# Ichthyop User Guide (3.3.17)

Nicolas Barrier  Philippe Verley  Gwendoline Andres
Christophe Lett

# Table of contents

# Preface

# Part I

# User guide

# 1 Getting started

In this section, download and install instructions are provided.

## 1.1 Prerequisites

### 1.1.1 Java

In order to run Ichthyop, **Java (>= 11)** needs to be installed. Beforehand, let us clarify some of the acronyms regarding the Java technologies.

`JVM`: Java Virtual Machine. It is a set of software programs that interprets the Java byte code.

`JRE`: Java Runtime Environment. It is a kit distributed by Sun to execute Java programs. A `JRE` provides a `JVM` and some basic Java libraries.

`JDK` or `SDK`: Java (or Software) Development Kit bound to the programmer. It provides a `JRE`, a compiler, useful programs, examples and the source of the API (Application Programming Interface: some standard libraries).

It is strongly recommended to download a `JDK`, in order to both compile and run the model. Builds for different platforms can be found here.

(nc-inst)=

### 1.1.2 NetCDF4

The Java library that manages input/outputs of NetCDF files requires the external NetCDF C library, which can be installed as follows:

#### 1.1.2.1 Mac Os X

To install the library on a Mac Os system, open a Terminal and type:

```
sudo port install netcdf4
```

#### 1.1.2.2 Linux

To install the library on a Linux system, open a Terminal and type:

```
sudo apt-get install netcdf4
```

#### 1.1.2.3 Windows

To install the library on a Windows system, download the pre-built libraries ib Unidata website

> **Caution**
>
> During the install process, make sure that the location of the library is added to the PATH

### 1.1.3 Conda environmenmt

There also is the possibility to use Conda environments in order to install Maven, OpenJDK and NetCDF4 easily. Instructions can be found on https://github.com/ichthyop/ichthyop-conda

## 1.2 Downloading Ichthyop

The Ichthyop model is available on GitHub. There is two ways to recover Ichthyop:

- Using executable files (`.jar` files).
- From source files.

### 1.2.1 Using executables

Ichthyop users can download Ichthyop executables here. Choose a version, and download the {samp}`ichthyop-X.Y.Z-jar-with-dependencies.jar` file (replacing {samp}`X.Y.Z` by the version number).

### 1.2.2 From source

To get the source code, type in a Terminal (Unix/MacOs) or Git Bash prompt (Windows):

```
git clone https://github.com/ichthyop/ichthyop.git
```

The code can then be compiled either using IDE (NetBeans, VSCode) or using the following command line:

```
mvn package
```

The executable will be generated in the `target` folder.

> **Warning**
>
> To use the command line, Maven needs to be installed (see instructions on https://maven.apache.org/install.html)

## 1.3 Downloading hydrodynamical files

To run Ichthyop templates, sample hydrodynamical files are required (`input` folder). They were previously part of the GitHub repository but have been migrated to Zenodo. They can be downloaded here.

## 1.4 Running Ichthyop

### 1.4.1 Clicking on file (Windows)

Open the Ichthyop folder and double click on the `ichthyop-X.Y.Z-jar-with-dependencies.jar` file, where `X.Y.Z` is the Ichthyop version. You should see the Ichthyop console.

### 1.4.2 From command line (Unix/Mac Os X)

Open a command line prompt (Terminal or CMD prompt) and navigate to the Ichthyop folder using `cd`.

Then, type:

```
java -jar ichthyop-X.Y.Z-jar-with-dependencies.jar
```

with `X.Y.Z` the Ichthyop version.

This will prompt the Java console. In order to run Ichthyop without the console, you need to specify a supplementary argument, which is the XML configuration file.

```
java -jar ichthyop-X.Y.Z-jar-with-dependencies.jar cfg-roms3d.xml
```

# 2 Ichthyop configuration

## 2.1 Simulation configuration file

Ichthyop simulations are configured using XML configuration files. It should always start as follows:

```
<icstructure>
<long_name>Generic Ichthyop configuration file</long_name>
<description>The file has few pre-defined parameters.</description>

</icstructure>
```

### 2.1.1 Configuration blocks

Ichthyop is configured by blocks, each block managing a specific aspect of the model. The blocks are as follows:

- ACTION: Block of parameters related to the `action` classes (cf. {numref}`process`).
- RELEASE: Block of parameters related to the `release` classes (cf. {numref}`release`)
- DATASET: Block of parameters related to the `datasets` classes.
- OPTION: Block of parameters related to the remaining parameters.

New blocks can be added in the XML file as follows:

```
<block type="option">
        <key>app.transport</key>
        <enabled>true</enabled>
        <tree_path>Transport/General</tree_path>
        <description>Set the general transport options of the simulation.</description>
</block>
```

The `key` tag is used to identify the configuration block. The `tree_path` tag is used in the Ichthyop console to create the parameter tree. The `description` field is used to display the block description in the Ichthyop console.

The enabled tag specifies whether the block should be considered by Ichthyop or not. By default, all blocks are enabled. But for the RELEASE and DATASET, one and only one block must be activated.

### 2.1.2 Configuration parameters

To each block is associated a list of parameters. This list of parameter is added in the XML as follows:

```
<parameters>
</parameters>
```

Inside the parameters tags, new parameters are defined as follows:

```
<parameter>
    <key>output_path</key>
    <value>output</value>
    <long_name>Output path</long_name>
    <format>path</format>
    <default>output</default>
    <description>Select the folder where the simulation NetCDF output file should be saved.</desc
</parameter>
```

The key tag allows to identify the parameter, while the value tag specifies the value of the parameter. The remaining tags are only used by the Ichthyop console. The long_name and description tags are used by the console to provide informations about the parameter.

The format tag specifies the parameter format, which will be used by the console parameter editor. The accepted values are:

- path: For files and folders
- date: For dates (format must be year YYYY month MM day at HH:MM)
- duration: For duration (format must be ###### day(s) ### hour(s) ### minute(s))
- float: For real values
- integer: For integer values.
- class: For class parameters. It allow the user to choose an existing Ichthyop class in the configuration file.
- list: For a list of string parameters, separated by ,
- boolean: For boolean parameters. It allows the user to select true or false using a simple combo box.
- combo: For parameters with a limited set of values, which can be selected in the console with a combo box.
- lonlat: For geographical coordinates.

In the case of `combo` parameters, the list of accepted parameters is specified by providing as many `accepted` tags as necessary. For instance:

```
<parameter>
    <key>time_arrow</key>
    <long_name>Direction of the simulation</long_name>
    <value>forward</value>
    <format>combo</format>
    <accepted>backward</accepted>
    <accepted>forward</accepted>
    <default>forward</default>
    <description>Run the simulation backward or forward in time.</description>
</parameter>
```

If a parameter should appear as hidden in the Ichthyop console, it can be specified by adding the `hidden="true"` argument to the `parameter` tag, as shown below:

```
<parameter hidden="true">
</parameter>
```

### 2.1.3 Serial parameters

In order to test different values for a given parameter, a `serial` tag can be added as follows:

```
<parameter type="serial">
</parameter>
```

Different values are provided by replicating the `value` parameter as follows:

```
<parameter type="serial">
  <key>initial_time</key>
  <long_name>Beginning of simulation</long_name>
  <value>year 2001 month 10 day 20 at 00:00</value>
  <value>year 2001 month 10 day 21 at 00:00</value>
  <format>date</format>
  <description>Set the beginning date and time of the simulation. Format: year ###### month ### d
</parameter>
```

> **Caution**
>
> **Using the GUI, additionnal values can be provided to serial parameters only when hidden parameters are displayed**

When simulations are run with serial parameters, all possible combinations of parameters will be run. Output files will contain a _sX suffix, with X the simulation number. The parameters used in the simulation are provided as global NetCDF attributes.

## 2.2 Zone configuration file

Zone configuration files are also managed via a dedicated XML file. The file must be as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<zones>

</zones>
```

Each zone is defined on a `zone` tag, which contain the following tags:

- `key` is the name of the zone
- `enabled` specifies whether this zone must be considered or not.
- `type` specifies whether the zone should be used for release (see :numref:, ``release` value) or recruitment processes (`recruitment` value)
- `polygon` specifies the different points used to define the area
- `bathy_mask` specifies the bathymetric zone (for instance 0 to 200m, i.e. continental shelf) where the zone is defined.
- `thickness` specifies the upper and lower depths where this zone is defined (**only valid for 3D runs**).
- `color` specifies the display color of the zone (format is `[r=102,g=51,b=255]`).
- `proportion_particles` specifies the proportion (values in $[0-1]$) of particles to be released in the area. Only used when `type` is `release` and if the `user_defined_nparticles` parameter is set to True (cf. {numref}`zone-release`)

An example of a zone definition is provided below.

```
<zone>
    <key>Release zone 2</key>
    <enabled>true</enabled>
    <type>release</type>
    <polygon>
```

```
        <point>
            <index>0</index>
            <lon>54.0</lon>
            <lat>-11.5</lat>
        </point>
        <point>
            <index>1</index>
            <lon>54.0</lon>
            <lat>-12.5</lat>
        </point>
        <point>
            <index>2</index>
            <lon>53.0</lon>
            <lat>-12.5</lat>
        </point>
        <point>
            <index>3</index>
            <lon>53.0</lon>
            <lat>-11.5</lat>
        </point>
    </polygon>
    <bathy_mask>
        <enabled>true</enabled>
        <line_inshore>0.0</line_inshore>
        <line_offshore>12000.0</line_offshore>
    </bathy_mask>
    <thickness>
        <enabled>true</enabled>
        <upper_depth>0.0</upper_depth>
        <lower_depth>50.0</lower_depth>
        </thickness>
    <color>[r=102,g=51,b=255]</color>
    <proportion_particles>0.2</proportion_particles>
</zone>
```

## 2.3 Time configuration

### 2.3.1 Beginning of the simulation

In Ichthyop, the user provides the time at which the simulation should start. This `initial_time` parameter, which must be defined in the `app.time` option block, must be formatted as `year YYYY`

`month MM day DD at HH:MM`, with `YYYY` the year, `MM` the month, `DD` the day, `HH` the hour and `MM` the minutes where the simulation should start.

### 2.3.2 Reading NetCDF times

When reading a NetCDF file (ocean currents, temperature, wind, wave, etc.), Ichthyop will determine the units in which NetCDF time is stored. These units must meet the CF Metadata Conventions and therefore be provided as follows:

```
UNITS since YYYY-MM-DD HH:MM:SS
```

with `UNITS` the units in which the time is stored (usually `seconds`, `days` or `hours`), `YYYY` the year, `MM` the month, `DD` the day, `HH` the hour, `MM` the minutes and `SS` the seconds of the reference date.

If a NetCDF `time::units` attribute is defined, Ichthyop will try to infer the NetCDF reference date and time units using this convention.

If it fails (i.e. the `time::units` attribute does not follow the convention) or if no `time::units` attribute is found, Ichthyop will read the `time_origin` parameter from the `app.time` option block, which must be defined following the CF conventions.

> **Caution**
>
> When reading two datasets (an ocean currents dataset and a wind dataset for instance), if none meets the CF convention, Ichthyop will apply the units defined in the `time_origin` parameter to both datasets, even though they may have different time units. **Therefeore, in this case, it is strongly recommended to manually include CF-like time units attributes to each dataset (cf. below).**

Manually updating the `units` attribute can be done by using the ncatted command (**Linux users**):

```
#!/bin/bash
for file in *nc
do
    ncatted -O -a units,time,o,s,"seconds since 1900-01-01 00:00:00" $file
done
```

This can also be done by using Python as follows:

```
from glob import glob
import xarray as xr

units = 'seconds since 1900-01-01 00:00:00'
time = 'time'

filelist = glob('*nc')

for f in filelist:

    data = xr.open_dataset(f)
    data[time].attrs['units'] = units
    data[time]

