SEAScope user manual

v20260126

This document describes the configuration and user interface of the SEAScope viewer application.

It provides instructions to start the application, describes the graphical interface elements and the keyboard shortcuts that you can use to quickly display and browse your data in the 3D viewer.

More advanced subjects like configuration and extending capabilities using the SEAScope Python package are also covered in annexes.

Other useful resources are available on the SEAScope website, such as the Intermediate data Format definition, general concept and glossary

Quickstart

Linux	*macOS*	*Windows*
1. Open a terminal 2. Go to the directory where you unpacked the SEAScope release 3. Run the `./seascope` command	1. Hit CMD+Space to bring up Spotlight 2. Type «SEAScope» 3. Choose the SEAScope application (usually the first option)	1. Go to the directory where you unpacked the SEAScope release 2. Double-click on the `SEAScope` shortcut

Contents

Overview of the graphical interface **4**
- 1.1 3D map . . . 5
- 1.2 View info . . . 6
- 1.3 Toolbar . . . 6
- 1.4 Timeline control . . . 7
- 1.5 Side menu . . . 7
- 1.6 Rendering control . . . 7

Catalogue **8**
- 2.1 Dynamic collections . . . 9

Timeline **11**
- 3.1 Current datetime . . . 12
- 3.2 Timespan . . . 12
- 3.3 Browsing . . . 12
- 3.4 Colocation . . . 12
- 3.5 Minimal view . . . 13
- 3.6 Calendars view . . . 13

Rendering configuration **14**
- 4.1 Condensed view . . . 15
- 4.2 Advanced rendering control . . . 16

Granule interactions **19**
- 5.1 Selection . . . 20
- 5.2 Granule information . . . 20
- 5.3 Extraction and transect dialogs . . . 21

Annotations **23**
- 6.1 Create an annotation . . . 24
- 6.2 Annotation properties . . . 25

Settings **27**
- 7.1 Paths . . . 28
- 7.2 Rendering . . . 29
- 7.3 Timeline . . . 29
- 7.4 Misc . . . 32
- 7.5 Drawing . . . 32
- 7.6 Processor . . . 33

Key concepts **34**
- 8.1 Granule . . . 35
- 8.2 Collection . . . 35
- 8.3 Variable . . . 36

Collections configuration **37**
9.1 Configuration file . . . 38
9.2 Configuration file template . . . 39
9.3 List of settings . . . 40

Adding data **45**
10.1 Getting IDF files . . . 46
10.2 Importing IDF data in SEAScope . . . 47
10.3 Non-IDF data . . . 50

Troubleshooting **57**
11.1 Quick solutions . . . 58
11.2 Known issues . . . 58
11.3 How to report problems . . . 59
11.4 Contact and feedback . . . 59
11.5 Debug mode . . . 59

Appendices **61**

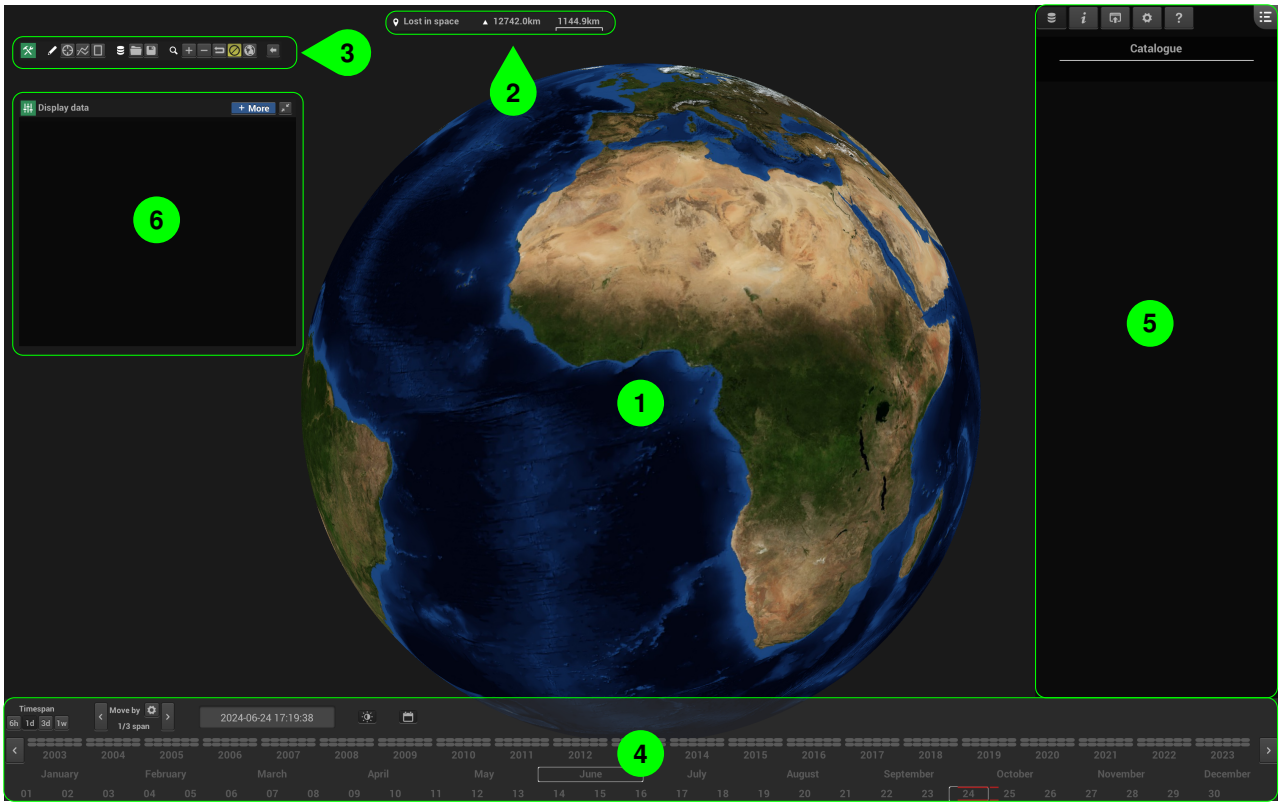
Annex I: Tips & tricks **62**
I.1 High resolution background . . . 63
I.2 Custom trajectory markers . . . 64
I.3 Multiple SEAScope instances . . . 64
I.4 Remote control . . . 65
I.5 Dialog boxes and file browsers on Linux . . . 65
I.6 Crafting annotations . . . 66

Annex II: SEAScope configuration file **71**
II.1 Configuration file template . . . 72
II.2 List of settings . . . 73

Annex III: Collection configuration examples **80**
III.1 AOML drifters . . . 81
III.2 ECMWF wind forecast . . . 82
III.3 ASCAT sea ice roughness . . . 84
III.4 SARAL / AltiKa sea level anomaly . . . 85
III.5 Sentinel-2 . . . 86

Overview of the graphical interface

The interface has been designed to maximize the screen area dedicated to rendering the data and allow users to hide or reduce most of the controls when they don't need them.
The first time you start SEAScope, when you have not added any data yet, your screen should look like this:

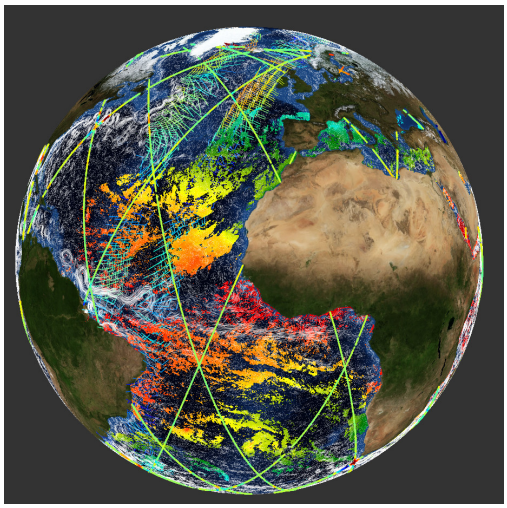


The viewer interface is composed of five main components that will be described in details in the next sections of the document:

1. 3D map
2. View info
3. Toolbar
4. Timeline control
5. Side menu
6. Rendering control

1.1 3D map

The 3D map is the canvas on which your data will be drawn and where you will be able to select granules, add custom annotations and make beautiful screenshots:



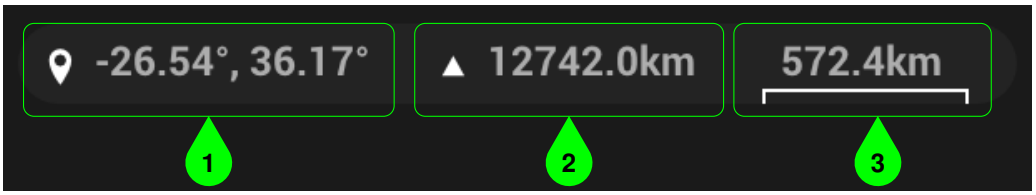
Use your mouse to navigate on the map:

- Place the mouse cursor on the globe, press the left mouse button and move the mouse to rotate the camera. When you have reached the desired point of view, release the left mouse button.
- Use the mouse wheel to zoom in/out. This can also be done using the keyboard: press Page Up for zooming in and Page Down for zooming out.

SEAScope conserves the viewing state between launches (i.e. the location and altitude or zoom level).

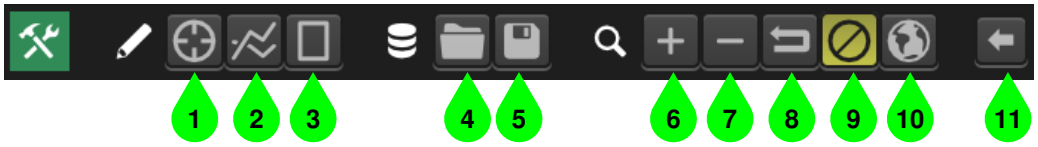
1.2 View info

Provides information about the parameters of the camera.



1. Location of the mouse cursor in terms of longitude and latitude (in decimal degrees).
2. The viewing altitude (i.e. distance between the camera and the surface of the globe) in meters or kilometers depending on its scale.
3. An indicator of the order of magnitude of the scale between pixels and geographical distances (in meters or kilometers depending on its scale).

1.3 Toolbar



Drawing controls

1. Draw point annotation.
2. Draw polyline annotation.
3. Draw polygon annotation.

Annotations load/save controls

4. Load annotations from file.
5. Save annotations to file.

Misc controls

11. Minimize toolbar.
    The minimized toolbar looks like this:

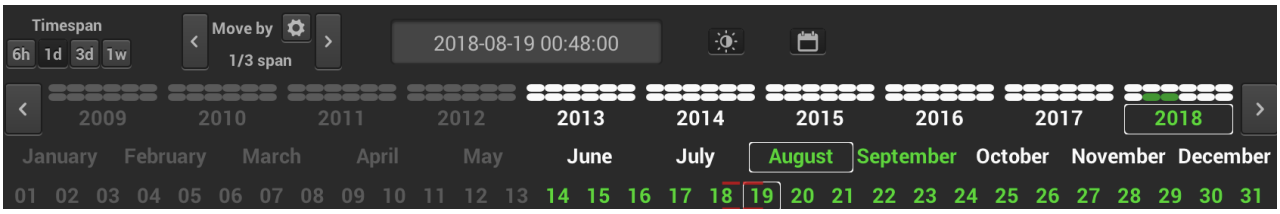
Camera controls

6. Zoom in.
7. Zoom out.
8. Reset the viewing state (i.e. the location and altitude or zoom level).
9. Aligned globe rotation toggle: when toggled, locks the globe rotation with longitude / latitude axis, otherwise the globe is free to rotate in any direction.
10. Rasterization mode selector:
    (a) F: full mode
    (b) W: wireframe mode
    (c) V: vertex mode

1.4 Timeline control

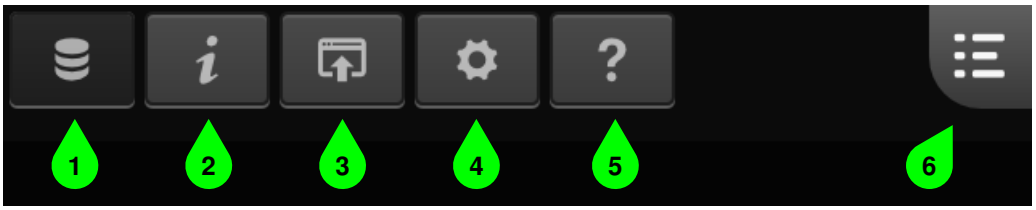
The timeline control shows the temporal availability of selected collections' granules and lets you configure how the viewer chooses which granules to display.

SEAScope conserves the timeline control's state between launches.



1.5 Side menu

The side menu is where most textual information and dialogs will be displayed.



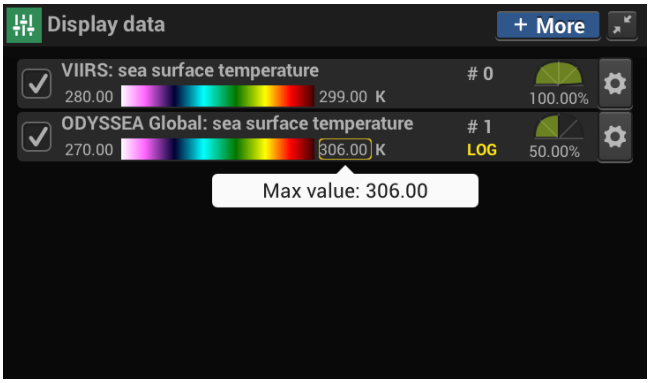
1. This button displays the data catalogue.
2. This button displays information about the selected granule.
3. This button displays information about the extracted data.
4. This button opens the settings panel.
5. The about button displays information about SEAScope, as well as ways to get help about the SEAScope.
6. The menu anchor has two roles: toggling the side menu (left click on the anchor) and switching the side of the screen on which the side menu is displayed (Shift key and left click on the anchor at the same time).

1.6 Rendering control

The rendering control lists all the variables selected in the catalogue and shows the settings used for rendering the associated data on the 3D map.

It also provides ways to change these settings interactively with an immediate visual feedback.

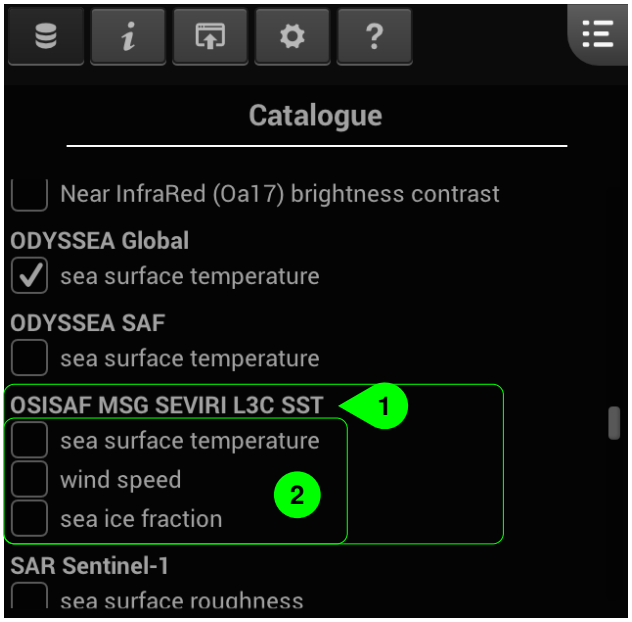
As it shows the values range and the colormaps (when relevant), the rendering control also acts as legend in SEAScope screenshots.



Catalogue

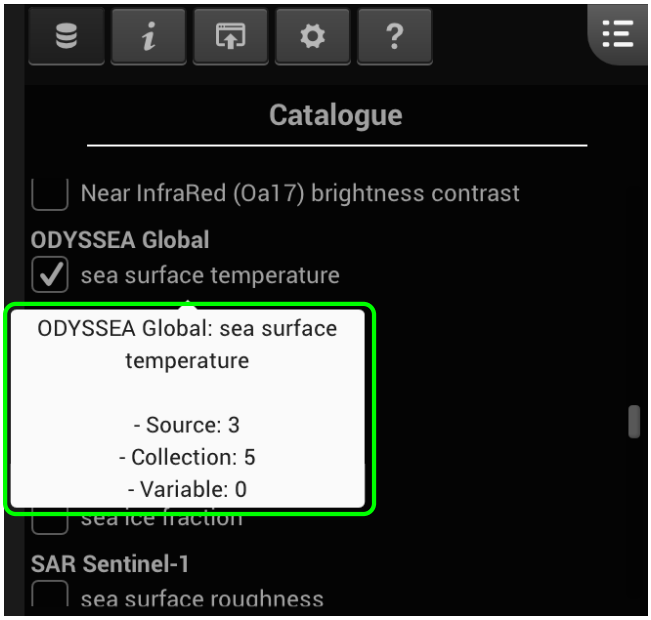
The catalogue content is built using the collection configuration files found in SEAScope's data directory. Its organization is based on a simple collection / variable hierarchy, i.e. variables are grouped under the label of the collection they belong to.

Note that having two collections with the same label is technically possible but can be a source of confusion: if two collections have the same labels, their variables will be grouped together, and if several variables also have the same label it will be impossible to distinguish between them only based on the information displayed by the catalogue.



The catalogue lists available variables for each collection.

1. Label of the collection.
2. Variables of the collection that you can display on the 3D map.



Place the mouse on a variable label to display a tooltip which gives information about the way SEAScope identifies the variable internally.

This information may be required if you want to write scripts that change the way a variable is displayed in the viewer.

Selecting the checkbox next to a variable label adds this variable to the rendering control and includes the temporal coverage of the collection's granules in the timeline.
Exclusive selection can be achieved by maintaining the Shift key pressed while clicking on the checkbox: in this case the catalogue unselects all variables but the one associated with the checkbox.

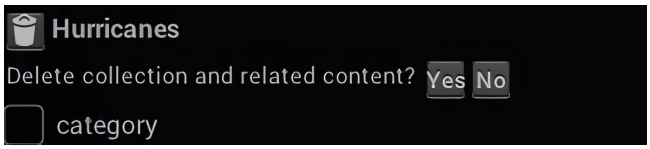
2.1 Dynamic collections

Collections can also be created while SEAScope is running, either for annotations or using the SEAScope API (Python bindings). Such collections have a different behavior than the ones created from the content of SEAScope's data directory.

Collections created using Python are volatile, they will disappear when you close the viewer, so you will have to run your Python script again to get them back on the next execution of SEAScope.

Annotation collections are persistent, SEAScope saves and restores them automatically so you can easily find them again on subsequent executions of the viewer.
Contrary to the volatile collections created using the API and the collections built from SEAScope's data directory, annotation collections can be deleted interactively using the trash bin icon next to the collection label:

As deleting an annotation collection also deletes all the associated annotations, SEAScope prompts for confirmation before actually deleting the collection:



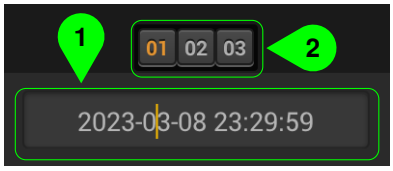
Here again, having several annotation collections with the same label will make it difficult to know which collection is actually deleted when clicking on the button. For example the default collections for points, polylines and polygons are all labelled "Annotations" so their variables all appear grouped together in the catalogue:

As there are actually three annotation collections but a single collection label, there is a single delete button: clicking on the button will only delete one of the three collections but SEAScope will hide the whole group of variables.
This is a caveat with the current version of SEAScope that will be fixed in the future.

Timeline

The timeline control lets you configure how the viewer chooses the granules to display based on temporal criteria.

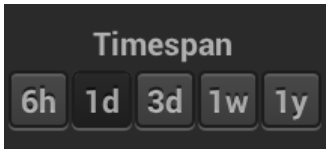
3.1 Current datetime



1. Datetime the timeline control considers as «current».
2. The timeline provides suggestions for dates while you edit the current datetime.

