# **MOHID – Lagrangian**

# Starter Guide for end Users

### VERSION STATUS: Revised on 3rd June 2025 by USC, MI, INTECMAR and COLAB+Atlantic

### PROLOGUE

This manual is a starter guide to operate the MOHID-Lagrangian model. I allows end-users to start to produce some outputs and explore the different setups and options in a fast way.

INDEX

[MOHID – LAGRANGIAN - Guide 1](#_Toc199751364)

[PROLOGUE 1](#_Toc199751365)

[INDEX 2](#_Toc199751366)

[1. INTRODUCTION: 3](#_Toc199751367)

[1.1 Test working simulations or template folder 3](#_Toc199751368)

[1.2 Create a new simulation from Case\_template 4](#_Toc199751369)

[1.3 Setup the Main\_Case.xml setup file. 5](#_Toc199751370)

[1.3.1 Setup the <Execution> block 6](#_Toc199751371)

[1.3.1.1 Change the Start time and end time 7](#_Toc199751372)

[1.3.1.2 Change the Integrator. 7](#_Toc199751373)

[1.3.1.3 Change the number of Threads 8](#_Toc199751374)

[1.3.1.4 Change the output frequency. 8](#_Toc199751375)

[1.3.1.5 8](#_Toc199751376)

[1.3.1.6 Change the buffer size 8](#_Toc199751377)

[1.3.1.7 Change the output format. 9](#_Toc199751378)

[1.4 Setup the netCDF input variable naming 9](#_Toc199751379)

[1.5 Setup the output particle properties fields 13](#_Toc199751380)

[1.6 Setup the post processor: the recipes 13](#_Toc199751381)

[1.6.1 Setup the recipes 13](#_Toc199751382)

[1.6.2 Recipe time range 14](#_Toc199751383)

[1.6.3 Recipe output fields 14](#_Toc199751384)

[1.6.4 Recipe grid region and resolution 15](#_Toc199751385)

[1.6.5 Recipe Polygon definition. 16](#_Toc199751386)

[1.6.6 Plotting 17](#_Toc199751387)

[1.6.7 Plotting – Time resample and group 17](#_Toc199751388)

[1.6.8 Recipe Weight. 19](#_Toc199751389)

[1.6.9 Plotting – Apply functions on data 19](#_Toc199751390)

[1.6.10 Convert files to hdf5 19](#_Toc199751391)

[1.7 Setup the <caseDefinitions> 20](#_Toc199751392)

[1.7.1 netCDF input files 20](#_Toc199751393)

[1.7.2 Setup the simulation 21](#_Toc199751394)

[1.7.3 Setup the sources 22](#_Toc199751395)

[1.7.3.1 Setup the sources ID 23](#_Toc199751396)

[1.7.3.2 Setup the source rate 24](#_Toc199751397)

[1.7.3.3 Setup the source geometry 25](#_Toc199751398)

[1.7.4 Setup the particle types 28](#_Toc199751399)

[1.8 Setup the constants 30](#_Toc199751400)

### INTRODUCTION:

The intended way to use MOHID – Lagrangian is to follow the scheme:

1. **Base template**
2. **Copy it**
3. **Personalize it**

That is, you must copy a folder with a template or other working simulation test. And then, modify the setup xml files inside to fit your needs.

Let's go step-by-step on how to prepare a working simulation:

### Test working simulations or template folder

We provide different working examples in the path:

**/MOHID-Lagrangian/RUN\_Cases/**

Inside, there are the following cases:

* Arousa\_2D\_ship\_test\_case
* Arousa\_2D\_test\_case
* PCOMS\_test\_case
* Tagus3D\_case
* Vigo\_3D\_test\_case
* Case\_template (do not run)

The first five cases are working setups of MOHID-Lagrangian with the files configured to work properly, just to test how MOHID-Lagrangian works. If everything is fine and there are no problems at the installation stage, each case should run fine just by running the scripts inside each test\_case folder:

**RunCase.sh (Linux)**

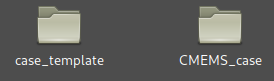
**RunCase.bat (windows)**

The Case\_template does not have a working executable version; however, it contains different templates or files to setup the simulation according to your needs. From here, we will use this folder as a template to create a new simulation.

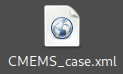
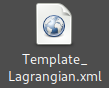
### Create a new simulation from Case\_template

Suppose we want to perform a new simulation using CMEMS surface current data. This hydrodynamic dataset contains the currents speeds (u, v and w) in *netcdf*. The first step is to:

1) copy the **case\_template** folder into a new one a rename it to **CMEMS\_case** for example:

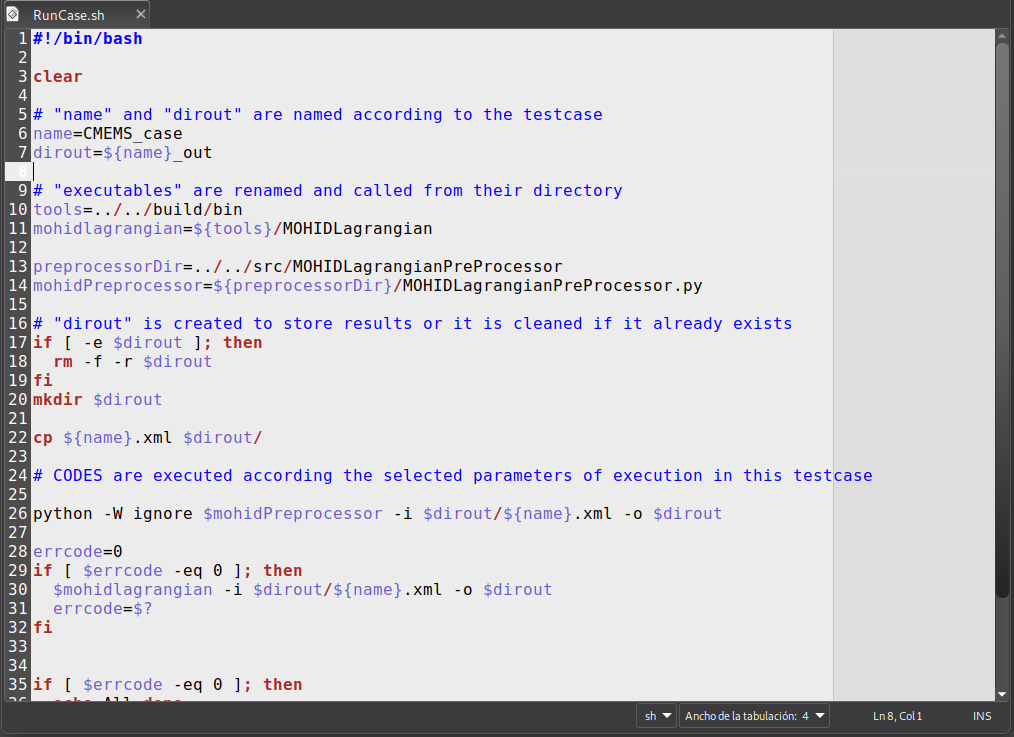


2) Then, to keep all names in agreement (optional but recommended) **rename the file Template\_Lagrangian.xml** to **CMEMS\_case.xml**

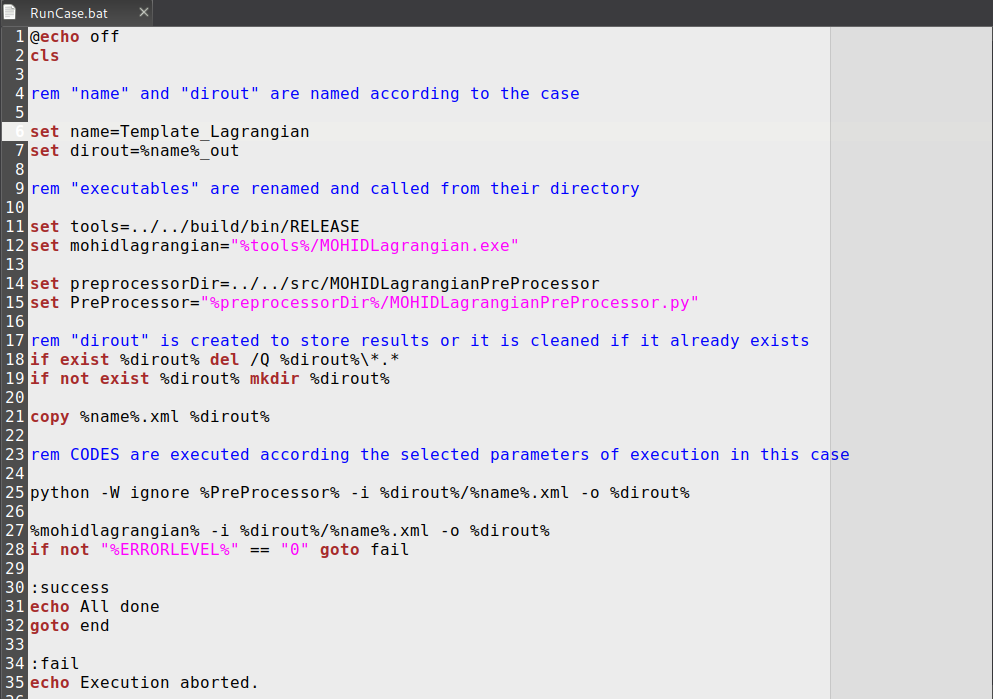


3) Open with your text editor the file **RunCase.sh (in Linux) or RunCase.sh (in windows)** and edit the line number 6) with:

* LINUX – (RunCase.sh) name=Template\_Lagrangian -----> name=CMEMS\_case
* Extra: if you always name your template .xml as the folder as it is in this example, you can also set name="$(basename "$(pwd)")". This way, you do not have to change the RunCase.sh every time you create a new example case.



