpenLayers.Feature[]`	List of all the trajectory features of the dataset.

Note: This function might be called multiple times, or just once and its return value cached.

Example

```
function argoMarker(color) {
  return {
    graphicName: 'left_arrow_tip',
    pointRadius: 4, // 2*4 = 8px wide and heigh on screen
    strokeWidth: 1,
    strokeColor: color,
    fillColor: color,
  };
}
```

MarkerStyle

Type: `MarkerStateStyle` or `object`

Field	Type	Default	Description
`default`	`MarkerStateStyle`	**Required**	The style to use for the marker by default.
`selected`	`MarkerStateStyle`	`default`	The style to use for the marker when the dataset is selected.

Note: If a single `MarkerStateStyle` is given (instead of an `object`), it will be used for all the states.

MarkerStateStyle

Type: `ImageMarkerStyle` or `PolygonMarkerStyle`

ImageMarkerStyle

Type: `object`

Field	Type	Default	Description
`rotation`	`number`	`0`	The angle in degrees between the marker pointing to the right and its position in the image CW. Use `NaN` to prevent rotation.
`externalGraphic`	`URL`	**Required**	The URL to the external image to use.
`graphicWidth`	`number > 0`	**Required**	The width in pixels of the marker when rendered on the screen.
`graphicHeight`	`number > 0`	**Required**	The height in pixels of the marker when rendered on the screen.
`graphicXOffset`	`number`	`-width / 2`	The pixel offset along the positive x axis (left to right) for displacing the image.
`graphicYOffset`	`number`	`-height / 2`	The pixel offset along the positive y axis (top to bottom) for displacing the image.
`graphicHFlip`	`boolean`	`false`	If set to `true`, the image will be flipped horizontally.
`graphicVFlip`	`boolean`	`false`	If set to `true`, the image will be flipped vertically.

Example

```
{
  rotation: 90,
  externalGraphic: 'css/images/marker-white.png',
  graphicWidth: 14,
  graphicHeight: 17,
}
```

PolygonMarkerStyle

Type: `object`

Field	Type	Default	Description
`rotation`	`number`	`0`	The angle in degrees between the marker pointing to the right and its defined position CW. Use `NaN` to prevent rotation.
`graphicName`	`string`	**Required**	The name of the symbol to render. Can be one of (`'circle'`, `'star'`, `'cross'`, `'x'`, `'square'`, `'triangle'`), or one of the fields defined in `markerSymbols`.
`pointRadius`	`number > 0`	**Required**	Half the width in pixels of the marker when rendered on the screen.
`fillColor`	`SVGPaint`	**Optional**	See MDN's SVG Paint
`fillOpacity`	`number ∈ [0, 1]`	`1`	The opacity of the fill color. `0.0` (fully transparent) to `1.0` (fully opaque).
`strokeColor`	`SVGPaint`	**Optional**	See MDN's SVG Paint

Field	Type	Default	Description
`strokeOpacity`	`number ∈ [0, 1]`	`1`	The opacity of the stroke color. `0.0` (fully transparent) to `1.0` (fully opaque).
`strokeWidth`	`string` or `number`	`1`	The width of the outline on the polygon. If a value of `0` is used the outline will never be drawn. MDN's SVG stroke-width
`strokeDashstyle`	`string`	**Optional**	Controls the pattern of dashes and gaps used to stroke the polygon. Can be one of (`'dot'`, `'dash'`, `'dashdot'`, `'longdash'`, `'longdashdot'`, `'solid'`) or a list of white space separated `<length>`s and `<percentage>`s that specify the lengths of alternating dashes and gaps. If an odd number of values is provided, then the list of values is repeated to yield an even number of values. Thus, `'5 3 2'` is equivalent to `'5 3 2 5 3 2'`.
`graphicHFlip`	`boolean`	`false`	If set to `true`, the polygon will be flipped horizontally.
`graphicVFlip`	`boolean`	`false`	If set to `true`, the polygon will be flipped vertically.

