MOHID – LAGRANGIAN - Short Guide

PROLOGUE

This manual is a short guide in order to operate with MOHID-Lagrangian code and start to produce some outputs and exploration the different setups and options in a fast way. Many of you have asked Ricardo and me about how to use the software so before a long technical manual I decided to write this short manual for CleanAtlantic members. I wrote it in two days so I am sure that it will contain many English errors. I correct it and improve it

INDEX

1. INTRODUCTION:

The way to use MOHID – Lagrangian is following 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, modified the setup xml files inside to fit in your needs.

Lets go step by step how to prepare a working simulation:

1.1 Test working simulations or template folder

Inside the path /MOHID-Lagrangian/RUN_Cases, we provide different working examples in the path :

/MOHID-Lagrangian/RUN_Cases/

Inside, there are the following cases:

- Arousa_2D_test_case
- PCOMS_test_case
- Tagus3D_case
- Vigo_3D_test_case
- Case_template (not run)

The four first cases are working setups of MOHID-Lagrangian with the files configured to work properly and test how MOHID-Lagrangian works. If everything is fine and there are not problems at the installation stage each case should run fine just by running the scripts inside each test_case folder:

RunCase.sh (Linux)

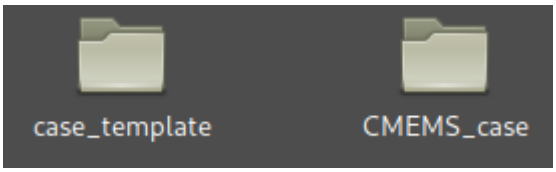
RunCase.bat (windows)

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

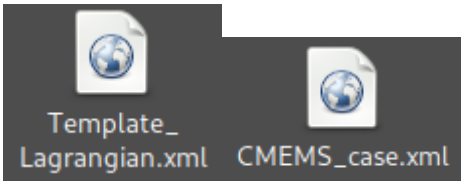
1.2 Create a new simulation from Case_template

Suppose we want to perform a new simulation using CMEMS surface current data for example. This netcdf hydrodynamic data contains the currents speeds u,v,w. The first step is to:

1) copy the **case_template** folder into a new one a rename to **CMEMS_case** for example:



2) Then, to keep an names 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

```
RunCase.sh
#!/bin/bash

clear

 named according to the testcase
name=CMEMS_case

# "executables" are renamed and called from their directory
tools=../../build/bin
mohidlagrangian=${tools}/MOHIDLagrangian

preprocessorDir=../../src/MOHIDLagrangianPreProcessor
mohidPreprocessor=${preprocessorDir}/MOHIDLagrangianPreProcessor.py

# "dirout" is created to store results or it is cleaned if it already exists
if [ -e $dirout ]; then
  rm -f -r $dirout
fi
mkdir $dirout

cp ${name}.xml $dirout/

# CODES are executed according the selected parameters of execution in this testcase

python -W ignore $mohidPreprocessor -i $dirout/${name}.xml -o $dirout

errcode=0
if [ $errcode -eq 0 ]; then
  $mohidlagrangian -i $dirout/${name}.xml -o $dirout
  errcode=$?
fi


if [ $errcode -eq 0 ]; then
```

- WINDOWS – (RunCase.bat) set name= Template_Lagrangian -----> set name = CMEMS_case