* WINDOWS – (RunCase.bat) set name= Template\_Lagrangian -----> set name = CMEMS\_case

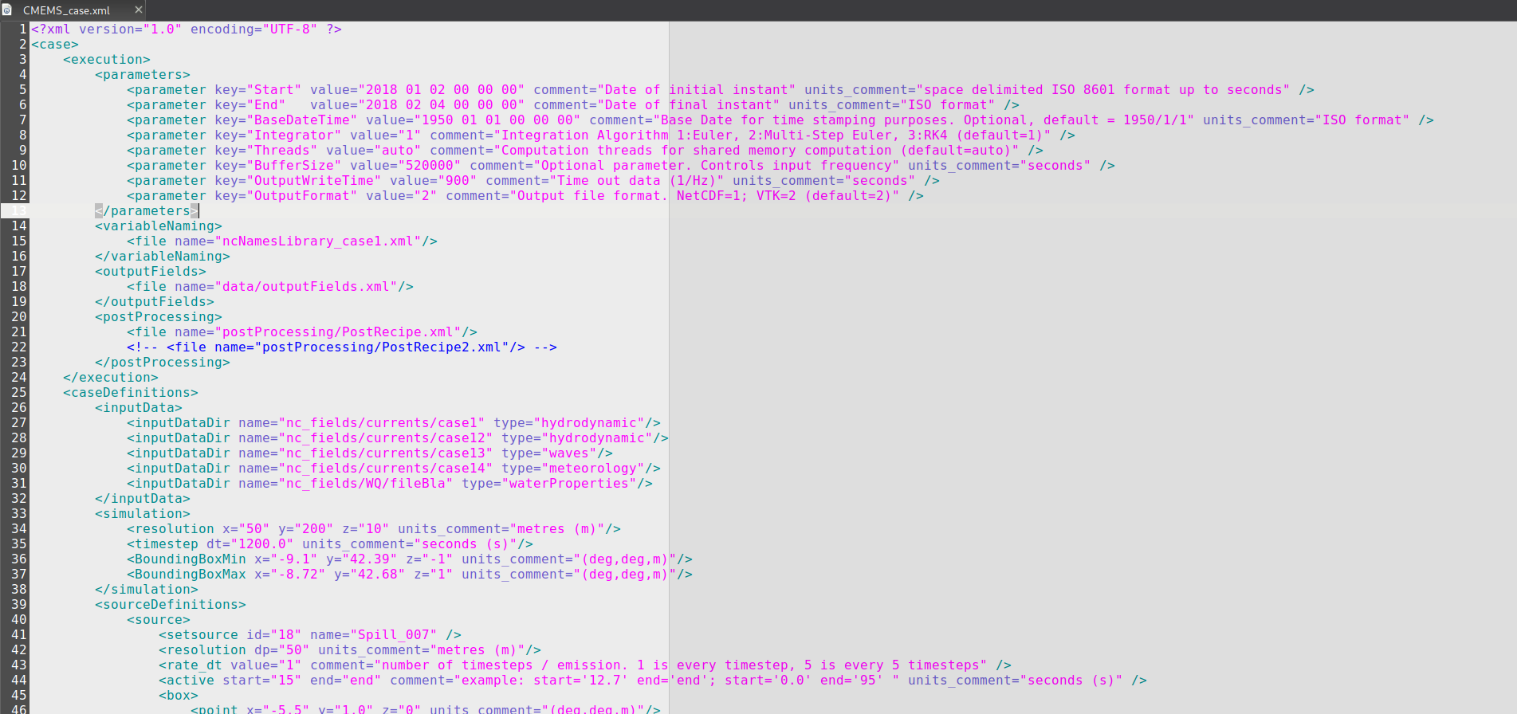


### Set up the Main\_Case.xml setup file.

Once we have renamed the executable files, then we can start to edit the main setup file of our simulation:

**CMEMS\_case.xml**

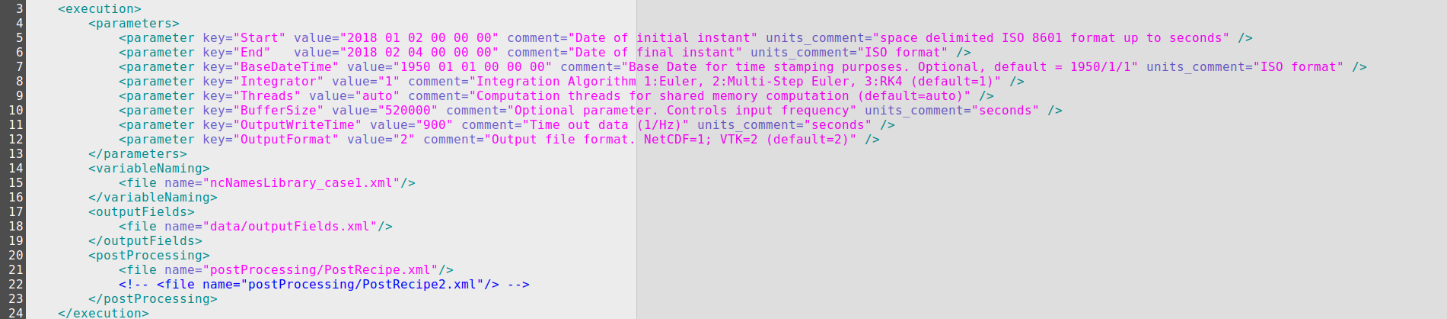
When you open it, you can observe the following text in xml format**:**



***For those who have no idea what xml is: the xml file has a tree structure, with blocks or sections and subblocks or subsections. Each section begins with <nameSection> and ends with </nameSection>. If it is an inline section (a section in only one line, it is <namesection -------/>. Inside each section, there is always a pair or many pairs of “nameofsomething”=”valueofsomething”.***

### 1.3.1 Setup the <Execution> block

We will start from the top to set up each condition or parameters. The first main block you can observe is the <execution> block.



This block controls, in accordance with its name, the execution parameters. In the *comment* field, there is a description about each entry. In the ***<parameters>*** section, there are sub-inline section called **<parameter>** with the template:

*<parameter key=”the name of the parameter” value=”its value” comment=”help comments” units\_comment=”and units if it neededs it” />*

Using this structure, we start to set up the different <parameters> of this section.

### Change the Start time and End time

These two parameters control the dates when the simulation starts and finishes.

*<parameter key="Start" value="2018 01 02 00 00 00" comment="Date of initial instant" units\_comment="space delimited ISO 8601 format up to seconds" />*

*<parameter key="End" value="2018 02 04 00 00 00" comment="Date of final instant" units\_comment="ISO format" />*

The format is always the same*: value=”Year Month Day Hour Minute Second”,* and it establishes the time domain for the simulation case.**Be sure that the time limits of when your simulation starts and/or ends is within your time range of *netCDF* input data.** The model then searches for hydrodynamic fields (wind, waves, properties, etc.) in the folder indicated below in the InputData subsection, in the block <*caseDefinitions*> (more on this later in the manual). It is advisable to have hydrodynamic data for a few moments prior to the simulation start date (at least 1 day), or in other words, not to start the simulation at the same time that the hydrodynamic data starts.

An example: to perform a two-year simulation from 2017 to 2019 we must change the time period from the previous one to:

*<parameter key="Start" value="2017 01 01 00 00 00" comment="Date of initial instant" units\_comment="space delimited ISO 8601 format up to seconds" />*

*<parameter key="End" value="2019 01 01 00 00 00" comment="Date of final instant" units\_comment="ISO format" />*

### Change the Integrator.

The next block controls the type of integrator to use:

*<parameter key="Integrator" value="2" comment="Integration Algorithm 1:Euler, 2:Multi-Step Euler, 3:RK4 (default=1)" />*

In the comment, you can observe the different options:

1. Euler (faster, but less precise)
2. Muli-Step Euler (slower than Euler, but more precise)
3. RK4 (slower than Euler, but more precise)

So, by changing the value number, you can set the desire integrator. In most cases, using option “2” is a well balance choice between accuracy and computational time demanding.