Example

```
{
  graphicName: 'left_arrow_tip',
  pointRadius: 4, // 2*4 = 8px wide and heigh on screen
  strokeWidth: 1,
  strokeColor: 'red',
  fillColor: 'red',
}
```

GranuleInfo

Type: `{[groupId: string]: GranuleInfoGroup}`

The key (`groupId`) is the granule info group's ID (corresponding to `metadata.syntool_id` in the API response).
The value is the granule info group containing metadata and resources. See GranuleInfoGroup.

Example

```
{
  default: {
    metadata: {
      satellite: 'Sentinel-3',
      instrument: 'OLCI',
    },
    register: {
      url: 'https://portal.creodias.eu/register.php',
    },
    download: {
      url: 'https://finder.creodias.eu/?productIdentifier=S3B_OL_2_LFR____20210425T102604_20210425T102904_2021(
      description: 'Get data',
    },
  },
}
```

GranuleInfoGroup

Type: `{[resourceId: string]: GranuleInfoResource, metadata: GranuleInfoMetadata}`

The key (`resourceId`) is the granule info resource's ID.
The value is the granule info resource containing fields like `url` and `description`. See GranuleInfoResource.

Note: The field "metadata" is special and contains key-value metadata. See GranuleInfoMetadata.

Example

```
{
  metadata: {
    satellite: 'Sentinel-3',
    instrument: 'OLCI',
  },
  register: {
    url: 'https://portal.creodias.eu/register.php',
  },
  download: {
    url: 'https://finder.creodias.eu/?productIdentifier=S3B_OL_2_LFR____20210425T102604_20210425T102904_202104
    description: 'Get data',
  },
}
```

GranuleInfoResource

Type: `object`

Field	Type	Default	Description
`url`	`URL`	**Required**	The URL of the linked resource.
`description`	`string`	**Optional**	The description of this resource. **Text only**

Examples

```
{
  url: 'https://portal.creodias.eu/register.php',
}
```

```
{
  url: 'https://finder.creodias.eu/?productIdentifier=S3B_OL_2_LFR____20210425T102604_20210425T102904_20210425T
  description: 'Get data',
}
```

GranuleInfoMetadata

Type: `{[key: string]: string}`

The key (`key`) is the field's ID.
The value is the field's value, it **supports HTML**.

Example

```
{
  satellite: 'Sentinel-3',
  instrument: 'SLSTR',
}
```

ProductGranuleInfoConfig

Type: GranuleInfoConfig or GranuleInfoConfigFn

Specifies which granule info groups, resources and metadata fields should be available to the user in the interface, and how they should be presented.

GranuleInfoConfigFn

Type: `(dataset: Dataset, info: GranuleInfo) -> GranuleInfoConfig?`

This function will be called in order to get the configuration for a given dataset and its info, and must return an optional GranuleInfoConfig.

If this function returns `null` or `undefined`, the entire granule info will be ignored and not accessible to the user.

Parameter	Type	Description
`dataset`	`Dataset`	The dataset to which the info apply.
`info`	`GranuleInfo`	The info of the dataset that require the configuring.

Note: This function might be called multiple times, or just once and its return value cached.

Example

`capitalizeString` is defined here

```
function(dataset, info) {
  return {
    unknownGroup: function(dataset, group, groupId) {
      return {
        title: capitalizeString(groupId),
        metadata: function(value, key, metadata, dataset) {
          return {
            label: capitalizeString(key),
            value: value,
          };
        },
        unknownResource: function(dataset, resource, resourceId) {
          return {
```

```
            type: 'link',
            label: capitalizeString(resourceId),
          };
        },
      };
    },
  };
}
```

GranuleInfoConfig

Type: `object`

Field	Type	Description
`autoResourceId`	`string`	The full ID, in the form `${groupId}/${resourceId}`, of the resource that should be activated automatically upon the selection of its dataset.
`groups`	`GranuleInfoGroupsConfig`	If given, specifies the configuration for known groups.
`unknownGroup`	`GranuleInfoUnknownGroupConfig`	If given, will be call to specify the configuration for a group not in `groups` field.