    data.to_netcdf(f)
```

> **Note**
>
> The user must replace `time` by the name of the time variable which is used by Ichthyop. Common values are `time`, `scrum_time`, `time_counter`, `ocean_time`.
> The units value must also be chosen consistently with the dataset

# 3 Ichthyop console

In this section, the Ichthyop console is described.

## 3.1 Configuration

Here you will find the usual “File menu” functions : create a new configuration file, open an existing one, close it or save and save as the configuration file.

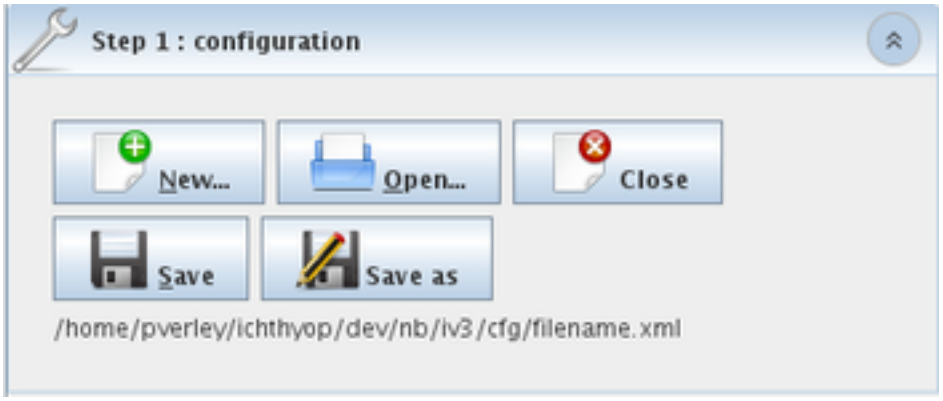



Figure 3.1: Step 1, configure

### 3.1.1 New configuration file

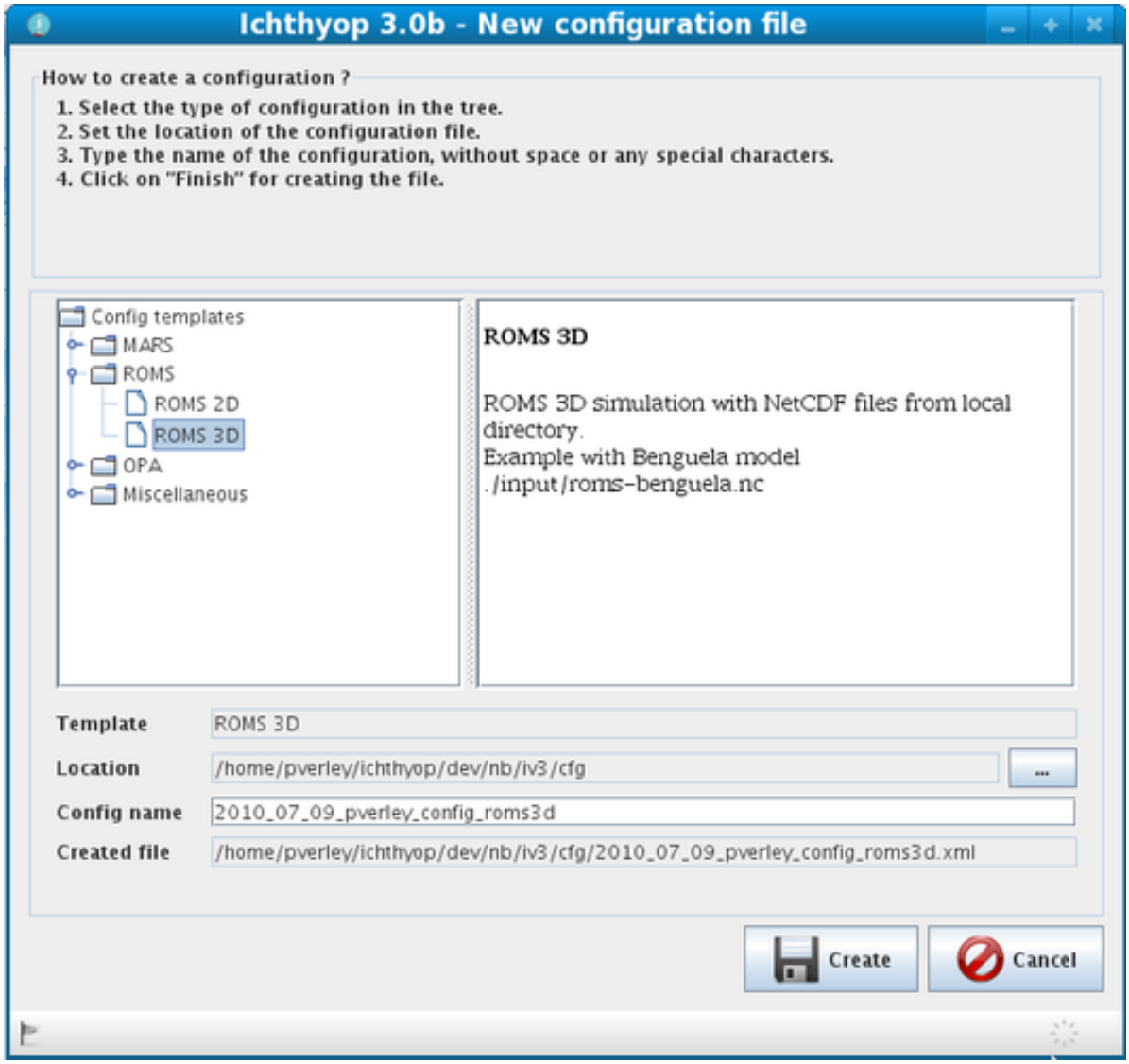



Figure 3.2: Step 1, create a new configuration file

The application comes with some preset examples of configuration files (the templates). Select one of the templates, change the name of the configuration if the suggested name does not suit you and click on the Create button.

### 3.1.2 Content of the configuration file

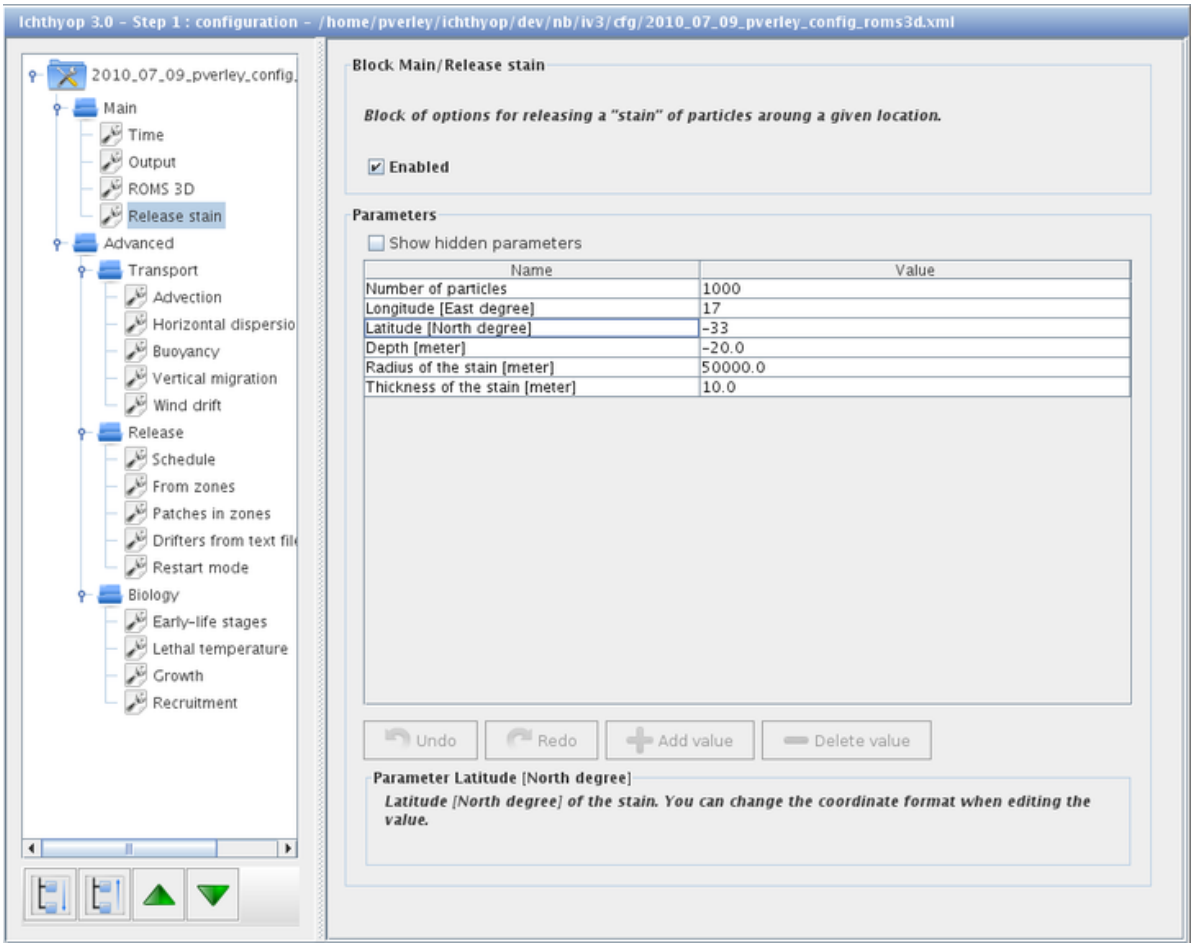



Figure 3.3: Step 1, edit the configuration file

The configuration file is organized in several categories and each category contains several blocks of parameters.

When the configuration file is created out of a template, it is ready to use and you do not need to change any parameter for running the simulation. Little by little you can explore the configuration file, starting with the "Main" blocks and change some parameters to see what is happening. On a second time you can start playing with the advanced parameters that activate and control the behaviors of the particles.

Main blocks:

- Time: Set the simulation time options, such as the beginning of the simulation, the duration of transport, the time step, etc.
- Output: Options that control the record of the particle tracks in a NetCDF output file.
- Dataset: Management of the hydrodynamic dataset.
- Release: Determine how and where the particles should be released.

Advanced blocks:

- Transport: Parameters for controlling the advection process, the dispersion, the vertical migration, the wind drift, etc.
- Release: Additional ways for releasing particles, from zones, with position recorded in a text file or in a NetDCF file, etc.
- Biology: Control the biological processes such as growth or cold water sensitivity, etc.

Each block is fully described and commented in the block information area. As well, you will find a description and the necessary explanations for each parameter in the parameter information area.

> **Warning**
>
> Do not forget to save the configuration file before going to the next step.

> **Caution**
>
> **Parameters cannot be added from within the Ichthyop console. Only paramters that are already defined on the XML file can be edited using the console**

## 3.2 Zone definition

In Ichthyop, the user can define zones, either release zones or recruitment zones.

The zones can be edited using the GUI, as shown in Figure 3.4

### 3.2.1 Adding, removing and renaming zones

The number of zones is managed on the left part of the panel.

New zones can be added by clicking the  button. When a zone is selected, it can be removed by clicking on the  button.

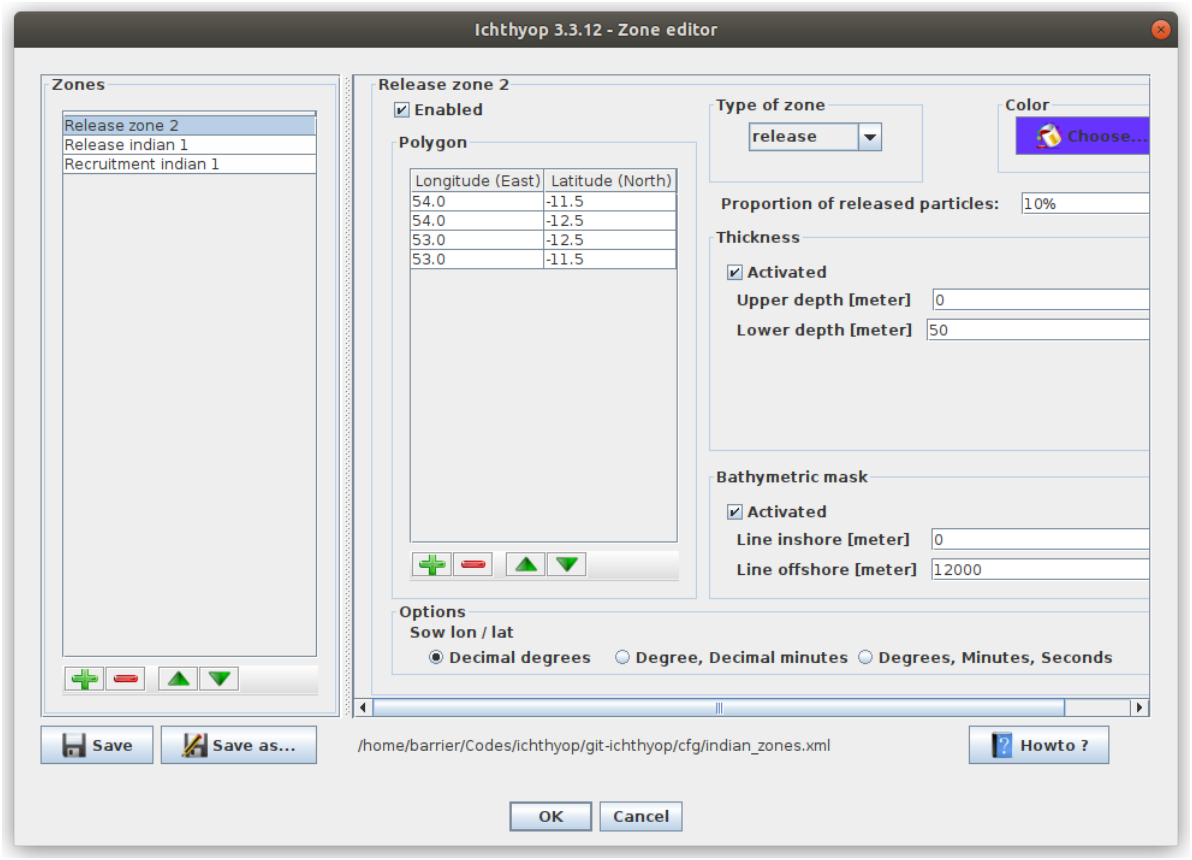
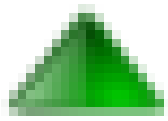
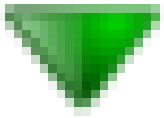



Figure 3.4: Ichthyop Zone editor

The reordering of the zones is achieved by clicking on the and buttons.

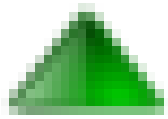
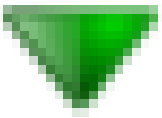
When double-clicking on the name of the zone on the left panel, the user can edit the zone name.

### 3.2.2 Editing a zone

When a zone is selected, the user can edit different parameters associated with the zone.

First, the user can enable or disable a zone by clicking on the {guilabel}`Enabled` tick box.

The zones are defined by providing the points coordinates. Points can be added, removed and re-

ordered by using the  ,  , and buttons, respectively. The user can also change the format of the points coordinates by clicking on the radio buttons in the `Options` bottom panel.

Ichthyop defines two types of zones: one for release (see Section 5.2) problèmeand one for recruitment purposes. This type is chosen by using the `Type of zone` combo box. For release zones, the user can specify the number of particles that will be released in the zone (only if the `user_defined_nparticles` parameter is set equal to true, cf Section 5.2). It is done by filling the `Number of released particles` textbox and pressing `ENTER`

Each zone is associated with a color, that will be used to its representation in the graphical interface during the preview and the display of the simulation results. This color can be edited by using the

 button.

In the case of 3D simulations, you can specify the depth range to use in the zone. To activate this feature, click on the `Activated` tick box of the `Thickness` panel. You can provide the lower and upper depth that must be considered in the given zone (negative values).

In 3D simulations, you can also specify the bathymetric range that you want to include, for instance if you want to release particles only on the ocean shelf (i.e depth less than 200m). This can be done by activating the feature by clicking on the `Activated` tick box of the `Bathymetric mask` panel.

## 3.3 Running


Figure 3.5: Step 2, simulation

You may want to preview the simulated area. Click on the "Preview" button. The main interest in previewing the area is that the application will check if the simulation is correctly set up. Here "correctly" does not mean you made a relevant parametrization in terms of physics or biology, but at least the application had found all the parameters required for starting the simulation. More specifically, Ichthyop will attempt to read the geographical variables (longitude, latitude and depth) from the dataset in order to draw the area. It should also display the release and the recruitment zones if they have been defined and activated in the configuration file. Make sure what you see is what you expect, and go back to "Step 1: configure" in case not.

When the preview is satisfactory, click on "Run simulation" for starting the simulation. The progress bar will give an estimation of the remaining time for the simulation to complete. You can interrupt (but not pause) the simulation anytime by a click on "Stop simulation".

Depending the capabilities of your computer, the number of released particles, how many actions are implemented, etc. the simulation might requires a large amount of the available dynamic memory and the application might look like it is frozen. Wait until the simulation run to completion. Refer to section "Java Heap Space" if the application crashes because of memory problem.

## 3.4 Visualize results

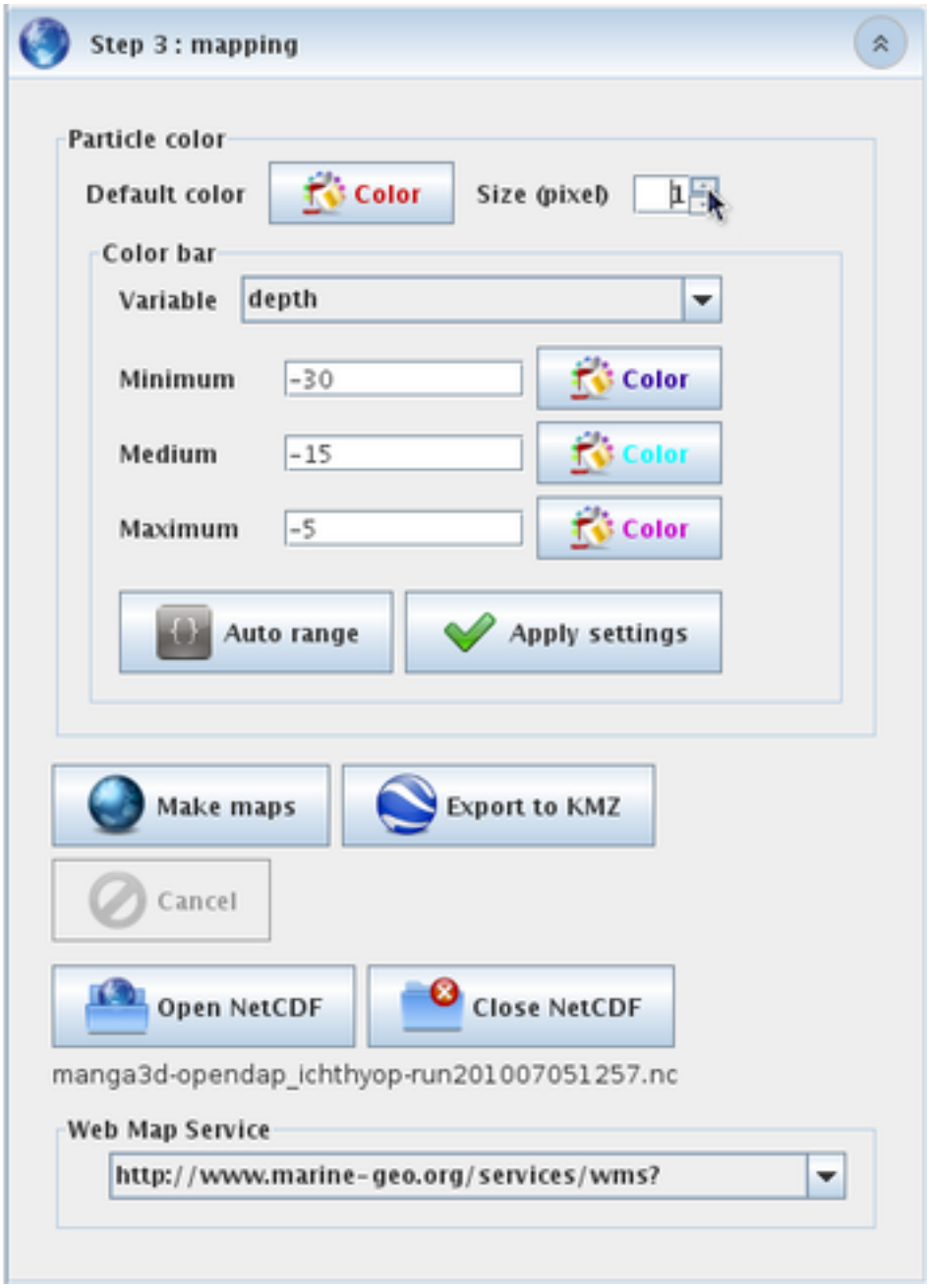



Figure 3.6: Step 3, mapping

When the simulation is completed, the application automatically opens the current Ichthyop output file for visualizing the results. If your computer is connected to Internet, you should see the map being centered above the simulated area. Otherwise, it only displays a Grey background.

You may want to skip that step or keep it for later. In that case, just click on "Close NetCDF" and go to any other steps or exit the application. Any time, you can go back to this step: click on "Open NetCDF" and select the Ichthyop output file you wish to visualize. When the NetCDF file is opened, the application brings you back to the exact point where it was when the simulation just completed.

The application offers two ways for visualizing the results : draw the particle trajectories with a Web Map Service or export the particle trajectories in a KMZ file that can be opened with Google Earth. Both functions are completely independent one from another.

### 3.4.1 Results

Ichthyop archives the particle trajectories in NetCDF format, a machine-independent data formats for sharing array-oriented scientific data. The NetDCF file is recorded in the output folder (set in the Output section of the configuration file) and the file name contains the date and time of creation of the file.

Default contents of the NetCDF output file: time of the simulation, longitude, latitude, and depth at particle position, and mortality status.

### 3.4.2 Set particle color

The `Default color` button determines the particle color for visualizing the trajectories.

Particles are plotted as small circles. `Particle size` determines the diameter of the circle in pixel.

To use a colorbar, select in the Combo box a variable archived in the Ichthyop output file you wish to visualize as a tri-color range. The `Auto range` button will scan the values of the variable and suggest the following range : [mean - 2 * standard deviation; mean + 2 * standard deviation]. Do not forget to click on `Apply settings` for validating the changes of the color bar.

For taking off the color bar, select the `None` item in the Combo box and click on `Apply settings`. A click on `Default color` button should also deactivate the color bar.

### 3.4.3 Make maps using Web Map Service

According to Wikipedia, a Web Map Service (WMS) is a standard protocol for serving georeferenced map images over the Internet.

Ichthyop provides three different WMS for displaying the ocean bathymetry and the cost line as a background of the particle trajectories.

Maps can be intuitively zoomed in and out with the mouse wheel and re-centered doing a mere drag and drop.

Depending on the quality of the Internet connection and how busy is the Web Map Server, the display of the background tiles might take a while or even not work at all. In that case, try again with a distinct WMS and change the zoom scale.

When the settings of the map looks satisfactory, click on “Make maps” button. Ichthyop will create a folder that has exactly the same name than the simulation NetCDF output file (without the .nc extension) in the output directory. Then maps are recorded in this folder as PNG pictures.

Again, depending on the computer capabilities and the number of particles, the map creation might require a large amount of the available dynamic memory and the application looks like it is frozen. Wait for the application to complete this step. Refer to section “Java Heap Space” if the application crashes because of memory problem.

### 3.4.4 Export trajectories to KMZ format

By default, Ichthyop records the particle trajectories in NetCDF format. It is perfectly adapted for archiving and sharing scientific data since it is a machine independent and array-oriented format. But it is not much handy for visualizing results.

Click on “Export to KMZ” button for recording the particle trajectories into a KMZ file. The file is recorded in the same directory than the Ichthyop output file, with the same name and the “.kmz” extension. KMZ format is the standard file format for visualizing georeferenced information with GoogleEarth.

Color settings (default color or color bar) and particle size will also be stored in the KMZ file.

When the export has performed, browse to the output folder and click on the KMZ file for launching GoogleEarth (assuming the program is installed on your computer).

## 3.5 Animation

Here you will find the usual “File menu” functions : create a new configuration file, open an existing one, close it or save and save as the configuration file.

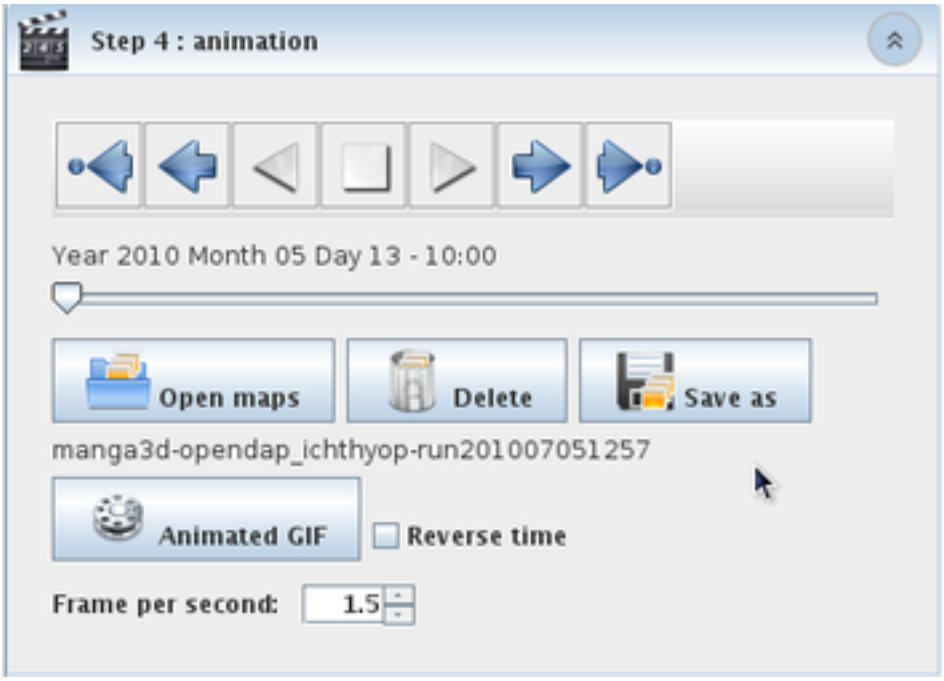



Figure 3.7: Step 4, Animation

You may want to skip that step or keep it for later. In that case, just go to any other steps or exit the application. Any time, you can go back to this step, click on “Open maps” and select the simulation output folder that contains the PNG pictures you wish to visualize. When the folder is opened, the application brings you back to the exact point where it was when the map creation just completed.

Set the number of frames per second of the animation with the spinner.

You can also create an animated GIF. The file is recorded in the same directory than the Ichthyop output file, with the same name and the “.gif” extension.

# 4 FAQ

## 4.1 What to do when Ichthyop does not manage my ocean dataset

Ichthyop developers cannot manage all the different ocean datasets that exist. First, there are too many of them which rely on different assumptions, such as grid layout, vertical coordinates, etc.

Therefore, the Ichthyop developers have decided to first focus on the most used datasets, i.e. NEMO, MARS, ROMS and ocean datatasets that are stored on regular grid.

If your model does not belon to the list, one possibility is to do a bit of pre-processing, in order to convert your input files to a regular, depth-based ocean grid. Different tools can help you with that, such as the XESMF Python package.

## 4.2 What to do when Ichthyop suddenly fails to launch ?

Situation: without apparent reason, Ichthyop fails to launch, either from GUI or command line. It looks like it starts and crashed instantly.

Solution: delete Ichthyop persistence files (the files that store the information to restart your ichthyop session as it was when you last closed it) In Windows environment such files are gathered in a hidden directory ~/AppData/Local. In this folder delete the ichthyop directory of the previmer/ichthyop directory. In Linux or Mac environement delete the ~/.ichthyop folder. Try to run Ichthyop again.

## 4.3 How to launch Ichthyop from command line ?

First you have to open a command prompt window on your computer.

For Linux or Mac users open a new Terminal (type Terminal in the Application search bar or the Finder if you are unsure how to open a terminal).

For Windows users, click on the start button > All programs > Accessories > Command Prompt (read more)

From the command prompt windows you need to change the current directory (by default your home directory) to the Ichthyop directory. For instance

```
cd projects/ichthyop/ichthyop-3.2
or
cd Mes\ Documents/Ichthyop/ichthyop-3.2
```

Assuming that in the first case that your ichthyop folder is in directory projects/ichthyop/ichthyop-3.2 and in the second case in directory MesDocuments/Ichthyop/ichthyop-3.2.

List the folder content to check that you are in the right directory : type dir in Widows and ls in Linux or Mac environment.

Launch Ichthyop, with UI, from command line:

```
java -jar ichthyop-3.2.jar
```

Launch Ichthyop, without UI, from command line:

```
java -jar ichthyop-3.2.jar cfg/your_configuration_file.xml
```

With your_configuration_file.xml the name of the XML configuration files that you first created from the UI and that you saved in the cfg/ folder.

## 4.4 What are the different coastline behaviours in Ichthyop?

First you may want to read {numref}`spatial-int`

Coastline behaviour manages what must be done in the event that the move of a particle takes it inland (which might happen because the simulation time step is not small enough or because there is some additional movement, such as diffusion or swimming, etc.)

Ichthyop offers four different behaviours at coastline:

- NONE: Ichthyop ignores the fact that it is land and just carries on moving the particle around.
- BEACHING: Ichthyop does move the particle inland but “kill” it. From now onward the particle is out of the simulation.
- BOUNCING: the coastline acts as a billard edge and the particle will bounce as a billard ball in the events that the move would take it beyond the coastline. The particle bounces back as much as it would penetrate inland.
- STANDSTILL: the particle gives up on the move that would take it inland and just wait until next time step for trying an other move.

(spatial-int)=

## 4.5 How does spatial interpolation work in Ichthyop?

A particle in Ichthyop only knows about its environment what is provided by the outputs of the hydrodynamic model. Such information, for instance the zonal and meridional current velocities, are usually provided on an Arakawa C-grid, every U-point and V-point respectively.

Here is the 2D scheme of cells (i, j) bottom left, (i+1, j) bottom right, (i, j+1) top left and (i+1, j+1) top right.

:::{figure} _static/spatial_interpolation.png :align: center

Particle current location at {samp}`X(x, y, z)` :::

Let's see how the interpolation works for both zonal and meridional velocities. The question we ask is what is the value of U and V at particle location?

We have i=round(x), j=truncate(y) and k=truncate(z), dx=x-i, dy=y-j, dz=z-k

Let's call t, the current time of the simulation, and t0 and t1 the values of the time NetCDF variable bounding t: t0 <= t < t1

We first interpolate the model velocity field at t0:

:::{figure} _static/spatial_interpolation_u(1).png :align: center :::

This large expression can be narrowed down to:

```
U(t0, x, y, z) = SUM( U(t0, i+ii-1, j+jj, k+kk) * |(0.5-ii-dx) * (1-jj-dy) * (1-kk-dz)| , ii in [0,1], j
```

with i=round(x), j=truncate(y), k=truncate(z), dx=x-i, dy=y-j, dz=z-k

Similarly, the meridional velocity can be expressed as:

```
V(t0, x, y, z) = SUM( U(t0, i+ii, j+jj-1, k+kk) * |(1-ii-dx) * (0.5-jj-dy) * (1-kk-dz)| , ii in [0:1], j
```

with i=truncate(x), j=round(y), k=truncate(z), dx=x-i, dy=y-j, dz=z-k

It means that the velocity, either zonal or meridional, at particle location is the result of a trilinear interpolation of the height (four above, four below) surrounding velocities in the grid.

Same with U(t1, x, y, z) and V(t1, x, y, z)

Let's take frac = (t - t0) / (t1 - t0). Then we have

```
U(t, x, y, z) = (1 - frac) * U(t0, x, y, z) + frac * U(t1, x, y, z)
```

```
V(t, x, y, z) = (1 - frac) * V(t0, x, y, z) + frac * V(t1, x, y, z)
```

This is the general case when all the surrounding cells are in water. Now what happened if the particle is in a cell adjacent to the coast? Let's say that in our example cell(i+1,j) and cell(i+1, j+1) are land. Basically the interpolation is limited to the four (two above and two below) closest surrounding velocity points:

```
U(t0, x, y, z) = SUM( U(t0, i+ii-1, j, k+kk) * |(0.5-ii-dx) * (1-dy) * (1-kk-dz)| , ii in [0,1], kk in [0