### Change the number of Threads

The next <parameter> controls the number of threads for shared memory computation (at the moment, MOHID-Lagrangian can run in one machine only).

*<parameter key="Threads" value="2" comment="Computation threads for shared memory computation (default=auto)" />*

To get better performance, this value should be equal to the number of cores available in your system. That is what the auto option does. If your machine has 8 physical cores but you want to use only 2 processes, change the value from ”auto” to “2”.

### Change the output frequency.

### 

The next parameter <key=”OutputWriteTime”> controls the **frequency to write data to disk**. And this means that we can use a different interpolation time step than the data writing time step, which can make the output lighter for further post-processing. For example, if your solution is computed every hour (dt=3600s), you could have an <OutputWriteTime value=”10800”>, which means that for every three time-steps (10800/3600 = 3) of computation, the third one is written to disk. In other words, the results in this case are calculated every hour, but the output file has the results grouped every 3 hours.

This value is **highly recommended to be an entire (integer) multiple** of the time step solution of the <simulation> block, to avoid a desynchronization between the solver and the writer.

*<simulation>*

*<timestep dt="1200.0" units\_comment="seconds (s)"/>*

*</simulation>*

### Change the buffer size

The parameter with <*key=”BufferSize*”> is important to perform long integrations. For large hydrodynamic fields, it allows you to control the amount of data to store in RAM memory (you cannot load 30GB of hydrodynamic fields if your computer has 8GB of RAM 😉) before going to disk to read a new chunk of data and continue the integration. This means that for example, if we have a month of hydrodynamic data contained in different files, we do not have to load the entire file into memory (which would be computationally very expensive) but take a fraction of this data. A buffer size of value=”520000” means that MOHID-Lagrangian is going to read a chunk of data of “520000” seconds and copy it into memory, use that data to simulate the particle motion and then if it requires data to continue, it adds it while the previous data is released from RAM memory. **This value should be changed only if you have some error** related to “not enough RAM memory”. You should make it lower in that case.

### Change the output format.

The value <key= ”outputformat”> controls the output format files to write to disk. At the moment, for the particle position at each time step, MOHID-Lagrangian just supports the output of *vtk* files, so the <*value>* must be kept at ”2”. Nevertheless, there is a postprocessing module (in python) under development to convert *vtk* output to *netCDF*. This postprocessing will yield particle concentrations in particular regions that can be specified by the user in the PostRecipe (see Section 1.6). The *vtu* format is a format intended to be read by fluid visualization programs such as *ParaView*. In a “rawer” form, *vtu* files can be converted to *csv* from python using the *pyvista* and *pandas* libraries.

### Set up the netCDF input variable naming

The variable *<variableName>* section consists in just a *filepath* to a *xml* file.

*<variableNaming>*

*<file name="ncNamesLibrary\_case1.xml"/>*

*</variableNaming>*

You can change the file name of the *xml* here, but you will also need to rename the actual *xml* file. This file controls the naming conventions of your netCDF input files. Why is this useful? Because different models can be used to generate input data, e.g. hydrodynamics, and they may attribute different names to the same variables. Commonly, the latitudinal component of the velocity is called “v” or “vo”, but it can also be defined as “v0” or any other name. How should MOHID-Lagrangian know that your “strange\_name\_v\_velocity\_component” or your “z\_evil\_dimension” inside the *netCDF* file means “vo” and simply “depth”, respectively? For the model to be able to read this latitudinal velocity variable, the way it is named in the input netcdf file must be previously defined in the *ncNamesLibrary* file. That’s why this library file was created. **This *xml* is a dictionary/translator for the CF compliant names of variables and dimensions, so that other variants of the variables can appear in *netCDFs*, which do not follow a common naming convention**. The pattern used is:

*<standandard\_cf\_variable\_name name=’variable\_name’>*

*<variant name=’variable\_name’>*

*</standandard\_cf\_variable\_name >*

The *xml* file contains the following text:



Take this example: Imagine that your input *netCDF* file(s) contain(s) the hydrodynamic fields **“utotal”** and **“vtotal”. These names are meant** to describe the CF compliant variables <eastward\_sea\_water\_velocity> and <northward\_sea\_water\_velocity> respectively. In that case, you should add the following lines to this *xml* file:

*<eastward\_sea\_water\_velocity name="u">*

*<variant name="u" comment="used in MOHID" />*

*<variant name="uu" />*

*<variant name="U" />*

*<variant name="uo" comment="used in CMEMS" />*

***<variant name="utotal">***

*</eastward\_sea\_water\_velocity>*

*<northward\_sea\_water\_velocity name="v">*

*<variant name="v" comment="used in MOHID" />*

*<variant name="vv" />*

*<variant name="V" />*

*<variant name="vo" comment="used in CMEMS" />*

***<variant name="vtotal">***

*</northward\_sea\_water\_velocity>*

Take another example: If you want to add the waves effect through the *stokes velocity drift*, you should add the variant:

*<stokes\_velocity\_drift\_x name=’vsdx’>*

*<variant name=’vsdx’*

*</stokes\_velocity\_drift\_x name>*

*<stokes\_velocity\_drift\_y name=’vsdx’>*

*<variant name=’vsdy’*

*</stokes\_velocity\_drift\_y name>*

The variant name allows MOHID-Lagrangian to seek those variant names instead of expecting the standard name. In the *<dimensions>* section*,* the same logic applies.



If, for instance, your depth dimension inside your *netCDF* file is *“z\_depth”,* you must add this to the *<vertical name=”level”>, with* the following information:

<vertical name="level">

<variant name="depth" />

<variant name="Depth" />

<variant name="DEPTH" />

<variant name="level" />

<variant name="Level" />

<variant name="z\_depth" />

</vertical>

### Set up the output particle properties fields

The block <*outputFields*>, like the previous one, contains the file path to the *xml* file which controls the outputs of the variables to be written in the *vtk* files.

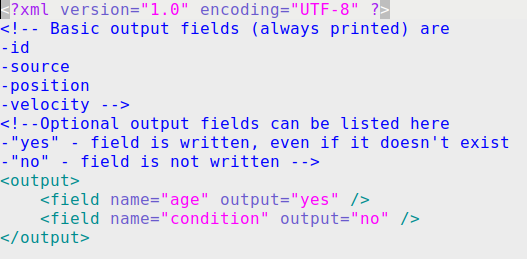
*<outputFields>*

*<file name="data/outputFields.xml"/>*

*</outputFields>*

If you **change** the path "*data/outputFields.xml*", please change the <*path/name.xml>* of the file outside the *xml* too.

This <*outputFields.xml>* file inside the path data contains:

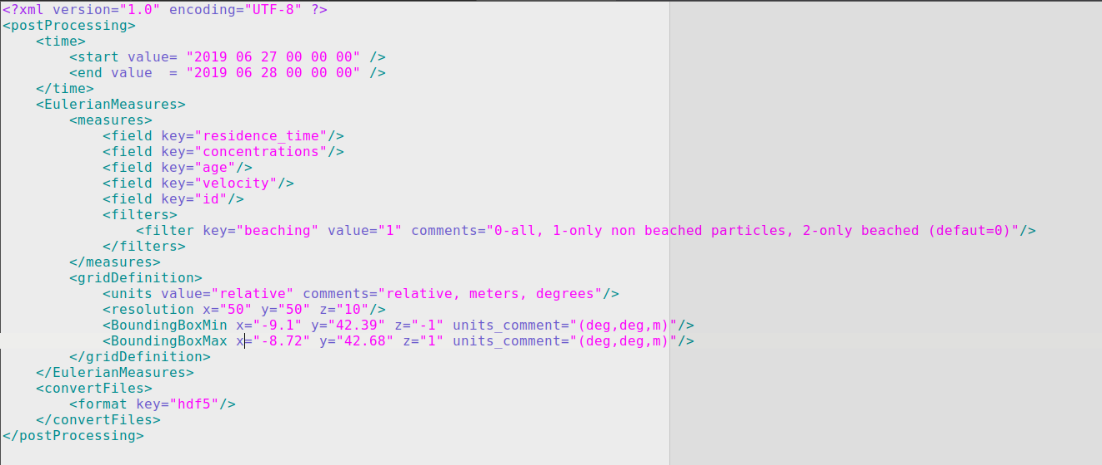


The *<output>* block controls whether a variable is written to disk or not. Notice that the basic output fields are always written to disk: id of the particle, source, position and velocity at any given time step. But you can add other variables by writing them in the outputFields.xml file.