Note: All fields are optional.
Note: If a group has no corresponding configuration, it will be ignored and not accessible to the user.

Example

`capitalizeString` is defined here

```
{
  autoResourceId: 'default/profile',
  groups: {
    default: {
      title: 'Profile',
      metadata: {
        'cycle number': 'Cycle Number',
        'data centre': 'Data Centre',
        'grounded': 'Grounded',
        'positioning inverstigator': 'Positioning Inverstigator',
        'project': 'Project',
        'wmo id': 'WMO ID',
        'wmo inst. type': 'WMO Instrument Type',
      },
      profile: {
        type: 'image',
        label: 'Profile',
      },
    },
  },
  unknownGroup: function(dataset, group, groupId) {
    return {
      title: capitalizeString(groupId),
      metadata: function(value, key, metadata, dataset) {
        return {
          label: capitalizeString(key),
          value: value,
        };
      },
      unknownResource: function(dataset, resource, resourceId) {
        return {
          type: 'link',
          label: capitalizeString(resourceId),
        };
      }
```

```
    };
  }
}
```

GranuleInfoGroupsConfig

Type: `{[groupId: string]: GranuleInfoGroupConfig}`

The key (`groupId`) is the group's ID referencing `groupId` of GranuleInfo.
The value is the configuration for that group. See GranuleInfoGroupConfig.

Example

```
{
  default: {
    title: 'Profile',
    metadata: {
      'cycle number': 'Cycle Number',
      'data centre': 'Data Centre',
      'grounded': 'Grounded',
      'positioning inverstigator': 'Positioning Inverstigator',
      'project': 'Project',
      'wmo id': 'WMO ID',
      'wmo inst. type': 'WMO Instrument Type',
    },
    profile: {
      type: 'image',
      label: 'Profile',
    },
  },
}
```

GranuleInfoUnknownGroupConfig

Type: `(dataset: Dataset, group: GranuleInfoGroup, groupId: string) -> GranuleInfoGroupConfig?`

This function will be called in order to get the configuration for a given info group associated to a dataset, and must return an optional GranuleInfoGroupConfig.

If this function returns `null` or `undefined`, the entire info group will be ignored and not accessible to the user.

Parameter	Type	Description
`dataset`	`Dataset`	The dataset with which the info group is associated.
`group`	`GranuleInfoGroup`	The info group that require the configuring.
`groupId`	`string`	The ID of info group `group` referencing `groupId` of GranuleInfo.

Note: This function might be called multiple times, or just once and its return value cached.

Example

`capitalizeString` is defined here

```
function(dataset, group, groupId) {
  return {
    title: capitalizeString(groupId),
    metadata: function(value, key, metadata, dataset) {
      return {
        label: capitalizeString(key),
        value: value,
      };
    },
    unknownResource: function(dataset, resource, resourceId) {
      return {
```

```
        type: 'link',
        label: capitalizeString(resourceId),
      };
    }
  };
}
```

GranuleInfoGroupConfig

Type: `object`

Specifies how the info group along with its resources and metadata fields should be available to the user in the interface, and how they should be presented.

Field	Type	Default	Description
`title`	`string`	**Required**	The title of the group as presented to the user, it **supports HTML**.
`order`	`number`	`0`	Used to sort the groups before presenting them.
`metadata`	`GranuleInfoMetadataConfig`	**Optional**	If given, specifies the configuration for the metadata fields.
`resources`	`GranuleInfoResourcesConfig`	**Optional**	If given, specifies the configuration for known resources.
`unknownResource`	`GranuleInfoUnknownResourceConfig`	**Optional**	If given, will be call to specify the configuration for a resource not in

Field	Type	Default	`resources` fieldDescription
`metadataDefaults`	`GranuleInfoMetadata`	**Optional**	If given, specifies defaults for missing metadata fields.

Note: If no configuration is given for the `metadata` field, all the metadata information included in this group will be ignored and not accessible to the user.
Note: If a resource has no corresponding configuration, it will be ignored and not accessible to the user.