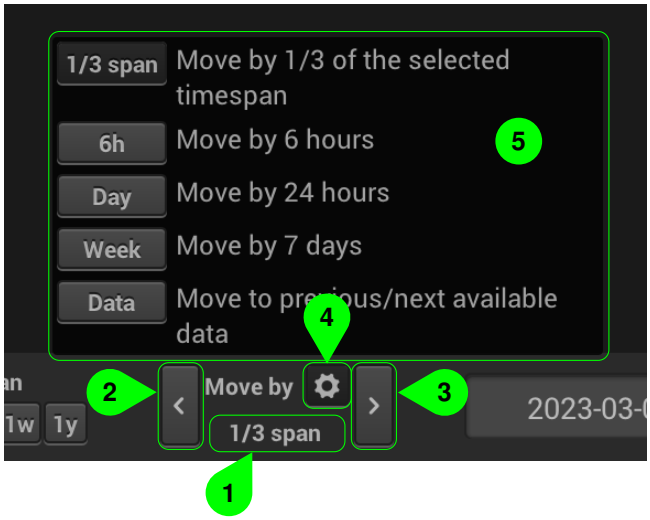
   Suggestions written in orange are years/months/days for which there is at least one granule of each selected collection available (temporal colocation).

3.2 Timespan



A timespan is a (potentially asymmetric) repartition of time around a reference datetime.
The viewer applies the selected timespan to the current datetime to compute the time window granules must intersect in order to be displayed.

3.3 Browsing



1. Timestep that will be used when navigating in time using the previous (2) and next (3) buttons.
2. Button to shift the current datetime in the past by one timestep.
3. Button to shift the current datetime in the future by one timestep.
4. Button for toggling the timestep selection panel (5).
5. Panel for selecting the timestep.

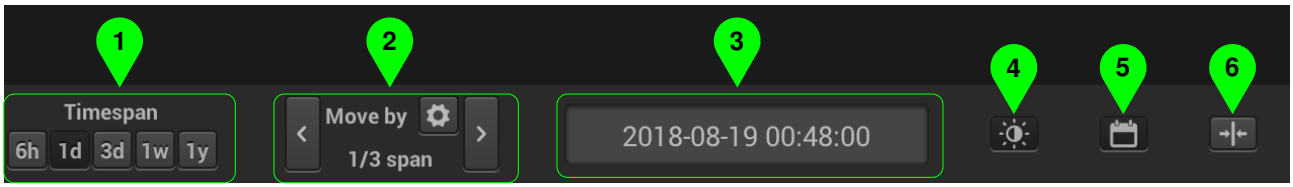
3.4 Colocation

Using the temporal coverage of the indexed granules, SEAScope is able to identify and highlight years, months and days within which all the variables selected in the catalogue are available.

This colocation is only temporal, no test is performed on the spatial coverage of the granules. It also uses temporal availability only at a daily resolution, so it cannot replace a colocation tool supporting finer resolutions and more flexible criteria, but it is still quite useful to quickly identify dates that show potential for further study.

Dates matching the colocation criteria are displayed in green in the timeline.

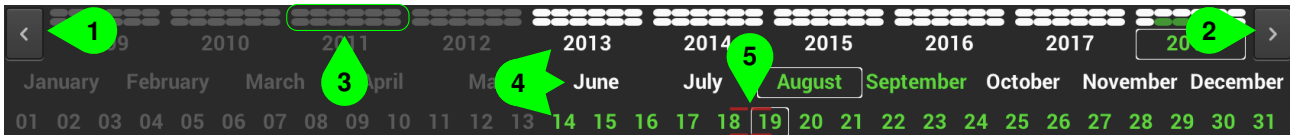
3.5 Minimal view



1. Controls the size of the time window granules must intersect in order to be displayed.
2. Allows temporal navigation in increments.
3. Datetime the timeline control considers as «current».
4. «Nearest» mode: when this button is pressed, the viewer centers the timeline on the datetime of the closest granule.
5. This button toggles the calendars view.
6. This button only appears when the year of the current datetime is not visible in the calendars view. Clicking on this button forces the calendars view to display a years range containing the current datetime.

3.6 Calendars view

The calendars view is an extension of the timeline control which provides an easy way to select a date. It also displays information about data availability.

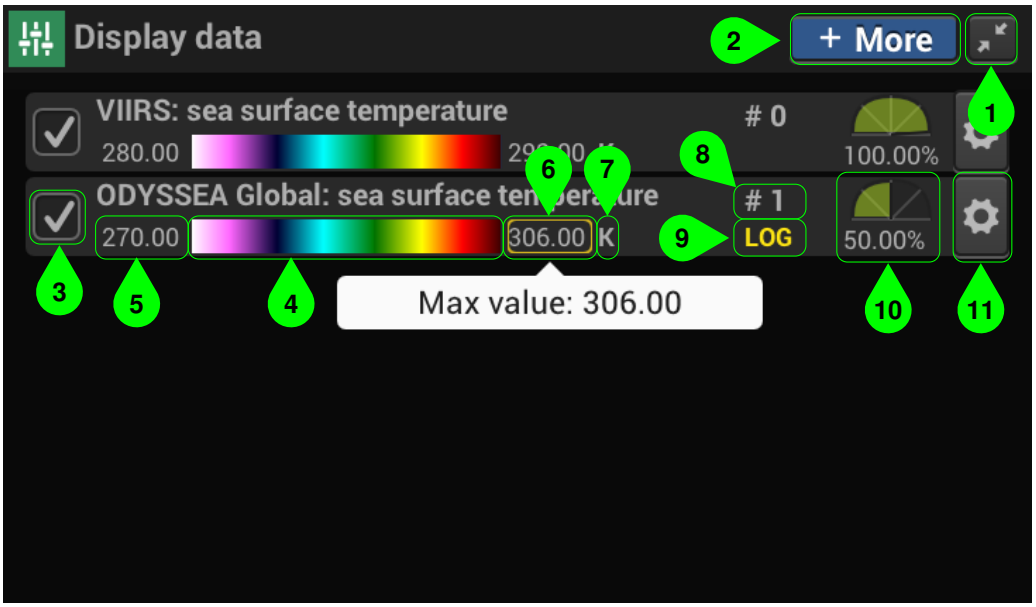


1. Display previous years range.
2. Display next years range.
3. Monthly data availability: a gray dot means that the associated month does not contain any suitable data, a white dot means that it contains data for at least one of the selected collections and a green dot means that there is at least one granule of each selected collection available during the associated month. Clicking on a dot centers the timeline on the associated month.
4. Year, month and day selection buttons. These buttons use the same color scheme as the monthly availability dots.
5. The red lines show the time window granules must overlap for the viewer to display them on the 3D map.

Rendering configuration

4.1 Condensed view

The rendering menu is a draggable window which contains one rendering control for each variable selected in the catalogue.

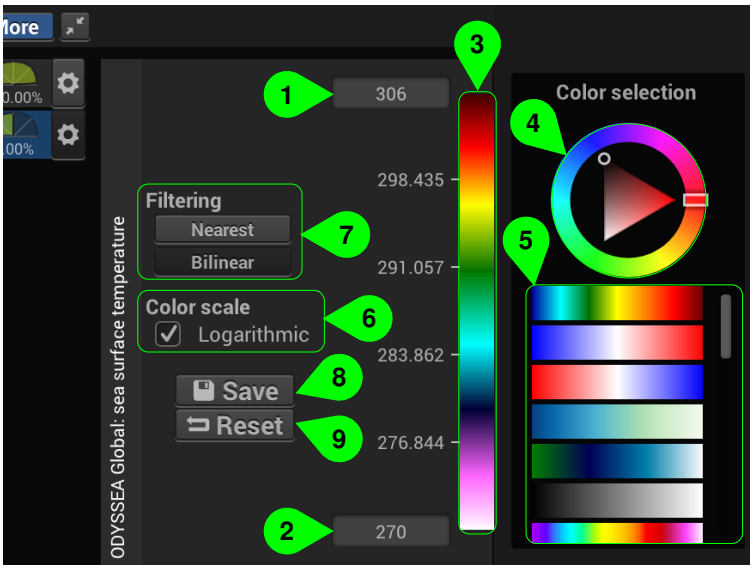


1. Minimize the rendering menu so that it looks like this: 
2. Shortcut to display the catalogue in the side menu
3. Visibility checkbox: if this box is unchecked, the associated data will be ignored when searching for granules to display.
4. The colormap used to render the variable.
5. Value associated with the first color of the colormap (min). Using the mouse wheel when the cursor is over this field changes the value.
6. Value associated with the last color of the colormap (max). Using the mouse wheel when the cursor is over this field changes the value.
7. Units of the variable.
8. Number of granules that contain the variable and are currently displayed on the 3D map.
9. The «LOG» flag is only visible if the mapping between data values and colors uses a logarithmic scale.
10. Opacity gauge: using the mouse wheel when the cursor is over this field changes the value.
11. Clicking on this button toggles the advanced rendering control.

Note that you can reorder the list elements using drag'n'drop: the order of the list is also the order in which data are drawn on the 3D map. Meaning that if a variable «A» is above another variable «B» in the list, data from variable «A» will be displayed on top of data from variable «B» on the map.

4.2 Advanced rendering control

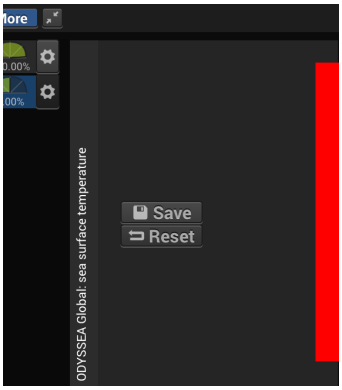
The advanced rendering menu provides a finer control over the rendering parameters of a given variable.



1. Value associated with the last color of the colormap (max). Using the mouse wheel when the cursor is over this field changes the value.
2. Value associated with the first color of the colormap (min). Using the mouse wheel when the cursor is over this field changes the value.
3. Colormap: clicking on the upper half changes the max value while clicking on the lower half changes the min value. Using drag'n'drop changes the value depending on mouse movements. Right-clicking on the colormap opens the color selection panel.
4. The color wheel can be used to select a uniform color.
5. Select the colormap to use for the selected variable.
6. Check this box if you want to map the data values range to the colormap using a logarithmic scale.
7. If the «Nearest» button is selected, pixels will be displayed using the color of the nearest data value. If the «Bilinear» button is selected, the color of the pixel will be the result of a bilinear interpolation filter applied on the surrounding data values.
8. Save button writes the rendering parameters to the configuration file of the collection. Not shown for volatile data (not relevant) nor for annotations (changes saved automatically).
9. Reset button re-initializes the rendering parameters using the values defined in the configuration file of the collection. Not shown for volatile data nor for annotations.

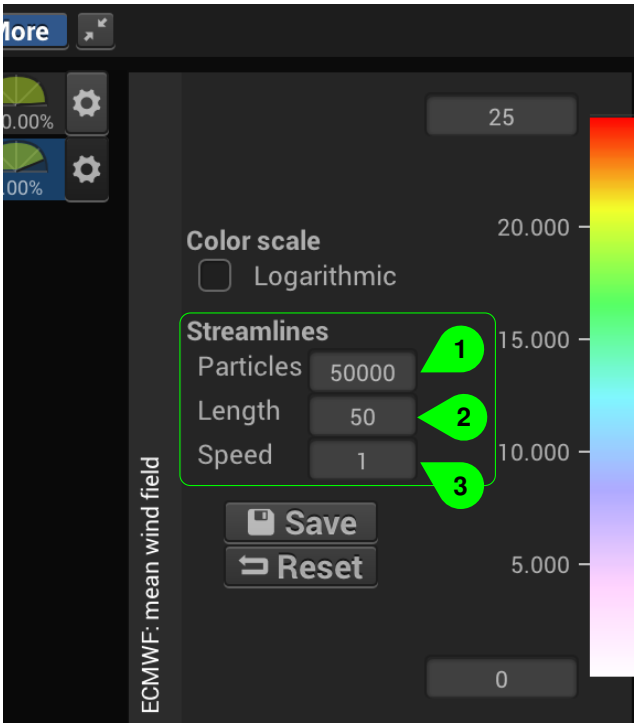
When a uniform color is selected instead of a colormap, elements that became irrelevant are hidden:

- the «Filtering» (7) section
- the «Color scale» (6) section
- the min (1) and max (2) fields
- the ticks for intermediate values



Streamlines settings

Rendering parameters for streamlines can be set in the «Streamlines» section.

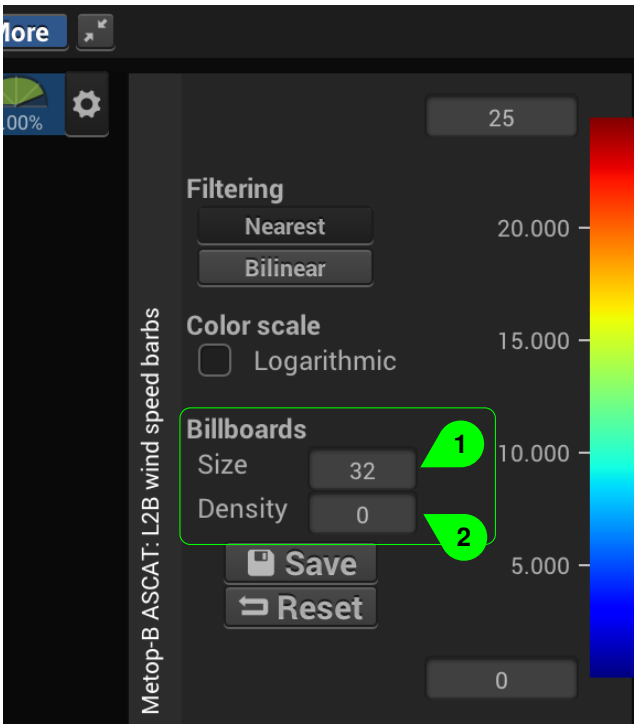


1. Size of the pool of animated particles, controlling their density. Increasing the value adds a toll on the GPU and may significantly reduce fluidity. Value must be 1000 or greater.
2. Number of segments of a streamline. Value must be 1 or greater.
3. Factor applied to the default speed of streamline particles: a value above 1.0 increases the speed of particles whereas a value between 0.0 and 1.0 slows down particles movement. Value must be 0.1 or greater.

Note that the «Filtering» section doesn't apply to streamlines and is therefore hidden.

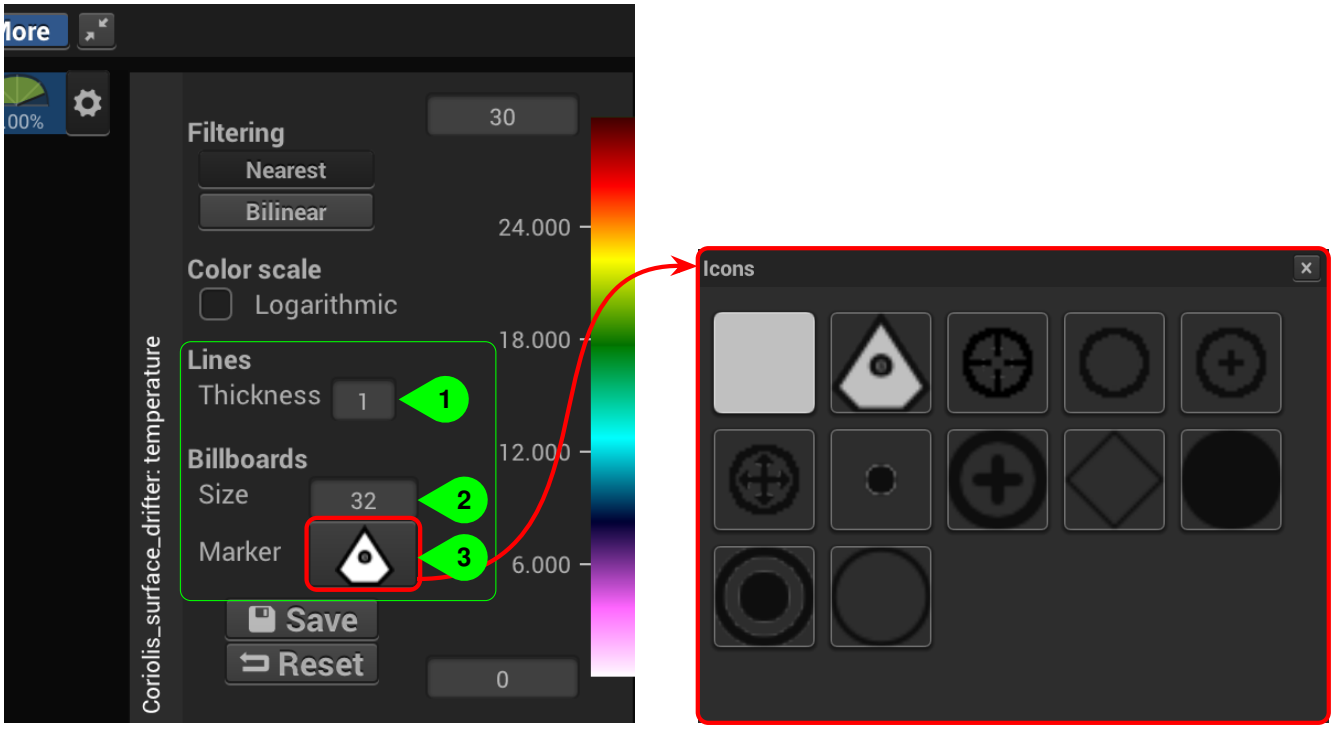
Barbs and arrows settings

Rendering parameters for barbs and arrows can be set in the «Billboards» section.



1. The size (ie. width and height) of each symbol (in pixels). Value must be 5 or greater.
2. Controls the density of the symbols: a value of 0 corresponds to a maximum spacing between symbols, whereas a value of 1 corresponds to a minimum spacing. Value must be between 0 and 1.

Trajectory settings



Rendering parameters for trajectories can be set in the «Lines» and «Billboards» sections.

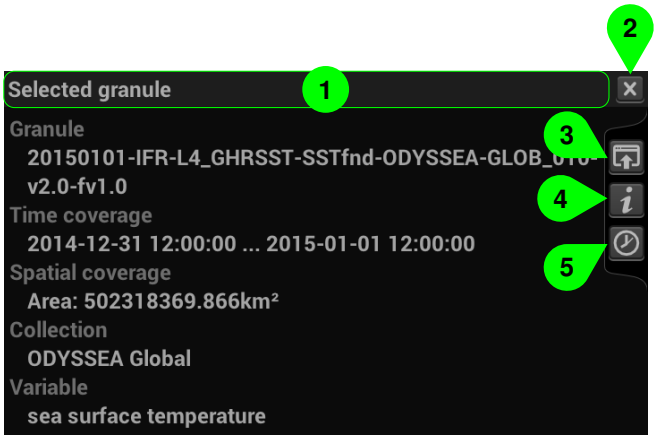
1. Thickness of the lines (in pixels). Value must be 1 or greater.
2. The size (ie. width and height) of the marker (in pixels). Value must be 5 or greater.
3. The marker used to indicate the position at the current datetime.

The marker can be changed by clicking on (3), which opens a dialog with all the available marker to choose from. Clicking on one of them, sets it as the marker of the trajectories.