### Setup the post processor: the recipes

Like the previous block, the *<postprocessing>* block controls the “recipes” file paths. The recipes are *xml* files controlling the post processing stage: the output fields produced, their type, and the conversion of the output files to other file formats. You can add many “recipes” with different setups. Each recipe produces one *netCDF* output field placed inside the output folder where the *vtu* files are stored and it contains the different fields specified in the *xml* recipe. The output will be written in a structured *netCDF* data with dimensions [*time, depth, latitude, longitude*] in a rectangular grid with boxes (*depth, latitude, longitude*) with size specified inside the *recipes.xml* files.

### Setting up the recipes



### Recipe time range

The first part of the <postProcessing> block looks like this:

*<time>*

*<start value= "2018 01 01 00 00 00" />*

*<end value = "2018 02 01 00 00 00" />*

*<step =2 />*

*</time>*

It controls the time range of your simulation to send to the postprocessor. If you have a long simulation (one year for example), you can make a sub-selection in time and focus in one month (e.g., a simulation could go, in the *CMEMS\_case.xml*, from *2017 01 01 00 00 00* to *2019 01 01 00 00 00* and here we decided to use particle trajectories in the date range *2018 01 01* to *2018 02 01*). Also, we add the <*step>* keyto sub select every *“n”* steps and to avoid memory overload for simulations with too many *vtu* files. In case a time range is not specified, the <*step>* keyword can be used for all timesteps. Also, you can set <*start value= "start">* and *<end value = "end">* to avoid writing the precise dates.

### Recipe output fields

The block “*Eulerian measures*” consist of translating the particle measures or the properties carried out by the particles to instantaneous grid measures.

*<measures>*

*<field key="residence\_time"/>*

*<field key="concentrations"/>*

*<field key="age"/>*

*<field key="velocity"/>*

*<field key="id"/>*

*<filters>*

*<filter key="beaching" value="1" comments="0-all, 1-only non beached particles, 2-only beached (defaut=0)"/>*

*</filters>*

*</measures>*

The lines starting with *<field key=>* control the variables to compute in this simulation and save them to the *netCDF* output file. If you include the basic variables from the *outputFields.xml* section, then the postprocessor will take the particles from each grid-cell and average it out, to provide the average value inside the cell at each timestep. For “*concentrations”* and “*residence\_time”* it will compute the number of particles per area/volume grid and the time that a cell is active by the presence of particles respectively.

If you do not want to compute a measure, you can delete the line(s). For example, if you do not want to compute the average ”id”, just delete the line from the *xml*.

The most important parameter of this <*measures*> block, is the <*filters*> parameter. It allows you to control which particles are going to be used for postprocessing:

1. All particles (beached and non-beached).
2. Just particles that do not reach beach condition (non-beached).
3. Just particles that reach the beach condition (beached).

For example, to measure the concentrations **ONLY** in beaches replace the *value=”0” with “2”.*

*<filters>*

*<filter key="beaching" value="0" comments="0-all, 1-only non beached particles, 2-only beached (default=0)"/>*

*</filters>*

### Recipe grid region and resolution

The *<gridDefinition>* block controls how to slice your simulation domain into boxes or cells to count the particles and to obtain the associated quantities mentioned above.

*<gridDefinition>*

*<units value="relative" comments="relative; meters; degrees"/>*

*<resolution x="50" y="50" z="10"/>*

*<BoundingBoxMin x="-9.1" y="42.39" z="-1" units\_comment="(deg,deg,m)"/>*

*<BoundingBoxMax x="-8.72" y="42.68" z="1" units\_comment="(deg,deg,m)"/>*

*</gridDefinition>*

The <*units value*> field refers to the resolution specified in the next line <*resolution*> and allows you to control how to split the domain.

1) “*relative*”: Slices the domain in 50 pieces in x-direction, 50 pieces in y-direction and 10 pieces in z-direction using the bounding box domain limits.

2) “*meters*”: Slices the domain in steps of 50 meters in x-direction, 50 meters in y-direction and 10 meters in z-direction.

3) “*degrees*”: Slices the domain in steps of 50 degrees in x-direction, 50 degrees in y-direction and 10 meters in z-direction.

The <*BoundingBoxMin*> and <*BoundingBoxMax*> lines control the area within the domain where you want to perform the postprocessing stage of the computation. If you do not provide a Bounding Box here, then the MOHID-Lagrangian Postprocessor will take these boundary limits from the *CMEMS\_case.xml*. If your main bounding Box from *CMEMS\_case.xml* takes the whole ocean, then here you could select a smaller box around the Azores Island for example, to compute concentrations around it.

### Recipe Polygon definition.

In case you want to perform operations over polygons, using the postprocessor allows you to introduce a polygon in the block <*EulerianMeasures*>. The <*polygonDefinition*> counts the number of particles inside each polygon provided by a shapefile. If a polygon is provided, it will override the grid counting option.

To provide a polygon we should edit the block in the following way:

<EulerianMeasures>

<measures>

<field key = "concentrations"/>

<filters>

<filter key = "beaching" value= "0" comments = "0-all, 1-only non beached particles, 2-only beached (defaut=0)"/>

</filters>

</measures>

<polygonDefinition>

<file name= "./../data/OSPAR\_Subregions/OSPAR\_subregions\_20160418\_3857.shp" comments="shape file path"/>

</polygonDefinition>

</EulerianMeasures>

Notice that in the <*polygonDefinition*> the “*file name”* key must contain the path to the shapefile.

### Plotting

The *<plot>* block controls the plotting stage, which should be specified in case you want to produce map outputs with the results. At this moment, the plotter works only with concentrations. The <*plot*> block has the following keys:

*<plot>*

*<time key='groupby' value='time.hour' comments='key: group, value: time.season, time.month, time.year. Resample: ' />*

*<weight file='Post\_scripts/weights.csv' comments='Weights data by a source value'/>*

*<measure key='cumsum' comments='any implicit method, mean, std, diff, cumsum'/>*

*<measure key='diff' comments='any implicit method, mean, std, diff, cumsum'/>*

*<measure key='mean' comments='any implicit method, mean, std, diff, cumsum'/>*

*<type value='imshow' comments='contour,contourf,pcolormesh,imshow'/>*

*</plot>*

### Plotting – Time, resample and group

The first key <*time*>, allows you to perform time-based operations. That is, it allows you to group the concentrations into months, years, etc., and perform basic statistical operations on it. This is due to the use of *pandas-xarray* grouper and resample methods from *python*. So here we have three options for this parameter:

key =’’,’resample’,’groupby’

1) The empty **key =’’**, and value=’all’ just use all time-steps to perform the operations.

2) The **key=’resample’**, allow you to use any of the keys that *pandas* use for timeseries. For an entire list of keywords, please refer to <https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html>. Here are the main keys that can be used:

B business day frequency

C custom business day frequency (experimental)

D calendar day frequency

W weekly frequency

M month end frequency

SM semi-month end frequency (15th and end of month)

BM business month end frequency

CBM custom business month end frequency

MS month start frequency

SMS semi-month start frequency (1st and 15th)

BMS business month start frequency

CBMS custom business month start frequency

Q quarter end frequency

BQ business quarter end frequency

QS quarter start frequency

BQS business quarter start frequency

A year end frequency

BA, BY business year end frequency

AS, YS year start frequency

BAS, BYS business year start frequency

BH business hour frequency

H hourly frequency

T, min minutely frequency

S secondly frequency

L, ms milliseconds

U, us microseconds

N nanoseconds

These keys can be combined with a number. For example, ‘3M’ will resample your data every ‘three months’ with the operator desired.

3) The **key=’groupby’,** allows you to group the data based on the argument passed through “*value*”. It allows us to group all the concentrations by month, week, season, year, etc… to perform operations grouping all timesteps that are inside a given interval. For example, if we group by season, it will take all the concentrations in winter and apply the desired method over all those timesteps (and do the same for spring, summer and autumn).

The available values are: ‘time.season, time.year, time.month, time.week, time.day, time.hour’

### Recipe Weight.

The <*weight*> block allows us to introduce a constant weight to each source, to change the weight of concentrations. For example, two sources emitting the same number of particles could have a different weight based on any source properties (such as population). This weight is introduced via *csv* file, which allows for specifying a weight so that each particle emitted by that source will “count more” based on the factor provided on the list. The <*weight*> key should be edited as so:

<weight file='Post\_scripts/weights.csv' comments='Weights data by a source value'/>

And the format of the weights.csv (separated by commas) file should be:

*id,sources,weight*

*1,Box1,5*

*4,PolygonTest,10*

*2,ReleaseLine10,1000*

*20,Polyline20,25*

### Plotting – Apply functions on data

The <*measure*> keys allow you to combine different implicit *xarray* methods together and in sequence.