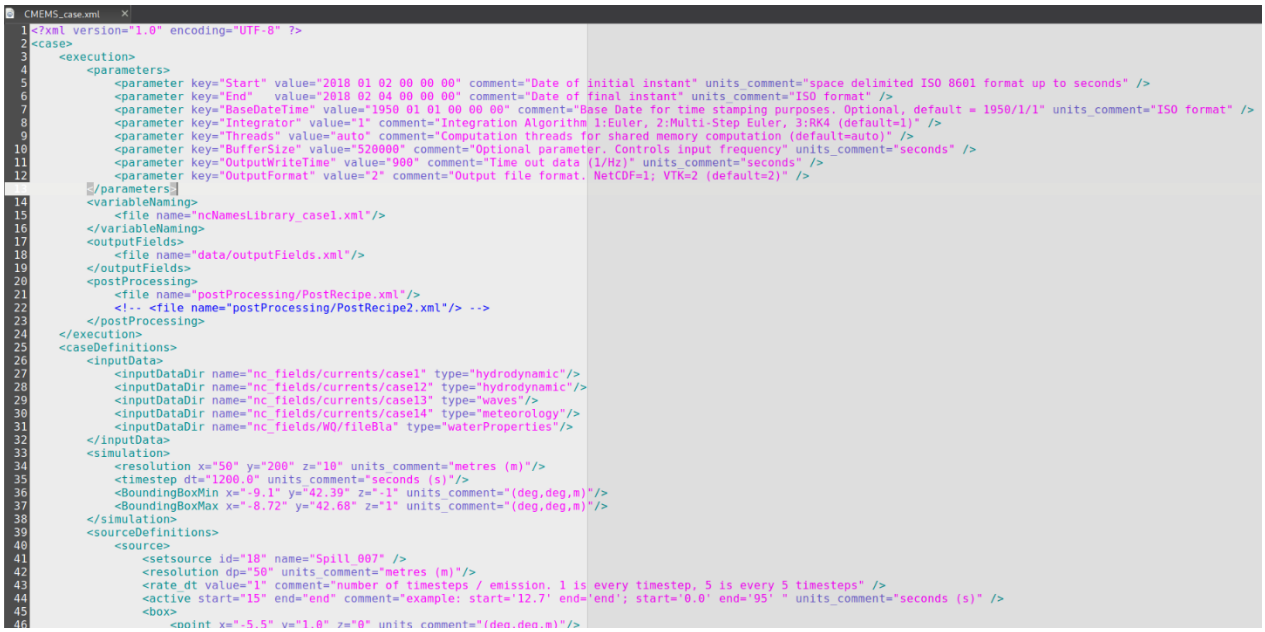
```
RunCase.bat
@echo off
cls

rem "name" and "dirout" are named according to the case

set name=Template_Lagrangian
set dirout=%name%_out

rem "executables" are renamed and called from their directory

set tools=../../build/bin/RELEASE
set mohidlagrangian="%tools%/MOHIDLagrangian.exe"

set preprocessorDir=../../src/MOHIDLagrangianPreProcessor
set PreProcessor="%preprocessorDir%/MOHIDLagrangianPreProcessor.py"

rem "dirout" is created to store results or it is cleaned if it already exists
if exist %dirout% del /Q %dirout%\*.*
if not exist %dirout% mkdir %dirout%

copy %name%.xml %dirout%

rem CODES are executed according the selected parameters of execution in this case

python -W ignore %PreProcessor% -i %dirout%/%name%.xml -o %dirout%

%mohidlagrangian% -i %dirout%/%name%.xml -o %dirout%
if not "%ERRORLEVEL%" == "0" goto fail

:success
echo All done
goto end

:fail
echo Execution aborted.
```

1.3 Setup the Main_Case.xml setup file.

Once we rename the executable files, then we start setup the main setup file of our simulation:

CMEMS_case.xml

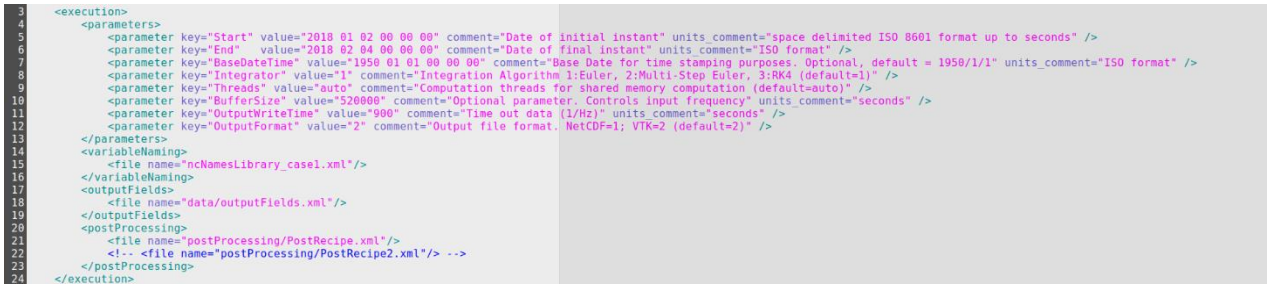
When you open it you can observe the following text in xml format:



For those who have not idea about what xml is…. the xml file has a tree structure with blocks or sections, with subblocks or subsections. Each section begins with <nameSection> and ends with </nameSection>. If it is an inline section (a section just in 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 top to setup each condition or parameters. The first main block you can observe is the <execution> block.



This block controls, as it is named, 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 needed it" />

Using this structure, we start to setup the different <parameters> in this section

1.3.1.1 Change the Start time and end time

These two parameters control the date 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" />

Be sure that the time limits when your simulation start and/or ends fits in with your time range of netCDF input data (later, we will check this). An example: to perform a two-year simulation from 2017 to 2019 we have to 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" />

1.3.1.2 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)

In most of the cases use the option “2” is enough.

1.3.1.3 Change the number of Threads

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

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

To get a better performance this value should be equals 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”.

1.3.1.4 Change the output frequency.

The next parameter key=”OutputWriteTime” controls when the data is written to disk, it controls the **frequency to write data to disk**. For example, if your solution is computed every hour (dt=3600s), you could have an <OutputWriteTime value=”10800”. It makes that every three time steps (10800/3600 = 3) of computation, the third one is written to disk.

It is recommended that this value is a entire multiple of the time step solution in the <simulation> block, to avoid a desynchronization between the solver and the writer.

```
<simulation>

    <timestep dt="1200.0" units_comment="seconds (s)"/>

</simulation>
```

1.3.1.5 Change the buffersize

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. A buffer size of value=”520000” it 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 add it while the previous data is released from ram memory. This value should be change just if you have some error related to “not enough RAM memory” and make it lower.

1.3.1.6 Change the outputformat.

This value controls the output format files to write on disk. At this moment, MOHID-Lagrangian just supports *vtk* output file format so the *value* must be kept at ”2”.

1.4 Setup the netCDF variable naming

The variable *<variableName>* section just have a filepath to a xml file. Here it is: ncNamesLibrary_case1.xml

```
<variableNaming>

  <file name="ncNamesLibrary_case1.xml"/>

</variableNaming>
```

(you can rename the file inside the xml but make it also in the filesystem name)

This file controls the naming convention in your netCDF files. What is the role of this file? Imagine that you have a netCDF file, with a variable name for your hydrodynamic velocity fields or your depth

dimension... How could MOHID-Lagrangian knows that your “strange_name_u_velocity_component” or your “z_evil_dimension” inside the netCDF file means “u_velocity_field” or simple “depth”?

That was why this library file was created. **This xml is dictionary/translator or dictionary between the CF compliant names for variables and dimensions and other variants for the variables that can appear for another netCDF outputs 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 >