Granule interactions

5.1 Selection

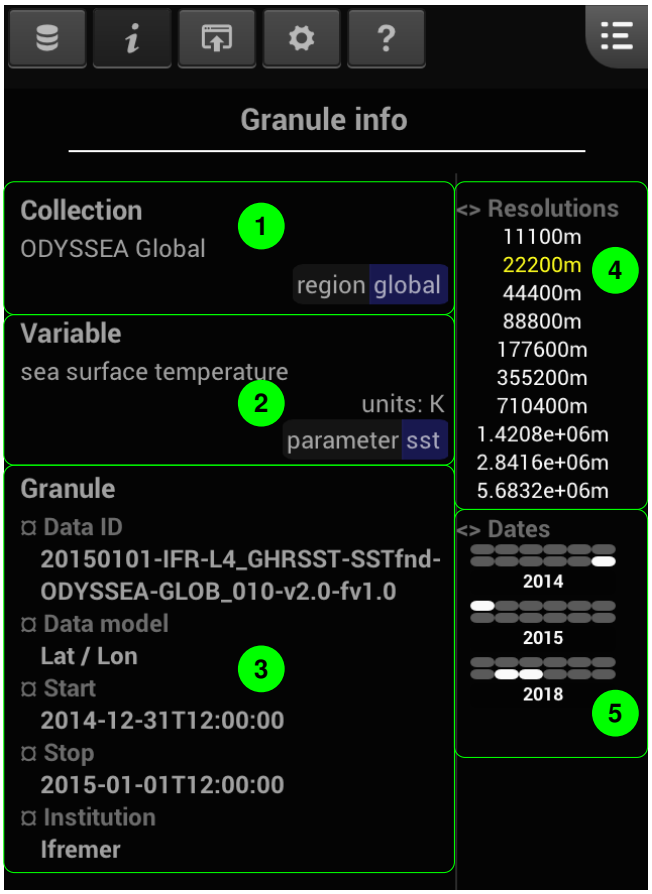
If you click on a granule (except ones rendered as streamlines, arrows or barbs), a contextual menu will appear:



1. Use drag'n'drop on the header to move the menu on the screen.
2. This button closes the menu. Pressing the Escape key has the same effect.
3. This button performs an extraction based on the granule's footprint.
4. This button displays information about the granule in the side menu.
5. The clock button centers the timeline on the granule's datetime.

5.2 Granule information

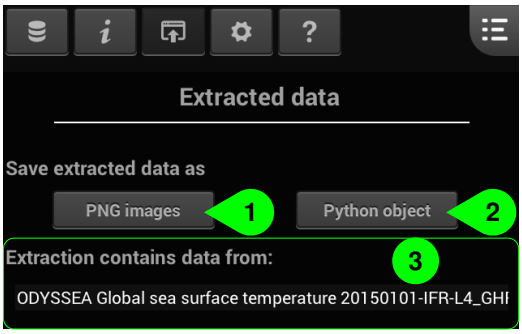
If you click on the information button of the contextual menu which pops after selecting a granule on the 3D map, you should see a new element in the side menu:



1. Collection details.
2. Variable details.
3. Granule details extracted from the attributes of the NetCDF files (cf. IDF specifications).
4. Available resolutions for the granule. The resolution written in yellow is the one used by the viewer when you clicked on the information button.
5. Years and months when the collection has granules. Clicking on a dot centers the timeline on the associated month and year.

5.3 Extraction and transect dialogs

The following dialog appears when you click on the Extract button on the contextual menu which pops after selecting a granule on the 3D map.



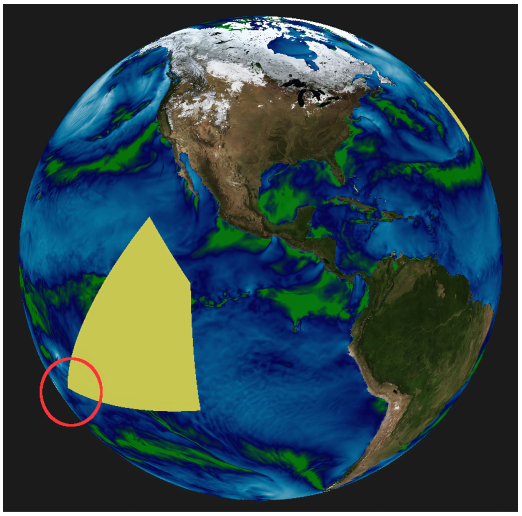
1. Save extracted data as PNG images.
2. Save extracted data as Python objects (pickle format).
3. List of granules from which data have been extracted.

SEAScope extracts data from the pixels projected on the computer screen, not from the original data matrices.
It allows SEAScope to perform the coregistration between all the data "for free" as this operation is already performed for rendering values on the screen, but it has two very important caveats:

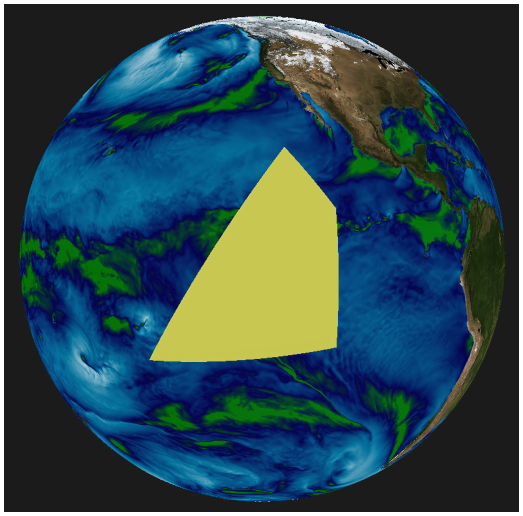
- due to the curvature of the globe and to the geometry of the screen, extracted pixels cannot have the same spatial resolution, especially when zoomed-out.
- by zooming-in it is possible to extract data at a higher resolution than the native resolution of the granules, which can lead to the creation of artefacts in downstream analyses.

  Depending on the filtering mode chosen in the rendering options, the value of extra pixels is computed automatically by the graphics card using either the value of the nearest pixel or the result of a bilinear interpolation.

It is also important to note that SEAScope includes geolocation data with the extraction results, but it might fail to compute some coordinates when zoomed-out if the footprint on which the extraction is performed has vertices close to the globe limits.
If geolocation cannot be computed, SEAScope will refuse to extract data so it is recommended to move the footprint as close as possible to the center of the screen before requesting an extraction.
For the same reason, selecting a granule with global coverage and using its footprint for extracting data will only work if the globe's limit are out of the screen.



Here the extraction will fail due to the vertex too close to the globe's limit...



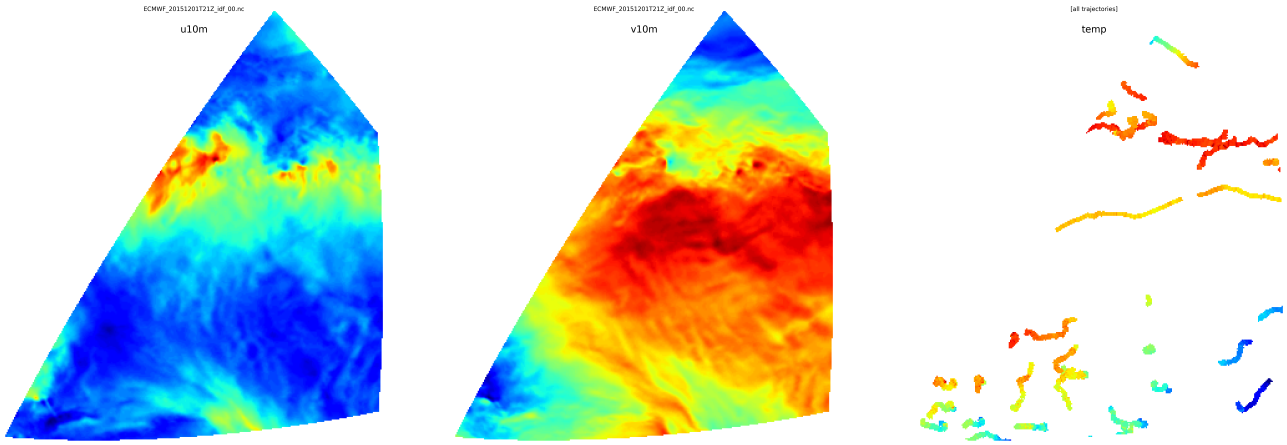
... but it will work with the very same footprint if the globe is rotated so that vertices are as far as possible from the globe's limits.

The extraction results are not exploited directly by SEAScope, they are stored in a temporary buffer that can be accessed using the API (Python bindings). This buffer is overwritten each time a new extraction is performed.

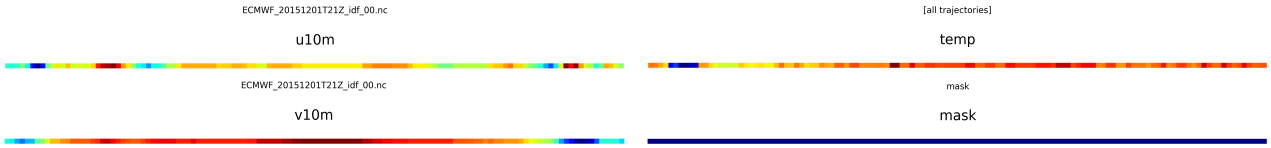
When it is installed with the `[processor]` option, the pySEAScope package includes not only the Python bindings for SEAScope but also a small command named `seascope-processor` meant to be a demonstration of how SEAScope capabilities can be extended using the API.

The "PNG images" (1) and "Python object" (2) buttons will only appear if the SEAScope processor option is enabled in the configuration file (`enabled = true` under the `[processor]` section). These buttons will do nothing unless the `seascope-processor` command is running in another terminal.

The "PNG images" button (1) will save one plot per NetCDF variable used to render data on the globe, for each granule that intersects the extraction area. These plots will use matplotlib's jet colormap, on a linear scale defined between the min and max values of the extracted data. Collections containing trajectories are treated differently: there is a single plot for all the trajectories that intersect the extraction area.



If the extraction footprint is a trajectory / polyline, then the plots will use the distance from the origin of the line as horizontal axis. One additional plot will be included to provide information on the masked portions of the footprint.



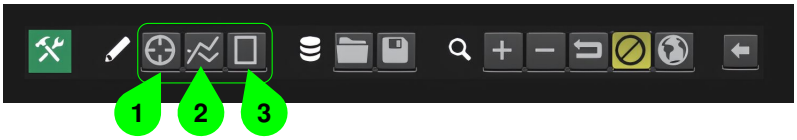
The "Python object" button (2) will convert the extraction results into a single Python dictionary and serialize it using the `pickle` module.

The result can then be loaded in Python with three lines of code:

```
import pickle

with open('path_of_the_file.pyo', 'rb') as f:
    extracted_data = pickle.load(f)
```

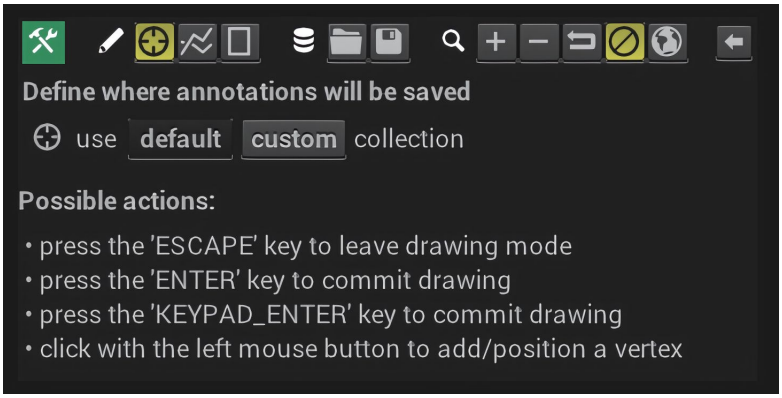
Annotations

6.1 Create an annotation



1. Create a point.
2. Create a polyline.
3. Create a polygon.

Clicking on any of the three buttons opens the same dialog and launches the creation mode for the selected annotation type.



This dialog box lets you define where to save the new annotation and summarizes the keyboard shortcuts.

Let's keep the default collection and look at the procedure for creating an annotation.

Create a point

Left-click on the globe to position the point. A green circle appears. Left-click elsewhere on the globe to move the point.

Confirm the location by pressing Enter or Enter on the numeric keypad. The new point has now been added to the variable and you leave point creation mode. To add a new point, repeat the procedure.

To cancel point creation, press the Escape key. The current point will be deleted and you will leave point creation mode.

Create a polyline

Draw the polyline by left-clicking on the globe.

Each click adds a point to the polyline. Once the polyline is complete, validate it by pressing Enter or Enter on the numeric keypad.

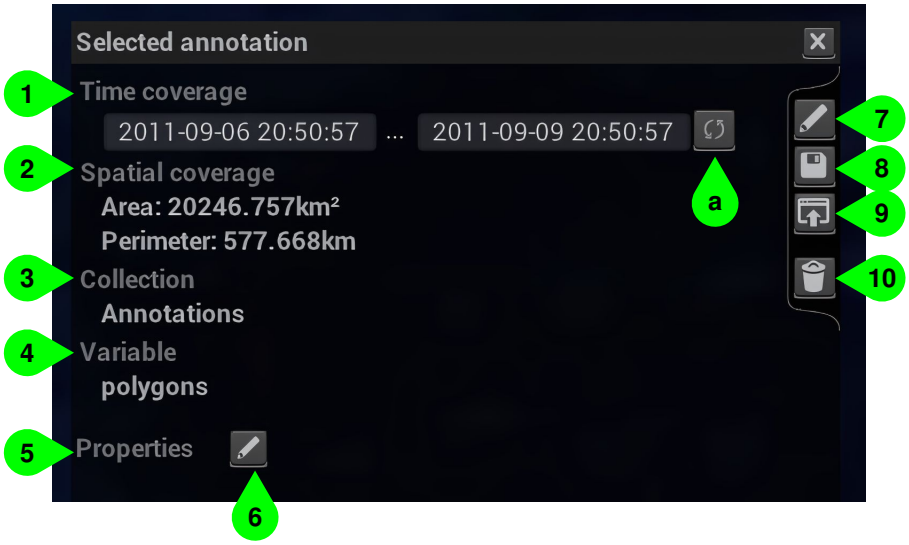
To cancel polyline creation, press the Escape key. The current creation will be deleted and you will exit polyline creation mode.

Create a polygon

Follow the same procedure as for polyline. Note that SEAScope will not allow the creation of a polygon whose sides intersect each other.

6.2 Annotation properties

Left-clicking on an annotation displays a properties dialog. The dialog is the same for all annotations.



1. The time coverage of the annotation. The shape will only be displayed if the current time is within this time range. Fields can be modified but changes will only be saved and applied when clicking on the update button (a).
2. Spatial coverage information.
3. The name of the collection to which the annotation belongs.
4. The name of the variable to which the annotation belongs.
5. List of properties.
6. Button to edit properties.
7. Button to edit the annotation on the globe.
8. Button to save the annotation as a JSON or WKT file.
9. Button to extract data that intersect the shape of the annotation and opens a dialog in the side menu.
10. Button to delete the annotation.

Create properties

Click on the edit properties button to open the properties editor.

1. To add a property, first select its type:

   bool: the property will have a checkbox to represent its state.

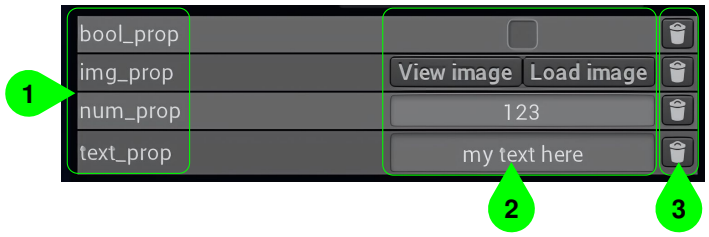
   dict: create a sub-category in the list of properties.

img: an image which can be loaded in jpg, png, gif, tga, bmp and ppm formats.
num: a property with a numerical value.
text: a property with a text field.

2. Then enter a name.
3. Finally click to the plus button to validate the new property or click on the red cross button to reset the dialog.
4. Button to save the changes and exit the properties edition mode.
5. Button to discard the changes and exit the properties edition mode.

Modify properties

Once created the values assigned to the properties can be modified:



1. Name of the property.
2. Editable fields.
3. Delete buttons.
   Deletes the corresponding property.

When leaving the properties edition mode, controls for adding/removing/modifying properties are hidden:

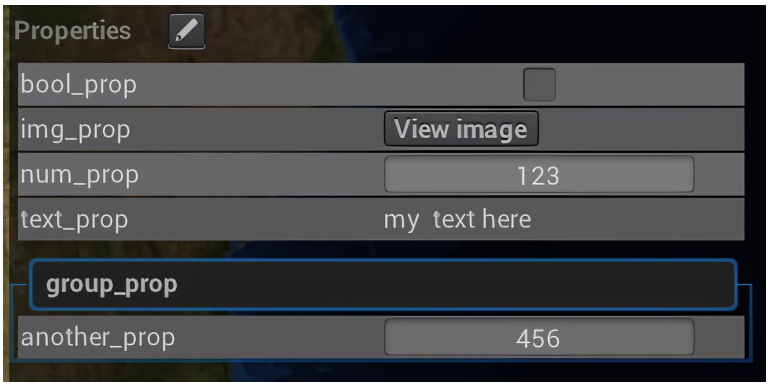


Image property

The image property is different from the other property types since it has no editable field, but two buttons:

1. View image: opens the image in a new dialog box.
2. Load image: opens a new dialog box with controls to select an image file:

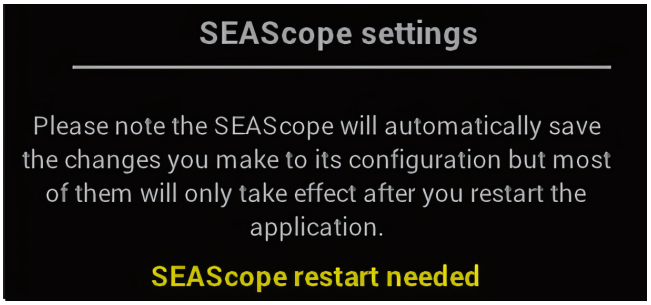
(a) Open a dialog box for selecting an image file.

(b) Load the image file.

(c) Close the dialog.

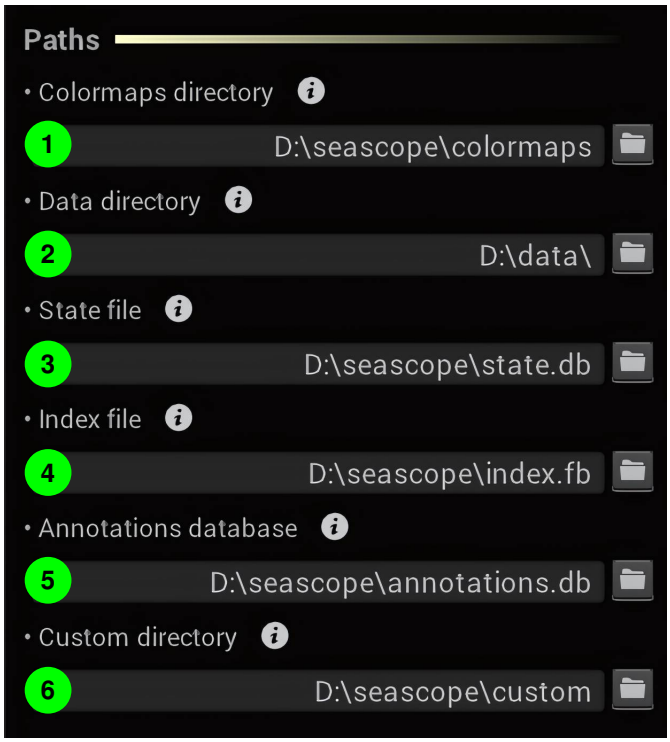
Settings

The settings panel in the side-menu provides access to the configuration of SEAScope.
Changes to these settings are saved immediately but in most cases they will only be applied after closing and restarting the application. SEAScope will display a notification in yellow when changes will only take effect after a restart:



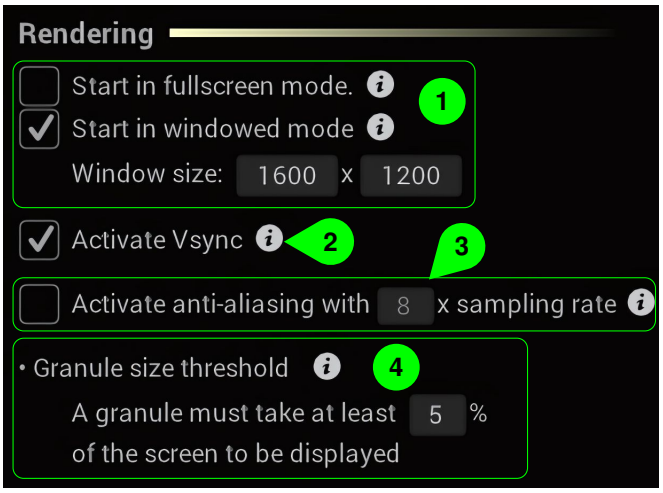
The settings tab contains several subsections: Paths, Rendering, Timeline, Misc, Drawing and Processor.