The main available methods are: ‘cumsum, sum, mean, std, quartile,’. The operands should be organized in such a way that the last operand collapses the time dimension. For example, if we want to perform the ‘derivative of the concentrations’ to observe where concentrations grow with time, we should do the following:

<measure key='diff' comments='any implicit method, mean, std, diff, cumsum'/>

<measure key='mean' comments='any implicit method, mean, std, diff, cumsum'/>

The first key applies the differentiation operator on time dimension over the timeseries obtained from

<*time*> key. Then, once the “*diff*” is applied, the mean is computed on time, collapsing it.

The last line of the <*plot*> block, the key “*type*” allows you to control the type of plot you want to perform. The available options are “*contour, contourf, pcolormesh, imshow”.*

### Convert files to hdf5

This block controls the conversion of *vtu* files to *hdf5* format at the post-processing stage.

*<convertFiles>*

*<format key="hdf5"/>*

*</convertFiles>*

If you do not want to convert the *vtu* output files, just delete this block.

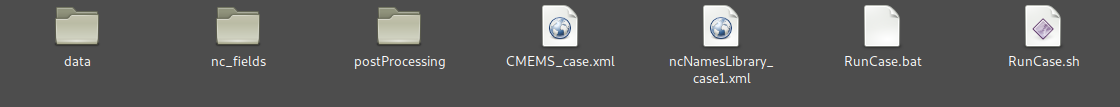
### Setup the <caseDefinitions>

The <*caseDefinitions*> block controls the specific features for your scenario. It controls the path for input data, simulation domain limits (bounding boxes), timesteps and resolution, as well as the definition of sources, together with other parameters to control some physical aspects.

### *netCDF* input files

In order to move the particles with the water currents, we need to add *netCDF* files with fields at least for currents velocities. To use them in our simulation, we must follow these steps:

1) Create a folder inside the *CMEMS\_case* main folder, called (for example) “*nc\_fields”.*



1. Inside the *nc\_fields* folder, we create different subfolders, depending on the type of input that we are going to use. At this moment, we support 4 different input types.
   * **hydrodynamic**: just for water velocities u, v, w;
   * **waves**: read the vsdx, vsdy (velocity stokes drift x and y);
   * **meteorology**: read the wind surface or wind at 10 m (u10, v10);
   * **water properties**: read salinity and temperature: salt, temp.

3) For each type, we recommend you create a subfolder with this structure:

├── nc\_fields

│ ├── currents – for the *netCDFs* fields with the currents’ velocity u,v and w .

│ ├── water\_properties – for the *netCDFs* with temperature and salinity fields

│ ├── waves – for the *netCDFs* with stokes velocity drift

│ └── winds – for the *netCDF* files with wind speed at surface

4) **Inside each folder, place the *netCDF* required**. The name of the *netCDF* is not important.

The *netCDF* files inside each subfolder can be in one file, or in multiple *netCDF* files. You do not have to worry about this. The MOHID-Lagrangian preprocessor analyses the time dimension of all *netCDF* files to make a continuous integration between the different files.

**Once you placed the *netCDFs* files inside each folder, please, check that your variables and dimension names inside the *netCDFs* are names defined in the *ncNamesLibrary\_case1.xml* described above.** The program reads the full path to the hydrodynamic files, such as */path/hydrodynamic1.nc*, but it assumes a fixed-length string (e.g., 18 characters). If the path exceeds this limit, it gets truncated (e.g., to */path/hydrodynami*), causing an error when trying to load the file**. It is also important to ensure that the data has no gaps in the time dimension** and that the time axis is consistent (i.e., regular time steps without discontinuities).

Here is the block <*inputData*> that controls where your netCDFs with your fields are stored.

*<inputData>*

*<inputDataDir name="nc\_fields/currents" type="hydrodynamic"/>*

*<inputDataDir name="nc\_fields/waves" type="waves"/>*

*<inputDataDir name="nc\_fields/winds" type="meteorology"/>*

*<inputDataDir name="nc\_fields/waterProperties" type="waterProperties"/>*

*</inputData>*

In the simplest case, if we want to perform a simulation just using the hydrodynamic currents to track the motion of water masses, we just need to set as input data:

*<inputData>*

*<inputDataDir name="nc\_fields/currents" type="hydrodynamic"/>*

*</inputData>*

### Set up the simulation

The *<simulation>* block controls some key parameters on the domain simulation. The most important ones are the *<timestep>* and the *<BoundingBoxes>*

*<simulation>*

*<resolution x="50" y="200" z="10" units\_comment="metres (m)"/>*

*<timestep dt="1200.0" units\_comment="seconds (s)"/>*

*<BoundingBoxMin x="-9.1" y="42.39" z="-1" units\_comment="(deg,deg,m)"/>*

*<BoundingBoxMax x="-8.72" y="42.68" z="1" units\_comment="(deg,deg,m)"/>*

*<VerticalVelMethod value="1" comment="1:From velocity fields, 2:Divergence based, 3:Disabled. Default = 1" />*

*<RemoveLandTracer value="0" comment="Remove tracers on land 0:No, 1:Yes. Default = 0" />*

*</simulation>*

Different axes can have different resolutions, but most commonly you can use the same for x and y dimension. The *<timestep>* controls when the solution is computed, or in other words, the integration time step. It is your choice and depends on the spatial and time resolution of your input data.

The *<BoundingBoxMin and Max>*, defines the corners of your simulation domain (simulation box) or in other words, defines the domain where your trajectories of particles are computed. If a particle is outside this bounding box, it will be deleted from the simulation. *BoundingBoxMin* corresponds with the *LowerLeftCorner* coordinates and *BoundingBoxMax* with the *UpperRightCorner* coordinates.

The *BoundingBox* can be smaller than the field domain inside your *netCDF* files. **If a particle reaches the *BoundingBox* limits despite data outside of it being available, it will be deleted from the domain**.

The *BoundingBox* can be bigger than your *netCDF* data. However, **if the particles leave the domain of your *netCDF*, they will also be deleted** because there is no data to integrate the particles.

The bounding box is not set automatically. If you consider using the whole domain of your *netCDF* data, you must set it correctly according to your *netCDF* field data limits.

Some datasets don’t include the vertical component of the velocity field in the currents (for example, the Copernicus Marine Service). To be able to compute the vertical component, we add the option to compute the Lagrangian divergence at each point using the perturbation of the point at 4 positions on horizontal plane to compute the derivatives involved.

<VerticalVelMethod value="1" comment="1:From velocity fields, 2:Divergence based, 3:Disabled. Default = 1" />

Particles can also reach land. You can choose if you want to remove them or not. By default, the particles stay on land, and they are not removed.

<RemoveLandTracer value="0" comment="Remove tracers on land 0:No, 1:Yes. Default = 0" />

### Set up the sources

The <*source*> definition works in the following way: each <*source*> defined needs a block describing the source. If you want to put two different sources, you require two blocks:

**<source>**

**Source definition 1**

**Source rate**

**Source geometry**

**</source>**

**<source>**

**Source definition 2**

**Source rate**

**Source geometry**

**</source>**

With this idea in mind, you can add as many sources as you want.

### Set up the sources ID

All sources can be identified by an id and a name, that will be included in the *vtk* outputs and in the postprocessed results.

*<source>*

*<setsource id="18" name="Spill\_007" />*

*</source>*

Each source requires a unique id and name to identify it. This is very important to identify the origin of the particles and, to set the type of material of the particle. The type of material is defined in accordance with this id and must be set later in the <*sources type*> definition.

**IMPORTANT**: Each source can emit just one type of particle. If you want to emit two types of particles from one source, define two sources with two different ids, give a good name to identify each of them and then add the corresponding material in the <*sourcesTypes*> block.

MOHID Lagrangian supports the following types of emissors:

* Point
* Line
* Box
* SphereKMZ-polygon (example: area emissions, works as the box emission but is not a regular one, the user can adapt to a specific geospatial area)
* XY-Polygon (example: area emissions, works as the KMZ-polygon)
* Csv file (example: river emissions, time-variable point sources)
* Position time series (example: a boat emitting)

These different kind of emissors will be further described in the section 1.7.3.3.

### Setup the source rate

After defining the <*setsource*> block, then the important part is the emission rate. This rate of emission can be set in two ways, directly on the template or by reading a *csv* file using the <*rateTimeSeries*> block. If you want to assume an ideal, both continuous or interrupted emission, you can do so using the variables rate or *rate\_dt*, but if, for example, you want to simulate the emission of a discharge by a river, you may be interested in loading that river flow discharge through a *csv* file.

If you don’t want to specify the emission rate in an input file, you can also use *xml* blocks to define a fixed personalized emission using:

<rate value="0.0333" comment="emission rate (Hz)" />

Or

<rate\_dt value="1" comment="number of timesteps / emission. 1 is every timestep, 5 is every 5 timesteps" />In other words:

* ***rate\_dt*** *-->* Sets a fixed time interval between emissions from a certain source (more specifically, the number of timesteps between emissions).
* ***rate*** *-->*  Sets an emission rate (particles/second) at which the particles are released.