Inside, it contains the following text:

```
<?xml version="1.0" encoding="UTF-8" ?>
<naming>
    <variables>
        <eastward_wind name="u10">
            <variant name="u10" comment="used in ECWMF">
        </eastward_wind>
        <northward_wind name="v10">
            <variant name="v10" comment="used in ECWMF">
        </northward_wind>
        <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" />
        </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" />
        </northward_sea_water_velocity>
        <upward_sea_water_velocity name="w">
            <variant name="w" comment="used in MOHID and CMEMS" />
            <variant name="W" />
        </upward_sea_water_velocity>
        <sea_water_temperature name="temp">
            <variant name="temp" comment="used in MOHID and CMEMS" />
            <variant name="Temp" />
            <variant name="temperature" />
            <variant name="Temperature" />
        </sea_water_temperature>
        <sea_water_salinity name="salt">
            <variant name="salt" comment="used in MOHID and CMEMS" />
            <variant name="salinity" />
            <variant name="Salt" />
            <variant name="Salinity" />
        </sea_water_salinity>
        <emission_rate name="rate">
            <variant name="rate" />
            <variant name="Rate" />
            <variant name="RATE" />
        </emission_rate>
    </variables>
```

An example: Imagine that your future netCDF file/files contains the hydrodynamic fields and their names are **“utotal”** and **“vtotal”** to describe the CF compliant variables <eastward_sea_water_velocity> and <northward_sea_water_velocity> respectively. In that case, you should add the following to this 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>

Another example:

If you want to add for example the waves effect through the *stokes velocity drift*, you should add the variant such as:

<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 for those variant names of the standard name.

In the *<dimensions>,* it is the same case.

```
<dimensions>
    <longitude name="lon">
        <variant name="lon" />
        <variant name="Lon" />
        <variant name="LON" />
        <variant name="longitude" />
        <variant name="Longitude" />
        <variant name="LONGITUDE" />
    </longitude>
    <latitude name="lat">
        <variant name="lat" />
        <variant name="Lat" />
        <variant name="LAT" />
        <variant name="latitude" />
        <variant name="Latitude" />
        <variant name="LATITUDE" />
    </latitude>
    <vertical name="level">
        <variant name="depth" />
        <variant name="Depth" />
        <variant name="DEPTH" />
        <variant name="level" />
        <variant name="Level" />
    </vertical>
    <time name="time">
        <variant name="time" />
        <variant name="Time" />
        <variant name="TIME" />
    </time>
</dimensions>
```

Your depth dimension inside your netCDF file is *"z_depth".* You must add in the *<vertical name="level">* 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>
```

1.5 Setup the output fields

The block <outputFields>, like the previous one, contains the file path of the xml field 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.

This file inside the path data contains:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Basic output fields (always printed) are
-id
-source
-position
-velocity -->
<!--Optional output fields can be listed here
-"yes" - field is written, even if it doesn't exist
-"no" - field is not written -->
<output>
    <field name="age" output="yes" />
    <field name="condition" output="no" />
</output>
```

The *<output>* block controls when a variable is written to disk or not. As it is written in the xml comment section, the basic output fields are always written to disk

1.6 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 outputs 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 a size specified inside the recipes.xml files.

1.6.1 Setup the recipes

```
<?xml version="1.0" encoding="UTF-8" ?>
<postProcessing>
    <time>
        <start value= "2019 06 27 00 00 00" />
        <end value  = "2019 06 28 00 00 00" />
    </time>
    <EulerianMeasures>
        <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>
        <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>
    </EulerianMeasures>
    <convertFiles>
        <format key="hdf5"/>
    </convertFiles>
</postProcessing>
```

1.6.2 Recipe time range

The first part of the <postProcessing> block:

<time>

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

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

<step =2 />

</time>

Controls the time range of your simulation to send it to postprocessor. If you have a long simulation (one year for example), you can make a sub selection in time and focuses in one month (for example. Our 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 ke*y to subselect *every n steps* and to avoid memory overload for simulations with so many vtu files. In case that a time range is not specify the *step* keyword can be used for all timesteps.

1.6.3 Recipe output fields

The Eulerian measures consist on translate the particle measures or the properties carried out by the particles to instantaneously 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 field key=”value” controls the variables to compute and add to the netCDF output file. If you include variables that are basics from *outputFields.xml* section, the postprocessor use the particles each grid or cell and it makes an average 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. For example, if you do not want to compute the average ”id”, just delete the line from the xml.