V(t0, x, y, z) = SUM( U(t0, i, j+jj-1, k+kk) * |(1-dx) * (0.5-jj-dy) * (1-kk-dz)| , jj in [0:1], kk in [0
```

In order to determine whether a particle is close to the coastline, Ichthyop proceeds in two steps: it first determines in which quater of the cell the grid point is located. Then it checks wether or not the three adjacent cells to this quater are in water.

:::{figure} _static/spatial_interpolation_coast.png :align: center :::

X1 will be considered as "close to coast" if any of cells (i,j+1) (i-1,j) (i-1,j+1) is on land.

X2 will be considered as "close to coast" if any of cells (i,j+1) (i+1,j) (i+1,j+1) is on land.

X3 will be considered as "close to coast" if any of cells (i,j-1) (i+1,j) (i+1,j-1) is on land.

X4 will be considered as "close to coast" if any of cells (i,j-1) (i-1,j) (i-1,j-1) is on land.

## 4.6 How does Ichthyop manage time ?

Time management is tricky to handle because on the computer side a given time is usually expressed as a number of seconds elapsed since a time origin (e.g. 13629116520 seconds elapsed between 1900/01/01 00:00 and 2014/09/04 09:42), whereas the user expects to read time in a human readable format (e.g. 2014/09/04 09:42).

Ichthyop is no different: the program itself only understands a time as a number of seconds elapsed since a time origin, just like the hydrodynamic datasets ROMS, MARS, NEMO, etc. and time displayed in the console or the GUI uses a human readable format. Since time in the hydrodynamoc dataset is expressed as a number of seconds elapsed since a time origin, Ichthyop must be able to convert a human readable time (for example the time of begining of the simulation) into a number of seconds, so that it can compare this given time value to the time vector of the hydrodynamic dataset and interpolate the velocity fields at the correct time step. The key issue is how to convert a human readable time into a number of seconds elapsed since a time origin ?

In order to do so, we need a calendar that basically details how many days (a day is always considered as a 24h period) are there in each month for each year. Since Ichthyop has to read some variables from the hydrodynamic dataset at a given time, we must make sure that Ichthyop uses the same calendar than the hydrodynamic dataset.

The default calendar used by Ichthyop is the Gregorian calendar (the most widely used civil calendar), the one we use for our daily life. The time origin is set by default at 1900/01/01 00:00. This value can be changed in the configuration file, in the {guilabel}`Time section`: tick the {guilabel}`Show hidden parameters` checkbox, change parameter {guilabel}`Type of calendar` to *Gregorian calendar* and adjust the value of parameter {guilabel}`Origin of time` so that it matches the origin of time set in the hydrodynamic dataset. Such information usually comes as an attribute of the time variable in the NetCDF output files of the hydrodynamic dataset.

Nonetheless some hydrodynamic simulations run with a different calendar than the Gregorian calendar. So far, Ichthyop includes an other calendar that we called the *Climatology calendar*. It is a commonly used calendar for climatological simulations, a year is divided in *12 months of 30 days each*. In order to select this calendar from the editor of configuration, go to the {guilabel}`Time section`, tick the {guilabel}`Show hidden parameters` checkbox and select the *Climatology calendar* for parameter {guilabel}`Type of calendar`. The origin of time for the climatology calendar is set at 01/01/01 00:00 and cannot be changed.

Let's sum up the steps involved in the time management, using the example of the time of beginning of the simulation:

- user provides a time for the beginning of the simulation 2014/09/04 09:42 ;
- user sets up the calendar to be used in Ichthyop, the same one that has been used in the hydrodynamic dataset, e.g. Gregorian calendar with origin of time 1900/01/01 00:00 ;
- Ichthyop converts 2014/09/04 09:42 into a number of seconds using the user-defined calendar, 13629116520 seconds ;
- Ichthyop scans the time variable of the hydrodynamic dataset and identifies that time value 13629116520 falls in between time step 5 and 6 of the hydrodynamic time step (time step 5 and 6 are just an example) ;
- Ichthyop can perfom the time integration of the velocity fields between time steps 5 and 6 of the hydrodynamic dataset and starts advecting the particles.

We provide a very simple utiliy programs in the Time converter repository, that illustrates how Ichthyop performs the time conversion, given a calendar and a time of origin.

Last bu not least: what if your hydrodynamic dataset uses an other calendar than Gregorian or Climatology calendars? Two options:

1. Contact the developpers and ask how much work would it be to include your calendar in Ichthyop ?
2. Select an existing calendar that is the most similar to yours and trick Ichthyop by providing a human readable time that you know it will be converted in the correct time value for the hydrodynamic dataset (thanks to the **Time converter** utility program).

## 4.7 How does Ichthyop interpolate the hydrodynamic dataset in time ?

Let's say that the hydrodynamics output dataset is archived with a 5 days time step and Ichthyop runs with a 1 hour time step.

Let's call tn a given time index in the hydrodynamic dataset and tnp1 the following one. And let's call Tr a variable from the dataset (either current velocity, temperature, free surface elevation, etc.). Let's call time the time variable of the hydrodynamic dataset, always expressed in seconds elapsed from a given origin.

Ichthyop does a linear interpolation to estimate the value of Tr at any given time between time(tn) and time(tnp1). For t, a given time index such as time(t) >= time(tn) and time(t) < time(tnp1), we have :

```
Tr(t) = (1 - x) * Tr(tn) + x * Tr(tnp1)
```

with `x = ( time(t) - time(tn) ) / ( time(tnp1) - time(tn) )`

## 4.8 Why do I get warning "CFL broken for W 1.208" ?

CFL stands for Courant–Friedrichs–Lewy condition. It is a necessary condition for stability while solving the equation of movement in Ichthyop.

We want at all time `(U * dt) / dX` stricly inferior to 1 with U the velocity (vertical in this specific case) and dX the move. **The warning informs you that the CFL condition has been broken for the vertical velocity and that it might jeopardize the numerical stability of the model.**

Try to decrease the time step (dt) in your configuration file (Time > Time step), let's say divide it by two. As explained in the configuration editor, an acceptable estimation for dt could be dt = 0.7 * dGrid / Umax with dGrid the average length of the grid cells and Umax the order of magnitude of the fastest current velocities in the hydrodynamic model (locally and punctually the vertical velocities can be intense). Nonetheless the smaller the better and as long as decreasing the time step does not slow down too much your simulations, you should always go for a smaller value that the previous estimation.

# Part II

# Documentation

# 5 Particle release

In the present section, the different release processes that are implemented within Ichthyop are described. The parameters that are associated with the release processes must be included within `release` blocks (cf. Section 2.1).

## 5.1 Stain release

The release stain (`StainRelease.java`) consits of releasing particles within a given circle. The user provides the longitude and latitudes of the release stain, the radius of the release stain (in *m*) and the number of particles to release.

For a 3D simulation, the user also provides the depth and the thickness of the release stains (both in *m*).

An example of a stain release is provided in Figure 5.1

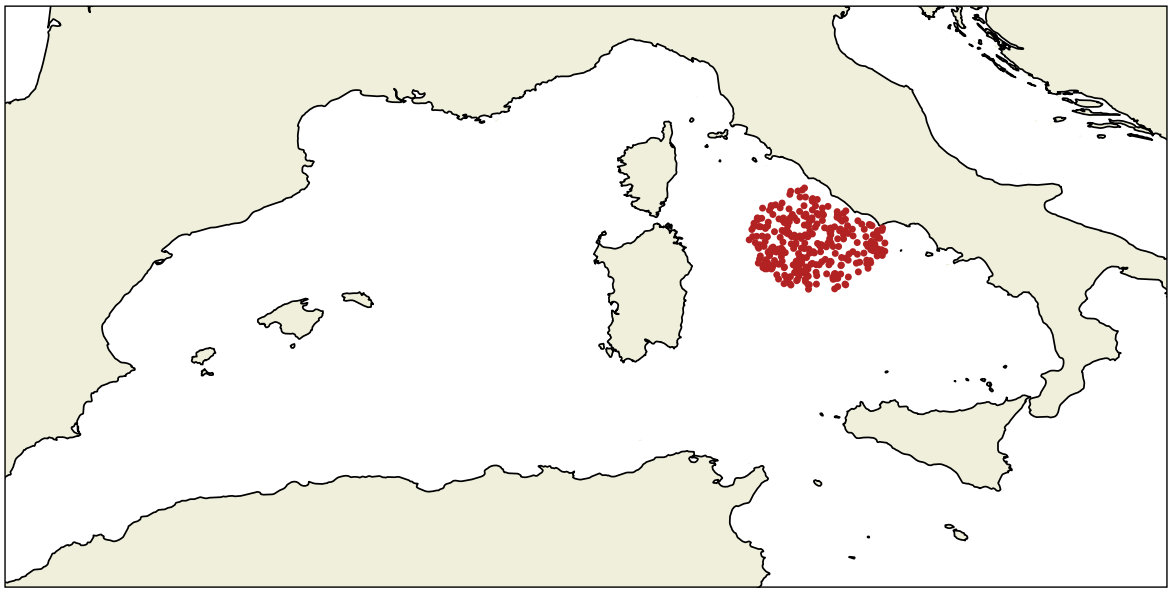


Figure 5.1: Example of a stain release.

## 5.2 Zone release

The zone release method (`ZoneRelease.java`) allows to release particles in different areas, which are defined in an XML zone file (`zone_file` parameter).

By default, the number of particles released in each zone ($N_k$) is equal to

$$N_k = N_{tot} \times \frac{S_k}{\sum_{i=1}^{N} S_i}$$

with $N_{tot}$ the total number of released particles, $S_k$ the surface of the $k^{th}$ release area and $N$ the number of release areas.

However, if the `user_defined_nparticles` parameter is set equal to `true`, then the proportion of the particles to release in each zone is defined by the user.

An example of zone release is provided below.

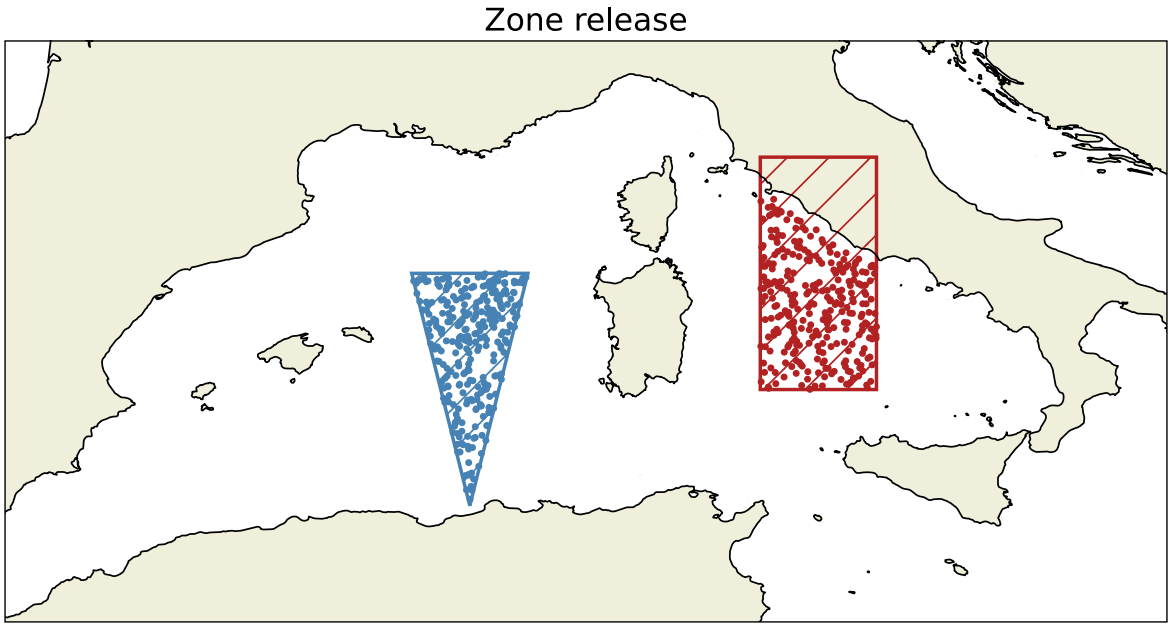



Figure 5.2: Example of a zone release.

## 5.3 Text release

If the user wants to simulate some specific trajectories, for instance DCPs or buoys, particles can be relased by providing a text file containing the release coordinates (`TxtFileRelease.java`). The

name of the text file is given by the `txtfile` parameter.

The file must be formatted as follows:

```
# 3D simulations
# longitude latitude depth
-5.45 48.30 -5
-5.45 48.30 -10
-5.45 48.30 -15
-5.45 48.30 -20
```

Each line is a drifter. Each character starting with # is considered as a comment. If the depth column is not provided, depth will be set equal to 0.

For 2D simulations, the depth column will be ignored.

> **Caution**
>
> Note that the columns must be separated by spaces.

## 5.4 Patchy release

The patchy release method (`PatchyRelease.java`) can be viewed as an extension of the release stain method, where several stains are randomly created.

The number of stains is given by the `number_patches` parameter. The number of particles per stain is given by the `number_agregated` parameter. The radius and thickness (if 3D simulation) for each patch is given by the `radius_patch` and `thickness_patch` parameters, respectively.

Additionnally, there is the possibility to release patches in each release zone. This is achieved by setting the `per_zone` parameter to true. An example is provided in Figure 5.3.

If this parameter is set to false, the bounding box around the defined release zones (if any) or the full domain is used to release the patches, as shown in Figure 5.4.

## 5.5 Surface and bottom releases

The surface and bottom release methods (`SurfaceRelease.java` and `BottonRelease.java`) allow to randomly release particles over the entire domain, either on the surface or at the bottom (for 3D simulations only).

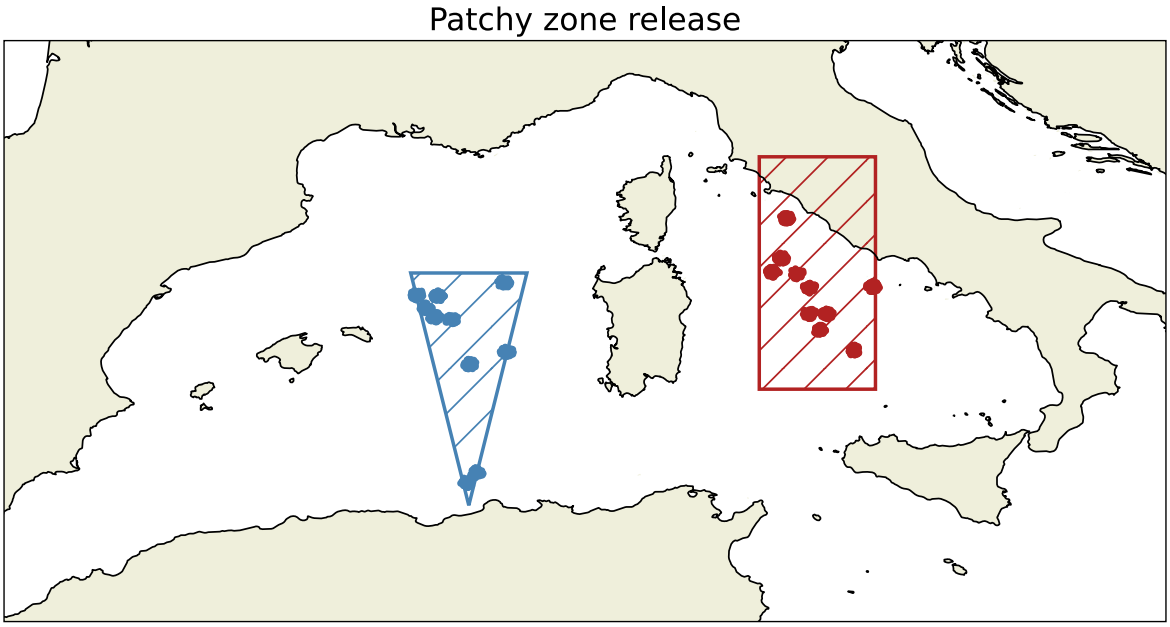



Figure 5.3: Example of a patchy zone release.

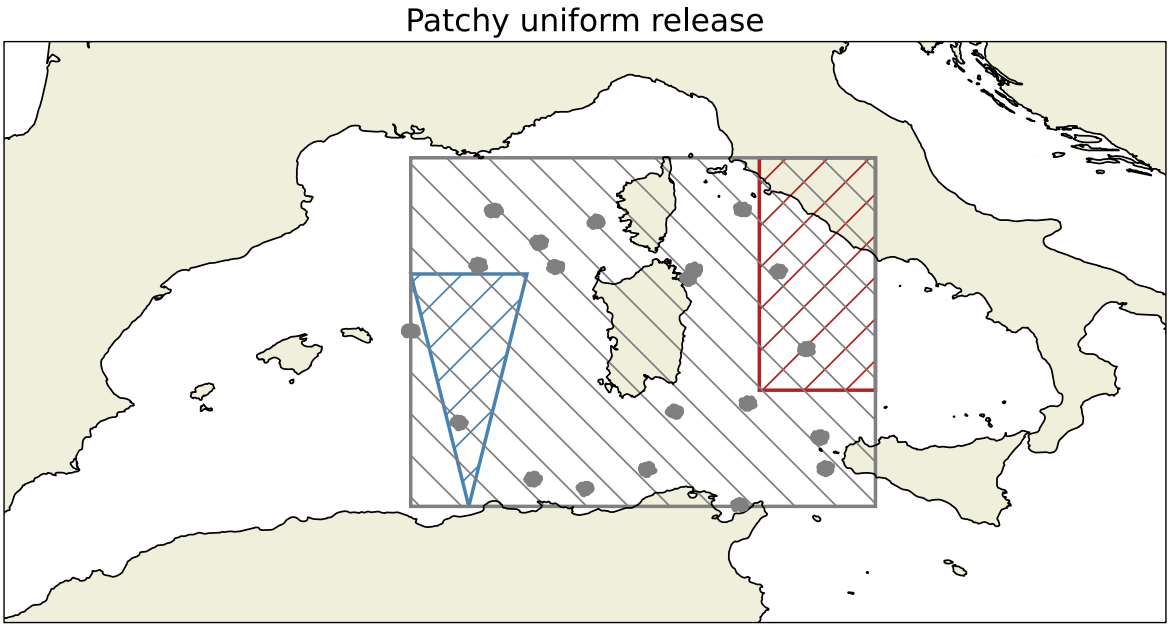



Figure 5.4: Example of a patchy uniform release.

The only parameter required by these methods is the number of particles (`number_particles` parameter). There is no control over the area where the particles are released.

An example of a surface release is shown in Figure 5.5.

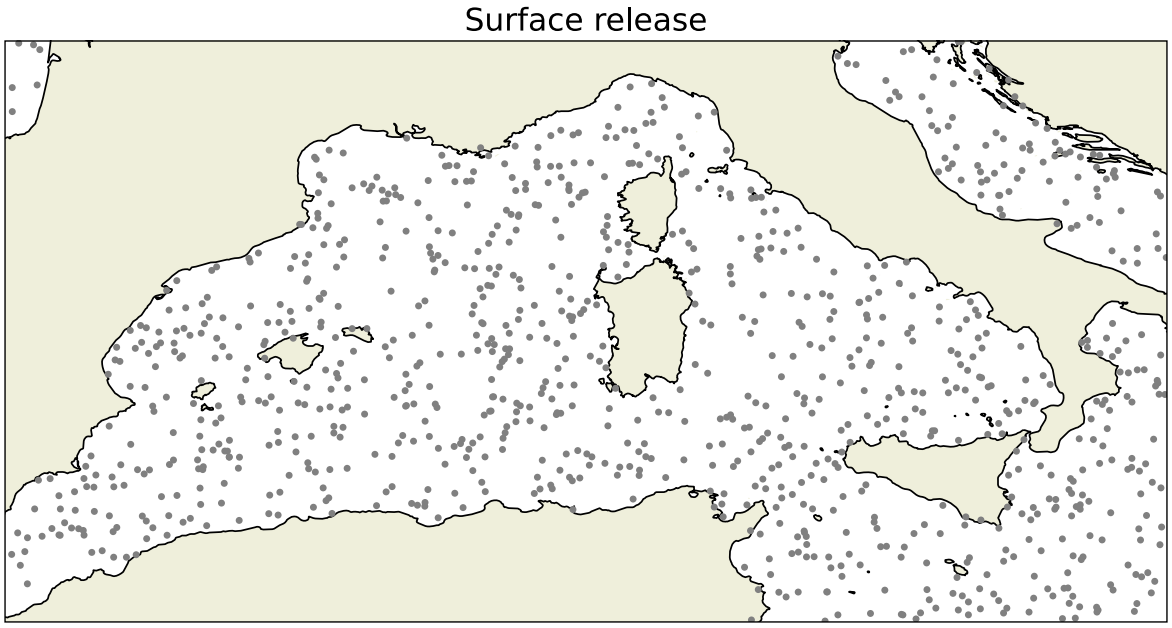



Figure 5.5: Example of a surface release.

## 5.6 Netcdf release

Ichthyop also offers the possibility to use an Ichthyop output file as a simulation starting point (`NcFileRelease.java`). This initialization method only takes a `ncfile` parameter, which provides the location of the NetCDF to use as a restart.

# 6 Release schedule

In Ichthyop, there is the possibility to schedule release events. These events are parameterized in the `release.schedule` option block.

It mainly contains two parameters:

- `is_enabled` specifies whether release events schedule is activated or not
- `events` is list of release date strings, which are formatted as the beginning simulation parameter (`year YYYY month MM day DD at YY:MM`).

The resulting output file will contain the same number of time-steps as in the case of a simple release, but the number of particles will be multiplied by the number of release events.

> **Warning**
>
> The output time array will be consistent with the beginning simulation time (`initial_time` parameter), **not with the release schedule dates**.

# 7 Ichthyop processes

In this section, the different Ichthyop processes are described. These processes are implemented in the classes within the `action` folder.

Parameters associated with these processes must be included in `action` blocks. They can be activated or deactivated by setting the `enabled` tag (cf. Section 2.1).

## 7.1 Growth

There is 2 different implementations of the growth module. They can be selected by setting the `class_name` parameter in the `action.growth` action configuration block.

### 7.1.1 Linear growth

To use the linear growth method described in Lett et al. (2008), choose the `LinearGrowthAction.java` class.

Length increment is provided as follows:

$$\Delta L = C_1 + C_2 \times \frac{F}{F + K_S} \times max(T, T_{thres}) \times \Delta t$$

where $\Delta t$ is the time-step (in days), $C_1$ and $C_2$ are parameters (`coeff1` and `coeff2`), $T_{thres}$ is a temperature threshold (`threshold_temp` parameter), $F$ is the food quantity and $K_S$ is a half-saturation constant (`half_saturation` parameter). If the latter is not defined or equals 0, $Q$ is assumed to be 1.

The name of the food and temperature variables are provided by the `food_field` and `temperature_field` parameters.

### 7.1.2 Sole growth

If the {samp}`SoleGrowthAction.java` class is selected, the growth model used in Tanner et al. (2017) is used. It relies on the growth equation from Fonds (1979) and given by:

$$\Delta L = C_1 \times T^{C_2} \times \Delta t$$

with $L$ the length, $T$ the temperature and $\Delta t$ the time step in days, and $C_1$ and $C_2$ are parameters (`c1` and `c2` respectively).

The temperature field is provided by the `temperature_field` parameter.

## 7.2 Lethal temperature and salinity

In Ichthyop, there is the possibility to define a range of temperature and salinity beyond which the particle is killed.

### 7.2.1 Lethal temperature

The functionning og the lethal temperature module depends on whether the growth module is enabled or not.

#### 7.2.1.1 Growth disabled

lethal temperature can either be provided in a CSV file (`lethal_temp_file` parameter), which provides the lower and upper temperature values that can be supported by the particle as a function of age (in hours). The file must have the following format:

```
Time(hour);Cold temperature (C);Warm temperature (C)
0;14;22
48;13;22
96;12;22
```

In this case, three age classes will be considered: $[0, 48[$, $[48, 96[$ and $[96, \infty[$

If the `lethal_temp_file` parameter is not defined, single values will be used independently of the age. These values are provided in the `cold_lethal_salinity_egg` and `warm_lethal_salinity_egg` parameters.

Note that the `temperature_field` parameter, providing the name of the temperature variable, must also be provided.

#### 7.2.1.2 Growth enabled

If the growth module is enabled, two cold and warm lethal temperatures must be provided. One for eggs (`cold_lethal_temperature_egg` and `hot_lethal_temperature_egg`), one for larva (`cold_lethal_temperature_larva` and `hot_lethal_temperature_larva`).

The stage (`egg` or `larva`) is determined by the growth module, and the right temperature range is applied to the particle.

### 7.2.2 Lethal salinity

Lethal salinity can either be provided in a CSV file (`lethal_salt_file` parameter), which provides the lower and upper salinity values that can be supported by the particle as a function of age (in hours). The file must have the following format:

```
Time(hour);Fresh salinity (PSU);Salty salinity (PSU)
0;35;40
48;30;40
96;30;45
```

In this case, three age classes will be considered: $[0, 48[$, $[48, 96[$ and $[96, \infty[$

If the `lethal_salt_file` parameter is not defined, single values will be used independently of the age. These values are provided in the `fresh_lethal_salinity_egg` and `saline_lethal_salinity_egg` parameters.

Note that the `salinity_field_` parameter, providing the name of the salinity variable, must also be provided.

## 7.3 Buoyancy

The buoyancy module allow to vertically displace a particle, depending on it's density and on the sea water density, following Parada et al. (2003).

The buoyancy-induced vertical velocity of the particle is given by:

$$W_{buoy} = \frac{1}{24} \times g \times a \times b \times \frac{\rho_{water} - \rho_{part}}{\rho_{water}} \mu^{-1} \log\left(2\frac{a}{b} + 0.5\right);$$

with $a$ and $b$ the semi-major axis and semi-minor axis of an ellipse (`mean_major_axis` and `mean_minor_axis` parameters), $\mu$ the molecular viscosity (`molecular_viscosity` parameter), $\rho_{water}$ the water density, $\rho_{part}$ the particle density and $g$ the gravitational acceleration ($cm.s^{-2}$).

If the particle density varies with age, it can be provided in a CSV file (`density_file` parameter), formatted as follows:

```
Age(hour);Density (g/cm3)
0;1.0235625
24;1.02374
48;1.023335
60;1.0239
66;1.025672
```

If the `density_file` parameter is not found, a constant density will be assumed (`particle_density` parameter).

The buoyancy process is only applied for the early life stages. If the growth process is disabled, it is controlled by the `age_max` parameter (provided in days). If this parameter is not provided, the buoyancy process will always be applied.

If the growth process is enabled, the application of the buoyancy module will be automatically managed.

## 7.4 Daily vertical migration

The `MigrationAction.java` manages the daily migration of the particles. It is done by providing daytime and nighttime depths, and the timing of the sunset and sunrise.

If the `daytime_depth_file` parameter is defined, it provides the depth of the particle during daytime. It is formatted as follows:

```
Age (day);Depth (m)
0.0;-20
3.0;-25
5.0;-30
8.0;-35
```

If this parameter is not found, a constant daytime depth, provided by the `daytime_depth` parameter, is assumed.

Same thing for the nighttime depths, which can be set by either the `nighttime_depth_file` or the `nighttime_depth` parameters.

The sunset and sunrise hours are set by the `sunset` and `sunrise` parameters, which must have a `HH::mm` format.

If the growth module is deactivated, the user must provide the minimum age (in days) at which the particle starts to migrate (`age_min` parameter). If the growth module is activated, it manages the activation or deactivation of the daily migration.

> **Warning**
>
> When the target depth is greater than the total depth, the particle does not move.

## 7.5 Ontogenetic vertical migration

The ontogenetic migration module controls the migration to different habitats as a function of age. Its implementation in Ichthyop follows the CMS one. It uses the CMS configuration file (`cms_ovm_config_file` parameter), which is formatted as follows:

```
nTime
nDepth
z1 z2 z3 z4 z5 z6 ... znDepth
t1 t2 t3 t4 t5 t6 ... tnTime

P1,1 P1,2 ... P1,nTime
P2,1 P2,2 ... P2,nTime
P3,1 P3,2 ... P3,nTime
          ...
PnDepth,1 PnDepth,2 ... PnDepth,nTime
```

The first line provides the number of time steps in the CMS file, the second line provides the number of vertical levels in the CMS file. The third line provides the depth values and the fourth lines provides the time step values.

The remaining lines provide the probability matrix $P_{z,t}$.

> **Note**
>
> The sum of the probability matrix along the depth dimension should equal 100.

At each time step, the index of the CMS time step, $k$, is determined by comparing the simulation and CMS times as follows:

$$t_{CMS}(k) < t \leq t_{CMS}(k+1)$$

When the CMS time index changes, all the particles are randomly distributed on the vertical, following the probability distribution of the given CMS time. If the CMS time index remains unchanged, nothing is done.

## 7.6 Wind drift

The wind-drift is implemented in the `WindDriftFileAction.java` class. This class requires that 2D (time, latitude, longitude) wind fields are provided. This is done by providing a `input_path` and a `file_filter` parameter, which specify the location and names of the wind files.

The user also provides `field_time`, `wind_u`, `wind_v`, `longitude`, `latitude` parameters, which specify the names of the time, zonal wind, meridional wind, longitude and latitude variables in the NetCDF files.

The user also provides a `depth_application` parameter, which specifies the depth at which the wind will impact the trajectories (only valid for 3D simulations), a `wind_factor` ($F$, multiplication factor), an `angle` ($\theta$) and a `wind_convention` parameter ($\varepsilon$, +1 if convention is ocean-based, i.e. wind-to, -1 if convention is atmospheric based, i.e. wind-from).

The changes in particle longitude ($\lambda$) and latitude ($\phi$) is provided as follows:

$$\Delta\lambda_0 = \frac{\Delta t \times U_W}{\frac{R\pi}{180} \times \cos(\frac{\pi\phi}{180})}$$

$$\Delta\phi_0 = \frac{\Delta t \times V_W}{\frac{R\pi}{180}}$$

$$\Delta\lambda = \varepsilon \times F \times \left(\Delta\lambda_0 \times \cos\left(\frac{\theta\pi}{180}\right) - \Delta\phi_0 \times \sin\left(\frac{\theta\pi}{180}\right)\right)$$

$$\Delta\phi = \varepsilon \times F \times \left(\Delta\lambda_0 \times \sin\left(\frac{\theta\pi}{180}\right) + \Delta\phi_0 \times \sin\left(\frac{\theta\pi}{180}\right)\right)$$

> **Note**
>
> $R$ is the Earth Radius and equals 6367.74 km. In the code, $\frac{R\pi}{180}$, which is the distance in m of a 1 degree cell, is approximated to 111138 m

## 7.7 Random swimming

Ichthyop allows particles to randomly swim. This is achieved by using the `SwimmingAction.java`. The velocity may vary with the age of the particle.

The user provides a CSV absolute velocity file (`velocity_file` parameter), which must me formatted as follows:

```
Age (days);Speed (m/s)
0;0.1
5;0.2
15;0.3
```

The user also provides a boolean parameter (`constant_velocity`) that specifies whether the velocity should remain as defined in the CSV file, or if the absolute velocity should be randomly selected as follows:

$$\|U\| = \|U\|_{file} \times \kappa$$

with $\kappa$ a random value in the $[0, 2]$ interval and $\|U\|_{file}$ the velocity defined in the CSV file.

At each time-step, the zonal and meridonal velocities are defined as follows:

$$U = \kappa' \varepsilon \|U\|$$

$$V = \varepsilon' \sqrt{\|U\|^2 - U^2}$$

with $\kappa'$ a random value between 0 and 1, and $\varepsilon$ and $\varepsilon'$ random values either equal to 1 or -1.

## 7.8 Wave drift

Ichthyop can take into account the effects of waves on the particles trajectories, following the Stokes drift equations of Stokes (2009):

$$U_S = w \times k \times a^2 \times \exp(2 \times k \times z)$$

In Ichthyop, the user provides zonal and meridional wave stokes drift and the wave periods. The horizontal displacement of particles is computed as follows:

$$\|U_{wave}\| = U_{wave}^2 + V_{wave}^2$$

$$\lambda_{wave} = \|U_{wave}\| \times T_{wave}$$

$$k_{wave} = \frac{2\pi}{\lambda_{wave}}$$

$$\Delta X = F \times U_{cur} \times \Delta t \times \exp\left(2 \times k_{wave} \times z\right)$$

$$\Delta Y = F \times V_{cur} \times \Delta t \times \exp\left(2 \times k_{wave} \times z\right)$$

with $U_{wave}$ and $V_{wave}$ the zonal and meridional stokes components, $T_{wave}$ the wave period, $U_{cur}$ and $V_{cur}$ the zonal and meridional ocean current components, $z$ the depth and $F$ a multiplication factor provided by the user.

## 7.9 Coastal behaviour

### 7.9.1 Bouncing

In the bouncing mode, the particle will bounce on the coast. First, whether the bouncing occurs on a meridional or a zonal coastline is determined.

In case of a meridional coastline, as shown in Figure 7.1, the calculation of the new position is performed as follows.

Let's assume that the particle is at the position $(x, y)$ and is moved at the postion $(x + \Delta x, y + \Delta y)$. We suppose that

$$\Delta y = \Delta y_1 + \Delta y_2, \tag{7.1}$$

where $\Delta y_1$ is the meridional distance between the particle and the coastline, and $\Delta y_2$ is the distance that the particle will spend on land.

In the bouncing mode, the position increment can be written as

$$\Delta_{cor} y = \Delta y_1 - \Delta y_2 \tag{7.2}$$

By replacing $\Delta y_2$ using Equation 7.1, we can write:

$$\Delta_{cor}y = \Delta y_1 - (\Delta y - \Delta y_1)$$

$$\Delta_{cor}y = 2\Delta y_1 - \Delta y$$

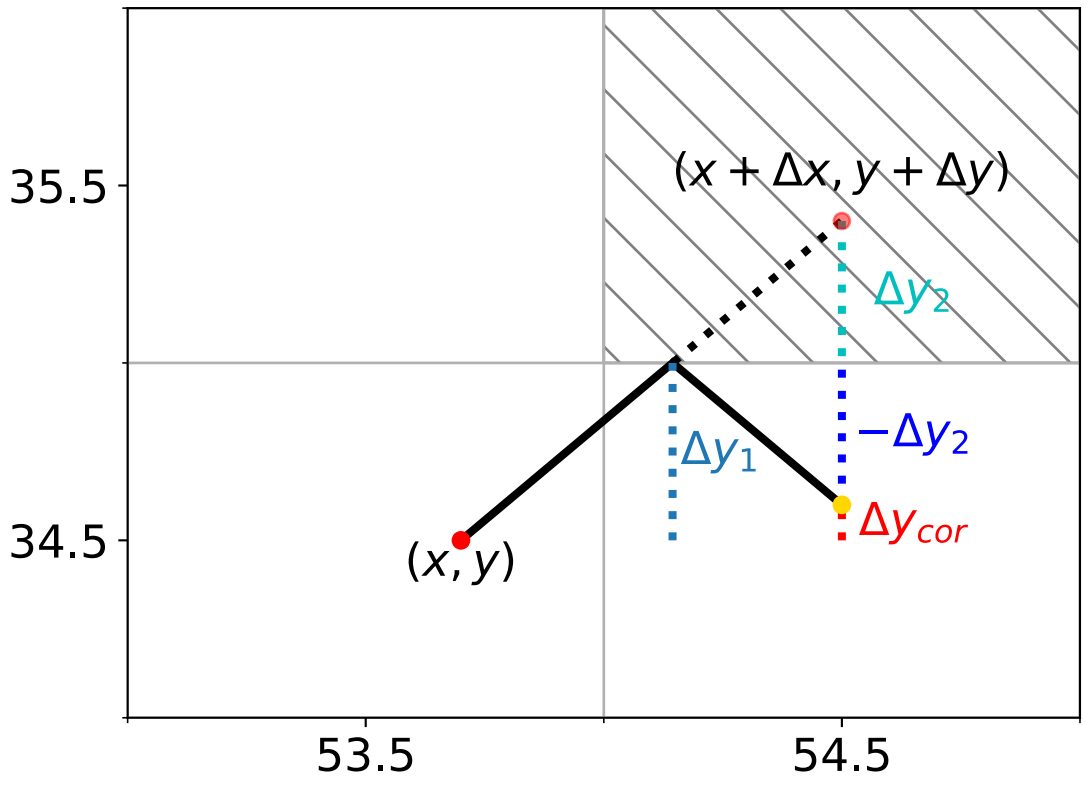



Figure 7.1: Coastal behaviour in bouncing mode.

## 7.10 Orientation

Active swimming has been implemented in Ichthyop following the work of Romain Chaput. Three active swimming behaviours have been implemented: the rheotaxis orientation (i.e. against the current), the cardinal orientation (i.e toward a given direction) and the reef orientation, i.e. orientation toward points of interests.

These three implementations involve the computation of a swimming velocity and direction. The former is common to all three methods, but the directions depend on the method considered.

### 7.10.1 Swimming velocity

The orientation processes all share common features. They both depend on a swimming velocity and random directions. The methods described below differ on the way the random directions are drafted.

Swimming velocity is computed following Staaterman, Paris, and Helgers (2012):

#|echo: false

$$V = V_{hatch} + (V_{settle} - V_{hatch})^{\log(age)/\log(PLD)}$$

with $V_{hatch}$ and $V_{settle}$ the larval velocity at hatching and settle, $A$ the age of the larva and $PLD$ the transport duraction.

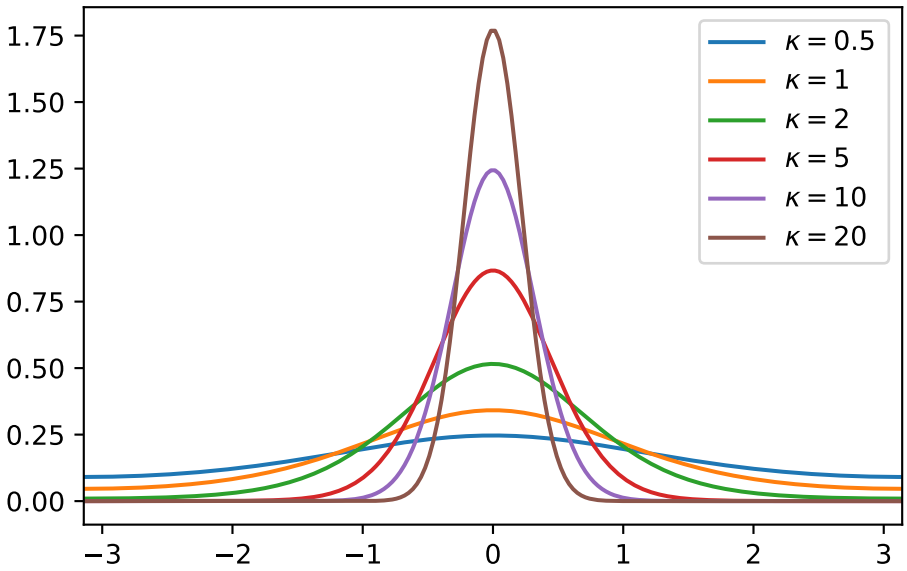
### 7.10.2 Von Mises distributions

Von Mises distribution is used in all three methods.The Von Mises distribution is given by:

$$f(\theta, \mu, \kappa) = \frac{\exp(\kappa \cos(\theta - \mu))}{2\pi I_0(\kappa)}$$

where $I_0(\kappa)$ is the modified Bessel function of the first kind of order 0, $\mu$ is angle where the distribution is centerred and $\kappa$ is the concentration parameter. The distribution is as follows:

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.special import i0

x = np.linspace(-np.pi, np.pi, 200)

def von_misses(kappa):
    mu = 0
    y = np.exp(kappa*np.cos(x-mu))/(2*np.pi*i0(kappa))
    return y

plt.figure()

for i in [0.5, 1, 2, 5, 10, 20]:
    plt.plot(x, von_misses(i), label=f'$\kappa = {i}$')
plt.legend()
plt.xlim(x.min(), x.max())
plt.show()
```




For computation purposes, all the Von Mises drafts performed in Ichthyop are done by using $mu = 0$. The angles are thus centerred around 0. Then, the *mu* value is added.

> **Note**
>
> $\theta = 0$ is eastward, $\theta = \frac{\pi}{2}$ is northward, etc.

### 7.10.3 Computation of displacement

Given a swimming velocity $V$ and a direction $\theta$, the larva displacement (in $m$) is computed as follows:

$$\Delta X = V \times \cos(\theta) \times \Delta t$$

$$\Delta Y = V \times \sin(\theta) \times \Delta t$$

with $\Delta t$ the time step. Next, the corresponding change in longitude ($\lambda$) and latitude ($\varphi$) is computed as follows:

$$\Delta \lambda = \frac{\Delta X}{111138 \times \cos \varphi}$$

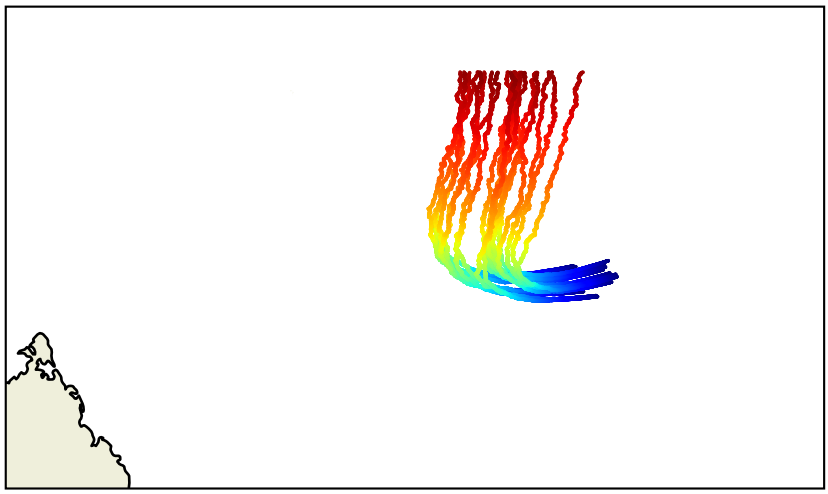
$$\Delta\varphi = \frac{\Delta Y}{111138}$$

### 7.10.4 Cardinal orientation

In cardinal orientation, the user provides a fixed heading $\theta_{card}$ and a fixed $\kappa$ parameter. Then, at each time step, a new angle is randomly drafted following a Von Misses distribution $f(\theta, \theta_{card}, \kappa)$.

```
import xarray as xr
import cartopy.crs as ccrs
import cartopy.feature as cfeature
import os
from glob import glob
import matplotlib.pyplot as plt
import numpy as np
import matplotlib as mpl

config = 'card'

if os.path.isdir('ichthyop_output'):
    pattern = os.path.join('ichthyop_output', config, '*nc')
else :
    pattern = os.path.join('..', '..', 'ichthyop_output', config, '*nc')
pattern

filelist = glob(pattern)
filelist

data = xr.open_mfdataset(filelist, decode_times=False)
data

# +
lon = data['lon'].values
lat = data['lat'].values
mort = data['mortality'].values

lon = np.ma.masked_where(mort > 0, lon)
lat = np.ma.masked_where(mort > 0, lat)
ntime, ndrifters = lon.shape

# +
time = np.arange(ntime)
```

```
drifters = np.arange(ndrifters)

d2d, t2d = np.meshgrid(drifters, time)

# +
plt.figure()

projin = ccrs.PlateCarree()
projout = ccrs.PlateCarree()

ax = plt.axes(projection = projout)
ax.scatter(lon[:, :], lat[:, :], c=t2d, marker='.', transform=projin, s=0.5, cmap=mpl.colormaps['
feat = ax.add_feature(cfeature.LAND)
feat = ax.add_feature(cfeature.COASTLINE)
ax.set_extent([49, 55.25, -13.13, -9.45], crs=projin)
```



### 7.10.5 Rheotaxis orientation

In the rheotaxis orientation method, the particles swim against the current. The user only provides a kappa parameter.

First, the angle of the current is computed as follows:

$$\theta_{current} = \arctan 2(V_{current}, U_{current})$$

Then, the angle that the particle must follow is given by adding $\pi$:

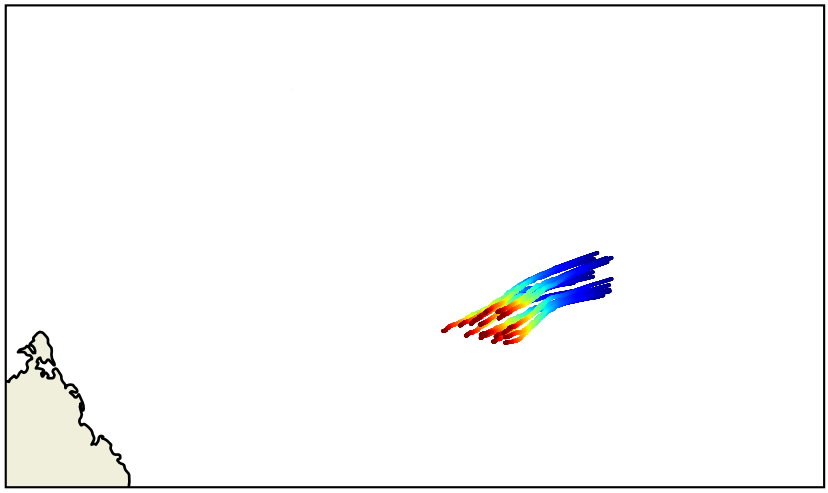
$$\theta_{direction} = \theta_{current} + \pi$$

Finally, a random angle is picked up following a Von Mises distribution $f(\theta, \theta_{direction}, \kappa_{reef})$

(ref-rheo)=

```
import xarray as xr
import cartopy.crs as ccrs
import cartopy.feature as cfeature
import os
from glob import glob
import matplotlib.pyplot as plt
import numpy as np
import matplotlib as mpl

config = 'rheo'

if os.path.isdir('ichthyop_output'):
    pattern = os.path.join('ichthyop_output', config, '*nc')
else :
    pattern = os.path.join('..', '..', 'ichthyop_output', config, '*nc')
pattern

filelist = glob(pattern)
filelist

data = xr.open_mfdataset(filelist, decode_times=False)
data

# +
lon = data['lon'].values
lat = data['lat'].values
mort = data['mortality'].values

lon = np.ma.masked_where(mort > 0, lon)
lat = np.ma.masked_where(mort > 0, lat)
ntime, ndrifters = lon.shape

# +
time = np.arange(ntime)
drifters = np.arange(ndrifters)
```

```
d2d, t2d = np.meshgrid(drifters, time)

# +
plt.figure()

projin = ccrs.PlateCarree()
projout = ccrs.PlateCarree()

ax = plt.axes(projection = projout)
ax.scatter(lon[:, :], lat[:, :], c=t2d, marker='.', transform=projin, s=0.5, cmap=mpl.colormaps['
feat = ax.add_feature(cfeature.LAND)
feat = ax.add_feature(cfeature.COASTLINE)
ax.set_extent([49, 55.25, -13.13, -9.45], crs=projin)
```



### 7.10.6 Reef orientation

In the reef orientation method, the larva will target the closest target area (for instance coral reef). These areas are defined in an XML zone file by a polygon and a zone-specific $\kappa$ parameter. The user also provides the sensory detection threshold of the larva (maximum detection distance $\beta$).