However, if this *rate* generates emissions which account for less than 1 emission/timestep, these emissions will add up until a “whole” particle is generated and will then be emitted. For instance, if our simulation timestep is 1h=3600s, and rate=1/7200 Hz (1 particle every 2 hours), a particle will be released every 2 timesteps. Of course, **this rate can be set no to be a multiple of the timestep**. This results in a non-uniform-like emission. For instance: if <*rate=1*>, a particle is emitted every 2.5 hours, but if the simulation’s timestep is 1h, MOHID will generate the emission in the 3rd timestep. The “leftover” particles of the last half-hour will be saved for the next emission. This method implies that, even though the first emission was in the 3rd timestep, the following ones might not be generated every 3 timesteps. Of course, this method also applies to the ***rateTimeSeries*** option.

If you want to use a *csv* file, write these lines inside your *xml* file:

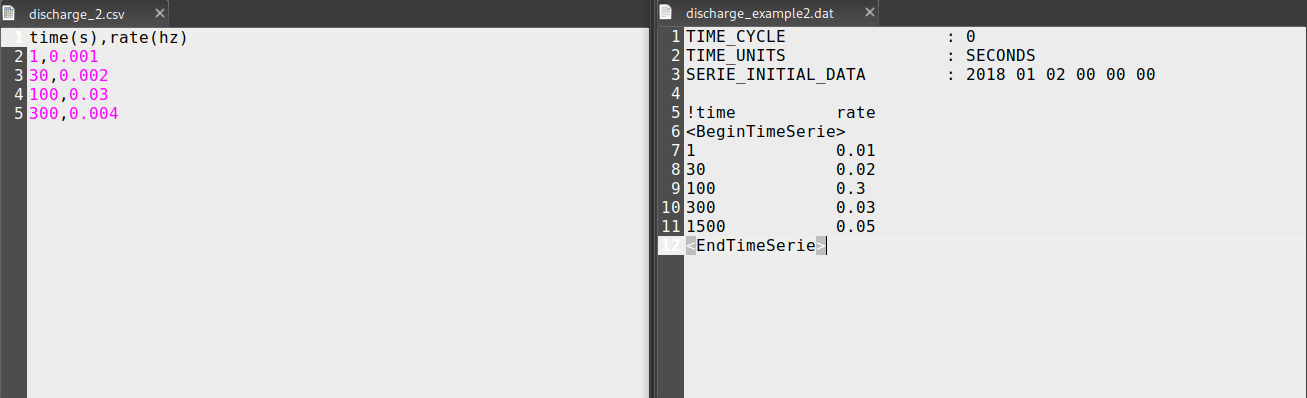
<rateTimeSeries>

<file name="data/discharge\_example.csv" comment="name of csv file with discharge information (time and rate columns)"/>

<scale value="1.01" comment="scales the data on the file by this factor (not time)" />

</rateTimeSeries>

The *csv* file or *dat* file should look like these two types of files, respectively:



On the left, it is a pure *csv* file. In the left column, the time is in seconds from the beginning of the simulation specified in

<execution>

<parameters>

<parameter key=”Start” value=”2017 01 01 00 00 00”/>

In the right-hand side of the figure (which contains the .*dat* file) the initial time for emission is specified at the *SERIE\_INITIAL\_DATA* attribute.

### Setup the source geometry

For all source types, emissions can be switched on and off during the simulation period using the <*active*> line. This can be combined with a time interval (in seconds from “*Start time*”) to set where the source must be active:

<active start="start" end="end" comment="example: start='12.7' end='end'; start='0.0' end='95' " units\_comment="seconds (s)" />

Or

<active start="start" end="3600" comment="example: start='12.7' end='end'; start='0.0' end='95' " units\_comment="seconds (s)" /> for 1hour of activity

Or

<active start="3600" end="end" comment="example: start='12.7' end='end'; start='0.0' end='95' " units\_comment="seconds (s)" /> from 1 hour from the starting point of the simulation to the end.

Finally, after setting the emission rate, we must define the geometry of the source:

* Point. We just need the coordinates for a point which is inside of the domain (be sure of that) and, optional, you can set the spatial resolution of this point.

*<resolution dp="50" units\_comment="metres (m)"/>*

*<point x="2.5" y="5.5" z="0.75" units\_comment="(deg,deg,m)"/>*

* Box. This just simulates the extension of the point example. We define the Box from the *LowerLeftCorner* and set its size in meters, and the spatial resolution between points inside the box domain (also define in metes):

<resolution dp="50" units\_comment="metres (m)"/>

*<box>*

*<point x="-5.5" y="1.0" z="0" units\_comment="(deg,deg,m)"/>*

*<size x="0.5" y="3" z="4.5" units\_comment="metres (m)"/>*

*</box>*

* Line. Similar to the box case, for the line type emission we set two points and the resolution of the emission points between them.

With “*dp”* we set the distance between points along the line.

*<resolution dp="50" units\_comment="metres (m)"/>*

*<line>*

*<pointa x="1" y="2" z="-10" units\_comment="(deg,deg,m)"/>*

*<pointb x="1.001" y="2.00001" z="7" units\_comment="(deg,deg,m)"/>*

*</line>*

* Sphere. To set up a sphere, you need a centre point and a radius.

*<sphere radius="0.95" units\_comment="metres (m)">*

*<point x="9.05" y="2.0" z="0" units\_comment="(deg,deg,m)"/>*

*</sphere>*

* KMZ-Polygon and XY-polygons. These are geospatial tools that can be defined by *QGIS*, *ArcGIS* or other similar tools to define a precise emission area. For example, if we can simulate the emission over a maritime specific demarcation, we can import this maritime demarcation as a KMZ-Polygon and then set the spatial resolution for the emission points inside it. To use a KMZ-polygon in <*file name=”data/polygon.kmz”*/> add the file path where your *kmz* file is stored. You can also specify the resolution in meters in the x,y,z components to fill the polygon with a distribution of points.

*<resolution x="80" y="50" z="150" units\_comment="metres (m)"/>*

*<polygon>*

*<file name="data/polygon1.xy"/>*

*<verticalBoundingBox min="0.0" max="0.0"/>*

*</polygon>*

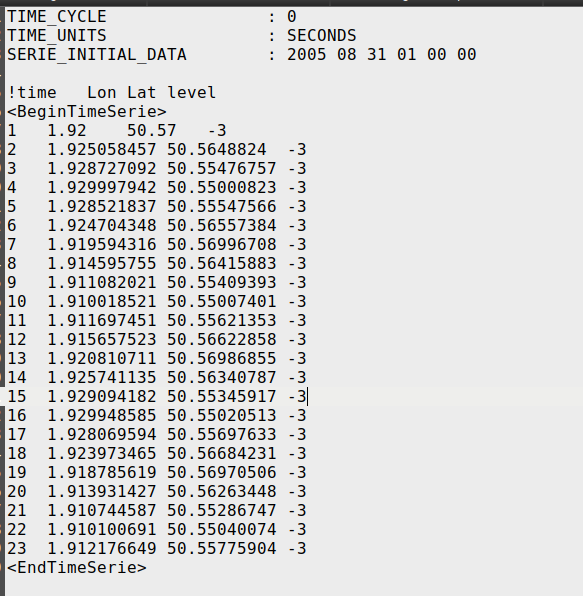
* Position time series. The position time series allows you to define a moving emission. To consider a moving emissors (e.g., a ship), just use the following block:

*<positionTimeSeries>*

*<file name="data/spill\_trajectory.csv" />*

*</positionTimeSeries>*

The format inside the *spill\_trajectory.csv* must be:



### Set up the particle types

The <*sourceTypes*> contain the information about the particle properties for each source. All the particle types must be placed in a *xml* file to act as a database for future plastic types or objects.

So, the *<sourceTypes>* block has the following syntax:

*<sourceTypes>*

*<types>*

*<type source="1" type='plastic' property="bag\_1" comment="" />*

*<type source="2" type='plastic' property="bag\_1" comment="" />*

*<type source="3" type='paper' property="cardboard\_1" comment="" />*

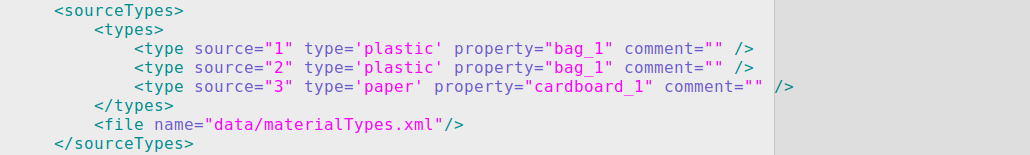
*</types>*

*<file name="data/materialTypes\_example.xml"/>*

*</sourceTypes>*

The default option for a (base) lagrangian tracer is the following:

*<type source="4" type='base' property="" comment="" />*



The important key here is the *<file name="data/materialTypes\_example.xml"/>.* It is used as a database for the different types of particles. Each particle type has its template with its own properties.