Example

```
{
  title: 'Data access',
  metadata: {
    satellite: 'Satellite',
    instrument: 'Instrument',
  },
  resources: {
    download: {
      type: 'link',
      label: 'Get Data',
      order: 2,
    },
    register: {
      type: 'link',
      label: 'Register',
      order: 2,
    },
  },
}
```

GranuleInfoMetadataConfig

Type: GranuleInfoMetadataFieldsConfig or GranuleInfoMetadataConfigFn

Specifies which metadata fields should be available to the user in the interface, and how they should be presented.

GranuleInfoMetadataFieldsConfig

Type: `{[key: string]: string}`

The key (`key`) is the metadata field's ID referencing `key` of GranuleInfoMetadata.
The value is the label of that metadata field, it **supports HTML**.

Note: All metadata fields *not* specified, will be ignored and not accessible to the user.

Example

```
{
  'cycle number': 'Cycle Number',
  'data centre': 'Data Centre',
  'grounded': 'Grounded',
  'positioning inverstigator': 'Positioning Inverstigator',
  'project': 'Project',
  'wmo id': 'WMO ID',
  'wmo inst. type': 'WMO Instrument Type',
}
```

GranuleInfoMetadataConfigFn

Type: `(value: string, key: string, metadata: GranuleInfoMetadata, dataset: Dataset) -> GranuleInfoMetadataFieldConfig?`

This function will be called in order to get the configuration for a given metadata field, and must return an optional GranuleInfoMetadataFieldConfig.

If this function returns `null` or `undefined`, the metadata field will be ignored and not accessible to the user.

Parameter	Type	Description
`value`	`string`	The value of the metadata field.
`key`	`string`	The ID of metadata field referencing `key` of GranuleInfoMetadata.
`metadata`	`GranuleInfoMetadata`	All the metadata fields in the group.
`dataset`	`Dataset`	The dataset with which the metadata field is associated.

Note: This function might be called multiple times, or just once and its return value cached.

Example

`capitalizeString` is defined here

```
function(value, key, metadata, dataset) {
  if (key === 'wmo id') {
    return {
      label: 'WMO ID',
      description: 'World Meteorological Organization identifier',
      value: value,
    };
  }
  if (key === 'wmo inst. type') {
    return {
      label: 'WMO Instrument Type',
      value: value,
    };
  }
  return {
    label: capitalizeString(key),
```

```
    value: value,
  };
}
```

GranuleInfoMetadataFieldConfig

Type: `object`

Specifies how a metadata field should be presented to the user in the interface.

Field	Type	Default	Description
`label`	`string`	**Required**	The label of the metadata field, it **supports HTML**.
`description`	`string`	**Optional**	The description of the metadata field. **Text only**
`value`	`string`	**Required**	The value of the metadata field, it **supports HTML**.
`order`	`number`	`0`	Used to sort the metadata fields and resources of the group before presenting them.

See GranuleInfoMetadataConfigFn for examples.

GranuleInfoResourcesConfig

Type: `{[resourceId: string]: GranuleInfoResourceConfig}`

The key (`resourceId`) is the resource's ID referencing `resourceId` of GranuleInfoResource.
The value is the configuration for that resource. See GranuleInfoResourceConfig.

Example

```
{
  profile: {
    type: 'image',
    label: 'Profile',
  },
}
```

GranuleInfoUnknownResourceConfig

Type: `(dataset: Dataset, resource: GranuleInfoResource, resourceId: string) -> GranuleInfoResourceConfig?`

This function will be called in order to get the configuration for a given resource associated to a dataset, and must return an optional GranuleInfoResourceConfig.

If this function returns `null` or `undefined`, the resource will be ignored and not accessible to the user.

Parameter	Type	Description
`dataset`	`Dataset`	The dataset with which the resource is associated.
`resource`	`GranuleInfoResource`	The resource that require the configuring.
`resourceId`	`string`	The ID of response `resource` referencing `resourceId` of GranuleInfoResource.

Note: This function might be called multiple times, or just once and its return value cached.