7.1 Paths



1. Path of the folder containing the colormaps to be used in false color rendering.
2. Path of the folder containing the data to be displayed in SEAScope.
3. Path of the file where SEAScope will save its state when the application shuts down. The file will be created automatically but the parent directory must already exist.
4. Path of the file where SEAScope will store the index of all the granules found in the data directory. SEAScope will create the file but the parent directory must already exist.
5. Path of the file where SEAScope will store your annotations. SEAScope will create the file but the parent directory must already exist.
6. Directory containing custom resources loaded by SEAScope (like icons). The path must be an existing directory.

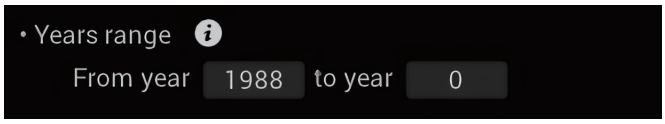
7.2 Rendering



The following section deals with window settings and 3D rendering.

1. You can configure SEAScope to run in full-screen or windowed mode, specifying the window size in pixels.
2. Vertical synchronization can be enabled or disabled. This means that 3D rendering will be synchronized or not with the screen refresh rate. For example, if the screen is at 60Hz, SEAScope will not exceed 60 frames per second if synchronization is enabled. Otherwise, there will be no limit to rendering and the number of frames per second will depend on the power of the computer.
3. Anti-aliasing can be enabled or disabled. Quality is adjustable: valid values are 2, 4 and 8.
4. You can define the minimum size of a granule before it is displayed. If its size is at least equal to X% of the screen size, it will be displayed.

7.3 Timeline



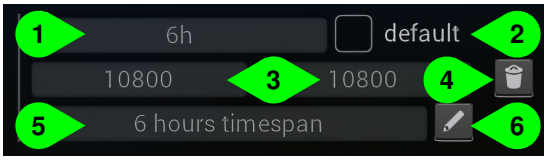
You can set the range of years displayed in the timeline. Granules with years outside this range will not be displayed. If the end year is zero, SEAScope will use the current year.
The minimum year is 1970 and the maximum year is 3000. The minimum valid date is January 1, 1970 at 0:00:00 and the maximum valid date is December 31, 3000 at 23:59:59.

Time spans

In the timeline the "timespan" corresponds to the period of time around the current time when the granules will be displayed on the globe. By default, SEAScope displays four different time periods centered on the current time: six hours, one day, three days and one week.

Time spans edition

The list of time spans can be edited. Each time span has an editing area as shown in the following image.

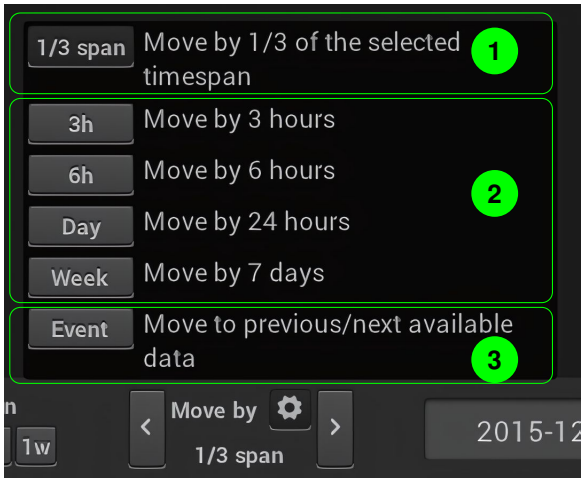


1. Text to be displayed on timeline button.
2. Use this time span as the default when you open SEAScope if the state.db file does not exist. The current state of the interface is saved and restored the next time SEAScope is opened, including the time span.
3. The number of seconds before and after the current time.
4. Delete time span.
5. Button tooltip text in the timeline.
6. Edit the time span. Fields will become editable. Confirm modifications with the Ok button.

Time steps

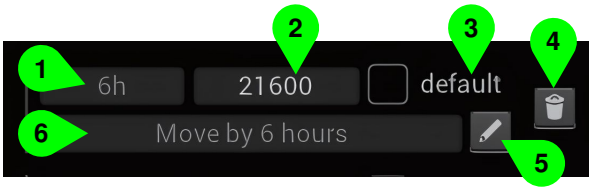
Section 3.3 of the manual describes time navigation and time step selection. Now let's see how to edit the time step list.

Time steps in the timeline



1. Not editable. Move by a third of the selected timespan.
2. Editable list. All timespans with a fixed time can be edited.
3. Editable. Move to previous / next available data. It's a special case when the time step equal zero.

Edit time steps



1. Text to be displayed on timeline button.

2. Time in seconds. If the value is zero, there will be no time shift. The timeline will move to previous / next available data.

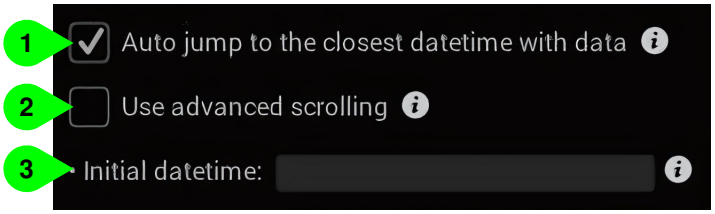
3. Use this time step as the default when you open SEAScope if the state.db file does not exist. The current state of the interface is saved and restored the next time SEAScope is opened, including the time step.

4. Delete the time step.

5. Edit the time step. Fields will become editable. Confirm modifications with the Ok button.

6. Button tooltip text in the timeline.

Other parameters



1. Activating this checkbox will cause SEAScope to automatically go to dates where there is data when:

   - selecting a variable from the catalog, if there is no data at the current time.

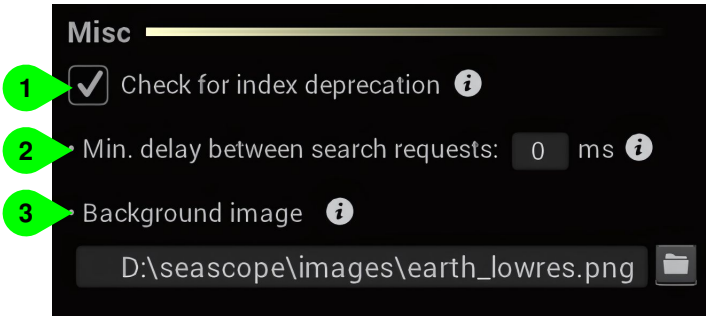
   - deselecting a variable from the catalog, if there is no data at the current time and there is at least one other variable selected.

   - clicking on a year or month in the timeline that contains data.

2. When this parameter is disabled, the mouse wheel scrolls the current time when the mouse cursor is over the timeline. The offset per scroll wheel step is equal to a third of the timespan. When this parameter is enabled, the behavior is the same when the cursor is over the days, but when the cursor is over the months/years the wheel shifts the current time by one month/year per wheel step.

3. This field defines the date and time used by SEAScope at its first launch when the state.db file does not exist. If this field is left empty, SEAScope will use the current time. After the first launch, SEAScope saves the time used at closure in state.db and reuses it the next time it is opened. The initial datetime field is no longer used.

7.4 Misc

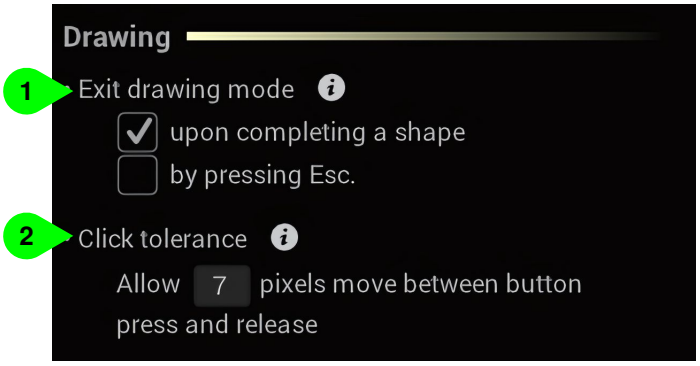


1. If the checkbox is checked, at launch, SEAScope tries to detect whether there have been any changes in the data directory. If so, SEAScope displays a message asking whether or not to update the index.

2. As soon as you change the date, timespan, zoom or catalog variables, SEAScope performs queries to find out what to display on the globe. Applying a delay between each query can, in some cases, avoid overloading SEAScope. The delay time is expressed in milliseconds.

3. The path to the image used for the background map. The image can be in png, jpg, bmp or tga format. To be rendered correctly, the image must have a 2:1 ratio, i.e. its width must be twice its height. To be displayed, the image must be smaller (in width and height) than the maximum texture size supported by your graphics card. At the date of this manual, the minimum size supported is generally 8096 pixels, rising to 32384 pixels on the latest graphics cards. By default SEAScope use the worldmap earth_lowres.png which has a size of 4096x2048 pixels.

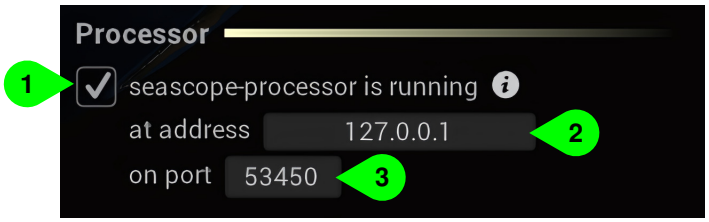
7.5 Drawing



1. Allows you to configure the annotations drawing mode, either in single mode or allowing drawing several annotations in succession. In single mode (the default), drawing mode exits once the annotation is validated. In the other mode, validating a annotation with the Enter key keeps SEAScope in drawing mode, which allows you to draw several in succession. To exit the drawing mode, press the Escape key.

2. This parameter is used to set the pixels tolerance allowed for a click to be considered valid. In other words, the number of pixels between the cursor position when the button is pressed and the position when it is released.

7.6 Processor



1. Check to activate functionalities related to the `seascope-processor` command ("PNG images" and "Python object" buttons available when extracting data)

2. Address on which the `seascope-processor` command listens to. It must be an IPv4 address, or a computer name that the operating system can turn into an IPV4 address using a DNS (Domain Name System) resolver.

   If the viewer and `seascope-processor` are running on the same computer, then the default value `127.0.0.1` should be used.

3. Port on which the `seascope-processor` command listens to. It must be an integer value, the default is `53450`.

Key concepts

SEAScope's data management is based on three concepts: granules, collections and variables.

Understanding these concepts and how they relate to each other is important to grasp the inner workings of the viewer.

8.1 Granule

> A granule is a set of data fields that share the same dimensions, as well as the same spatial and temporal coverage.

For example, several spectral bands measured simultaneously by the same instrument onboard a satellite would be stored as data fields in a single granule.

Or multiple parameters computed by a numerical model on the same grid for a specific time step.

One file, several granules
Be aware that data may regrouped in single file and yet belong to different granules.

For example, a file may contain data from multiple instruments, or instruments with multiple modes of measurements: in that case each tuple of (spatial coverage, spatial resolution, temporal coverage) should be a separate granule.

Another case would be a model or an analysis output file containing a timeseries of 2D grids: for these there is actually one granule for each time step.

One granule, several files
The Intermediate Data Format (IDF) has been designed according to SEAScope's philosophy: one IDF file only contains a single granule, with a convention on dimension names so that data structures can be quickly identified.

SEAScope also supports granules available in different resolutions (to avoid Moiré effects and wasting memory when zoomed out), which is not possible with a single IDF file.

The IDF specifications include conventions to let SEAScope know when several IDF files actually contain a variation of the same granule with a different degree of downscaling, so the viewer can load the file whose spatial resolution is best suited for the current zoom level in the 3D map.

8.2 Collection

> A collection is a set of granules that contain the same data fields (even if they are filled with masked values) and the same set of dimensions.

A collection defines characteristics that are common to all the granules it contains, e.g. whether they should be displayed as soon a their temporal coverage intersects SEAScope's time window (generally good for satellite swaths) or only if SEAScope's current datetime is within their temporal coverage (more suitable for global data, analyses over an area, or geostationary satellite acquisitions).

If some granules have data fields with no usable values (e.g. a satellite acquisition with an optical sensor completely filled with clouds), these data fields must still be defined in the granule because the structure of all the granules within a collection must be homogeneous.

Even if all the granules contain the same data fields, these data fields must be defined on the same dimensions: the size of the dimensions may vary from granule to granule, but it is not possible to have a 1D granule and a 2D granule in the same collection.

8.3 Variable

> A variable is the association of a set of data field names with information on how SEAScope should assemble and display these fields on the 3D map.

A variable is attached to a single collection: SEAScope's catalogue displays the collection's label followed by one labelled checkbox for each variable associated with that collection.

When a variable is selected in the catalogue, SEAScope first identifies the attached collection, then searches the granules owned by that collection that satisfy the criteria for appearing on the 3D map.

For each matching granule, the viewer reads the data fields whose names are listed by the variable (hence the importance of having an homogeneous collection).

The data fields and geolocation data are then transformed in a form that is suitable for the graphics card (GPU) and for the rendering settings defined in the variable.

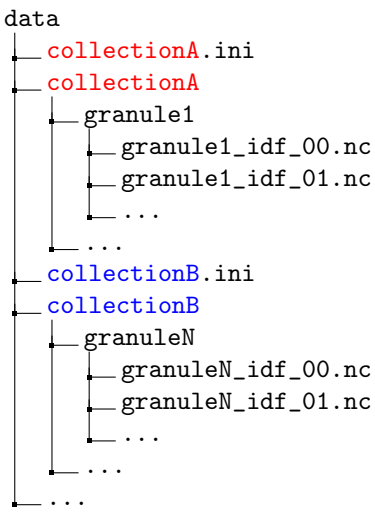
Collections configuration

9.1 Configuration file

For each data collection, SEAScope expects the data directory to contain:

- one sub-directory containing one or several granules
- a configuration file defining how SEAScope should display the granules that belong to the collection

The collection configuration file must have the same name as the sub-directory that contains the granules, with a `.ini` extension.



Without the configuration file SEAScope will not detect the collection and will ignore its granules.

Please note that SEAScope will not detect the availability of new granules or new collections unless you restart the application!

Upon restart the application should detect that new granules/collections are available and display a prompt to ask if the index should be rebuilt. Agree and SEAScope will update its index with the new data: the new collections and granules should be available for visualisation.

9.2 Configuration file template

Configuration files for collections use the `INI` format, i.e. each line of the file must contain either a section name between square brackets (`[section]`) or a setting defined using the `key = value` syntax.

The configuration file has a mandatory section called `[general]` and one additional section for each variable that should appear in the data catalogue of the viewer.
An optional section named `[reader]` is also supported for collections composed of NetCDF files containing data geolocated by a regular lat/lon grid, i.e. granules not stored as IDF files.

An incomplete configuration file can cause errors and prevent SEAScope from completing the indexation process, so please make sure that your configuration files have all the settings described below, even if the associated value remains empty.

Users are strongly advised to copy/paste and adapt the following template when creating a configuration file for a new collection:

```
[general]
label = Collection label
xSeamless = false
ySeamless = false
mustBeCurrent = false
tags =
variables = VARID_N

[VARID_N]
label = Variable label
fields =
units =
defaultRendering = RASTER
opacity = 1.0
zindex = 0.8
tags = parameter:unknown
filterMode = NEAREST

; Colorscale
min = 0.0
max = 100.0
logscale = false
colormap = jet
color = 255,128,128

; Streamlines
particlesCount = 5000
streamlineLength = 50
streamlineSpeed = 1.0

; Trajectories
lineThickness = 2
currentTimeTrajectoryMarker =

; Barbs and arrows
billboardsDensity = 0.3

; Trajectories, barbs and arrows
billboardsSize = 24
```

9.3 List of settings

Supported settings are described hereafter.

[general] section

label

Name used in the viewer to designate the collection.

xSeamless

Tell the viewer that the granules of the collection have a 360° longitudinal cover. Should be a boolean value (true|false).

ySeamless

Tell the viewer that the granules of the collection have a 180° latitudinal cover. Should be a boolean value (true|false).

mustBeCurrent

This setting controls the viewer behavior when it searches for granules to display and uses temporal constraints to filter the results. Should be a boolean value (true|false).

When this setting is set to true, the viewer will ignore granules whose time range does not contain the current datetime. It is the recommended behavior for collections whose granules cover the whole globe.

When it is set to false, the viewer displays the granules of the collection as long as their time range intersects the current time window (cf. timespans in the list of settings for the application configuration file). Use this behavior for collections of granules with a smaller spatial footprint or a sparse temporal coverage.

tags

Comma-separated list of tags associated with the collection, shown in the viewer when the detailed information of a granule is displayed.

Tags are defined with a key:value syntax. For example: `tags = area:global,timeliness:nrt`

variables

Comma-separated list of identifiers for the variables which can be extracted from the granules of the collection and be displayed in the viewer.

Identifiers must be unique within a collection and can be chosen arbitrarily. In addition to the `[general]` section, the configuration file of the collection must contain one section per variable identifier in this list.

Variable sections

label

Name used in the viewer to designate the variable.

fields

Comma-separated list of NetCDF variables to extract from the IDF files in order to compute the SEAScope variable.

In some cases it may be necessary to extract more than one NetCDF variable to compute a SEAScope variable. For example, displaying vectorfield as arrows/streamlines requires u and v components of the vectors.
Another example would be a variable composed from three fields which are interpreted as R, G and B channels to create an image.

units

Units of the values contained in the variable.

defaultRendering

Rendering method used to display the variable. Must be one of the following:

- RASTER
- BARBS
- STREAMLINES
- ARROWS
- TRAJECTORIES
- RAWRGB

opacity

Opacity of the variable representation. Should be a float value between 0.0 (transparent) and 1.0 (completely opaque).

zindex

Z-indices are used to define the order in which the representations are drawn on the screen. Should be a float value.

Representations of a variable with a low z-index are drawn first and may be hidden by the representations of another variable with a higher z-index if they overlap.

You can choose an arbitrary range to define the z-indices, but z-indices must be unique within a collection AND between collections.

tags

Comma-separated list of tags associated with the variable, shown in the viewer when the detailed information of a granule is displayed.

Tags are defined with a key:value syntax. For example: `tags = parameter:temperature,quality:1_or_2`

filterMode

Method used by SEAScope to find the value associated with a pixel. Either NEAREST to use the nearest data value or BILINEAR to use the result of a bilinear interpolation applied on surrounding values.

min

Minimum value for the color palette applied to the representation of the variable. Should be a float value.

max

Maximum value for the color palette applied to the representation of the variable. Should be a float value.

logscale

Use a logarithmic scale for the colormap. Should be a boolean value (truelfalse).