**IMPORTANT**: Notice that particle types different from the ‘base’ tracer might need temperature and salinity fields in order to compute the buoyancy and other processes affecting these particles.



The different properties associated with each type of particle are passed to the particle properties of the MOHID-Lagrangian code. To add new particles, just copy a block of code and rename it to fit your needs.

### Setting up the constants

The block <*constants*> refers to global values affecting the whole integration and can be changed to adapt some values to your needs. The description in the comment section will guide you to make any necessary changes.

For example, if your data does not start at 0 level or start above zero, replace the *Z0* value:

<Z0 value="2.16" comment="Reference local 'zero' level. Default = 0.0" units\_comment="m" />

If you want to increase the diffusion effect, change the following block:

<DiffusionCoeff value="0.75" comment="Horizontal diffusion coefficient. Default = 1.0" units\_comment="m2/s" />

Particles can be resuspended if they are on the floor through the key <*landIntMask = 1>*. At the moment, the resuspension in the vertical is done by adding to the particle a fraction of the horizontal velocity module. The <*ResuspensionCoeff>* factor controls it. *0* means no resuspension and *1* means full horizontal module to push particles up in the vertical direction.

<ResuspensionCoeff value="0.0" comment="Resuspension amplitude velocity factor. Default = 0.0" units\_comment="n.u" />

Other global constants can be defined to change default values or to activate different processes. For instance, the beaching probability can vary depending on the coastal type of our study domain (calm beaches, rocky cliffs, etc.). Also, one might want to add the wind influence in our particles’ behaviour (typically set between 1-5%). These two parameters can be changed as follows:

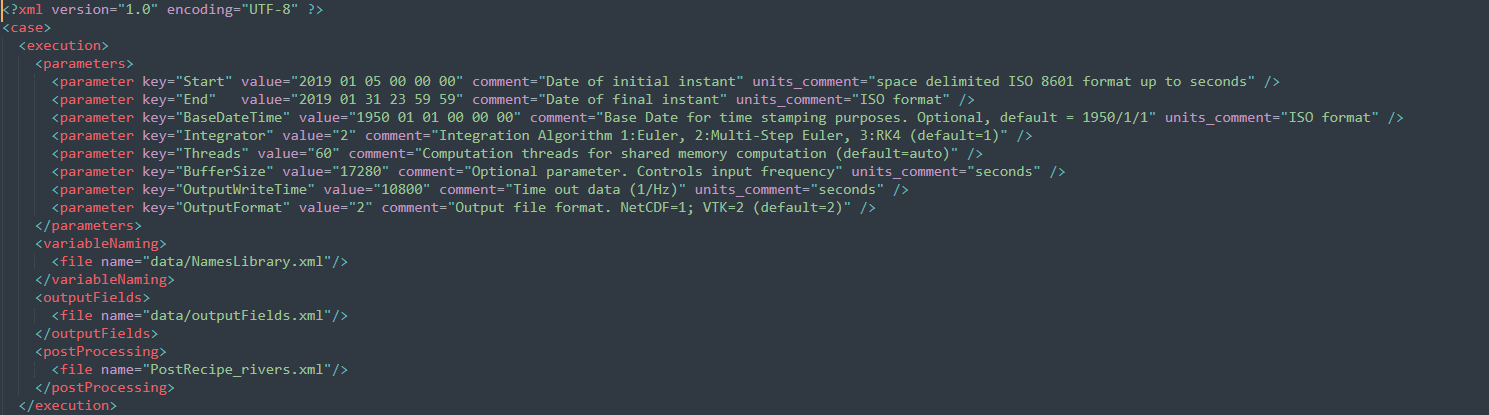
<BeachingStopProb value="80" comment="Resuspension amplitude velocity factor. Default = 50" units\_comment="%" />

<WindDragCoeff value="0.03" comment="Wind drag coefficient. Default = 0.0" units\_comment="n.u." />

### Self-contained example

So now, after all this information, we present to you a self-contained example to show some of the particularities of the MOHID-Lagrangian model. This is a self-contained example of a time-varying point emission in the Arousa estuary domain. The folder /*Arousa\_example/hidro\_wind\_caudal/* contains the hydrodynamic and meteorological data for the period from 01/01/2019 to 01/31/2019 with a temporal resolution of 1 day, as well as the *dat* file corresponding to the flow of the Ulla River. (If you need further information about this test case and the domain area, just contact to the University of Santiago de Compostela.)

Let’s start with the execution module:



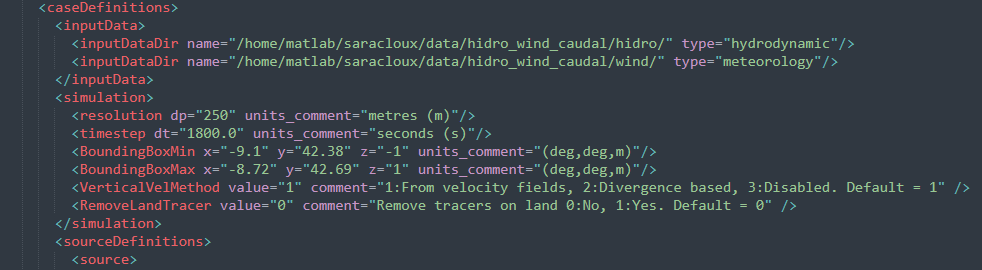
Setting the temporal domain, we have the simulation starting on the 2019 01 05 00 00 00 and ending on the 2019 01 31 23 59 59. *BaseDateTime* is just a formalism in computation to set the starting day from which the seconds are counted from. Nothing to worry about. *Integrator* is set as value=”2”, which means we are using Multi Step Euler; *Threads* is set to 60 but this depends on the computer/machine/server you are running this model on. Please update according to tips given in section 1.3.1.3. Our *buffersize* is set to “17280” seconds, which means that the model is saving around 5-hours hydrodynamic data in the RAM memory. In our case, since we are working with 1-day input data files, the model will read one file at a time. If on the other hand we would have an input file per hour, then the model would load into RAM 5 files per calculation round.

We save the output data every 3h, corresponding to *OutputWriteTime* = “10800” (10800s/3600 = 3h). And the output format is *vtu* files.

We must indicate where the file *NamesLibrary.xml* and the *outputFields.xml* are located, as well as the *PostRecipe\_rivers.xml*. More on these files later. And with this we can conclude the execution block.

Moving on to the <*case definition*>, the first thing we need is to indicate where the hydrodynamic and wind fields are located, as can be seen in the next picture. If we want to just turn off the wind field influence, we can either delete this line or just comment it as:

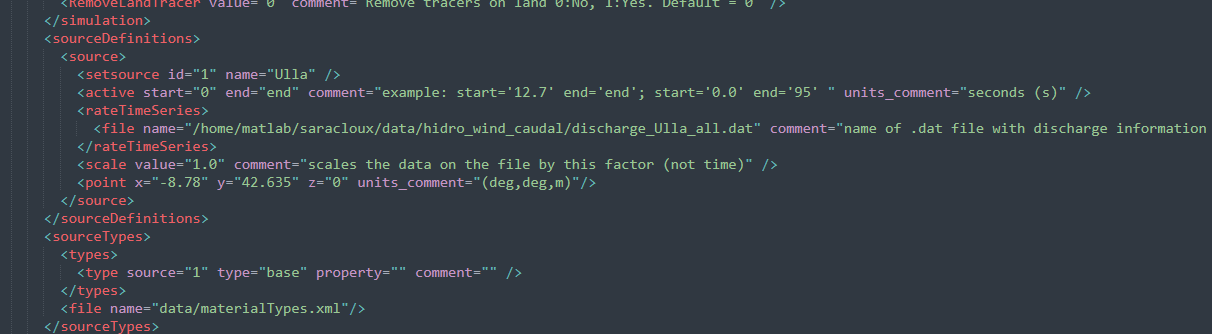
<!--inputDataDir name="/home/matlab/saracloux/data/hidro\_wind\_caudal/wind/" type="meteorology"/-->



The next step is to define the simulation spatial and temporal domain, or in other words, the spatial and temporal interpolation resolution. Here we have set a spatial resolution of <*resolution* *dp=*250> meters and a temporal resolution of 1800s (i.e., 0.5 h), specified in the <*timestep dt*=> key. We also set a box domain with a lower left corner corresponding to <*BoundingBoxMin x=“-9.1” y=”42.38” z=”-1”*> and a upper right corner <*BoundingBoxMax x=“-8.72” y=”42.69” z=”1”*>.