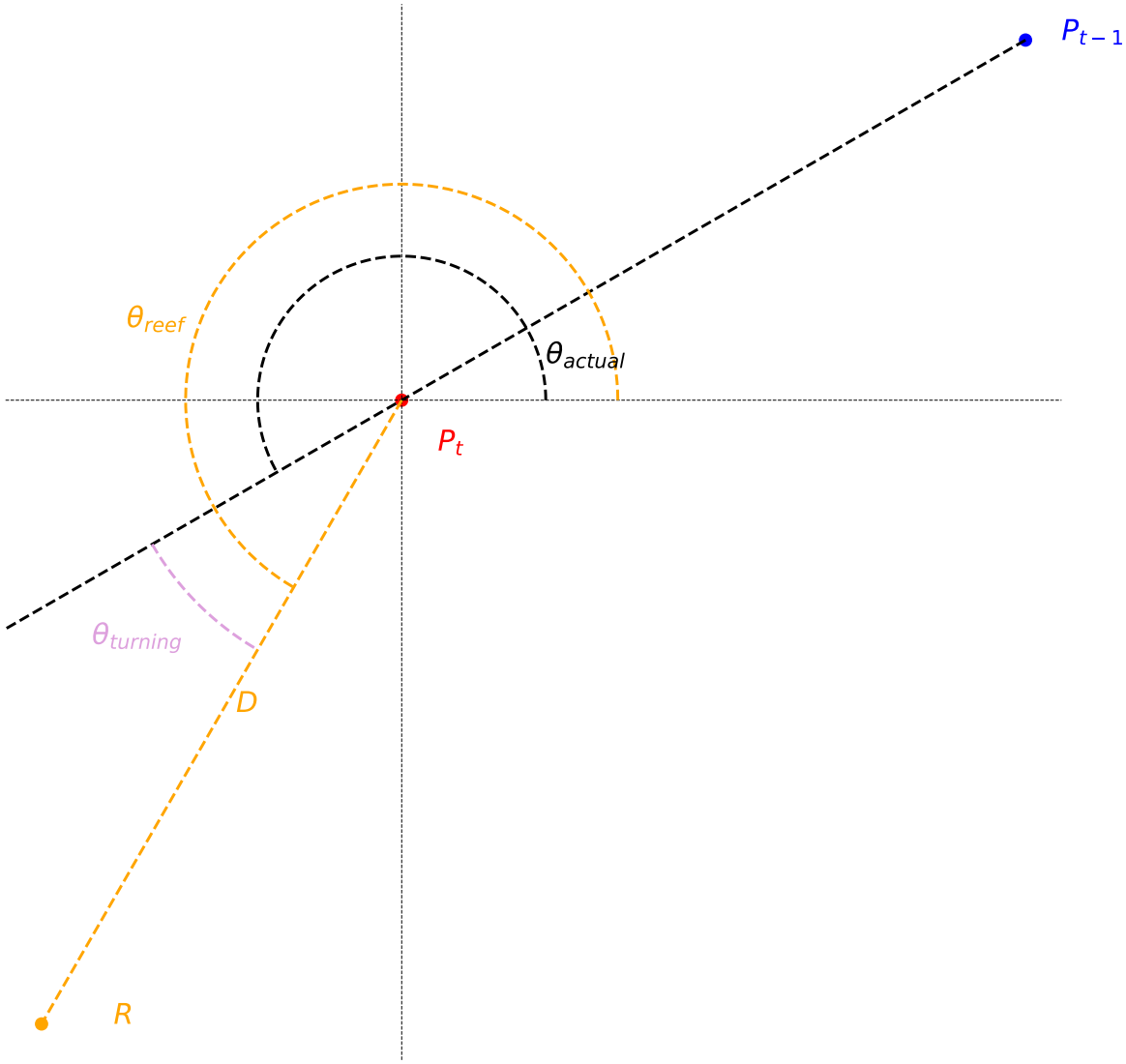
If the distance between the particle and the barycenter of the closest reef ($D$) is below the detection thereshold $\beta$, the larva will swim in the direction of the reef.

(ref-orientation)=

```
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams['font.size'] = 15

plt.figure(figsize = (10, 10))

xnew = 0
ynew = 0

#draw point at orgin
plt.plot(xnew, ynew, color = 'red', marker = 'o')
plt.gca().annotate('$P_{t}$', xy=(0 + 0.05, 0 - 0.07), xycoords='data', color='red')

#draw circle
r = 1.5
angles = np.linspace(0 * np.pi, 2 * np.pi, 100 )
xs = r * np.cos(angles)
ys = r * np.sin(angles)
plt.plot(xs, ys, color = 'k', ls='--', lw=0.5)

angle_old = np.pi / 6.
rold = 1.
xold = rold * np.cos(angle_old)
yold = rold * np.sin(angle_old)

plt.plot(xold, yold, marker='o', color='blue')
plt.gca().annotate('$P_{t - 1}$', xy=(xold + 0.05, yold), xycoords='data', color='blue')

angle_reef = np.pi + np.pi / 3.
print(np.rad2deg(angle_reef))
rreef = 1
xreef = rreef * np.cos(angle_reef)
yreef = rreef * np.sin(angle_reef)
print(yreef)

plt.plot(xreef, yreef, marker='o', color='orange')
plt.gca().annotate('$R$', xy=(xreef + 0.1, yreef), xycoords='data', color='orange')

plt.gca().annotate(r'$D$', xy=(0.5*(xreef) + 0.02, 0.5*yreef), xycoords='data', color='orange', l

plt.axvline(xnew, color='k', ls='--', lw=0.5)
```

```
plt.axhline(ynew, color='k', ls='--', lw=0.5)

p = np.polyfit([xold, xnew], [yold, ynew], deg=1)
xtemp = np.linspace(xold, xold -2, 100)
plt.plot(xtemp, np.polyval(p, xtemp), color='black', ls='--')

angle_current = angle_old + np.pi
tmp_angle = np.linspace(0, angle_current, 100)
rtmp = 0.2
plt.plot(0 + rtmp * np.cos(tmp_angle), 0 + rtmp * np.sin(tmp_angle), color='k', ls='--')
plt.gca().annotate(r'$\theta_{actual}$', xy=(xnew + 0.2, ynew+0.05), xycoords='data', color='k')


plt.plot([xnew, xreef], [ynew, yreef], color='orange', ls='--')
angle_current = angle_reef
tmp_angle = np.linspace(0, angle_current, 100)
rtmp = 0.3
plt.plot(0 + rtmp * np.cos(tmp_angle), 0 + rtmp * np.sin(tmp_angle), color='orange', ls='--')
plt.gca().annotate(r'$\theta_{reef}$', xy=(xnew - 0.3, ynew+0.1), xycoords='data', color='orange'

angle_current = angle_old + np.pi
tmp_angle = np.linspace(angle_current, angle_reef, 100)
rtmp = 0.4
plt.plot(0 + rtmp * np.cos(tmp_angle), 0 + rtmp * np.sin(tmp_angle), color='plum', ls='--')
plt.gca().annotate(r'$\theta_{turning}$', xy=(-0.43, -0.34), xycoords='data', color='plum')

off = 0.05
plt.xlim(xreef - off, xold + off)
plt.ylim(yreef - off, yold + off)
plt.gca().set_aspect('equal')
plt.axis('off')
plt.show()
```

```
239.99999999999997
-0.8660254037844384
```




First, the angle of the current trajectory, $\theta_{actual}$, is computed by using the particle position at the previous time step (blue point) and the actual position (red point).

$$\Delta_X = (X_{t-1} - X_t)$$

$$\Delta_Y = (Y_{t-1} - Y_t)$$

$$\theta_{actual} = \arctan 2(\Delta Y, \Delta X) + \pi$$

The direction toward the reef, $\theta_{reef}$ is also computed.

$$\Delta_X = (X_{reef} - X_t)$$

$$\Delta_Y = (Y_{reef} - Y_t)$$

$$\theta_{reef} = \arctan 2(\Delta Y, \Delta X)$$

> **Warning**
>
> The angles are computed in the $(X, Y)$ space. Therefore, longitude and latitude coordinates are converted in $(X, Y)$ using the `latlon2xy` Dataset methods.

The turning angle $\theta_{turning}$ is given by:

$$\theta_{turning} = \theta_{reef} - \theta_{actual}$$

The turning angle is then ponderated by the ratio of the distance from the reef to the detection threshold as follows:

$$\theta_{ponderated} = \left(1 - \frac{D}{\beta}\right)\theta_{turning}$$

$$\theta_{ponderated} = \left(1 - \frac{D}{\beta}\right)\left(\theta_{reef} - \theta_{actual}\right)$$

Therefore, the closest to the reef, the strongest the turning angle.

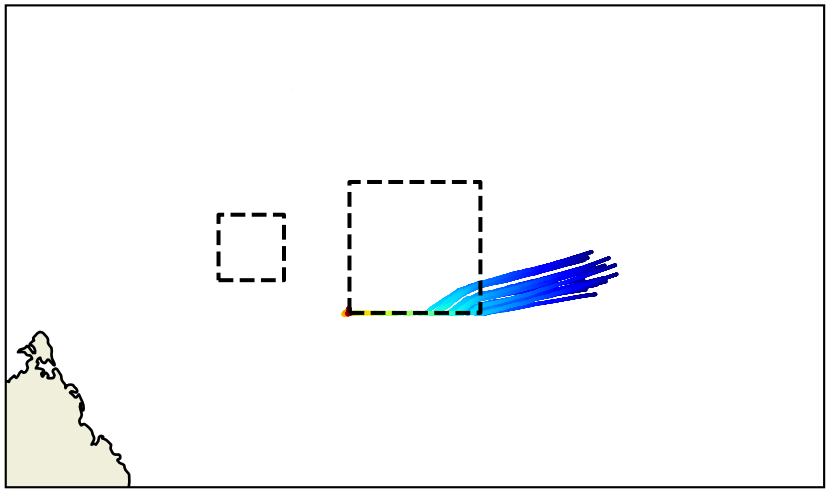
Then, a random angle is picked up following a Von Mises distribution $f(\theta, \theta_{ponderated}, \kappa_{reef})$

An example of a trajectory is provided below. In this case, two target destinations are provided (black boxes). The same $\kappa$ value was used for both ares (1.2) and the $\beta$ parameter has been set equal to 3 km.

(ref-orientation-2)=

```
import xarray as xr
import cartopy.crs as ccrs
import cartopy.feature as cfeature
import os
from glob import glob
import matplotlib.pyplot as plt
import numpy as np
import matplotlib as mpl

config = 'reef'

if os.path.isdir('ichthyop_output'):
    pattern = os.path.join('ichthyop_output', config, '*nc')
else :
    pattern = os.path.join('..', '..', 'ichthyop_output', config, '*nc')
pattern

filelist = glob(pattern)
filelist

data = xr.open_mfdataset(filelist, decode_times=False)
data

# +
lon = data['lon'].values
lat = data['lat'].values
mort = data['mortality'].values

lon = np.ma.masked_where(mort > 0, lon)
lat = np.ma.masked_where(mort > 0, lat)
ntime, ndrifters = lon.shape

# +
time = np.arange(ntime)
drifters = np.arange(ndrifters)

d2d, t2d = np.meshgrid(drifters, time)

# +
plt.figure()

projin = ccrs.PlateCarree()
```

```
projout = ccrs.PlateCarree()

ax = plt.axes(projection = projout)
ax.scatter(lon[:, :], lat[:, :], c=t2d, marker='.', transform=projin, s=0.5, cmap=mpl.colormaps['
feat = ax.add_feature(cfeature.LAND)
feat = ax.add_feature(cfeature.COASTLINE)
xp = np.array([51.625, 52.625, 52.625,51.625, 51.625])
yp = np.array([-1.179878E1, -1.179878E1, -1.079878E1, -1.079878E1, -1.179878E1])
ax.plot(xp, yp, transform=projin, color='k', ls='--')
xp = np.array([51.625, 52.625 - 0.5, 52.625 - 0.5,51.625, 51.625]) - 1
yp = np.array([-1.179878E1 + 0.5, -1.179878E1 + 0.5, -1.079878E1, -1.079878E1, -1.179878E1 + 0.5]
ax.plot(xp, yp, transform=projin, color='k', ls='--')
ax.set_extent([49, 55.25, -13.13, -9.45], crs=projin)
```



# Part III

# Developer documentation

In this section, the Ichthyop console is described.

# 8 Manager initialization

When an Ichthyop simulation is launched, managers are mobilized in the following order (cf. `SimulationManager.mobiliseManagers()` method):

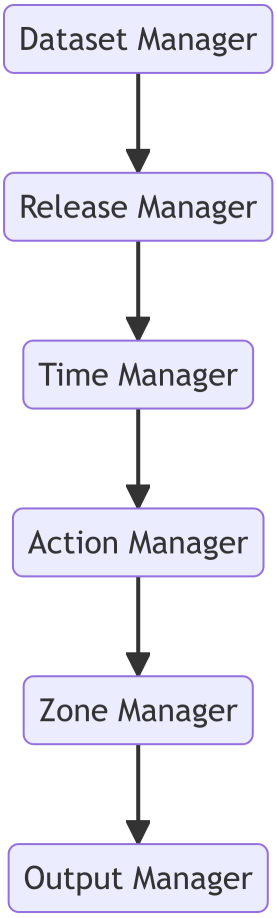



Figure 8.1: Order in which managers are mobilized.

This order is the one in which the managers will be setup and then initialized.

During the setup process, the `setupPerformed` methods, implemented on all the manager classes, will be called.

After this setup process, the managers will be initialized. This will be achieved by calling the `initializePerformed` methods, implemented on all the manager classes.

# 9 Particles

Ichthyop particles (`Particle.java` class) contains the attributes described in Table 9.1.

Table 9.1: Particle state variables

| Variable | Description | Type | Units |
|---|---|---|---|
| x | Particle x position within the grid | float, $\in [0, N_x - 1]$ | |
| y | Particle y position within the grid | float, $\in [0, N_y - 1]$ | |
| z | Particle z position within the grid | float, $\in [0, N_z - 1]$ | |
| dx | Particle increment/decrement of x position | float | |
| dy | Particle increment/decrement of y position | float | |
| dz | Particle increment/decrement of z position | float | |
| lat | Particle longitude | float | Degrees East |
| lon | Particle latitude | float | Degrees North |
| depth | Particle depth | float, $< 0$ | m |
| index | Particle index | int, $\in [0, N_{part} - 1]$ | |
| age | Particle age | int | seconds |
| deathCause | Mortality status | `ParticleMortality` | |
| living | True if a particle is alive | bool | |
| locked | True if a particle is locked | bool | |
| layers | List of additional particle layers | `List<ParticleLayer>` | |

When using Ichthyop with additional processes, such as growth or DEB processes, additional variables need to be tracked.

This is done by firsr creating a layer class, which should extend the `ParticleLayer.java` class. An exemple if provided as follows:

```
package org.previmer.ichthyop.particle;

public class LengthParticleLayer extends ParticleLayer {
```

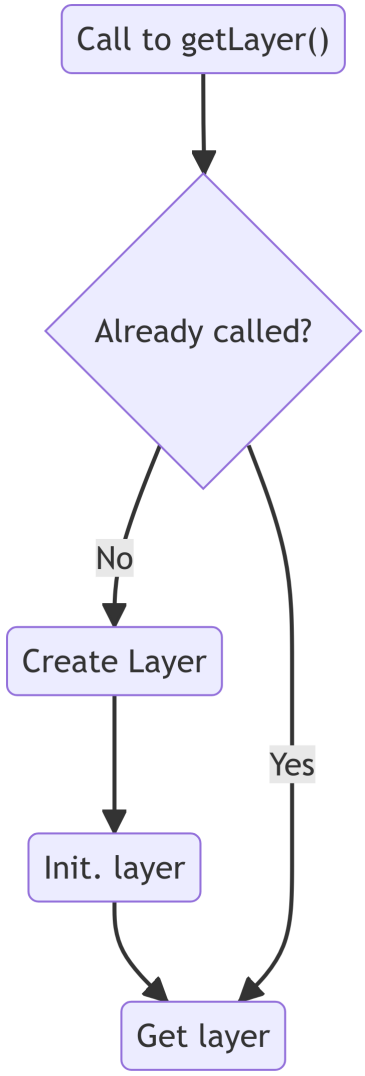
```
    private double length;

    public LengthParticleLayer(IParticle particle) {
        super(particle);
    }

    @Override
    public void init() {
        length = 0;
    }

    public double getLength() {
        return length;
    }

    public void incrementLength(double dlength) {
        length += dlength;
    }
}
```

This class contains additional particle attributes (here, for instance `length`) and methods that can return and modify these attributes.

These new variables can be accessed as follows:

```
LengthParticleLayer lengthLayer = (LengthParticleLayer) particle.getLayer(LengthParticleLayer.cla
lengthLayer.setLength(length_init);
```

During the first call to the `getLayer` method applied to a given layer class, the latter will be instanciated, initialized and add to the particle's `layers` list attribute, as shown below.




Order in which managers are mobilized.

# 10 Grid management

### 10.0.1 NEMO grid

In this section, the main features of the NEMO grid and the implications in Ichthyop are summarized.

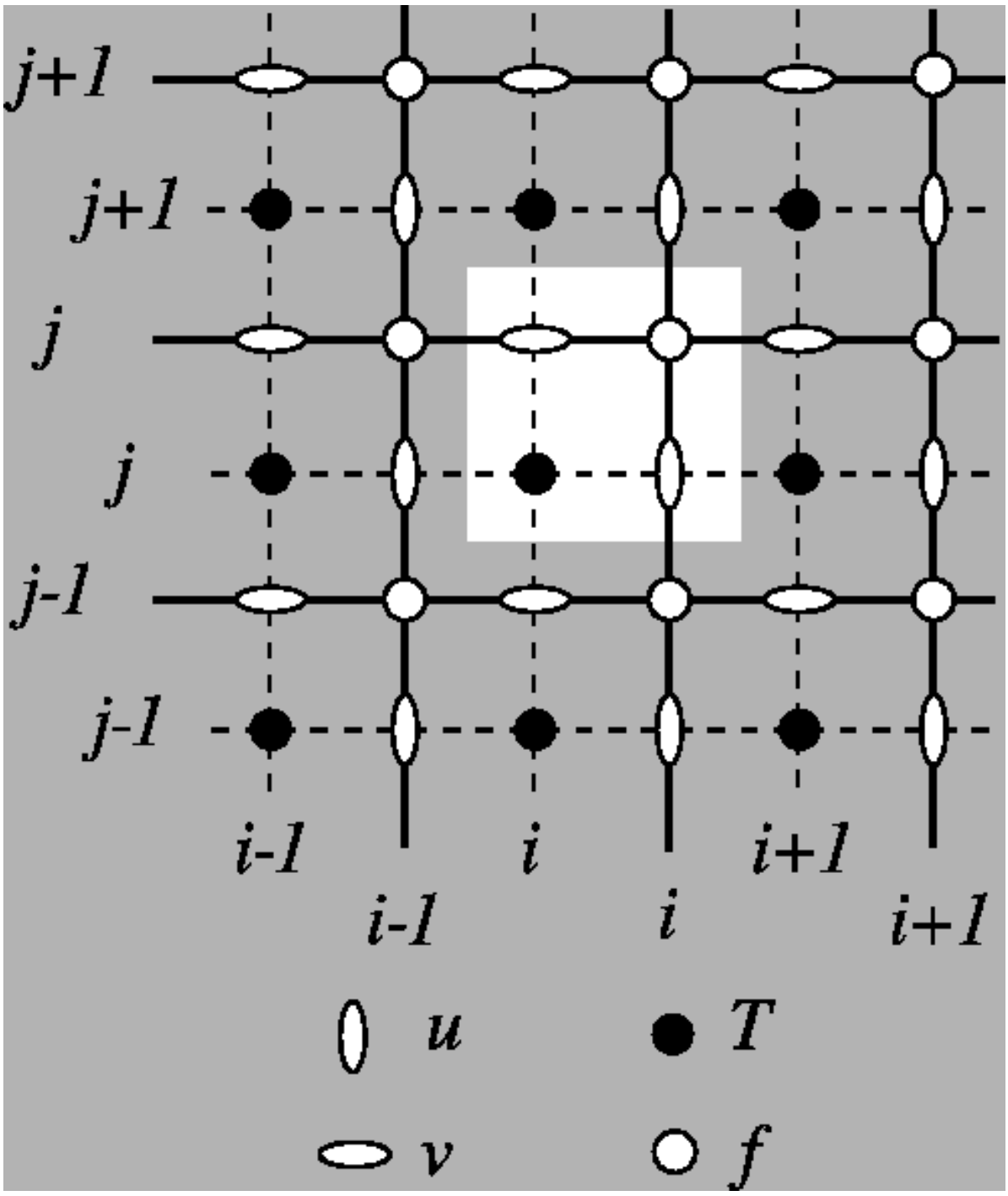
#### 10.0.1.1 Horizontal

##### 10.0.1.1.0.1 Computation domain

In the NEMO grid, the computation domain extends from the western edge of the western T cell (`i=0`) to the eastern edge of the eastern T cells (`i=nx`), and from the southern edge of the southern T cells (`j=0`) to the northern edge of the nothern T cells (`j=ny`).

##### 10.0.1.1.0.2 Indexing

The horizontal layout of the NEMO grid is as follows:




Tracer points (T points) are stored at the center of the cell. Zonal velocities (U points) are stored on the eastern face, while meridional velocities (V points) are stored on the northern face. **U, V and T points have the same number of elements!**

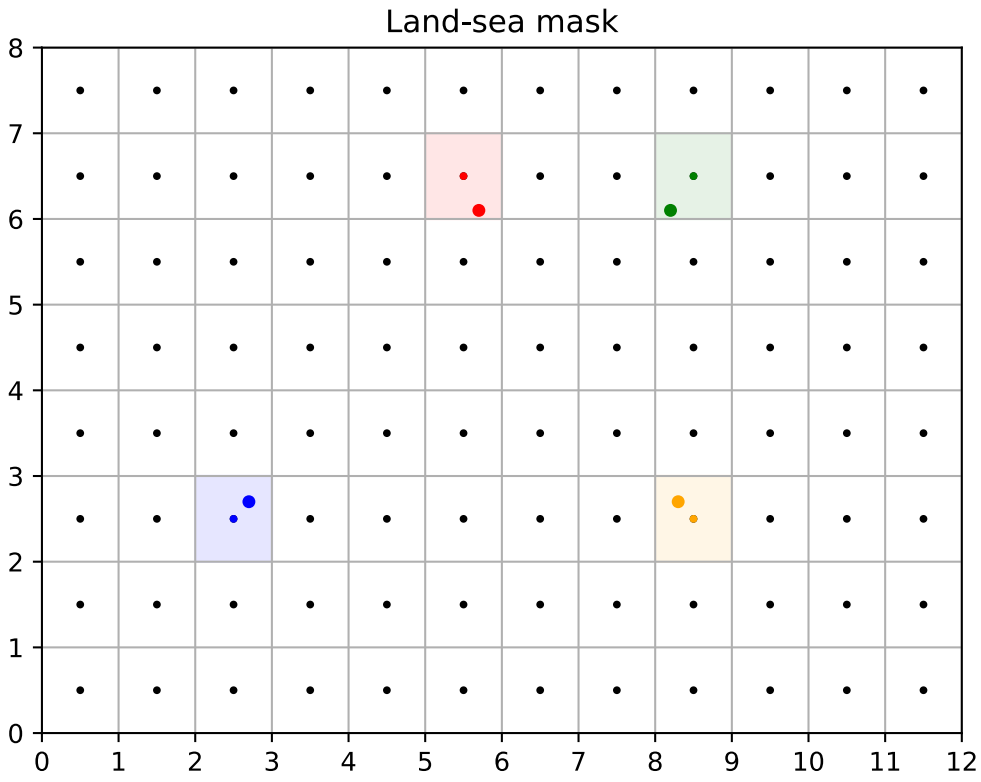
#### 10.0.1.1.0.3 Scale factors

In NEMO, the zonal and meridional length of the cells are stored in the `e1x` and `e2x` variables, with `x` equals to `t`, `u` or `v` depending on the point considered.