Only applied if `min` is strictly greater than 0.

colormap

Identifier of the colormap applied to the representation of the variable.

A colormap identifier is only valid if there is a file in the `[paths]colormaps` directory whose name is the identifier suffixed by the `.rgb` extension.
For example, you can use:

```
colormap = rainbow
```

only if there is a file named rainbow.rgb in the `[paths]colormaps` directory.

If both a uniform color and a colormap are defined for a variable, the colormap will be used, i.e. leave this field empty if you want to use a uniform color.

color

Uniform RGB color applied to the representation of the variable. Should be a comma-separated list of three integer values in the [0, 255] range.

If both a uniform color and a colormap are defined for a variable, the colormap will be used.

particlesCount

Number of animated particles. Should be an integer value greater or equal to 1000.

This value controls the density of the particles when using the STREAMLINES rendering method. Increasing the density adds a toll on the GPU and may significantly reduce fluidity.

streamlineLength

Number of segments of a streamline. Should be an integer value greater or equal to 1.

This value controls the length of the semi-transparent tail of an animated particle. Increasing the length adds a toll on the GPU and may significantly reduce fluidity.

streamlineSpeed

Factor applied to the default speed of streamline particles. Should be a float value greater or equal to 0.1.

A value above 1.0 increases the speed of particles whereas a value between 0.1 and 1.0 slows down particles movement.

currentTimeTrajectoryMarker

Name of the icon used for representing the location for the current datetime on variables using the TRAJECTORY rendering method.
No marker is displayed for the current datetime if this field is left empty.
By default, the following icons are supported:

Names starting with an underscore _ are reserved for icons embedded in SEAScope (only the values listed above are valid).

Otherwise SEAScope will look for a user-provided icon whose name matches the value of `currentTimeTrajectoryMarker` in a subdirectory named `icons`, itself located inside the directory defined by the `[paths]custom` setting in SEAScope configuration file.

For example, with `customTimeTrajectoryMarker = another_custom_icon.png`, SEAScope will look inside the `[paths]custom` directory (assuming it is named "custom", which is the default):

```
custom
├── ...
├── icons
│   ├── my_icon.png
│   ├── another_custom_icon.png
│   └── ...
└── ...
```

lineThickness

Width (expressed in pixels) of the segments displayed on the globe for polyline annotations and variables using the TRAJECTORY rendering method. Should be a float value greater or equal to 1.0.

billboardDensity

Density of symbols drawn on the globe for variables using the BARBS or the ARROWS rendering methods. Should be a float value between 0.0 (empty space approximately equivalent to one symbol between adjacent symbols) and 1.0 (almost no space between adjacent symbols).

billboardSize

Size (expressed in pixels) of the symbols drawn on the globe for variables using the BARBS or the ARROWS rendering methods. Should be a float value greater or equal to 5.0.

[reader] section

In addition to IDF files, SEAScope natively supports NetCDF files with variables defined on a regular lat/lon grid. The structure of the files is well known for IDF, but there are multiple ways to define a regular lat/lon grid in NetCDF, so the collection configuration must describe the data layout to SEAScope.

The `[reader]` section is not required for collections of IDF files.

type

Identifier of the data type. The value must be `latlon` (other types will be added in the future).

SEAScope uses the `type` value to select a data reader compatible with the file format and structure.

timeVariable

Name of the NetCDF variable which defines the time of the granule(s) contained in the file.

The variable must have a single dimension.

latVariable

Name of the NetCDF variable which defines the major axis for geographical coordinates (latitude for regular lat/lon grids).

The variable must have a single dimension.

lonVariable

Name of the NetCDF variable which defines the minor axis for geographical coordinates (longitude for regular lat/lon grids).

The variable must have a single dimension.

outputGeolocMajorAxisSpacing

Spacing of control points along the major axis. Must be a strictly positive integer value, expressed in data cells.

When SEAScope extracts geographical coordinates from the NetCDF files, it subsamples them to create a coarse grid that is easier to manipulate and takes less memory. The points contained in the coarse grid are named "ground control points" (GCP) and are used as anchors to map the data matrix on the globe. By interpolating between GCPs, SEAScope can rebuild the coordinates that have been pruned by the subsampling operation.

outputGeolocMinorAxisSpacing

Spacing of control points along the minor axis. Must be a strictly positive integer value, expressed in data cells.

When SEAScope extracts geographical coordinates from the NetCDF files, it subsamples them to create a coarse grid that is easier to manipulate and takes less memory. The points contained in the coarse grid are named "ground control points" (GCP) and are used as anchors to map the data matrix on the globe. By interpolating between GCPs, SEAScope can rebuild the coordinates that have been pruned by the subsampling operation.

relativeTimeCoverageStart

Temporal offset between the datetime of the granule (read from the `timeVariable` variable) and the beginning of its temporal coverage. Must be a number of seconds expressed as a signed integer value.

The offset is added to the time value read in the NetCDF file, so the value of `relativeTimeCoverageStart` should probably be negative.

relativeTimeCoverageEnd

Temporal offset between the datetime of the granule (read from the `timeVariable` variable) and the end of its temporal coverage. Must be a number of seconds expressed as a signed integer value.

The offset is added to the time value read in the NetCDF file, so the value of `relativeTimeCoverageEnd` should probably be positive.

depthVariable

Name of the NetCDF variable which contains values for the depth dimension.

The variable must have a single dimension.

The variable values must be sorted in ascending order.

depthValue

Depth value associated with the collection. Should be a float value.

For NetCDF files that contain multiple 2D grids of data at different depths, one collection must be defined for each depth that should appear in SEAScope (a collection can only be associated with one depth value).

SEAScope will look in the variable identified by `depthVariable` to find the value closest to `depthValue`. If `depthValue` is at equal distance from two values, the bigger one is selected.

Adding data

10.1 Getting IDF files

The IDF Data page on the SEAScope website offers examples of data from various sources converted to IDF.
All IDF samples are available in both zip and tar.gz format, so simply choose the format that is more convenient for you.

Bundles

Two data bundles are available:

- A large bundle containing a large variety of data converted in IDF format: zip or tar.gz (approx. 860MiB)
- A subset of the large bundle without the heavier files, in case your Internet connection is limited: zip or tar.gz (approx. 105MiB)

Samples repository

The content of the large IDF data bundle has been split into separate archives for each collection, each archive including the collection configuration file and a directory containing the IDF granules:

`https://seascope.oceandatalab.com/data/idf_samples/collections/`

Case studies

Some samples have been selected to illustrate how SEAScope can be used for specific case studies:

- **Agulhas current**: zip or tar.gz
  Snapshots of Hi-Res Sentinel1 sea surface roughness modulation by the Agulhas current together with 1km IR SST and Chlorophyll concentration data from MODIS and VIIRS as well as regional ODYSSEA SST with Globcurrent geostrophic current and Jason 2 Sea Level Anomalies.
- **Global circulation**: zip or tar.gz
  One month (Dec 2015) of global Odyssea SST with surface current from Globcurrent and OSCAR, along track Sea Level Anomalies from Jason2 altimeter together with Mean Dynamic Topography and in-situ SVP drifters at 15m depth.
- **Ocean wind**: zip or tar.gz
  One day (1st Dec 2015) of ASCAT A/B 10m winds with additional ECMWF background.
- **Brittany tidal current**: zip or tar.gz
  Snapshot of clear sky Sentinel2 on 14th Feb 2017 over strong tidal currents in Britanny (western tip of France) with colocated Sentinel1 sea surface roughness.
- **Gibraltar strait**: zip or tar.gz
  Snapshot of clear sky Sentinel2 on 12th Jan 2017 over Gibraltar strait with strong currents, internal waves and ship wake signatures.
- **Gulf of Lion**: zip or tar.gz
  Snapshot of clear sky Sentinel2 on 3rd Jan 2017 over the Gulf of Lion with coastal submesocale signatures and wave breaking, with colocated Sentinel1 sea surface roughness.
- **Polar**: zip or tar.gz
  Wave propagation in sea ice with Sentinel1 TOPS modes along with WW3 sea state model wave partitions and medium resolution sea ice information from ASCAT scatterometer and AMSR radiometer.

10.2 Importing IDF data in SEAScope

Bundles, collection archives and case studies

Zip and tar.gz archives available on the IDF Data page of the SEAScope website have been packaged so that they can be installed simply by extracting their content in SEAScope's data directory and starting the application to let the viewer detect that new data are available. Here are some example for the three operating systems supported by SEAScope:

Linux

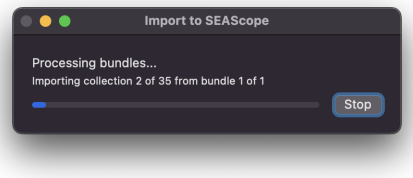
In a terminal:

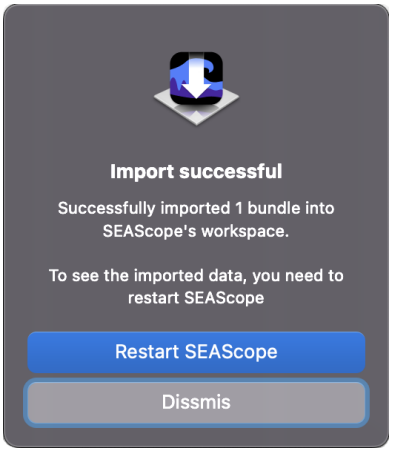
```
# Go to the seascope directory
cd seascope

# Download the bundle
wget 'https://seascope.oceandatalab.com/data/idf_samples/light_samples_OTC2023.tar.gz' \
     -O '/tmp/archive.tar.gz'

# Extract its content in the directory where you store IDF files (the "data" directory)
tar xzf /tmp/archive.tar.gz -C data

# Restart SEAScope (accept to update the index)
./seascope
```

---

macOS

- Download the data bundle
- Drag and drop the downloaded bundle on the SEAScope icon in Finder or the dock
- Wait for it to finish



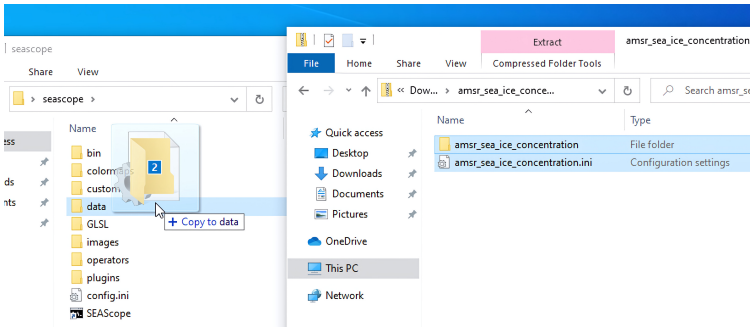


- Restart SEAScope and choose to rebuild the index if asked

Windows

- Download the data bundle in zip format
- Double-click on the downloaded file to display its content (it will open an Explorer window, do not close it yet)
- Open the SEAScope's directory in Windows Explorer

- Drag and drop all the files and directories from the archive's content to the data subdirectory in SEAScope directory Drag and drop archive content to SEAScope's data directory



- Restart SEAScope and choose to rebuild the index if asked

Other IDF files

IDF files must be placed inside a collection folder for SEAScope to detect them.

For a collection to be recognized by SEAScope, two elements are required:

- a folder whose name will be the identifier for the collection, located directly under SEAScope's data directory
- a collection configuration file whose name is the identifier of the collection followed by the `.ini` extension, also located directly under SEAScope's data directory

The collection folder can be organized however you see fit, SEAScope will browse it recursively looking for files with the `.nc` extension.
Yet, it is recommended to create subfolders to regroup IDF files that correspond to the same granule at different resolutions (same `idf_granule_id` attribute but different `idf_subsampling_factor`).

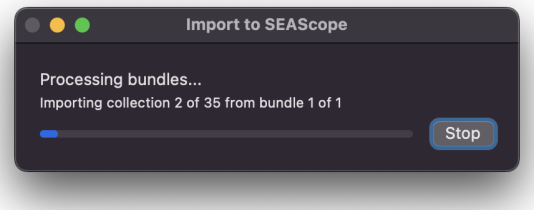
Manually

To add IDF files in SEAScope:

1. locate SEAScope's data directory
2. choose the collection to associate the IDF files with. Create it if it does not exist (folder + configuration file)
3. move or copy the IDF files in the collection folder
4. restart SEAScope to trigger a data index update

Using drag'n'drop over the SEAScope icon (macOS only)

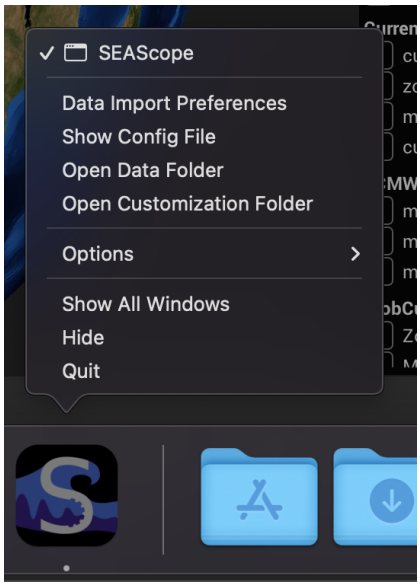
On macOS, SEAScope includes a tool which extracts data from IDF archives and places the results in SEAScope's data directory. Simply drop the folder or the archive you want to import on the icon of the SEAScope application and the tool will handle the rest:

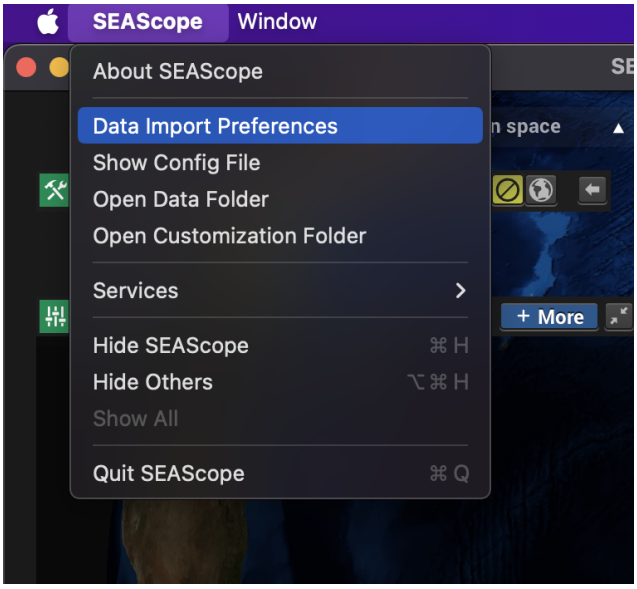


Note that the item you drop on the SEAScope icon must be a container (folder or archive) organized like SEAScope's data directory, i.e. with a folder and a file with the `.ini` extension for each collection.

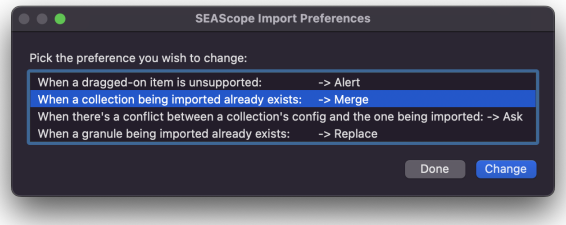
In case there is a conflict between the content of SEAScope's data directory and the content being imported, the tool will open a dialog box to ask how it should resolve the conflict:

The behavior of the tool in case of conflict can be customized: either right click on the SEAScope application icon or in the menu while SEAScope is running to open the preferences panel:





You will then be able to define the behavior of the import tool:



10.3 Non-IDF data

Regular lat/lon grids NetCDF

NetCDF files wherein data are defined on a regular lat/lon grid, with a 1-dimensional variable for latitude values and another 1-dimensional variable for longitude values, are supported natively by SEAScope.

The procedure for adding such files is exactly the same as for IDF files, the only difference being that the collection configuration file must contain a `[reader]` section describing the structure of the files.

For a regular lat/lon grid NetCDF file that contains variables with 3 dimensions (time, latitude and longitude), with the size of the time dimension equal to 1, assuming the coordinate variables have the same name as the dimensions they represent, the `[reader]` section would look like:

```
[reader]
type = latlon
timeVariable = time
latVariable = latitude
lonVariable = longitude
```

If the size of the time dimension is greater than 1, it means that the NetCDF file contains several granules (one per time value). In this case SEAScope needs to know how the temporal coverage of each granule is distributed around the time value:

- should one granule span from its time value to the next time value (i.e. the time value is the start of the granule's temporal coverage)?
- or should the time value be considered as the center of the granule's temporal coverage?
- is the distribution of the temporal coverage symmetrical around the time value?
- are there time gaps between the successive granules in this file?

Defining the `relativeTimeCoverageStart` and `relativeTimeCoverageEnd` fields in the collection configuration lets SEAScope know how it should handle these matters.

For an hourly product with a temporal coverage centered on the time values, the `[reader]` section would become:

```
[reader]
type = latlon
timeVariable = time
latVariable = latitude
lonVariable = longitude
relativeTimeCoverageStart = -1800
relativeTimeCoverageEnd = 1800
```

SEAScope can also read regular lat/lon grids with a depth dimension, but one collection must be defined for each depth value that must be available in the viewer.

For such collections, the `[reader]` section must include the `depthVariable` and `depthValue` fields.
Assuming that the variable providing values for the depth dimension is named `DEPTH` and that the collection should contain data for the depth value closest to 14.5m, the `[reader]` section of the collection configuration file from the previous example would become:

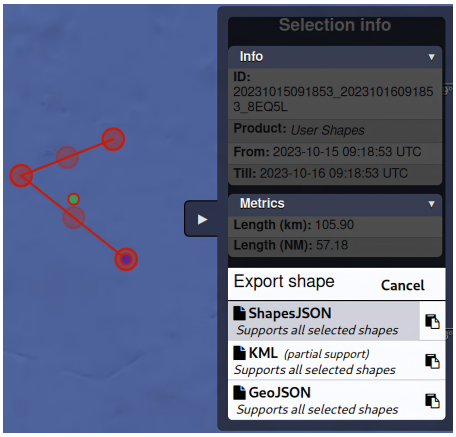
```
[reader]
type = latlon
timeVariable = time
latVariable = latitude
lonVariable = longitude
relativeTimeCoverageStart = -1800
relativeTimeCoverageEnd = 1800
depthVariable = DEPTH
depthValue = 14.5
```

Syntool annotations

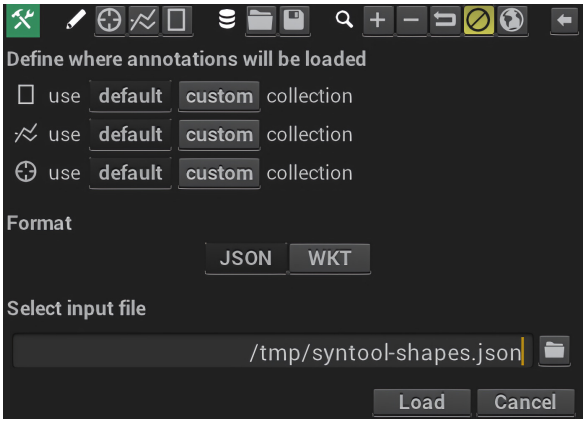
Syntool web portals offer more options than SEAScope regarding the shapes that can be drawn on the map to create annotations. But SEAScope can attach arbitrary properties to an annotation, including images, whereas Syntool cannot.

Despite these differences, it is possible to import/export annotations between Syntool and SEAScope, only for shape types supported by both tools (points, polylines and polygons).

Syntool annotations must be exported using the `ShapesJSON` output format:



Then the output file can be imported in SEAScope as JSON annotations:



If the JSON file contains annotations with shape types that SEAScope does not support, the application will discard these annotations when loading the file.