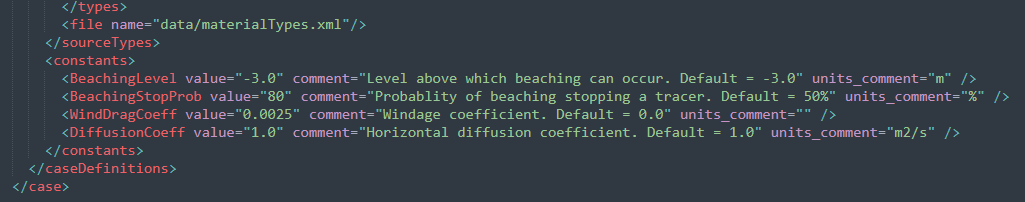
If we were interested on a 3D simulation, then we would need to supply the vertical motion, which can either be read from the *w* component of the hydrodynamic field or just calculated from the divergency. Here, we are running a simulation with pure Lagrangian particles on the surface, so this parameter has no effect now.

Another thing we can specify is what happens to a particle if and when it reaches the coast (it can be removed from the simulation), but still, we are not interest on that.



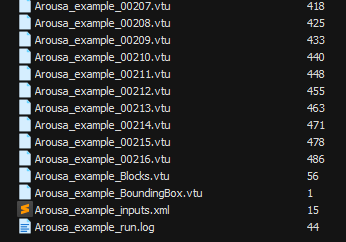
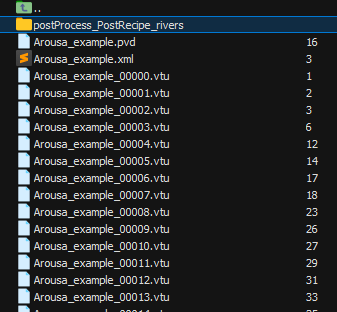
Moving on to the definition of sources, we use the *id=1* and the name of “*Ulla*” as we are modelling the Ulla River, which will be active from the start of the simulation until the end of it. The *rateTimeSeries* will follow the Ulla River flow discharge saved on the *discharge\_Ulla\_all.dat*, and since we want to test the simplest case (with no modulation), *scale\_value* is set to “1”. In another experiment, you could consider, for example, having 2 river sources with its 2 river flows. Suppose that we could also know that the population around one of them is double the population around the other river. We can assume that the marine litter of the first river could be, potentially, the double of the other one, so in that case we would be interested in modifying this *scale\_value*.

Be aware that it is important to maintain the header and the ending lines exactly as they are in the *discharge\_Ulla\_all.dat* file. We set the emission point at the mouth of the river, around the coordinate point (-8.78, 42.635, 0), and we choose the “base” *type*, which is not defined on the *materialTypes.xml* because it is already defined inside the model. You are free to choose another type from the list on that *xml* file, but depending on the specification, you might need to provide temperature and salinity as input fields as well (which are needed to compute density).

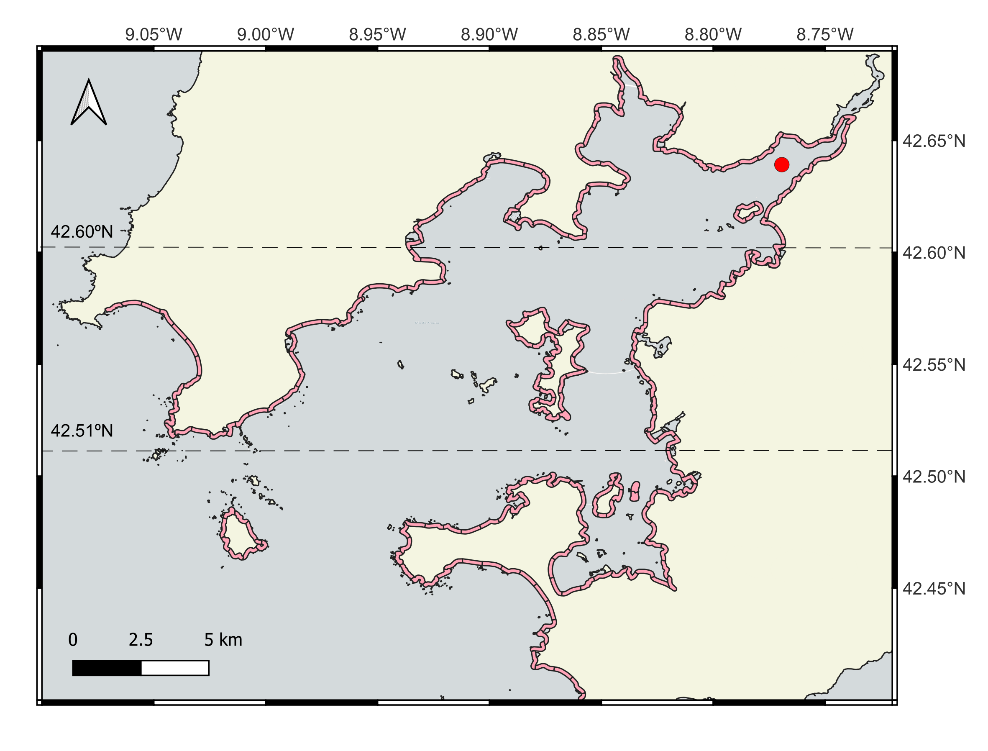


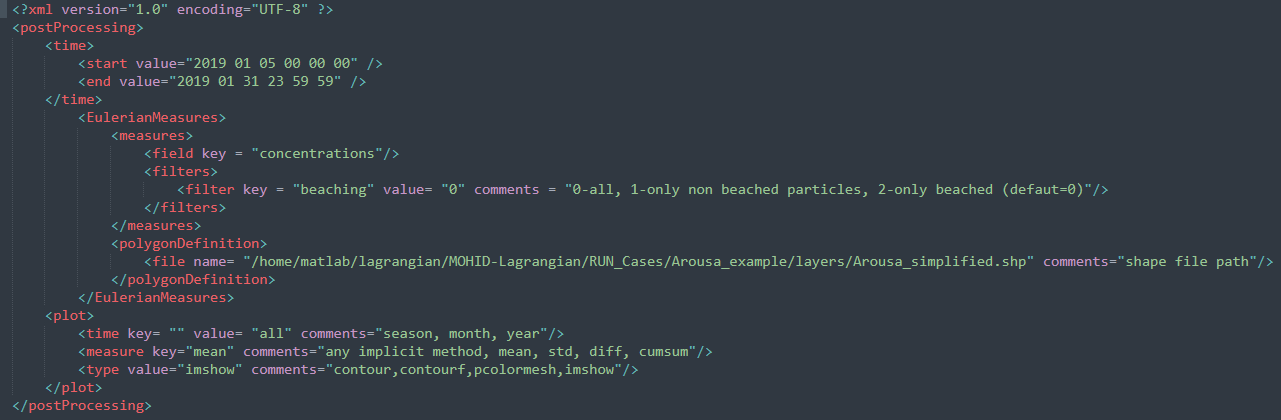
We conclude the setup of this example with the *constants* block of definition, *BeachingLevel* is set to “–3” meters, the *BeachingStopProb* is set to “80” % as we expect a high stranding probability on the area, *WindDragCoeff* is set to “2.5” % and finally *DiffusionCoeff* is set to “1”.

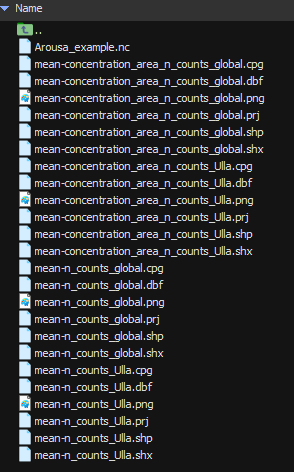
Once we run this simulation, the *Arousa\_example\_out* folder will be created. Inside we can find all the *vtu* files and other 6 files that are generated to save information about the simulation, and another folder with the postprocessing images:



Let’s continue now by setting up the *PostRecipe\_rivers.xml*. We want to know the concentrations on all time steps during the simulation, for all particles, both beached and non-beached. We could set a regular grid as was earlier specified in this guide, or we can also set our own grid or even our own region. In this case, we will set our own coastal line (pink segments), that we obtained using QGIS and saved as shapefile.







Inside the *postProcess* folder we find a *netCDF* file and the geospatial files (i.e., *shp*, *prj*, etc). The *netCDF* contains the information on both the number of particles and the concentration given by each source in each spatial object that we have described in our shapefile, for each time step indicated in the *PostRecipe*.xml. It also contains the information for the global accumulation of all sources, both *concentration\_area\_n* and *n\_counts*.

Note: geospatial objects were generated, which can be opened with different tools such as QGIS. This document shows the *png* generated by the model itself.