#### 10.0.1.1.0.4 Land-sea mask

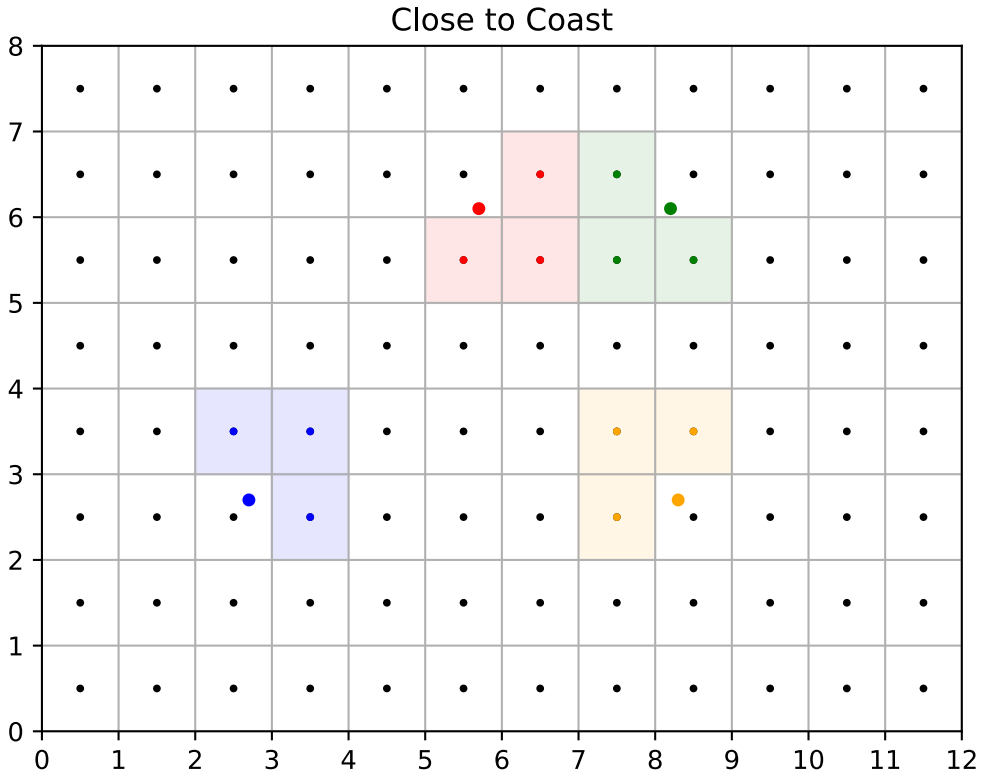
For a particle at a given location, we determine whether it is on land as follows:

- We extract the `i` index of the `T` land-sea mask to use by computing `round(x - 0.5)`.
- We extract the `j` index of the `T` land-sea mask to use by computing `round(y - 0.5)`.
- We extract the mask value at the `(i, j)` location.




#### 10.0.1.1.0.5 Close to coast

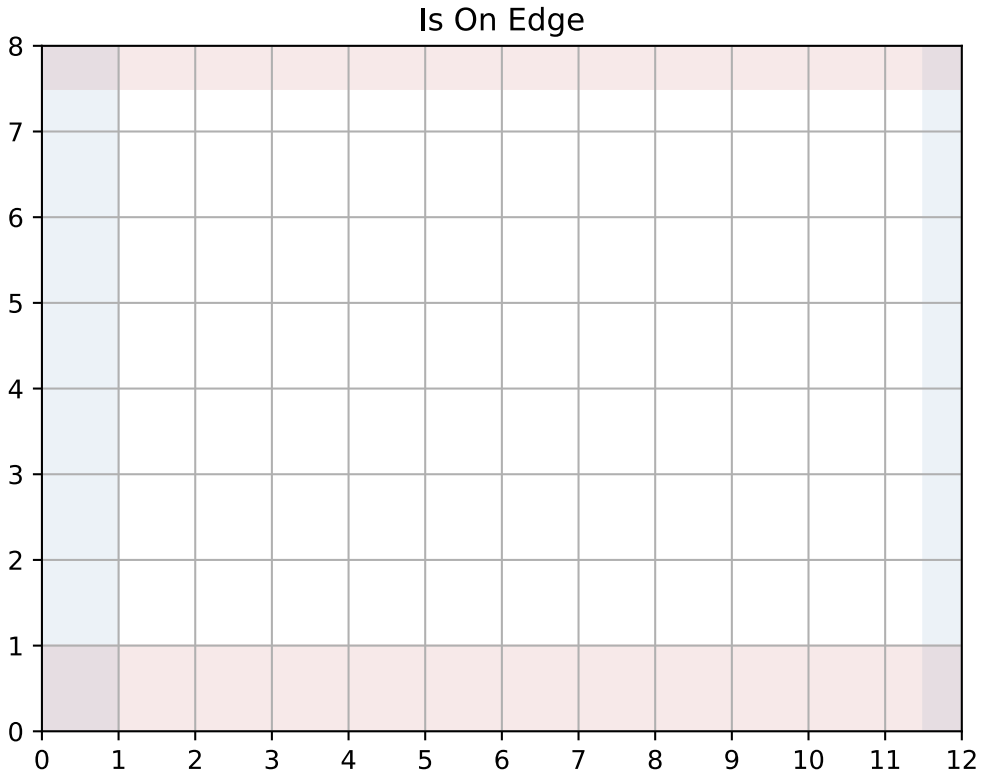
To determine whether a particle is close to coast, we extract the three neighbouring T cells. If one of them is land, then it assumed to be close to coast.




#### 10.0.1.1.0.6 Is On Edge

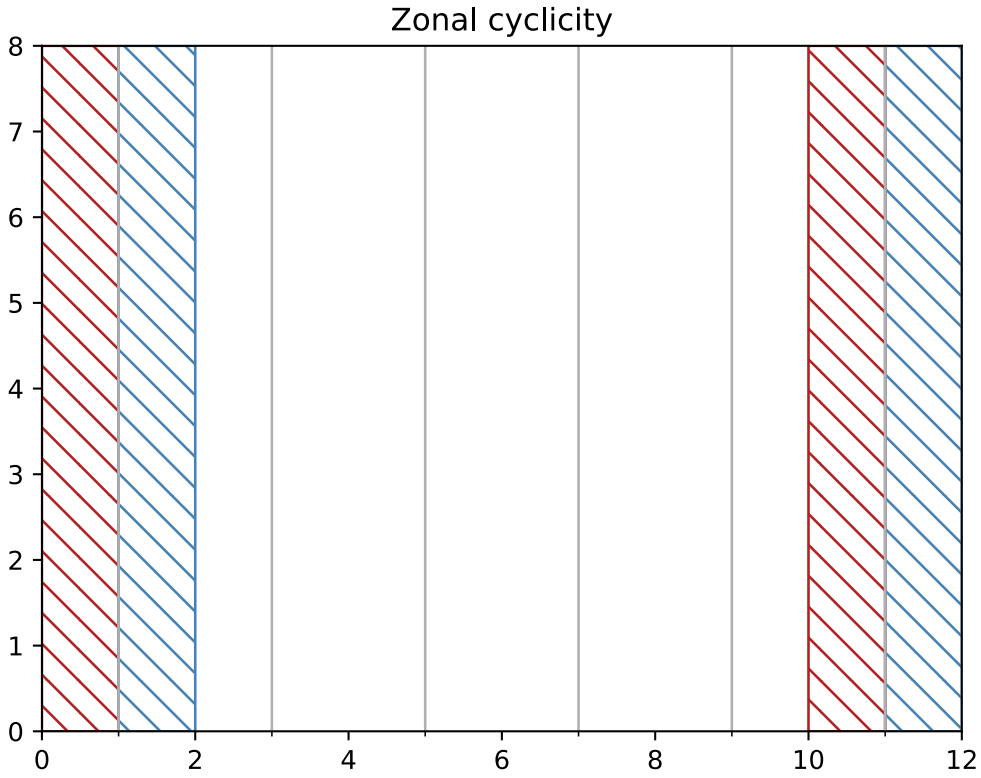
The particle is considered to be out of the domain if the y value is greater than ny - 0.5 (no possible interpolation of T points) or less than 1 (no possible interpolation of V points).

If there is no zonal cyclicity, the particle is also considered to be out of the domain if the x value is greater than nx - 0.5 (no possible interpolation of T points) or less than 1 (no possible interpolation of U points).




#### 10.0.1.1.0.7 Zonal cyclicity

For regional simulations, there is no zonal cyclicty. On the other hand, for global NEMO simulations, which runs on the ORCA grid, the zonal cyclicity is as follows (indexes are provided for T points):




Therefore:

- if $x \leq 1$, the particle is moved at $N_x - 2 + x$
- if $x \geq nx - 1$, the particle is moved at $x - N_x + 2$)

#### 10.0.1.1.0.8 Interpolation

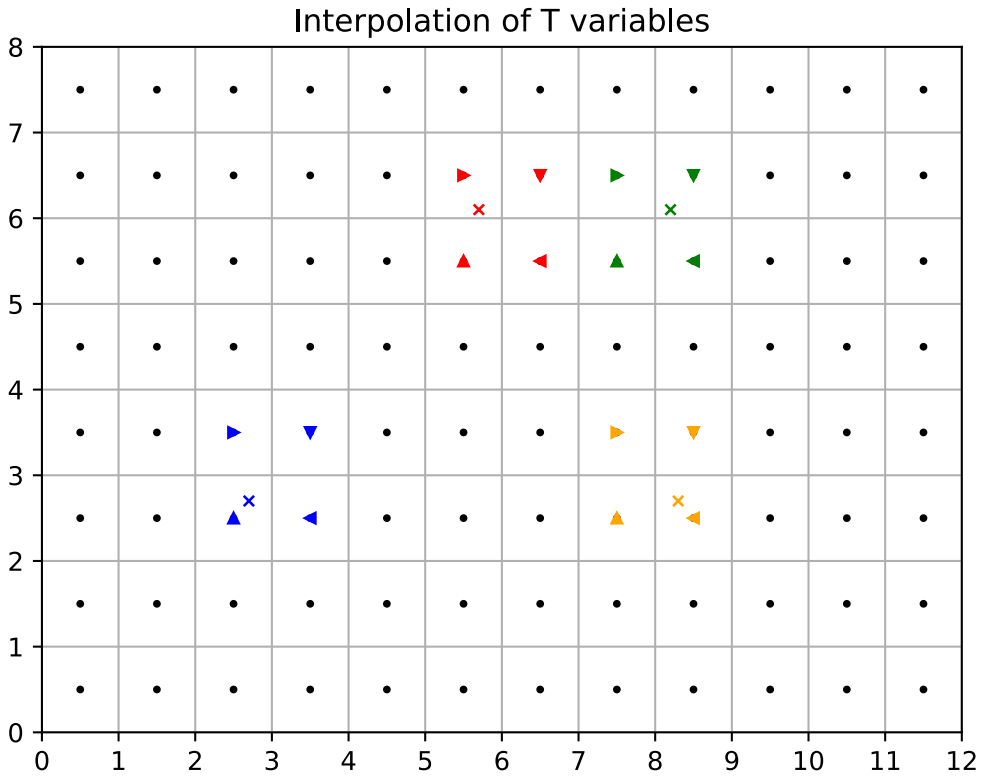
Interpolation of T variables

Given a given position index of a particle with the `T` grid, the determination of the interpolation is done as follows:

- First, the `i` index of the `T` grid column left of the particle is found. This is done by using `floor` on the `x - 0.5` value. The removing of 0.5 is to convert the x value from the computational grid to the `T` grid.
- Then, the `j` index of the `T` grid line below the particle is found. This is done by using `floor` on the `y - 0.5` value. The removing of 0.5 is to convert the y value from the computational grid to the `T` grid.

- The area to consider is defined by the `[i, i + 1]` and `[j, j + 1]` squares.

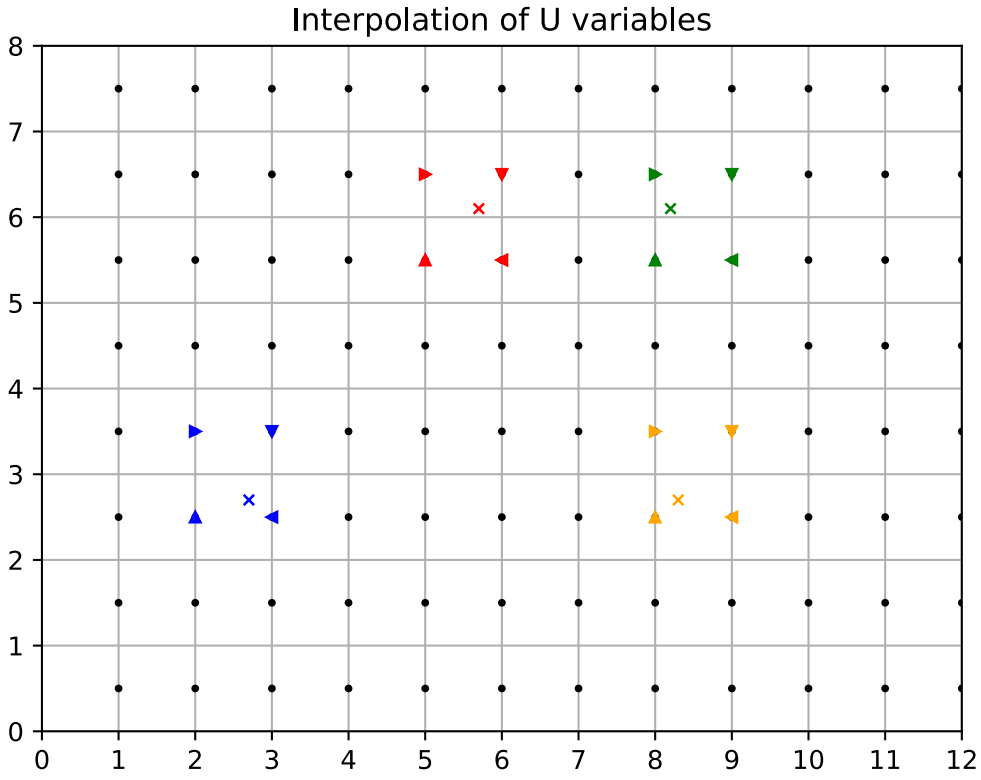
An illustration is given below




Interpolation of U variables

Interpolation of `U` variables is done as follows:

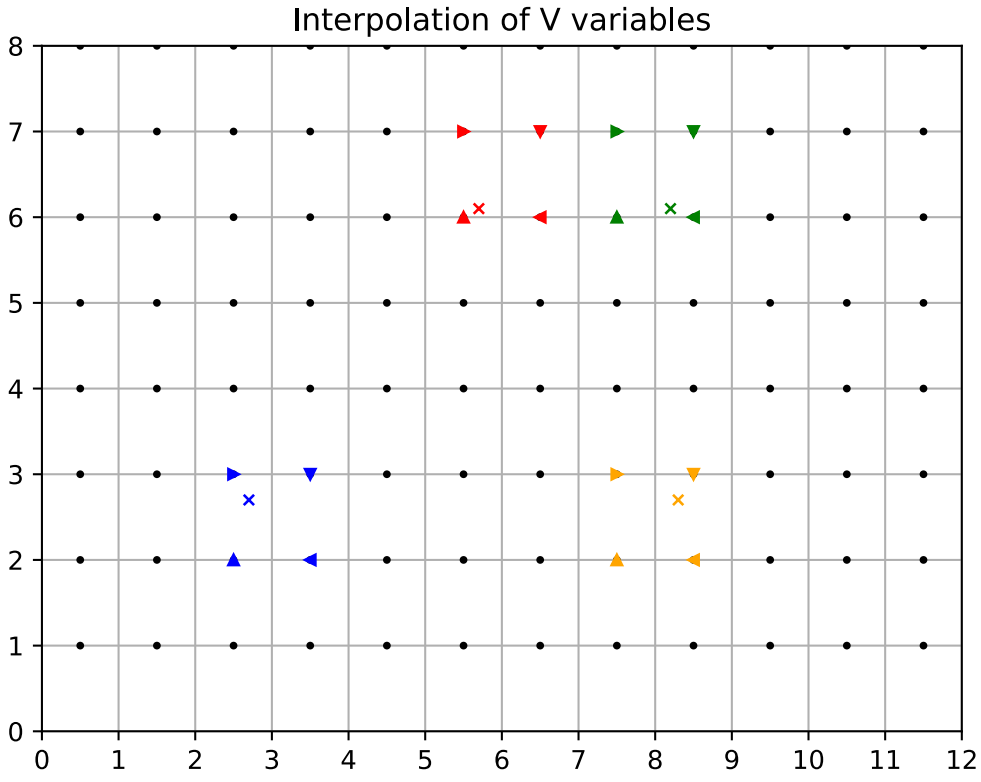
- First, the `i` index of the `U` point left of the particle is found by using `floor(x - 1)`. The `-1` is to move from the computation grid to the `U` grid system.
- Then, the `j` index of the `U` grid line below the particle is found. This is done by using `floor` on the `y - 0.5` value. The `-0.5` is to move from the computation grid to the `U` grid system.
- The box used to average the variable is therefore defined by the `[i, i + 1]` and `[j, j + 1]` squares.




Interpolation of V variables

Interpolation of `V` variables is done as follows:

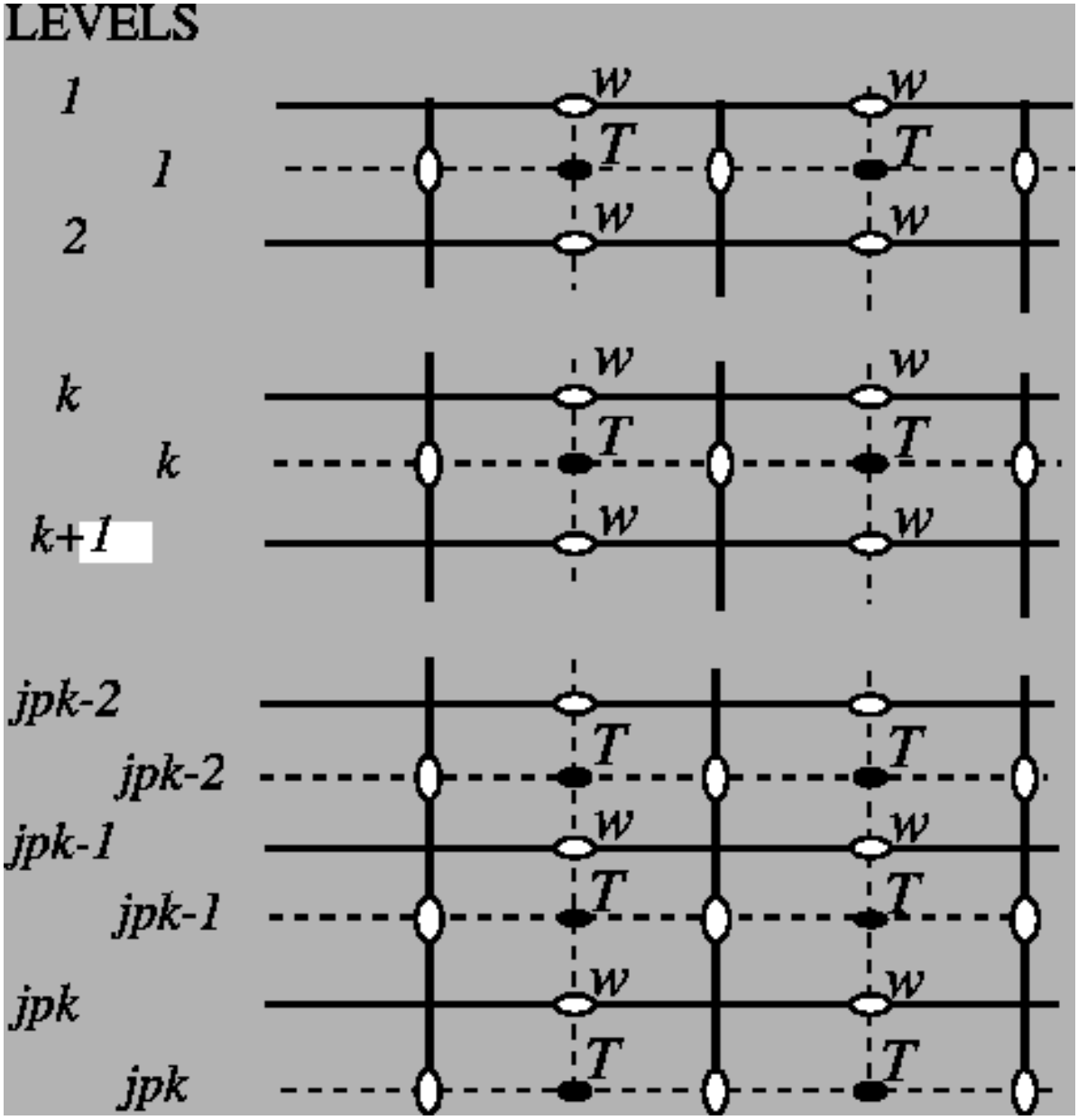
- First, the `i` index of the `V` point left of the particle is found by using `floor(x - 0.5)`. The `-0.5` is to move from the computation grid to the `U` grid system.
- Then, the `j` index of the `V` grid line below the particle is found. This is done by using `floor` on the `y - 1` value. The `-1` is to move from the computation grid to the `U` grid system.
- The box used to average the variable is therefore defined by the `[i, i + 1]` and `[j, j + 1]` squares.




#### 10.0.1.2 Vertical

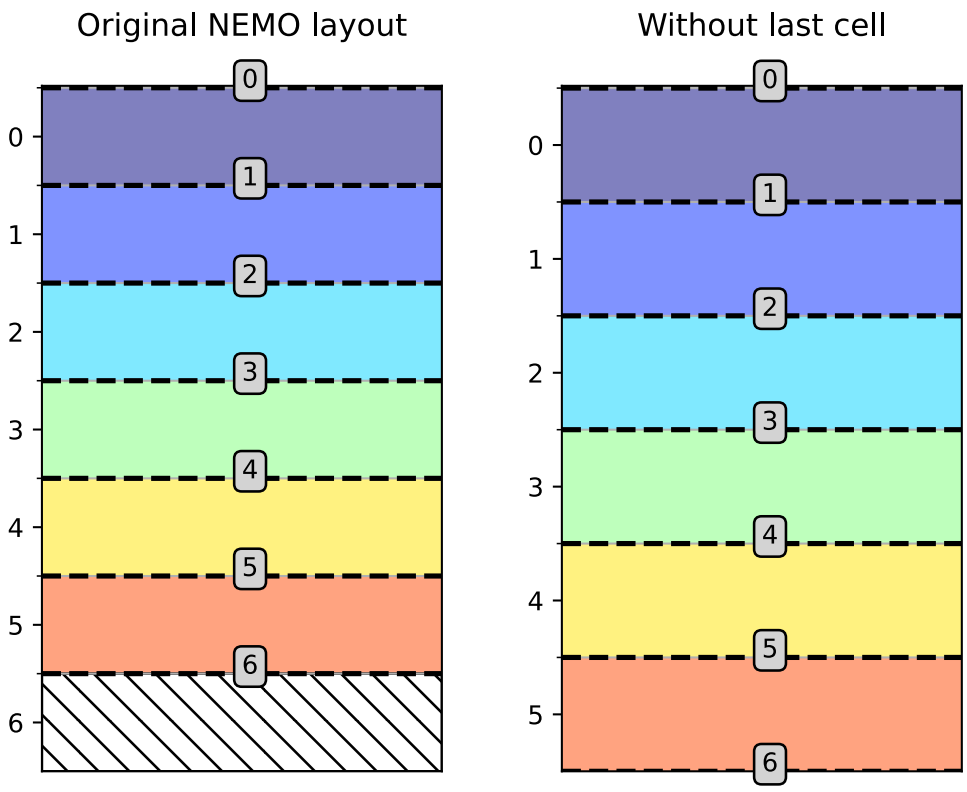
##### 10.0.1.2.0.1 Indexing

The original NEMO vertical indexing system is shown below:




Index starts at 0 (at the surface) and ends at $N_z - 1$ at depth. There are as many `W` levels as `T` levels. In NEMO, the `T` point situated at $N_z - 1$ **is always masked**. `W` levels are located *above* the corresponding `T` points.