Note that properties attached to a SEAScope annotation will be lost when they are imported in Syntool. On the other hand, importing a Syntool annotation in SEAScope preserves the Syntool-specific features (arrows density, text, etc...) even if SEAScope does not support them.

Formats supported by idf-converter

The `idf-converter` Python package (which can be installed with a simple `pip install idf-converter`) provides tools for converting files available in various formats into IDF files that SEAScope will be able to manipulate.

The list of supported formats changes over time, you can get up-to-date information by querying help from the `idf-converter` command: `idf-converter --help`

To get more details (available options, etc...) on a specific data reader, simply specify the reader with the `-t` option, for example: `idf-converter -t remss/L3/wind/netcdf --help`

To use `idf-converter` jointly with SEAScope:

1. if it does not exist yet, create a collection configuration file in SEAScope's data directory. We will assume the file is named `collection1.ini`.

2. when calling `idf-converter`, set the output path to SEAScope's data directory and the output collection to `collection1`:

   ```
   idf-converter -t reader_name -o path = seascope/data collection = collection1 [...]
   ```

   The collection folder will be created in SEAScope's data directory if it did not exist yet.

3. once the files are converted, start (or restart if it was already running) SEAScope to trigger a data index update.

Please refer to `idf-converter` help and documentation for more advanced uses of the converter.

Other formats

As a last resort, if you know how to get the geolocation, temporal coverage and data using Python, the `pySEAScope` package can be used to craft granules and send them to SEAScope.

Note that collections, variables and granules created using the Python bindings are volatile, they will be lost once the SEAScope application is closed, so make sure to save the Python script used to load the data if you will need them later on.

1D data

The following dummy trajectory and data will be used as an example:

```
import datetime
import numpy

start = datetime.datetime(2015, 12, 15)
stop = datetime.datetime(2015, 12, 30)

n_points = 100
lons = numpy.rad2deg([i * (numpy.pi/2) / n_points for i in range(n_points)])
lats = numpy.rad2deg([(i * numpy.pi/4) / n_points for i in range(n_points)])
time_step = (stop - start) / n_points
time = [1000 * int((start + _ * time_step).timestamp()) for _ in range(n_points)]

fieldA = numpy.array([i**2 for i in range(n_points)])
fieldB = numpy.array([i**3 for i in range(n_points)])
```

As with any data in SEAScope, a collection is needed to hold the granule(s):

```
from SEAScope.lib.utils import create_collection

# Make sure to keep the collection identifier, we will need it later
collection_id, collection = create_collection('User - Trajectory')
```

All the elements required to create the "shell" of the granule are now available, so create it:

```
from SEAScope.lib.utils import create_granule

# Each position of the trajectory will be used as GCP
gcps = []
for i in range(0, len(lons)):
    gcp = {'time': time[i], 'lon': lons[i], 'lat': lats[i], 'i': i, 'j': 0}
    gcps.append(gcp)

granule_id, granule = create_granule(collection_id, gcps, start, stop)
```

Data can then be attached to the granule, with a name to identify them (equivalent to a name for NetCDF variables):

```
from SEAScope.lib.utils import set_field

set_field(granule, 'square', fieldA)
set_field(granule, 'cube', fieldB)
```

The granule and the collection are ready, so they are sent to SEAScope:

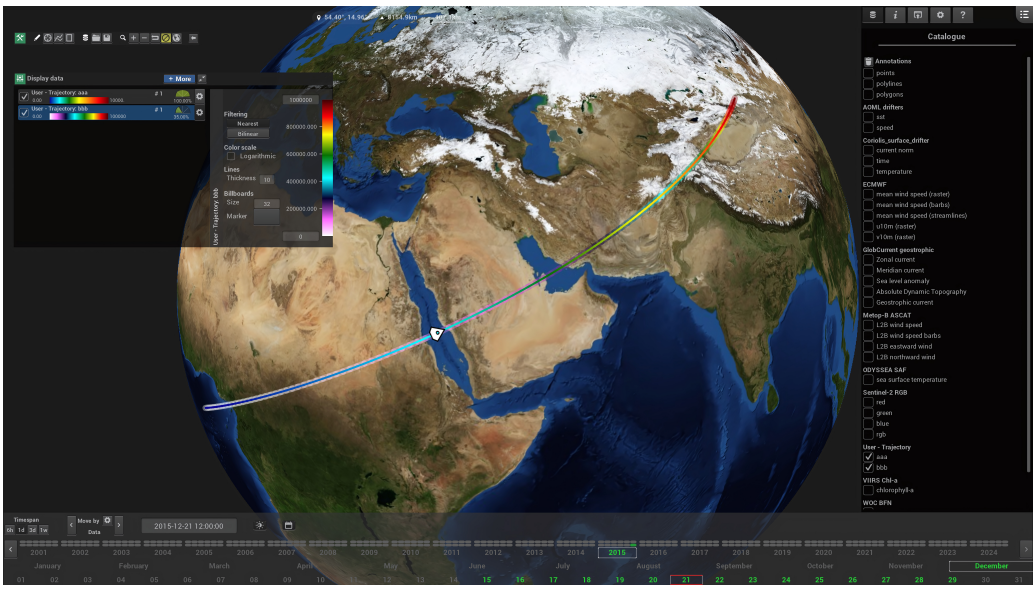
```
import SEAScope.upload

# IP address and port used to reach the SEAScope standalone application
host = '127.0.0.1'
port = 11155

with SEAScope.upload.connect(host, port) as link:
    SEAScope.upload.collection(link, collection)
    SEAScope.upload.granule(link, granule)
```

At this point nothing changes in SEAScope interface because it still lacks information on what it should do with the uploaded data. This is done by creating and uploading variables, here one per data field:

```
from SEAScope.lib.utils import create_variable

var1 = create_variable(collection, 'aaa', ['square'], dims=1)
var2 = create_variable(collection, 'bbb', ['cube'], dims=1)

with SEAScope.upload.connect(host, port) as link:
    SEAScope.upload.variable(link, var1)
    SEAScope.upload.variable(link, var2)
```

Notice the `dims=1` option passed to the `create_variable()` method: by default this method creates 2D variables, so it is required to specify that the granule is one-dimensional in this case.

Now SEAScope shows the collection with the two variables in the catalogue, here is the result after selecting the two variables and tweaking the rendering parameters so they can both been seen:



2D data

To illustrate SEAScope capabilities regarding 2D data, the following snippet creates data shaped like a donut:

```
import numpy
import datetime
import pyproj

start = datetime.datetime(2015, 12, 15)
stop = datetime.datetime(2015, 12, 30)

nrows = 500
ncols = 200

center_lon = -42
center_lat = 30
inner_radius = 3
outer_radius = 10

lons = numpy.ndarray((nrows, ncols), dtype=numpy.float32)
lats = numpy.ndarray((nrows, ncols), dtype=numpy.float32)

geod = pyproj.Geod(ellps='WGS84')
angular_step = 2.0 * numpy.pi / nrows
for row in range(nrows):
    _ = geod.inv_intermediate(center_lon + inner_radius * numpy.cos(row * angular_step),
                              center_lat + inner_radius * numpy.sin(row * angular_step),
                              center_lon + outer_radius * numpy.cos(row * angular_step),
                              center_lat + outer_radius * numpy.sin(row * angular_step),
                              npts = ncols,
                              return_back_azimuth=False)
    lons[row] = _.lons
    lats[row] = _.lats

fieldU = numpy.arange(nrows * ncols).reshape(nrows, ncols)
fieldV = numpy.arange(nrows * ncols)[::-1].reshape(nrows, ncols)
```

As with any data in SEAScope, a collection is needed to hold the granule(s):

```
from SEAScope.lib.utils import create_collection

# Make sure to keep the collection identifier, we will need it later
collection_id, collection = create_collection('User - Donut')
```

Defining the geolocation for 2D data is more complicated than for 1D data, but some tools have bee included in pySEAScope to make it easier.

The `SEAScope.lib.utils.geoloc_from_gcps` method can notably be used to compute a subsampled geolocation from matrices of latitudes and longitudes.

To do that it takes six matrices:

- the longitude values at full resolution
- the latitude values at full resolution
- a matrix containing the indices at which reference points will be read from the full resolution lat/lon matrices, for the major axis (i.e. the first dimension)

- a matrix containing the indices at which reference points will be read from the full resolution lat/lon matrices, for the minor axis (i.e. the second dimension)
- a matrix containing the desired indices on the major axis for the output (GCPs)
- a matrix containing the desired indices on the minor axis for the output (GCPs)

All the elements required to create the "shell" of the granule are now available, so create it:

```
from SEAScope.lib.utils import create_granule, geoloc_from_gcps

gcp_nrows = 100
gcp_ncols = 100

_in_dim0_indices = numpy.arange(0, lons.shape[0])
_in_dim1_indices = numpy.arange(0, lons.shape[1])
in_dim0_indices = numpy.tile(_in_dim0_indices[:, numpy.newaxis], (1, lons.shape[1]))
in_dim1_indices = numpy.tile(_in_dim1_indices[numpy.newaxis, :], (lons.shape[0], 1))

_out_dim0_indices = numpy.linspace(0, lons.shape[0], endpoint=True,
                                   num=gcp_nrows).round().astype(int)
_out_dim1_indices = numpy.linspace(0, lons.shape[1], endpoint=True,
                                   num=gcp_ncols).round().astype(int)
out_dim0_indices = numpy.tile(_out_dim0_indices[:, numpy.newaxis],
                              (1, _out_dim1_indices.shape[0]))
out_dim1_indices = numpy.tile(_out_dim1_indices[numpy.newaxis, :],
                              (_out_dim0_indices.shape[0], 1))

gcp_lon, gcp_lat = geoloc_from_gcps(lons,
                                    lats,
                                    in_dim0_indices,
                                    in_dim1_indices,
                                    out_dim0_indices,
                                    out_dim1_indices)

gcps = [{'i': out_dim1_indices[j][i], 'j': out_dim0_indices[j][i],
         'lon': gcp_lon[j][i], 'lat': gcp_lat[j][i]}
        for j in range(gcp_lon.shape[0]) for i in range(gcp_lon.shape[1])]

granule_id, granule = create_granule(collection_id, gcps, start, stop)
```

Data can then be attached to the granule, with a name to identify them (equivalent to a name for NetCDF variables):

```
from SEAScope.lib.utils import set_field

set_field(granule, 'u', fieldU)
set_field(granule, 'v', fieldV)
```

By default pySEAScope considers that there is the same number of GCPs along each axis. This is indeed the case here (`gcps_nrows = 100` and `gcps_ncols = 100`), but it might not be the case so it is important to fix the size of the GCPs matrix in the granule's metadata:

```
granule['metadata']['uris'][0]['shape']['xArity'] = gcp_ncols
granule['metadata']['uris'][0]['shape']['yArity'] = gcp_nrows
```

The granule and the collection are ready, so they are sent to SEAScope:

```
import SEAScope.upload

# IP address and port used to reach the SEAScope standalone application
host = '127.0.0.1'
port = 11155

with SEAScope.upload.connect(host, port) as link:
    SEAScope.upload.collection(link, collection)
    SEAScope.upload.granule(link, granule)
```

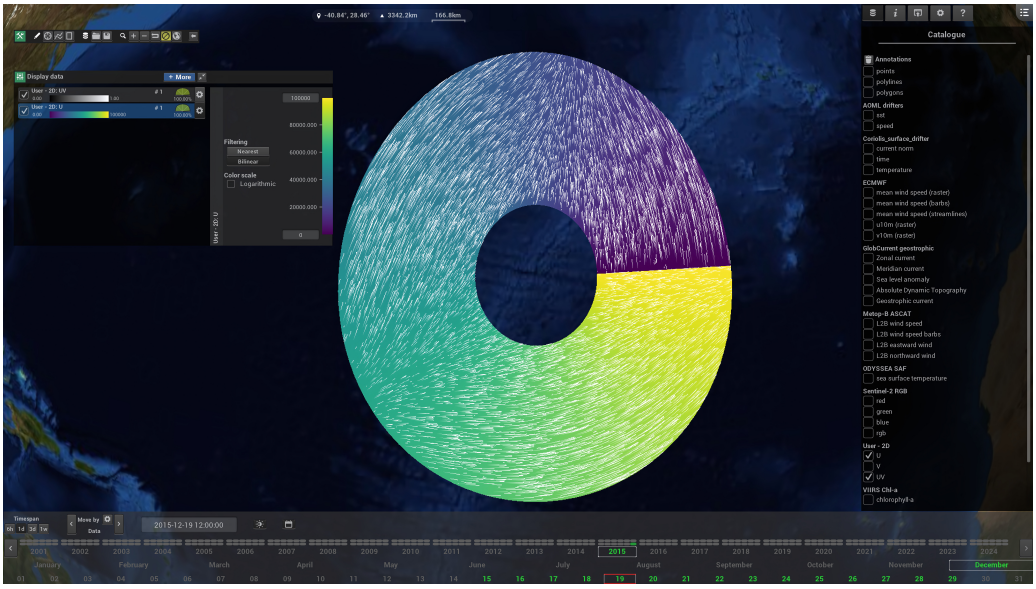
Now create and upload variables to inform SEAScope how it should display the granule's data, here one per data field:

```
from SEAScope.lib.utils import create_variable

var1 = create_variable(collection, 'U', ['u'])
var2 = create_variable(collection, 'V', ['v'])

with SEAScope.upload.connect(host, port) as link:
    SEAScope.upload.variable(link, var1)
    SEAScope.upload.variable(link, var2)
```

One extra step to show how to define a variable rendered as streamlines:

```
var3 = create_variable(collection, 'UV', ['u', 'v'])
var3['rendering']['renderMethod'] = 'STREAMLINES'
var3['rendering']['particlesCount'] = 10000
var3['rendering']['streamlineLength'] = 60
var3['rendering']['streamlineSpeed'] = 0.5

with SEAScope.upload.connect(host, port) as link:
    SEAScope.upload.variable(link, var3)
```

Select the variables in the catalogue, tweak the rendering settings and SEAScope will display the granule variables as raster and as streamlines:



Troubleshooting

11.1 Quick solutions

SEAScope uses absolute paths when it stores information about data and configuration files, so if SEAScope does not start anymore or if it starts but the catalogue is empty / only contains annotations, it might be because the files are not located where SEAScope expects them anymore.

So, in case you:

- renamed/moved the SEAScope directory:
  - if you did not change any setting, or don't mind losing your customizations, you can simply delete the `config.ini` file located in the SEAScope directory, the application will recreate it next time it starts.
  - if you modified the settings and want to preserve your changes, then you need to open the `config.ini` file in a text editor and update the values under the `[paths]` section.
- renamed/moved the data directory, which by default is inside the SEAScope directory (i.e. if you moved the SEAScope directory, you moved the data directory as well):
  - delete the index.fb file located in the SEAScope directory, the application will recreate and populate it automatically next time you start the application.

Also remember that the `data` directory should have the following structure:

```
data
├── collectionA.ini
├── collectionA
│   ├── granule1
│   │   ├── granule1_idf_00.nc
│   │   ├── granule1_idf_01.nc
│   │   └── ...
│   └── ...
├── collectionB.ini
├── collectionB
│   ├── granuleN
│   │   ├── granuleN_idf_00.nc
│   │   ├── granuleN_idf_01.nc
│   │   └── ...
│   └── ...
└── ...
```

When new files have been added in the data directory but they do not show up when you start the viewer, it is probably because the new files have not been placed according to this layout. For example, it is possible that the software you used to extract the IDF files from a tar.gz or a zip archive has created an additional directory where it put the extracted files.

It is also important that the names of the INI configuration files match the names of the collection directories, otherwise the collections will be ignored by SEAScope.

Finally, make sure to restart SEAScope when you add / modify / remove files in the data directory: the application will not check for changes in the data directory while it is running, so restart SEAScope and accept its request to rebuild the index (it prompts either in a dialog box or in the terminal, depending on the tools installed on your system).

11.2 Known issues

Most common issues are documented in the FAQ available on the SEAScope website: https://seascope.oceandatalab.com/faq.html.

Known issues that may not have a workaround yet are also listed on the website for the viewer at https://seascope.oceandatalab.com/viewer/seascope-viewer-20250709_known_issues.txt and for the Python bindings package at https://seascope.oceandatalab.com/python/pyseascope-0.3.229_known_issues.txt

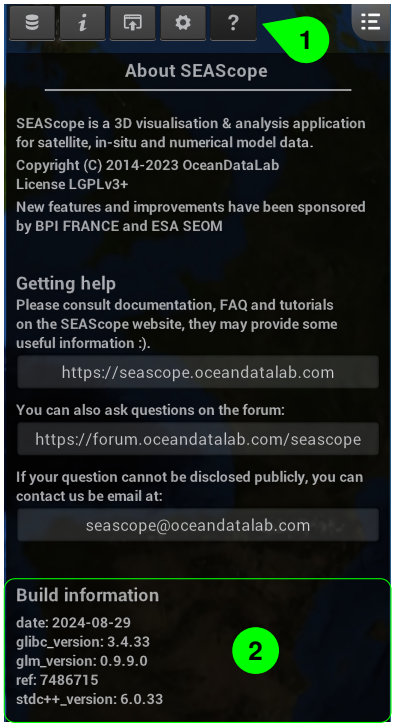
11.3 How to report problems

To identify the cause of a bug, it is really important that you provide as much information as possible regarding the context in which the problem occurred.

- a step by step description of the actions that must be performed to reproduce the problem
- the version of your operating system, e.g. Windows 10, Ubuntu 22.04, macOS Big Sur, etc...
- the content of your SEAScope configuration file
- SEAScope build information

  These details can be found in the "About" tab of SEAScope's side panel, it will open if you click on the button with a question mark icon (1.).

  The build information (2.) cannot be copied as text, and transcribing the version numbers is a cumbersome task, so don't hesitate to take a screenshot and include it in your message when you report an issue.



In case the problem only manifests with a specific collection or a specific granule, providing the collection configuration file will help analysing the problem.

If the data that trigger the bug are publicly available, sharing a link to one of the granules that trigger the bug also makes it easier to reproduce the malfunction.

For data that are not publicly available, sharing the schema of one of the granules (the output of `ncdump -hs THE_FILE.nc`) can already provide some hints that can move the investigation in the right direction.

11.4 Contact and feedback

- On the forum: https://github.com/oceandatalab/OVL/discussions/new/choose
- On X/Twitter: @oceandatalab
- By email: seascope@oceandatalab.com

11.5 Debug mode

If there is an issue with SEAScope and you want to investigate by yourself, enabling debug logs can provide some insights, notably on what the viewer was doing before a malfunction.

Enabling debug logs is quite simple if you run SEAScope from the command line, you only have to add the `-f debug` argument when you execute the command.