Example

`capitalizeString` is defined here

```
function(dataset, resource, resourceId) {
  return {
    type: 'image',
    label: capitalizeString(resourceId),
    inline: resource.url + '.thumb.png',
    order: 2,
  };
}
```

GranuleInfoResourceConfig

Type: `object`

Specifies how a resource should be presented to the user in the interface.

Field	Type	Default	Description
`type`	`GranuleInfoResourceType`	**Required**	The type of the resource.
`label`	`string`	**Required**	The label of the resource, it **supports HTML**.
`order`	`number`	`-1`	Used to sort the metadata fields and resources of the group before presenting them.
image			*Applies to image resources only*
`inline`	`boolean` or `URL`	`false`	When `true`, the same image will be shown inline. When a `URL`, the image located at its value will be shown inline. Otherwise the image is not shown inline.

See GranuleInfoResourceType for examples.

GranuleInfoResourceType

Type: `string`
Possible values:

- `'link'`
- `'image'`
- `'spectrum'`

Note: Any other type will be ignored and the resource will not accessible to the user.

link

Example

```
{
  type: 'link',
  label: 'Get Data',
}
```

image

Examples

```
{
  type: 'image',
  label: 'Profile',
}
```

```
{
  type: 'image',
  label: 'Profile',
  inline: true,
  order: 2,
}
```

```
{
  type: 'image',
  label: 'Profile',
  inline: 'https://example.com/image.png',
  order: 2,
}
```

spectrum

Example

```
{
  type: 'spectrum',
  label: 'Spectrum',
}
```

Helper JS functions

The following are some helper functions useful when configuring a Syntool portal.

capitalizeString

Type: `capitalizeString(input: string): string`

Returns the input string but with the first letter of each word capitalized.

```
function capitalizeString(input) {
  return input.replace(
    /(^|[^a-z])([a-z])(\w*)\b/ig,
    function(_, p1, p2, p3) {
      return p1 + p2.toUpperCase() + p3.toLowerCase();
    }
  );
}
```

makeValuesTicks

Type: `makeValuesTicks([[min: number,] max: number,] values: number[]): ColormapTick[]`

Returns a list of ticks such that all `values` are shown as ticks.

- If `min` is not given the first value will be used instead.
- If `max` is not given the last value will be used instead.

```
function makeValuesTicks(min, max, values) {
  if (Array.isArray(min)) {
    values = min;
    min = NaN;
    max = NaN;
  } else if (Array.isArray(max)) {
    values = max;
    max = NaN;
  }
  min = isNaN(min) ? values[0] : min;
```

```
  max = isNaN(max) ? values[values.length - 1] : max;
  return values.map(function(value) {
    return [(value - min) / (max - min), Math.round(value * 1e2) / 1e2];
  });
}
```

makeDeltaTicks

Type: `makeDeltaTicks(min: number, max: number, delta: number, multiple: boolean = false): ColormapTick[]`

Returns a list of ticks such that:

- The `min` value is included
- The `max` value is included
- If `multiple` is not set, all `min + n * delta < max` are included
  - except values that are visually too close to `max`
- If `multiple` is set, all multiples of delta between `min + delta` and `max` are included
  - except values that are visually too close to `max`

```
function makeDeltaTicks(min, max, delta, multiple) {
  var start = min + delta;
  if (multiple && start % delta !== 0) {
    start = ODL.nextMultiple(start, delta);
  }
  var values = [];
  values.push(min);
  for (var value = start; (value - min) / (max - min) < 0.92; value += delta) {
    values.push(value);
  }
  values.push(max);
  return makeValuesTicks(min, max, values);
}
```

makeMinMaxTicks

Type: `makeMinMaxTicks(count: number): ColormapTick[]`

Returns a list of ticks such that:

- The minimum tick is labeled "min"
- The maximum tick is labeled "max"
- There are in total `count` segments on the colormap

```
function makeMinMaxTicks(count) {
  var ticks = Array(count + 1);
  ticks[0] = [0, 'min'];
  for (var i = 1; i < count; i++) {
    ticks[i] = [i / count, ''];
  }
  ticks[count] = [1, 'max'];
  return ticks;
}
```