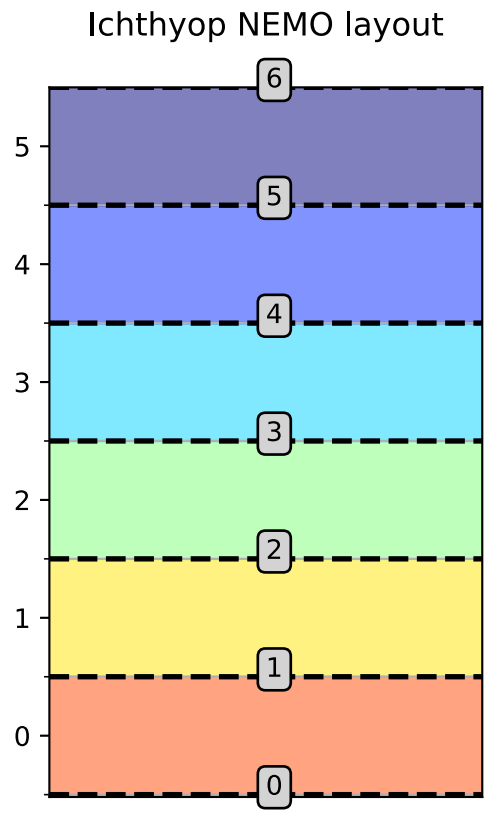
Therefore, in Ichthyop, only the first $N_z - 1$ `T` points are read, while all the $N_z$ `W` points are read. This is show below:




Vertical indexing system as used in Ichthyop. Red dashed lines represent the `W` levels (cell edges), whose index is given in the gray box.

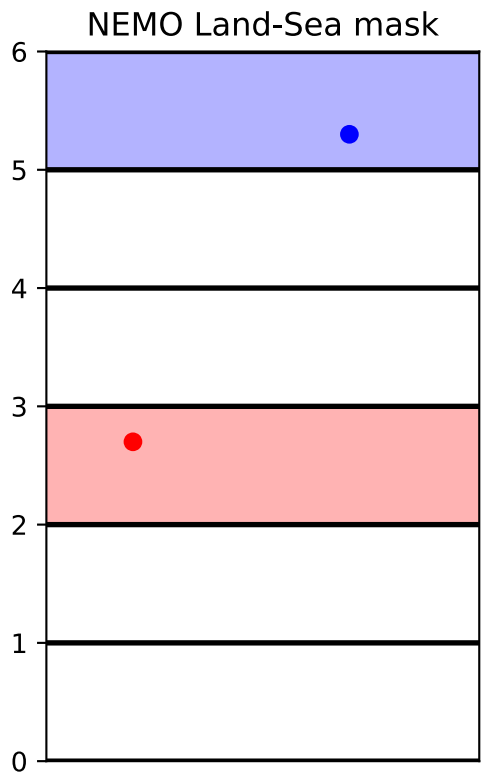
There is now $N_z$ W levels but $N_z - 1$ T levels.

Furthermore, since in Ichthyop the first index corresponds to the seabed, the arrays are vertically flipped. Consequently, the final structure of the vertical grid is as follows:




Corrected vertical indexing, with `k=0` associated with the bottom depth.

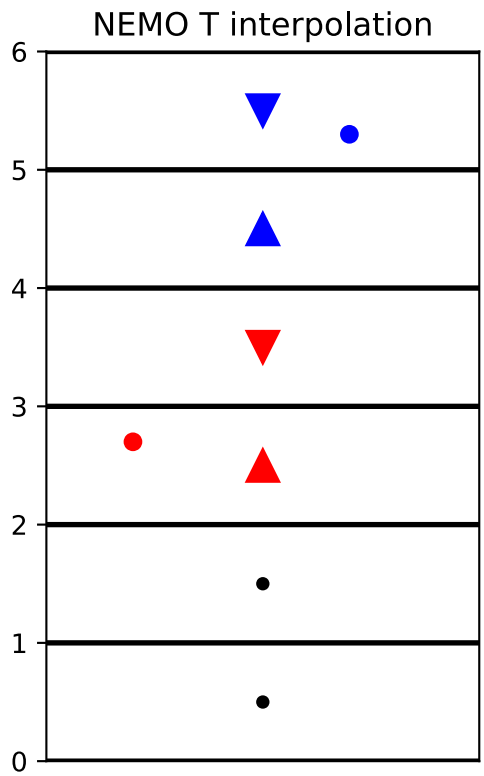
#### 10.0.1.2.0.2 Land-sea mask




Vertical land-sea mask

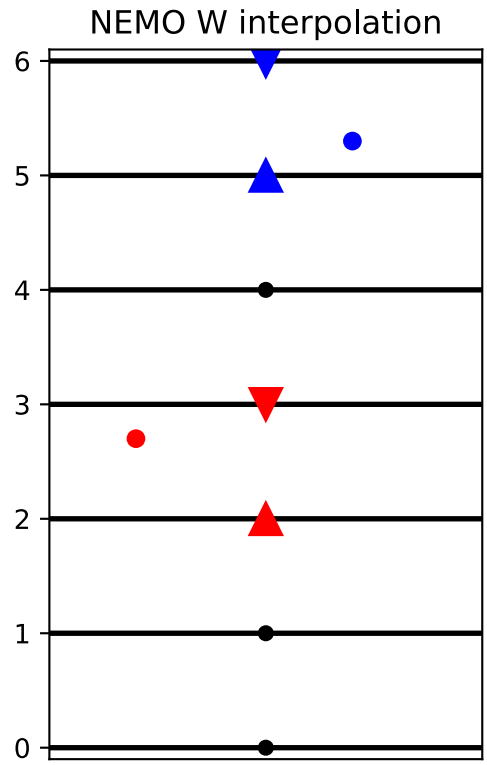
#### 10.0.1.2.0.3 Interpolation

T variables




Vertical interpolation of T variables.

W variables



:align: center :width: 250 px

Vertical interpolation of T variables.

### 10.0.2 ROMS grid

In this section, the main features of the ROMS grid and the implications in Ichthyop are summarized.

#### 10.0.2.1 Horizontal

The horizontal grid layout of the ROMS model is shown below:

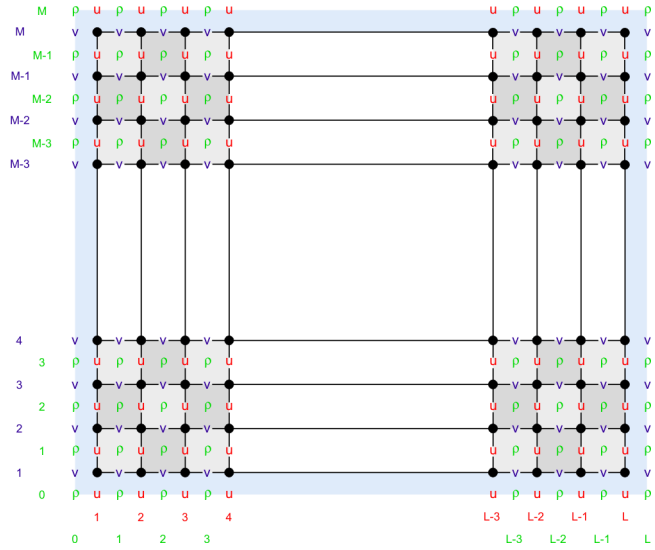


Figure 10.1: ROMS staggered grid structure

Contrary to NEMO and MARS, the U points are located on the *western* face, while velocities are located on the *southern* faces. However, However, as indicated on the Wikiroms page, while the T interior domain has $(N_y, N_x)$ dimensions, the U domain is $(N_y, N_x-1)$ points while and V domain is $(N_y-1, N_x)$ points. Indeed, the first row for V and the first column for U are discarded. Therefore, the structure of the input ROMS grid is as follows:

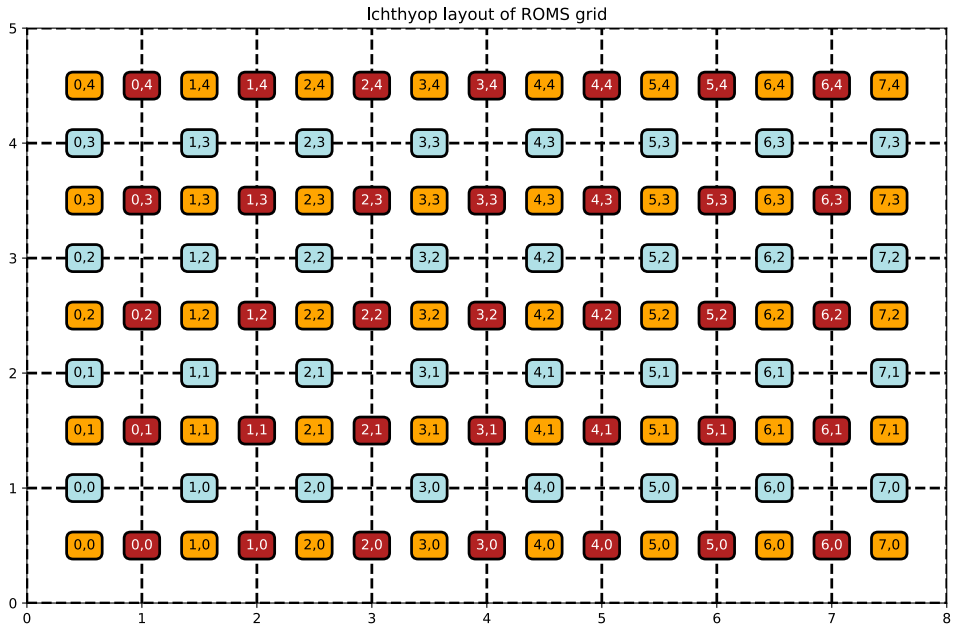



Figure 10.2: ROMS grid as interpreted by Ichthyop

#### 10.0.2.1.1 Land-sea mask

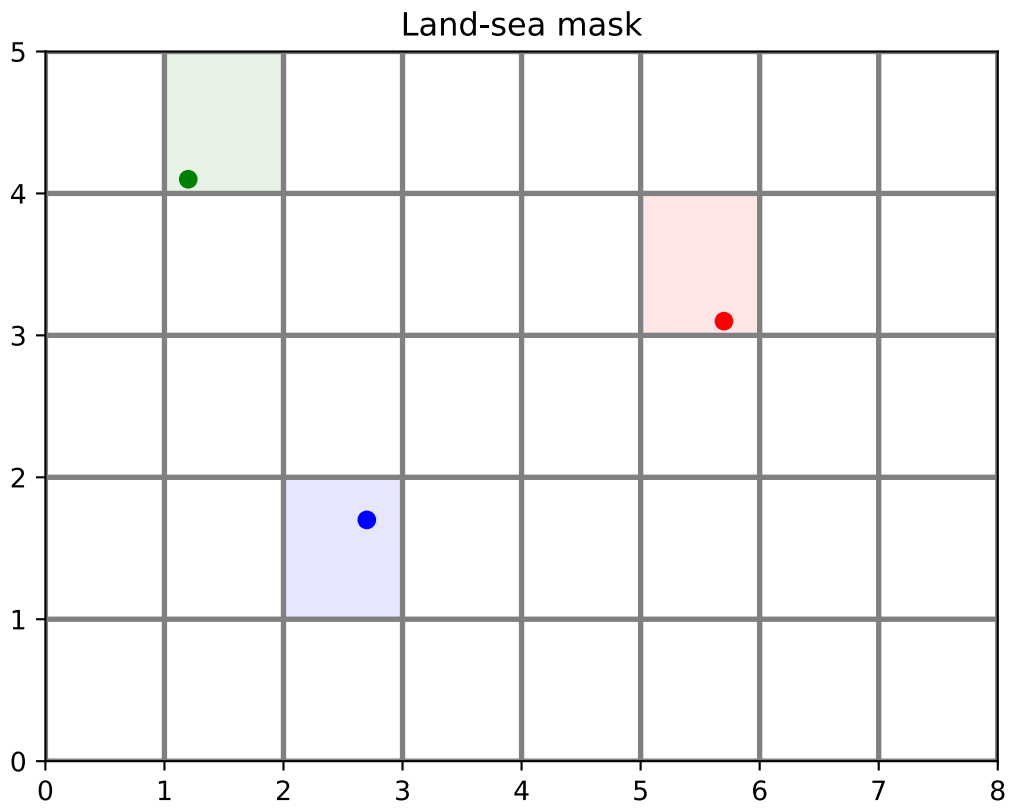



Figure 10.3: Land-sea masking for ROMS grid

#### 10.0.2.1.2 Interpolation

##### 10.0.2.1.2.1 T interpolation

Given a given position index of a particle with the `T` grid, the determination of the interpolation is done as follows:

- First, the `i` index of the `T` grid column left of the particle is found. This is done by using `floor` on the `x - 0.5` value. The removing of 0.5 is to convert the `x` value from the computational grid to the `T` grid.
- Then, the `j` index of the `T` grid line below the particle is found. This is done by using `floor` on the `y - 0.5` value. The removing of 0.5 is to convert the `y` value from the computational grid to the `T` grid.
- The area to consider is defined by the `[i, i + 1]` and `[j, j + 1]` squares.

An illustration is given below

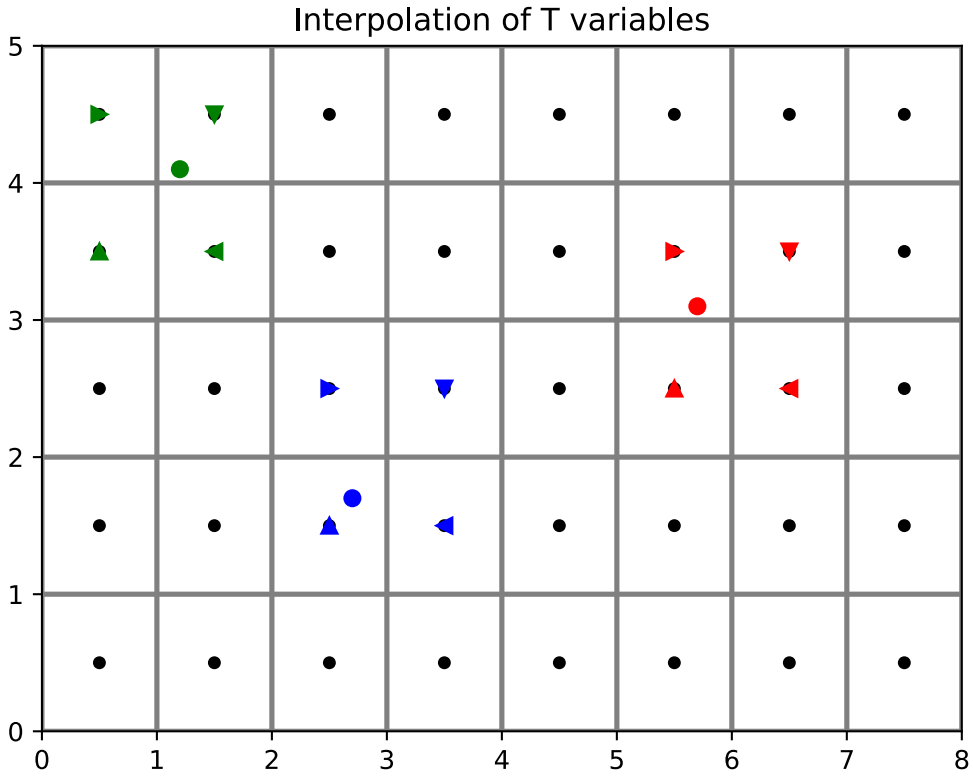



Figure 10.4: Interpolation of T points from ROMS grid

#### 10.0.2.1.2.2 U interpolation

Interpolation of `U` variables is done as follows:

- First, the `i` index of the `U` point left of the particle is found by using `floor(x - 1)`. The `-1` is to move from the computation grid to the `U` grid system.
- Then, the `j` index of the `U` grid line below the particle is found. This is done by using `floor` on the `y - 0.5` value. The `-0.5` is to move from the computation grid to the `U` grid system.
- The box used to average the variable is therefore defined by the `[i, i + 1]` and `[j, j + 1]` squares.

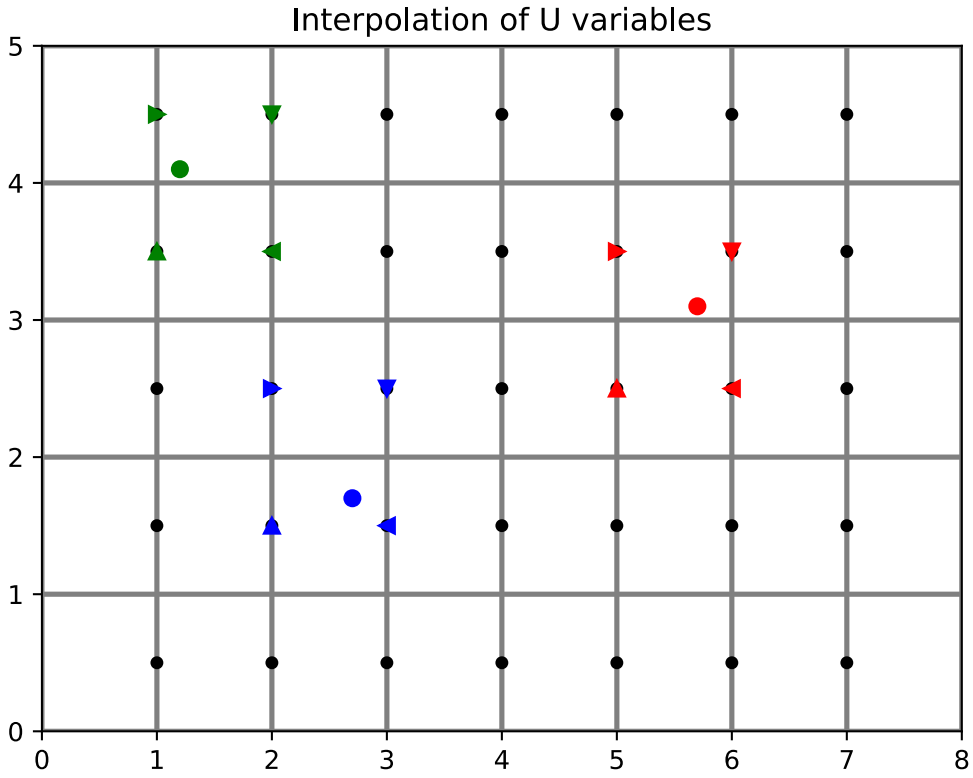



Figure 10.5: Interpolation of U points from ROMS grid

#### 10.0.2.1.2.3 V interpolation

Interpolation of `V` variables is done as follows:

- First, the `i` index of the `V` point left of the particle is found by using `floor(x - 0.5)`. The `-0.5` is to move from the computation grid to the `U` grid system.
- Then, the `j` index of the `V` grid line below the particle is found. This is done by using `floor` on the `y - 1` value. The `-1` is to move from the computation grid to the `U` grid system.
- The box used to average the variable is therefore defined by the `[i, i + 1]` and `[j, j + 1]` squares.

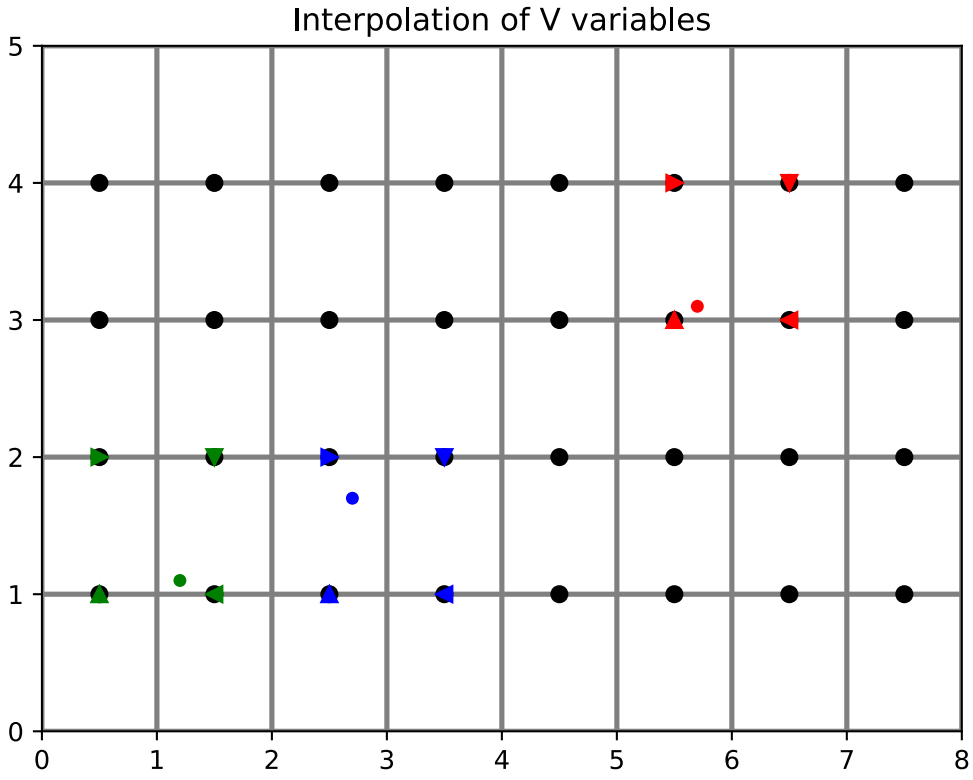



Figure 10.6: Interpolation of V points from ROMS grid

#### 10.0.2.1.3 Is on edge

A particle is considered to be out of domain when $x \leq 1$ (no possible interpolation of U on the western face), when $y \geq N_x - 1$ (no possible interpolation of U on the eastern face), when $y \leq 1$ (no possible interpolation of V on the southern domain) or when $y \geq N_y - 1$ (no possible interpolation of V on the northern part of the domain).