On macOS, assuming there is only one SEAScope application installed, the command line becomes:

```
open -a SEAScope --args -f debug
```

If there are more than one SEAScope available, you can use either:

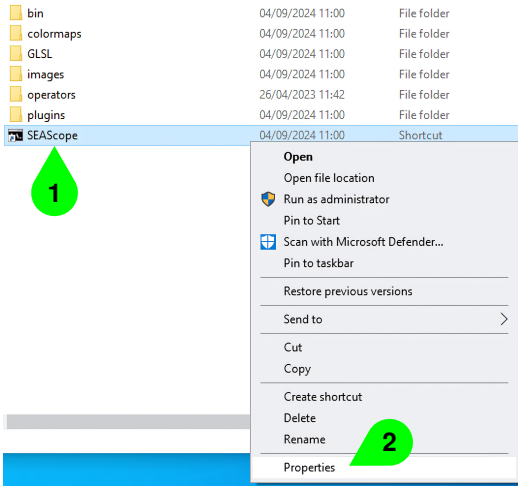
```
open path/to/SEAScope.app --args -f debug
```

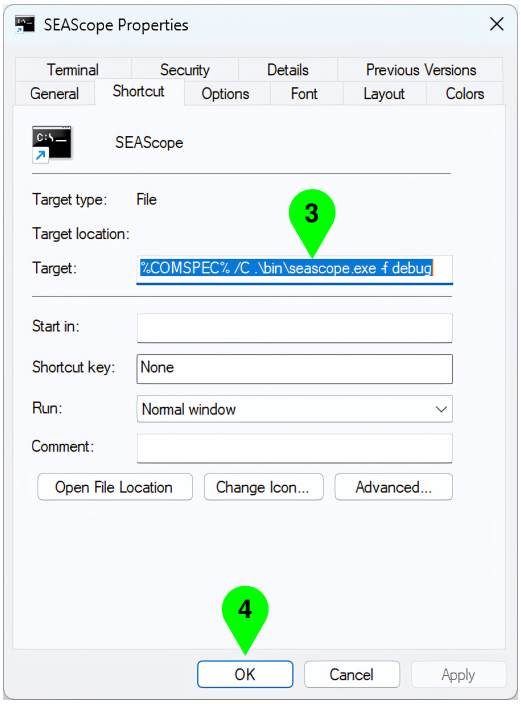
or

```
path/to/SEAScope.app/Contents/MacOS/SEAScope -f debug
```

On Windows, if you are not using the command line you have to change the target of the SEAScope shortcut. To do so:

- Right-click on your SEAScope shortcut (1).
- Select "Properties" (2), it will open a dialog box similar to the one in the screenshot on the right.





- In the "Target" field (3), append `-f debug` after `seascope.exe`.
- Click "OK" (4).

On macOS log messages are saved in a file (`~/Library/Logs/com.oceandatalab.SEAScope/seascope.log`, which can be accessed under `~/Library/Logs > com.oceandatalab.SEAScope` using the Console app found in "/Applications/Utilities") whereas they are printed in the terminal on Linux and Windows.

Debug logs are **very** verbose, especially when you move your mouse, so exploiting these logs while SEAScope is running might be difficult (it is probably easier to save the output in a file for later inspection).

In case SEAScope has issued error messages, it might be possible to narrow down the problem and filter logs accordingly. SEAScope log messages are all prefixed by an identifier between square brackets, and you can tell the application to only display log messages that are associated with this identifier if you replace `-f debug` by `-f debug:Identifier`.

For example, if you have the following error message:
`[CollectionINIParser] ...`
then you can use the `-f debug:CollectionINIParser` argument when you start SEAScope so that it only prints messages related to `CollectionINIParser` in the logs.

It is possible to accept several identifiers in the filter, by using a comma-separated list after the colon:
`-f debug:Identifier1,Identifier2,...`

Appendices

Annex I: Tips & tricks

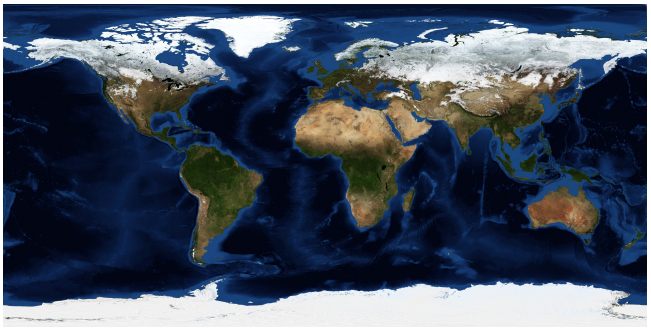
I.1 High resolution background

Several versions of the BlueMarble worldmap from NASA are available on ODL servers, we did not include them in the SEAScope package because these files are quite heavy and some of them are only supported by high-end graphic cards.

Graphics cards (GPUs) not only have a limit on the number of pixels an image can have, but also on the width and height of the image: even if the number of pixels fits in the GPU's memory, if the image width is much bigger than its height (or the opposite) then it might exceed the GPU's limit and it will not be possible to load it.

As all GPUs are not the same, the number of pixels and the dimensions limits vary between computers. We offer background images at various resolutions to accommodate most GPUs, both in png and jpg formats.

Note that at the same resolution both formats will take the same amount of memory on the GPU, the file sizes differ due to how each format compresses the data (lossy compression has been enabled for jpg, so file sizes are smaller but some compression artefacts may be visible when zoomed in):



- 4000x2000 PNG JPG
- 8000x4000 PNG JPG
- 16000x8000 PNG JPG
- 21000x10500 PNG JPG
- 32000x16000 PNG JPG

These worlmaps have been created from Blue Marble: Next Generation +Topography and Bathymetry (December 2004).

Blue Marble: Next Generation was produced by Reto Stöckli, NASA Earth Observatory (NASA Goddard Space Flight Center). When using or republishing Blue Marble: Next Generation please credit "NASA Earth Observatory."

Once you have downloaded a worldmap:

1. open SEAScope
2. go to the Settings tab in the side menu
3. under the Misc section, in the Background image field select/input the path of the image you downloaded
4. close and restart SEAScope

If you try to use a worldmap whose dimensions are not supported by your GPU, SEAScope will simply render the Earth as a black globe and your graphics driver may issue an error message in the logs.

In case these worldmaps don't suit your needs, you can generate your own, SEAScope only needs the image to:

- have a width twice the size of its height (width/height ratio = 2:1)
- have a width below the texture size limit of your GPU
- be in `png`, `jpg`, `bmp` or `tga` format

I.2 Custom trajectory markers

You can add you own pictograms to SEAScope and use them to represent the trajectory positions that match the current time.

SEAScope only supports pictograms in PNG format that contain 4 channels (3 for primary colors and 1 for transparency). The images should be square with a maximal size of 256x256 pixels.

Please note that SEAScope will rotate the images so that the bottom side faces the previous position in the trajectory and the top side faces the next position.

To add your own pictograms, you first have to locate your "custom" directory, the easiest way to achieve this is to:

1. start SEAScope
2. open the Settings tab in the side menu
3. copy the value of the "Custom" field in the "Paths" section

If you inspect the content of this "custom" directory you should see that it contains a directory named "icons": simply copy your pictograms in this "icons" directory, restart SEAScope and you should now be able to choose your pictograms in the rendering configuration of trajectory collections!

I.3 Multiple SEAScope instances

Running several instances of SEAScope at the same time, simply by clicking on the icon/shortcut or by executing the command from different terminals, will lead to incoherences or application crashes because these instances will try to modify the same files.

However, it is possible to have multiple instances of SEAScope running on the same computer under specific conditions:

1. each instance must have its own configuration file. You can simply start SEAScope so that it generates the configuration file, then create as many copies of that file as the number of instances you want.
2. it is also mandatory for each instance to have its own state file, so each configuration file must have a different value for the `[paths]state` setting.
3. each instance can have its own data directory, but sharing the data directory is also possible as long as none of the SEAScope instances is used to modify rendering settings permanently (the "Save" button in the Advanced rendering control).

   In case each instance must have its own data directory, set a different value for the `[paths]data` setting in each configuration file.
4. sharing the data index is not recommended because several instances could try to update it at the same time, so each configuration file should have a different value for the `[paths]index` setting.
5. the same goes for the annotations database, as SEAScope immediately updates the database when there is a change related to annotations, be it a shape modification, a property update or a change in the rendering settings (contrary to collections from the data directory, changes in rendering settings are saved automatically and immediately for annotations, there is no "Save" button).

   Configurations files should therefore have a different value for the `[paths]annotations` setting.

Once the configuration files have been modified, start each instance with a different configuration file by using the `-c` option: `./seascope -c path_to_seascope_config.ini`. If you are not using the command line, please refer to the "Debug mode" section as the procedure to use a specific configuration file is similar to the one for enabling debug mode.

If you have full control on how the instances are created, **if** the instances will not modify anything on annotations and **if** the "Save" button in the Advanced rendering control will not be used, then rules 3., 4. and 5. can be ignored. However you will have to start one instance, wait for it to complete the index update, then start the other instances to avoid any index update conflict. Use that information at your own risk.

I.4 Remote control

SEAScope listens on a network interface for incoming API requests that can be emitted with the Python bindings (`pySEAScope` package).

By default, for obvious security reasons, requests are only accepted if the script or notebook that emitted them is running on the same computer as the SEAScope instance.

In environments where the network is considered secure, it is possible to make SEAScope accept API requests coming from other computers, meaning that it is possible to run the viewer on a machine (one with a better GPU for example) and control it from another as long as they can reach each other on the network.

This can be achieved by adding the `-l` option followed by an IPv4 address, a colon and a port number when starting SEAScope. For example: `./seascope -l '192.168.1.32:33333'`

If you are not using the command line, please refer to the "Debug mode" section as it illustrates the procedure to pass an option to SEAScope.

Note that the IPv4 address cannot be chosen arbitrarily, it must be a valid address for one of the network interfaces of the computer.

On most systems, port numbers below 1024 are reserved and cannot be used unless you have an administrator account (simply use a port number greater than 1024 to avoid this issue).

Be sure to discuss this with the IT staff of your organization to confirm that it is in line with the network security policy, and agree on port numbers if communication between computers must pass through a firewall.

Once SEAScope is running, adapt you script or notebook on the remote computer so that it uses the same address and port as the ones specified with the `-l` option, for example to fetch extracted data then zoom-in:

```
import SEAScope.lib
import SEAScope.upload

address = '192.168.1.32'
port = 33333

extracted_data = SEAScope.lib.get_extracted_data(address, port)

with SEAScope.upload.connect(address, port) as link:
    SEAScope.upload.zoom_in(link)
```

I.5 Dialog boxes and file browsers on Linux

Contrary to Windows and macOS, Linux has no official graphical user interface, so by default SEAScope uses the terminal for the data index prompt and a very basic file browser for selecting paths.
It is however possible to get dialog boxes and more advanced file browsers if one of the following software is installed on the system:

- zenity
- matedialog
- qarma
- kdialog

Please refer to the documentation of your Linux distribution for instructions on how to install these tools.

I.6 Crafting annotations

SEAScope uses Syntool's conventions for annotations saved in JSON format, here are some examples of JSON data for each shape type supported by SEAScope:

Point annotation

```
{
    "end": 1449080999000,
    "start": 1448994599000,
    "type": "WKT",
    "wkt": "POINT(-129.087 -0.339614)"
}
```

start beginning of the annotation's temporal coverage, expressed as the number of milliseconds since 1970-01-01T00:00:00Z. Included in temporal coverage.

end end of the annotation's temporal coverage, expressed as the number of milliseconds since 1970-01-01T00:00:00Z. Excluded from temporal coverage.

type must be set to "WKT" for points.

wkt WKT description of the geometry, using the `POINT(longitude latitude)` pattern.

Polyline annotation

```
{
    "end": 1449080999000,
    "points": [
        [
            -153.98358223394598,
            -0.1693628647635171
        ],
        [
            -146.82281190810463,
            -20.169543496405776
        ],
        [
            -120.92112265077215,
            23.420925277196115
        ]
    ],
    "start": 1448994599000,
    "type": "LINE"
}
```

start beginning of the annotation's temporal coverage, expressed as the number of milliseconds since 1970-01-01T00:00:00Z. Included in temporal coverage.

end end of the annotation's temporal coverage, expressed as the number of milliseconds since 1970-01-01T00:00:00Z. Excluded from temporal coverage.

type must be set to "LINE" for polylines.

points a list of vertices, each vertex being a list of two coordinates provided in the longitude, latitude order.

Polygon annotation

```
{
    "end": 1449080999000,
    "start": 1448994599000,
    "type": "WKT",
    "wkt": "POLYGON((-139.52 -3.45,-124.14 7.47,-117.99 -8.46,-138.58 -18.27,-139.52 -3.45))"
}
```

start beginning of the annotation's temporal coverage, expressed as the number of milliseconds since 1970-01-01T00:00:00Z. Included in temporal coverage.

end end of the annotation's temporal coverage, expressed as the number of milliseconds since 1970-01-01T00:00:00Z. Excluded from temporal coverage.

type must be set to "WKT" for polygons.

wkt WKT description of the geometry, using the `POLYGON((longitude1 latitude1, longitude2 latitude2, ..., longitude1 latitude1))` pattern.

Note that polygons are closed geometries, which explains why the first and the last vertices have the same coordinates.

With a Python script

For some use cases, creating annotations by hand is not convenient, but it is absolutely possible to use a script for generating JSON files containing annotations that can be imported in SEAScope.

SEAScope expects JSON files to contain a list of annotations, so JSON files containing annotations should always start with a `[` character and end with a `]` character. All annotations, except the last one, should be followed by a comma.

The following snippet defines methods to create annotations of each type:

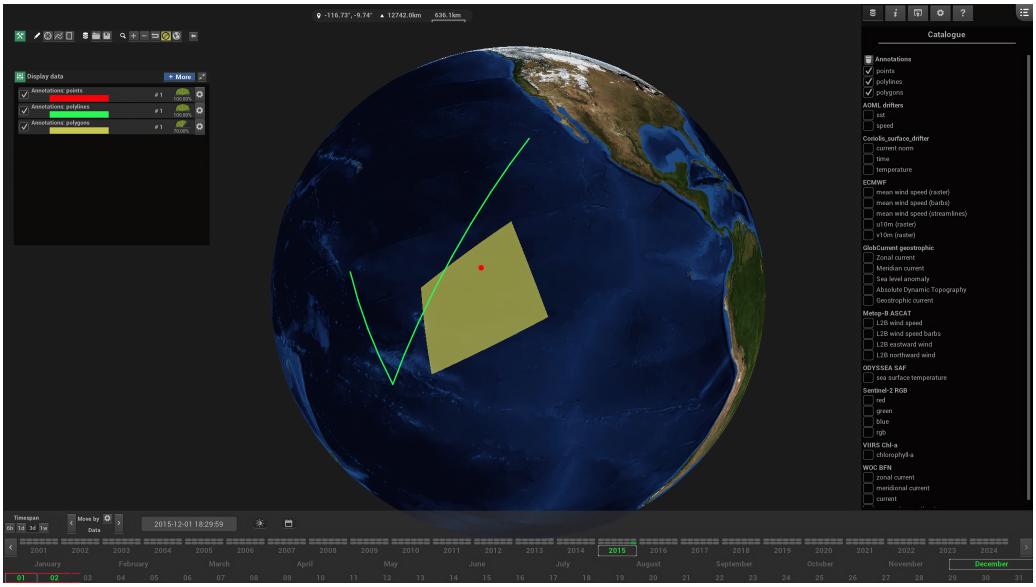
```
import datetime
import typing


def create_point(start: datetime.datetime,
                 end: datetime.datetime,
                 lon: float,
                 lat: float
                 ) -> typing.Dict[str, typing.Any]:
    """Create a point annotation that SEAScope can read"""
    annotation = {'start': int(1000 * start.timestamp()),
                  'end': int(1000 * end.timestamp()),
                  'type': 'WKT',
                  'wkt': f'POINT({lon} {lat})'
                  }
    return annotation


def create_polyline(start: datetime.datetime,
                    end: datetime.datetime,
                    coordinates: typing.Sequence[typing.Tuple[float, float]]
                    ) -> typing.Dict[str, typing.Any]:
    """Create a polyline annotation that SEAScope can read"""
    annotation = {'start': int(1000 * start.timestamp()),
                  'end': int(1000 * end.timestamp()),
                  'type': 'LINE',
                  'points': coordinates
                  }
        return annotation


def create_polygon(start: datetime.datetime,
                   end: datetime.datetime,
                   vertices: typing.Sequence[typing.Tuple[float, float]]
                  ) -> typing.Dict[str, typing.Any]:
    """Create a polygon annotation that SEAScope can read"""
    polygon_wkt_coords = ','.join([f'{_[0]} {_[1]}' for _ in vertices])
    annotation = {'start': int(1000 * start.timestamp()),
                  'end': int(1000 * end.timestamp()),
                  'type': 'WKT',
                  'wkt': f'POLYGON(({polygon_wkt_coords}))'
                  }
    return annotation
```

Please note that some elements that should be included in Python scripts (defining a logger, method documentation, wrapping code in a function, etc...) have been omitted to keep the snippet as concise as possible.

These methods can then be used to recreate the previous examples and save them in a single JSON file:

```
import json

point = create_point(datetime.datetime(2015, 12, 1, 18, 29, 59, 0, datetime.UTC),
                     datetime.datetime(2015, 12, 2, 18, 29, 59, 0, datetime.UTC),
                     -129.087,
                     -0.339614)


polyline = create_polyline(datetime.datetime(2015, 12, 1, 18, 29, 59, 0, datetime.UTC),
                           datetime.datetime(2015, 12, 2, 18, 29, 59, 0, datetime.UTC),
                           [[-153.98358223394598,-0.1693628647635171],
                            [-146.82281190810463,-20.169543496405776],
                            [-120.92112265077215,23.420925277196115]])

polygon = create_polygon(datetime.datetime(2015, 12, 1, 18, 29, 59, 0, datetime.UTC),
                         datetime.datetime(2015, 12, 2, 18, 29, 59, 0, datetime.UTC),
                         [[-139.52, -3.45],
                          [-124.14, 7.47],
                          [-117.99, -8.46],
                          [-138.58, -18.27],
                          [-139.52, -3.45]])

annotations = [point, polyline, polygon]
with open('annotations_example.json', 'wt') as f:
    json.dump(annotations, f)
```

Import the output file `'annotations_example.json'` and after selecting the variables that you chose to tie the annotations with, SEAScope correctly displays the point, polyline and polygon previously defined:



In SEAScope, annotations are also able to hold properties and these properties can be defined in the JSON file as well by creating a new key named `properties` and associated with a dictionary value.

Some caveats:

- Properties support at most one level of nesting, i.e. values directly under the `properties` object can be dictionaries, but these dictionaries cannot contain other dictionaries.
- Lists are not supported.
- JSON cannot contain binary data, so images must be encoded in text form using base64. The result must be prepended by `` otherwise SEAScope will consider that the property is only some text. Note that SEAScope only recognize this preamble for images, so use it even if the images are not in PNG format, SEAScope will read them anyway as long as their format is supported.

Here is an example showing how to add properties to an annotation with Python (using the method defined in previous snippets):

```python
import json
import base64

point = create_point(datetime.datetime(2015, 12, 1, 18, 29, 59, 0, datetime.UTC),
                     datetime.datetime(2015, 12, 2, 18, 29, 59, 0, datetime.UTC),
                     -119,
                     45)

point['properties'] = {}
point['properties']['some text'] = 'lorem ipsum...'
point['properties']['a random number'] = 42
point['properties']['boolean value'] = True
point['properties']['one sub level'] = {}
point['properties']['one sub level']['another number'] = 22.12345

with open('some_image.png', 'rb') as f:
    binary_png_image = f.read()
b64_encoded_png = base64.b64encode(binary_png_image).decode('utf-8')
point['properties']['a png image'] = f'{b64_encoded_png}'

with open('another_image.jpg', 'rb') as f:
    binary_jpg_image = f.read()
b64_encoded_jpg = base64.b64encode(binary_jpg_image).decode('utf-8')

# Use image/png in the prefix even if the image is in another format
point['properties']['a jpg image'] = f'{b64_encoded_jpg}'

# SEAScope expects a list of annotations
annotations = [point, ]
with open('annotation_properties_example.json', 'wt') as f:
    json.dump(annotations, f)
```

Once loaded in SEAScope, the annotation has all the properties defined in the Python script, including the images:

Annex II: SEAScope configuration file

II.1 Configuration file template

SEAScope settings are stored in a configuration file in INI format, each line containing either a section name between square brackets (`[section]`) or a setting defined using the `key = value` syntax.