The excluded domain is represented below:

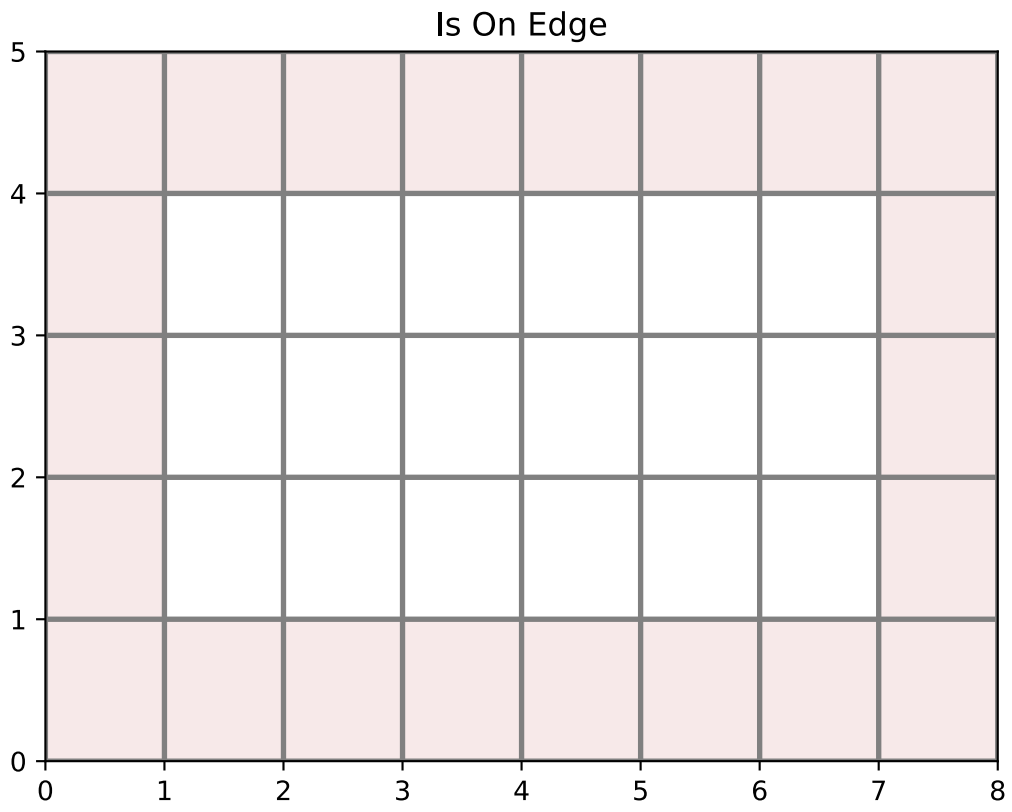



Figure 10.7: Excluded domain in the Ichthyop ROMS simulations.

#### 10.0.2.2 Vertical

##### 10.0.2.2.1 Sigma coordinate

The vertical coordinate system of ROMS is discussed on WikiRoms and shown below.

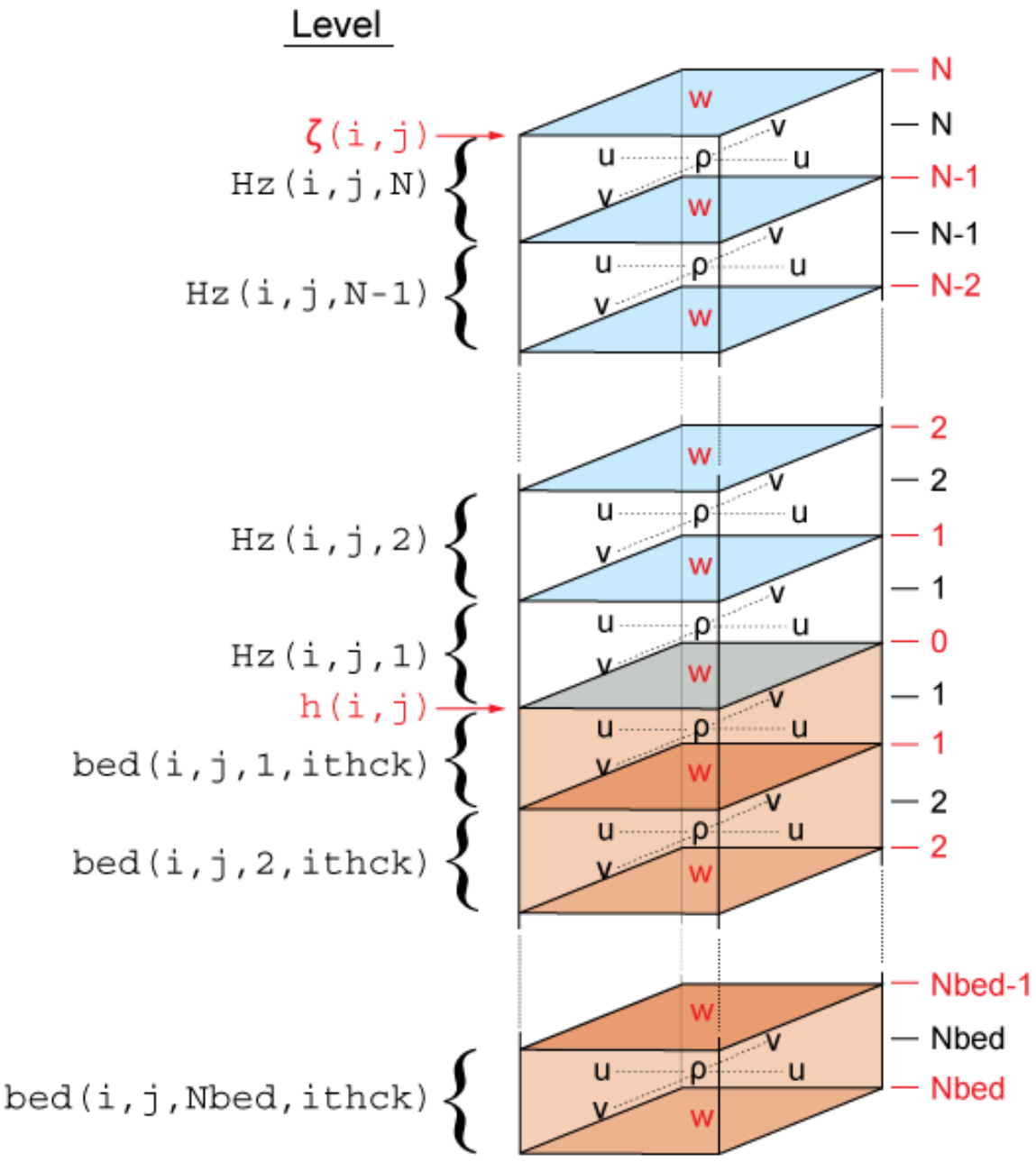



Figure 10.8: Vertical grid in the ROMS model

The vertical coordinate in ROMS is $\sigma$, which varies between $-1$ (ocean bottom) and 0 (ocean surface). There are two implementations of the $\sigma$ to $z$ conversion, both using sea-level anomalies ($\zeta$) and bathymetry ($h$).

The first one is available in ROMS since 1999 and is given by:

$$z(x,y,\sigma,t) = S(x,y,\sigma) + \zeta(x,y,t)\left[1 + \frac{S(x,y,\sigma)}{h(x,y)}\right]$$

with

$$S(x,y,\sigma) = h_c\,\sigma + [h(x,y) - h_c]\;C(\sigma)$$

and $h_c$ and $C(\sigma)$ parameters provided in the grid file.

The second transform, called UCLA-ROMS, is given by:

$$z(x,y,\sigma,t) = \zeta(x,y,t) + [\zeta(x,y,t) + h(x,y)]\;S(x,y,\sigma)$$

with

$$S(x,y,\sigma) = \frac{h_c\,\sigma + h(x,y)\,C(\sigma)}{h_c + h(x,y)}$$

and $h_c$ and $C(\sigma)$ parameters provided in the grid file.

It can be rewritten in the same form as the original one.

$$z(x,y,\sigma,t) = h(x,y)S(x,y,\sigma) + \zeta(x,y,t) + \zeta(x,y,t)S(x,y,\sigma)$$

$$z(x,y,\sigma,t) = h(x,y)S(x,y,\sigma) + \zeta(x,y,t)\,[1 + S(x,y,\sigma)]$$

$$z(x,y,\sigma,t) = h(x,y)S(x,y,\sigma) + \zeta(x,y,t)\left[1 + \frac{h(x,y)S(x,y,\sigma)}{h(x,y)}\right]$$

In this form, both formulations can be expressed as:

$$z(x,y,\sigma,t) = H_0(x,y,\sigma) + \zeta(x,y,t)\left[1 + \frac{H_0(x,y,\sigma)}{h(x,y)}\right]$$

with $H_0$ which is constant overt time, and which varies between the classical and the UCLA formulations. For the classical formulation:

$$H_0(x,y,\sigma) = S(x,y,\sigma)$$

For the UCLA formulation:

$$H_0(x, y, \sigma) = h(x, y)S(x, y, \sigma)$$

### 10.0.3 MARS grid

In this section, the main features of the MARS grid and the implications in Ichthyop are summarized.

#### 10.0.3.1 Horizontal

In MARS, the 3D structure of the grid is the same as the one in NEMO:

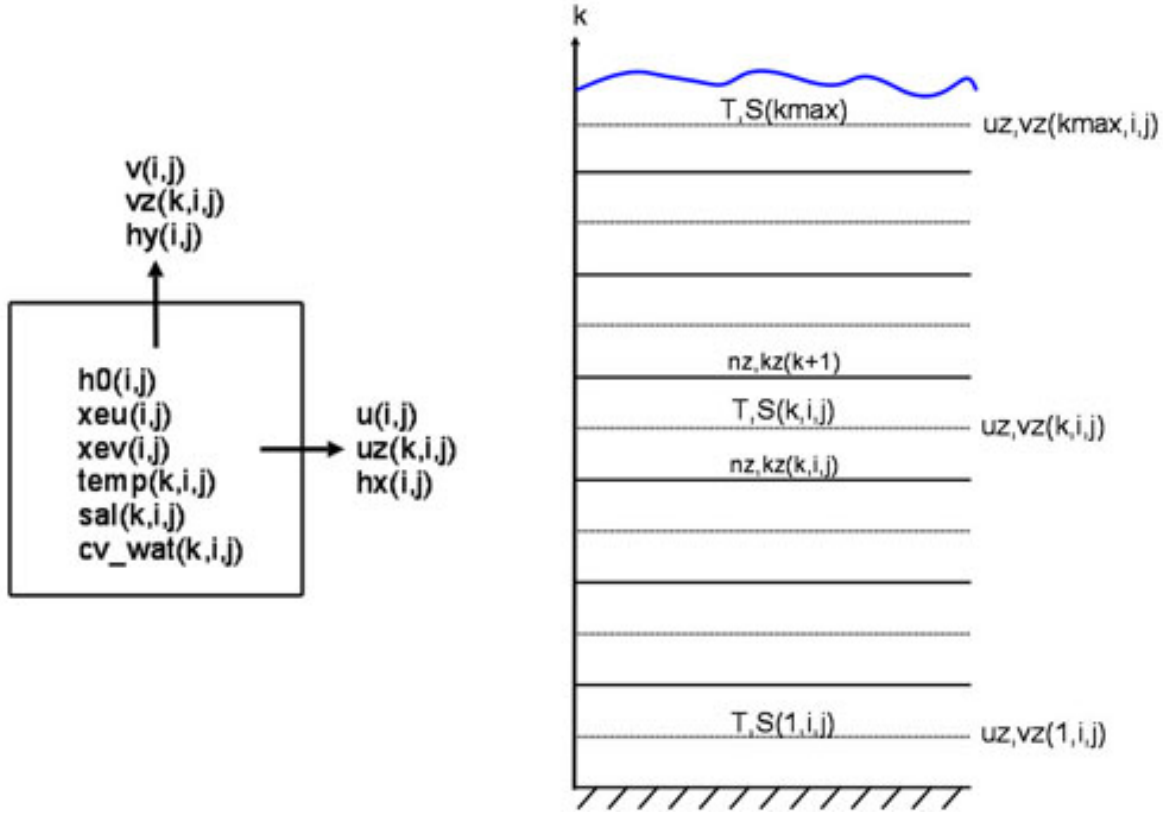



Figure 10.9: MARS staggered grid structure

#### 10.0.3.2 Vertical

The vertical coordinate of the Mars model is called $\sigma$, which varies from -1 at seabed to 0 at the surface. In MARS, if the number of `T` points on the vertical is `kmax`, the number of `W` points is `kmax + 1`. For a given `T` cell located at the `k` index, the corresponding `W` point is located below. `k=0` corresponds to the bottom, while `k=kmax` corresponds to the surface.

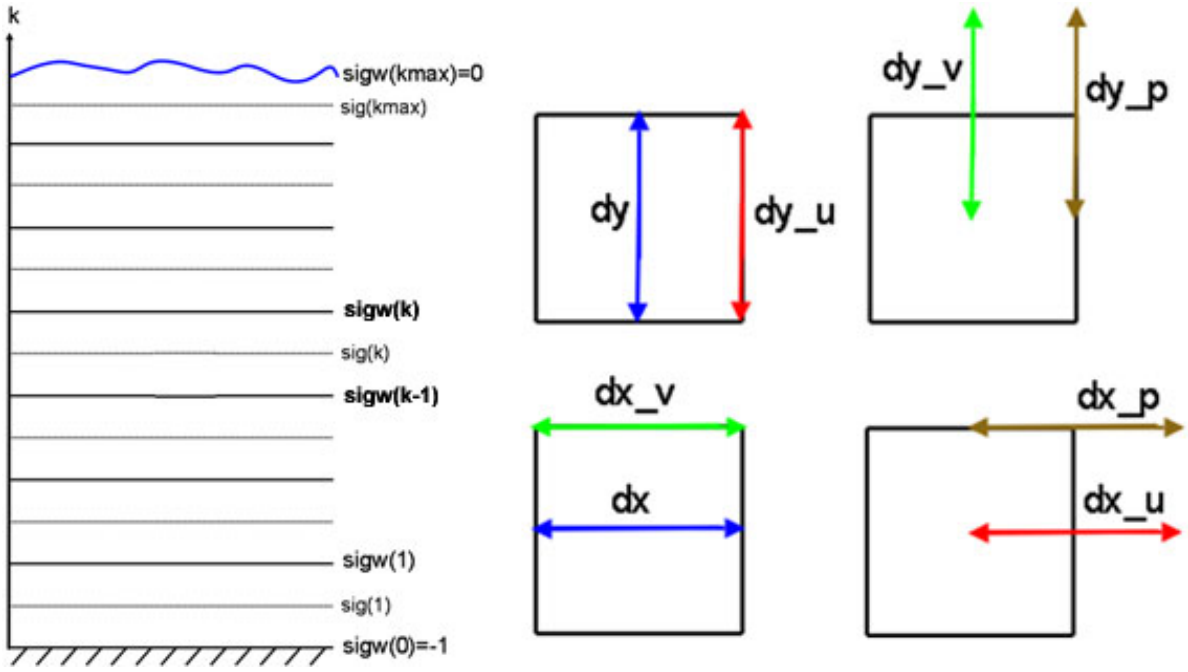



Figure 10.10: MARS grid structure.

The conversion from $\sigma$ to $z$, using generalized $\sigma$ levels, is given in Dumas (2009) (equation 1.29):

$$z = \xi(1+\sigma) + H_c \times [\sigma - C(\sigma)] + HC(\sigma)$$

where $\xi$ is the free surface, $H$ is the bottom depth and $H_c$ is either the minimum depth or a shallow water depth above which we wish to have more resolution. $C$ is defined as (equation 1.30):

$$C(\sigma) = (1-\beta)\frac{\sinh(\theta\sigma)}{\sinh(\theta)} + \beta\frac{\tanh[\theta(\sigma+\frac{1}{2})] - \tanh(\frac{\theta}{2})}{2\tanh(\frac{\theta}{2})}$$

However, if the $H_c$ variable is not found, the following formulation will be used:

$$z = \xi(1 + \sigma) + \sigma H$$

Note that the $C(\sigma)$ variable is read from the input file.

# 11 Adding new processes

To Do

# 12 Adding output variable

When including new processes to Ichthyop (cf. Chapter 11), the storage of additional variable may be required. For instance, in the growth processes (see Section 7.1), in which particle length is a state variable, it is necessary to save length in the output NetCDF file. This is done by creating a new Java class associated with a property file.

## 12.1 Creating java class

Creating the Java class depends on what type of variable you want to save, as shown in Figure 12.1.

### 12.1.1 General case

Adding new variables can be achieved by creating new tracker class in the `org.previmer.ichthyop.io` package, which inherits from the `AbstractTracker` java class. It must override the 4 methods, as shown below for the `LengthTracker` class:

```
public class LengthTracker extends AbstractTracker {

    @Override
    public void setDimensions() {
    }

    @Override
    public void addRuntimeAttributes() {
    }

    @Override
    public Array createArray() {
    }

    @Override
    public void track() {
    }
```

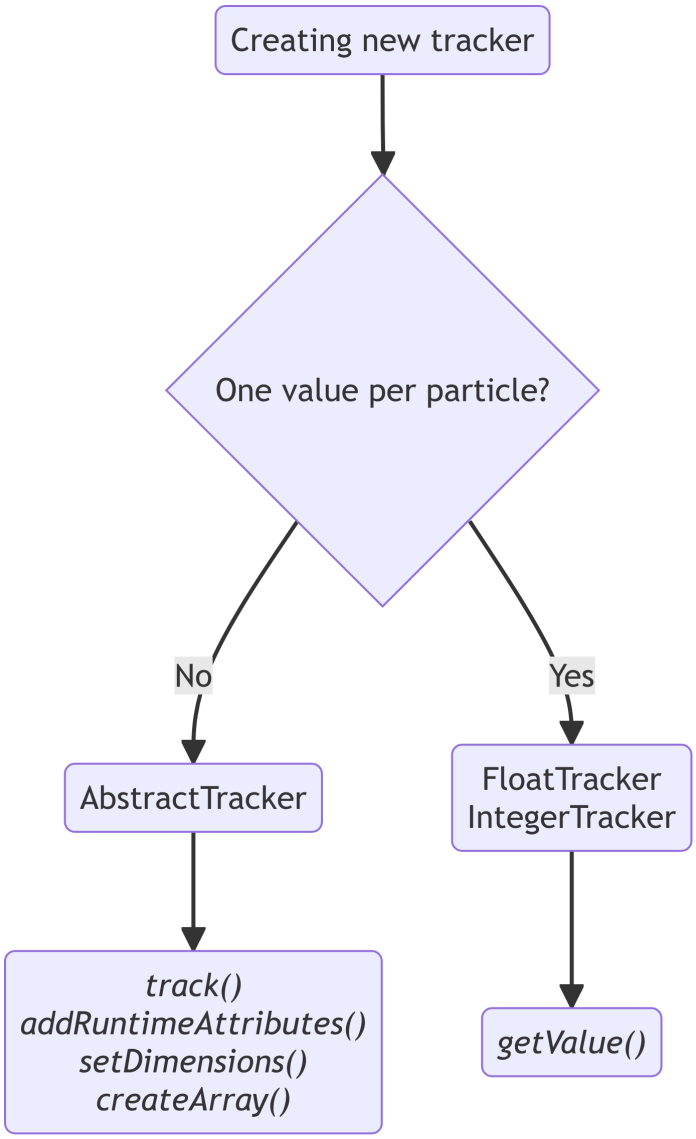



Figure 12.1: Adding a new tracker

```
}
```

`setDimensions` defines the dimensions associated with the variable. Time and drifter dimensions must be added by using the `addTimeDimension()` and `addDrifterDimension()` methods, respectively. A zone dimension can be added by calling the `addZoneDimension(TypeZone zoneType)` method. Custom dimensions can be added by using the `addCustomDimension(Dimension dim)` method.

`addRuntimeAttributes` defines additional attributes associated with the variable to be saved. Attributes are added by calling the `addAttribute(Attribute attribute)` method.

> **Note**
>
> Compulsory attributes and variable names are defined using properties files, see Section 12.2

`createArray` initializes the `Array` object that will be used to store the output variable. The dimensions of the array depends on the dimension of the output variables.

`track()` is the method that is called at each output time-step and which writes the variable in the NetCDF.

### 12.1.2 Simple case

Usually, new variables consist in tracking one single particle property, such as length for instance. In this case, the new tracker class can inherits either from the `FloatTracker` or the `IntegerTracker` classes as follows:

```
public class LengthTracker extends FloatTracker {

    @Override
    float getValue(IParticle particle) {
        ...
    }
}
```

In this case, the only method to define is the `getValue(IParticle particle)`, which specifies which particle's state variable is to be extracted for the given tracker.

## 12.2 Creating property file

In addition to the tracker java class, a property file must be included in the `io/resources/` folder. The name of this file must me the same as the Java class, except for the `.properties` suffix. For instance, the property file associated with the `LengthTracker.java` class will be named `LengthTracker.properties`. It must contain the following three lines:

```
tracker.shortname = length
tracker.longname = particle length
tracker.unit = millimeter
```

`tracker.shortname` is the name of the variable in the NetCDF, `tracker.longname` and `tracker.unit` are the values of the variable's longname and unit attributes.

# 13 References


Fonds, M. 1979. "Laboratory Observations on the Influence of Temperature and Salinity on Development of the Eggs and Growth of the Larvae of Solea Solea (Pisces)." *Mar. Ecol. Prog. Ser* 1 (9).

Lett, Christophe, Philippe Verley, Christian Mullon, Carolina Parada, Timothée Brochier, Pierrick Penven, and Bruno Blanke. 2008. "A Lagrangian Tool for Modelling Ichthyoplankton Dynamics." *Environmental Modelling & Software* 23 (9): 1210–14. https://doi.org/https://doi.org/10.1016/j.envsoft.2008.02.005.

Parada, C, CD Van Der Lingen, C Mullon, and P Penven. 2003. "Modelling the Effect of Buoyancy on the Transport of Anchovy (Engraulis Capensis) Eggs from Spawning to Nursery Grounds in the Southern Benguela: An IBM Approach." *Fisheries Oceanography* 12 (3): 170–84.

Staaterman, Erica, Claire B. Paris, and Judith Helgers. 2012. "Orientation Behavior in Fish Larvae: A Missing Piece to Hjort's Critical Period Hypothesis." *Journal of Theoretical Biology* 304 (July): 188–96. https://doi.org/10.1016/j.jtbi.2012.03.016.

Stokes, George Gabriel. 2009. "On the Theory of Oscillatory Waves." In *Mathematical and Physical Papers*, 1:197–229. Cambridge Library Collection - Mathematics. Cambridge University Press. https://doi.org/10.1017/CBO9780511702242.013.

Tanner, Susanne E., Ana Teles-Machado, Filipe Martinho, Álvaro Peliz, and Henrique N. Cabral. 2017. "Modelling Larval Dispersal Dynamics of Common Sole (Solea Solea) Along the Western Iberian Coast." *Progress in Oceanography* 156: 78–90. https://doi.org/https://doi.org/10.1016/j.pocean.2017.06.005.