There are six sections of settings in the configuration file: paths, rendering, timeline, misc, processor and drawing. SEAScope will not work properly and might crash if there are undefined settings, so please make sure that your configuration file has all the settings described below, even if the associated value stays empty.

```
[paths]
colormaps = SEASCOPE_WORKSPACE/colormaps
data = SEASCOPE_WORKSPACE/data
state = SEASCOPE_WORKSPACE/state.db
index = SEASCOPE_WORKSPACE/index.fb
annotations = SEASCOPE_WORKSPACE/annotations.db
custom = SEASCOPE_WORKSPACE/custom

[rendering]
glRenderer = OpenGL3
glContextMajor = 3
glContextMinor = 3
windowWidth = 800
windowHeight = 600
antialiasing = false
antialiasingType = msaa
antialiasingFactor = 8
fullscreen = false
vsync = true
minGranuleScreenEstate = 0.008

[timeline]
minYear = 1988
maxYear =
timespans = 1d|1 day timespan|43200|43200,1w|1 week timespan|302400|302400
timesteps = 6h|Move by 6 hours|21600,Data|Move to previous/next available data|0
defaultTimespan = 1d
defaultTimestep = 1/3 span
useNearestMode = true
initialDatetime =
advancedScrolling = false

[misc]
worldMap = lowres
skipIndexCheck = false
searchRequestMinDelay = 0.1

[processor]
enabled = true
address = 127.0.0.1
port = 53450

[drawing]
explicitDrawingModeExit = false
clickPixelsTolerance = 7
```

The configuration of the SEAScope application is stored in a file named "config.ini" which is located at the root of the SEAScope workspace:

- on Linux, where you extracted the seascope-viewer-20250709.tar.gz archive
- on macOS, at ~/SEAScope-workspace
- on Windows, where you extracted the seascope-viewer-20250709.zip archive

If this file does not exist, it will be generated automatically (with default settings) when you run SEAScope.

It is not recommended to edit the configuration file directly, instead you should use the Settings tab in the application's side menu to modify settings.
It might not be possible to access the Settings tab if the application refuses to start after changing some settings. Please note that the safest way to recover in this case remains to delete the configuration file and let SEAScope generate a new one automatically.

II.2 List of settings

[paths] section

colormaps

Path of the directory which contains the colormaps available in the viewer.
A colormap is a text file which contains a list of RGB triplets and its filename must end with the .rgb extension.
Each line of the colormap file must contain exactly three space-separated integer values between 0 and 255.

data

Path of the directory that contains the data that will be available in the application, organized according to a collection/granule hierarchy.

```
data
├── collectionA.ini
├── collectionA
│   ├── granule1
│   │   ├── granule1_idf_00.nc
│   │   ├── granule1_idf_01.nc
│   │   └── ...
│   ├── granule2
│   │   ├── granule2_idf_00.nc
│   │   ├── granule2_idf_01.nc
│   │   └── ...
│   └── ...
├── collectionB.ini
├── collectionB
│   ├── granuleN
│   │   ├── granuleN_idf_00.nc
│   │   ├── granuleN_idf_01.nc
│   │   └── ...
│   └── ...
└── ...
```

state

Path of the file where the state of the application will be stored. Edit only if you move the SEAScope workspace to another location.

index

Path of the file where the application will index the contents of the data directory. Edit only if you move the SEAScope workspace to another location, or if the index file cannot be stored in SEAScope's workspace due to its size.

In order to provide users with a way to explore data smoothly, the viewer needs to list the available IDF files and to extract some metadata from each of them. Doing so dynamically would not yield satisfactory performances since the time required to crawl the data directory would grow as new granules are added.

The crawling step may take a long time therefore the listing and extracted metadata are stored in an index file. When the viewer starts, it checks the availability of the index file:

- if it exists, it loads its contents and skips the crawling step completely.
- otherwise it performs the index step and creates the index file.

You have to be patient the first time you start the viewer as it will have to build the index.

After you add/remove/change files in the data directory, restart SEAScope: the application will detect that the content of the data directory has changed and ask you to confirm that the index needs to be updated.

You can use the `skipIndexCheck` entry (`[misc]` section) to bypass the detection of changes, which can save some time if your data directory contains a lot of IDF files, or if you are only modifying files that will have no effect on the index (changing a label in the configuration file of a collection for example). In this case SEAScope will not rebuild the index unless you delete it or set the `skipIndexCheck` setting back to false.

annotations

Path of the file where SEAScope will store the annotations you create or import. Edit only if you move the SEAScope workspace to another location, or if the annotations file cannot be stored in SEAScope's workspace due to its size.

custom

Path of the directory where SEAScope will look for custom resources. Edit only if you move the SEAScope workspace to another location, or if the custom directory cannot be stored in SEAScope's workspace due to its size.

The directory must exist and include a subdirectory named "icons".

```
custom
├── ...
├── icons
│   ├── my_icon.png
│   ├── another_custom_icon.png
│   └── ...
└── ...
```

[rendering] section

glRenderer

Identifier of the OpenGL renderer used by the viewer. Default value is OpenGL3.
This is a developer setting, do not change it unless you know what you are doing.

glContextMajor

Major version number of the GL context to create. Should be an integer value, default value is 3.
This is a developer setting, do not change it unless you know what you are doing.

glContextMinor

Minor version number of the GL context to create. Should be an integer value, default value is 3.

This is a developer setting, do not change it unless you know what you are doing.

windowWidth

Width in pixels of the viewer window. Should be an integer value, default value is 800.
Ignored if fullscreen is enabled.

windowHeight

Height in pixels of the viewer window. Should be an integer value, default value is 600.
Ignored if fullscreen is enabled.

antialiasing

Enable antialiasing filter. Should be a boolean value (true|false), default value is false.

antialiasingType

Identifier of the antialiasing filter to use. Leave it empty or use the default value (`msaa`)

antialiasingFactor

Parameter for the msaa antialiasing filter. Should be an integer value from the following list: 2, 4, 6, 8. Default value is 8.

Higher values generate smoother images but require a more powerful GPU.

fullscreen

Open the viewer in fullscreen mode. Should be a boolean value (true|false), default value is false.

vsync

Enable vertical synchronization. Should be a boolean value (true|false), default value is true.

Vertical synchronization prevents the GPU from generating output at a frequency higher than the screen refresh rate. We recommend to enable this option unless you are performing a benchmark.

minGranuleScreenEstate

Minimal value the (granule area / screen area) ratio must reach for the viewer to actually load data. Should be a float value between 0 and 1, default value is 0.008 (i.e. 0.8%).

Some high resolution granules may only be displayed as as bunch of pixels at low zoom levels. This setting tells the viewer to skip data loading in these cases, thus saving processing power and improving fluidity.

[timeline] section

minYear

Lower bound (inclusive) for the years list available in the timeline. Should be an integer value in YYYY format in the [1970, 3000] range, default value is 1988.

maxYear

Upper bound (exclusive) for the years list available in the timeline. Should be either an integer value in YYYY format in the [1970, 3000] range or left empty, default value is empty.
When this setting is left empty, SEAScope acts as if it was set to current year + 1, so that the current year is available in the timeline.

timespans

This setting defines a list of timespans the user will be able to choose from in the viewer.
A timespan is a (potentially asymmetric) distribution of time around a reference datetime.

The viewer applies the selected timespan to the current datetime to compute the time window granules must intersect in order to be displayed.

You must provide four elements to define a timespan (timespans are displayed as buttons in the timeline control):

- a label for the button (keep it short, 3-4 characters)
- a tooltip for the button
- the size, in seconds, of the left part (before current datetime) of the time window
- the size, in seconds, of the right part (after current datetime) of the time window

These four elements must be separated by pipe characters. Timespan definitions must be separated by commas.

For example, in order to create two buttons:

- one for a [T-3h, T+3h[time window
- one for a [T-30min, T+30min[time window

The value associated to the `timespans` key in the configuration file would be:

```
6h|6hours timespan|10800|10800,1h|1hour timespan|1800|1800,
```

Default value is (on a single line):

```
6h|6 hours timespan|10800|10800,1d|1 day timespan|43200|43200,3d|3 days timespan|129600|129600,
1w|1 week timespan|302400|302400
```

timesteps

This setting defines a list of timesteps the user will be able to choose from in the viewer.
A timestep is a number of seconds by which the current datetime is decreased (resp. increased) when the user clicks on the Prev (resp. Next) button.

You must provide three elements to define a timestep (timesteps are displayed as buttons in the timeline control):

- a label for the button (keep it short, 3-4 characters)
- a tooltip for the button
- the size of the step (in seconds)

These three elements must be separated by pipe characters. Timestep definitions must be separated by commas.

Note that by defining a timestep of zero second, you will create a button which has a special behavior: if it is selected, instead of decreasing/increasing the current datetime by a fixed number of seconds, it will tell the viewer to use the datetime of the closest granule in the past/future as current datetime.

For example, in order to create three buttons:

- one for a one-hour step
- one for a three-days step
- one for the nearest granule behavior

The value associated to the `timesteps` key in the configuration file would be:

`1h|Move by one hour|3600,3d|Move by three days|259200,Event|Move to closest data|0,`

SEAScope also adds a timestep labelled "1/3 span" automatically, even if it is not listed in the `timesteps` setting.

Default value is (on a single line):

```
6h|Move by 6 hours|21600,Day|Move by 24 hours|86400,Week|Move by 7 days|604800,
Data|Move to previous/next available data|0
```

defaultTimespan

Label of the timespan button which is selected by default in the viewer. Default value is "1d".

defaultTimestep

Label of the timestep button which is selected by default in the viewer. Default value is "1/3 span".

useNearestMode

Enable «nearest» mode. Should be a boolean value (true|false), default value is true.
When the «nearest» mode is active, the viewer looks for the closest datetime for which a granule is available and use it as current datetime.

initialDatetime

Datetime on which the timeline is centered when SEAScope starts.
Supported values are:

- empty (current datetime is used)
- a full datetime in YYYY-mm-ddTHH:MM:SS format
- a date in YYYY-mm-dd format (time will be set to 12:00:00)
- a time in HH:MM:SS format (date will be set to current day)

Note that SEAScope saves the state of the timeline when it closes and restores it on next execution, so this setting is only used for the first time SEAScope is executed, or if the state file is deleted.

advancedScrolling

Enable support for independent scrolling on years, months and days in the timeline. Should be a boolean value (true|false), default value is false.

When advanced scrolling is enabled each row of the timeline (years, months and days) can be scrolled independently. When it is disabled the scroll shifts the time by the same amount regardless on the hovered row.

[misc] section

worldMap

Image that SEAScope displays as background on the globe.
SEAScope has two predefined worldmap backgrounds:

- `lowres`: A worldmap with coarse details but very light in memory
- `black`: A black worldmap

A third predefined value, `highres`, is also supported but will be deprecated and removed in future versions of SEAScope. It had much more details but was not bundled in the SEAScope package due to its size.

Any other value will be considered as the path to the image that must be used as worldmap. The height of the image must be half its width and the file format can be PNG, JPG, BMP or TGA. Note that the image must fit in the memory of your graphics card.

If you want to use your own worldmap or one that is available on the SEAScope website, use a path to your image file as value for this setting.

The default value is `lowres`.

skipIndexCheck

Disable the detection of changes in the data directory at startup. Should be a boolean value (true|false), default value is false.

This can help the application start faster if the data directory contains a lot of IDF files but note that it prevents SEAScope from rebuilding the index, so it might lead to inconsistencies between the indexed information and the actual content of the data directory.

searchRequestMinDelay

Minimum number of seconds between two search requests. Should be a float value, default is 0.1.

SEAScope executes search requests every time you change the date, the timespan, the zoom or the variables to display.

Casually browsing data in SEAScope can generate a lot of requests and overwhelm the application, so SEAScope uses a throttling mechanism by postponing the processing of search requests if too many of them are created in a short time.

[processor] section

enabled

Enable processing options in SEAScope menus. Should be a boolean value (true|false), default value is true.

This setting tells SEAScope whether it should unlock the features supported by seascope-processor (i.e. the seascope-processor command is running) like generating a plot from a transect or exporting extracted data as a Numpy array.

address

IP address of the SEAScope processor. Should be an IPv4 address, or a computer name resolving to an IPv4 address, default value is 127.0.0.1.

Use 127.0.0.1 if the processor runs on the same machine as the SEAScope viewer.

port

Port that the SEAScope processor listens to. Should be an integer value, default value is 53450.

[drawing] section

explicitDrawingModeExit

Remain in drawing mode until the user presses the Escape key. Should be a boolean value (true|false), default value is false.

You can decide if SEAScope should remain in drawing mode until you choose to leave it (useful when drawing many shapes in a row) or if it should leave this mode as soon as you finish drawing a shape.

clickPixelsTolerance

Click tolerance expressed in pixels. Should be an integer value, default value is 7.

This setting allows you to define how many pixels your cursor can move between the press and release of the mouse button and the event still registers as a "click".

This is notably useful on high resolution screens where the slightest movement of the mouse causes the cursor to translate by a large amount of pixels.

Annex III: Collection configuration examples

III.1 AOML drifters

```
[general]
label = AOML drifters
xSeamless = false
ySeamless = false
mustBeCurrent = true
tags = type:drifter
variables = speed,sst

[sst]
label = sst
fields = temp
units = K
defaultRendering = TRAJECTORIES
logscale = false
min = 270.0
max = 315.0
opacity = 0.9
zindex = 0.96
particlesCount = 1000
streamlineLength = 1
colormap = medspiration
color = 0,0,0
tags = parameter:temperature
filterMode = NEAREST
billboardsDensity = 0.0
billboardsSize = 32
lineThickness = 1.0
streamlineSpeed = 1.0
currentTimeTrajectoryMarker =

[speed]
label = speed
fields = current
units = m/s
defaultRendering = TRAJECTORIES
logscale = false
min = 0.000000
max = 2.000000
opacity = 0.900000
zindex = 0.965099
particlesCount = 1000
streamlineLength = 1
colormap = jet
color = 0,0,0
tags = parameter:current
filterMode = NEAREST
billboardsDensity = 0.000000
billboardsSize = 32.000000
lineThickness = 1.000000
streamlineSpeed = 1.000000
currentTimeTrajectoryMarker = _diamond.png
```

III.2 ECMWF wind forecast

```
[general]
label = ECMWF
xSeamless = true
ySeamless = false
mustBeCurrent = true
tags = type:model, region:global
variables = mean_wind_field, mean_wind_field_raster, mean_wind_field_streamline

[mean_wind_field]
label = mean wind velocity barbs
fields = u10m, v10m
units = m/s
defaultRendering = BARBS
logscale = false
min = 0.000000
max = 25.000000
opacity = 0.900000
zindex = 0.940099
particlesCount = 1000
streamlineLength = 1
colormap = jet
color = 0,0,0
tags = parameter:wind
filterMode = NEAREST
billboardsDensity = 0.500000
billboardsSize = 32.000000
lineThickness = 1.000000
streamlineSpeed = 1.000000
currentTimeTrajectoryMarker =

[mean_wind_field_raster]
label = mean wind speed
fields = u10m, v10m
units = m/s
defaultRendering = RASTER
logscale = false
min = 0.0
max = 25.0
opacity = 0.9
zindex = 0.20
particlesCount = 1000
streamlineLength = 1
colormap = jet
color = 0,0,0
tags = parameter:wind
filterMode = BILINEAR
billboardsDensity = 0.0
billboardsSize = 32
lineThickness = 1.0
streamlineSpeed = 1.0
currentTimeTrajectoryMarker =

[mean_wind_field_streamline]
```

```
label = mean wind velocity streamlines
fields = u10m, v10m
units = m/s
defaultRendering = STREAMLINES
logscale = false
min = 0.0
max = 25.0
opacity = 0.9
zindex = 0.20
particlesCount = 50000
streamlineLength = 50
colormap = wind
color = 255,255,255
tags = parameter:wind
filterMode = NEAREST
billboardsDensity = 0.0
billboardsSize = 32
lineThickness = 1.0
streamlineSpeed = 1.0
currentTimeTrajectoryMarker =
```

III.3 ASCAT sea ice roughness

```
[general]
label = Ascat A/B
xSeamless = true
ySeamless = true
mustBeCurrent = true
tags = region:polar
variables = sea_ice_roughness

[sea_ice_roughness]
label = sea ice roughness
fields = sea_ice_roughness
units =
defaultRendering = RASTER
logscale = false
min = 0.0
max = 0.122409712058
opacity = 1.0
zindex = 0.401
particlesCount = 1000
streamlineLength = 1
colormap = pesket
color = 0,0,0
tags = parameter:ice
filterMode = NEAREST
billboardsDensity = 0.0
billboardsSize = 32
lineThickness = 1.0
streamlineSpeed = 1.0
currentTimeTrajectoryMarker =
```

III.4 SARAL / AltiKa sea level anomaly

```
[general]
label = SARAL SLA
xSeamless = false
ySeamless = false
mustBeCurrent = false
tags = type:altimeter
variables = sla

[sla]
label = sea level anomaly
fields = SLA
units = m
defaultRendering = TRAJECTORIES
logscale = false
min = -0.2
max = 0.2
opacity = 0.9
zindex = 0.988
particlesCount = 1000
streamlineLength = 1
colormap = jet
color = 0,0,0
tags = parameter:height
filterMode = NEAREST
billboardsDensity = 0.0
billboardsSize = 32
lineThickness = 1.0
streamlineSpeed = 1.0
currentTimeTrajectoryMarker =
```

III.5 Sentinel-2

```
[general]
label = Sentinel-2 RGB
xSeamless = false
ySeamless = false
mustBeCurrent = false
tags = type:satellite
variables = red, green, blue, rgb

[red]
label = red
fields = B04_TOA_reflectance
units = _
defaultRendering = RASTER
logscale = false
min = 0
max = 0.3
opacity = 1.0
zindex = 0.8
particlesCount = 1000
streamlineLength = 1
colormap = grayscale
color = 0,0,0
filterMode = BILINEAR
tags = parameter:red
billboardsDensity = 0.0
billboardsSize = 32
currentTimeTrajectoryMarker =
lineThickness = 1.0
streamlineSpeed = 1.0

[green]
label = green
fields = B03_TOA_reflectance
units = _
defaultRendering = RASTER
logscale = false
min = 0
max = 0.3
opacity = 1.0
zindex = 0.8
particlesCount = 1000
streamlineLength = 1
colormap = grayscale
color = 0,0,0
filterMode = BILINEAR
tags = parameter:green
billboardsDensity = 0.0
billboardsSize = 32
currentTimeTrajectoryMarker =
lineThickness = 1.0
streamlineSpeed = 1.0

[blue]
```

```
label = blue
fields = B02_TOA_reflectance
units = _
defaultRendering = RASTER
logscale = false
min = 0
max = 0.3
opacity = 1.0
zindex = 0.8
particlesCount = 1000
streamlineLength = 1
colormap = grayscale
color = 0,0,0
filterMode = BILINEAR
tags = parameter:blue
billboardsDensity = 0.0
billboardsSize = 32
currentTimeTrajectoryMarker =
lineThickness = 1.0
streamlineSpeed = 1.0

[rgb]
label = rgb
fields = B04_TOA_reflectance, B03_TOA_reflectance, B02_TOA_reflectance
units = _
defaultRendering = RAWRGB
logscale = false
min = 0
max = 0.5
opacity = 1.0
zindex = 0.8
particlesCount = 1000
streamlineLength = 1
colormap = grayscale
color = 0,0,0
filterMode = BILINEAR
tags = parameter:blue, parameter:red, parameter:green
billboardsDensity = 0.0
billboardsSize = 32
currentTimeTrajectoryMarker =
lineThickness = 1.0
streamlineSpeed = 1.0
```