<measures>

<field key="residence_time"/>

<field key="concentrations"/>

<field key="age"/>

<field key="velocity"/>

<filters>

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

</filters>

</measures>

The most important parameter here 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 *value=”0”*

<filters>

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

</filters>

1.6.4 Recipe grid region and resolution

The *<gridDefinition>* block controls how to slice your simulation domain of it 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> fields within the resolution, allows you to control how to split the domain.

1) “relative”: It 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”: It slices the domain in steps of 50 meters in x direction, 50 meters in y direction and 10 meters in z direction

3) “degrees”: It 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, controls the domain where you want to perform the postprocess stage computation. If you do not provide a BoundingBox here, the MOHID-Lagrangian Postprocessor will take these limits from the *CMEMS_case.xml*. If your main boundingBox from CMEMS_case.xml takes the whole ocean, here you could select a box around the Azores island for example to compute concentrations around it.

1.6.5 Recipe Polygon definition.

In case you to perform operations over Polygons, the postprocessor allows 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, this option overrides the grid counting.

To provide a polygon we should provided a 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>
```

Where in the polygon definition in the *name key* we must provide the path to the shapefile.

1.6.6 Recipe Weight.

The <weight> block allows to introduce a constant weight to each source in to change the weight of concentrations. For example, two sources emiting the same ammount of particles could have a diferente weight based on any source properties (such as population). This file allow to specify a weight so each particle emitted by that source will “count more” based on the factor provided on the list.

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

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

id,name,weight

1,Box1,5

4,PolygonTest,10

2,ReleaseLine10,1000

20,Polyline20,25

1.6.7 Plotting

The <*plot*> block control the plotting stage in case you want to produce map outputs with the results. At this moment, the plotter works only with concentrations. The plot block have 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>

1.6.8 Plotting – Time resample and group

The firts key, allows you to perform time-based operations. That is, it allows you to group the concentrations into months, years or any kind of group and perform basic-statistical operations on it.

This is due the use of pandas-xarray grouper and resample methods. So here we have three options:

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, refere 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 endfrequency
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 trhough value. It allows to group all the concentrations by month, week, season, year… to perform operations grouping all timestepss 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 (the same for spring, summer and autumm).

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

1.6.9 Plotting – Apply functions on data

Then the <measure> keys, allows you to combine different implicit xarray methods together in sequence.

The list of the main available methods are: 'cumsum, sum, mean, std, quartile,'. The operands should be in order 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 this concentrations grows on 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 key "type" allow you to control the type of plot you want to perform. The available options are: *contour, contourf, pcolormesh, imshow.*

1.6.10 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 does not want to convert the vtu output files, just delete this block.

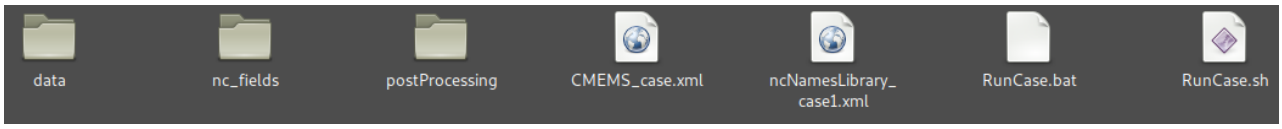
1.7 Setup the <caseDefinitions>

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

1.7.1 netCDF input files

In order to move the particles in within the water currents we need to add to add netCDF files with fields at least with currents velocities. To add them and to use it in our simulation, we will have to do in the following way.:

1) Create a folder inside the CMEMS_case called for example: nc_fields



2) Inside the nc_fields, we create different subfolders, depending the type of the 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
   - **waterproperties**: read salinity and temperature: salt, temp

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

├── nc_fields

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

│ ├── water_properties – place the netCDFs with temperature and salinity fields

│ ├── waves – place the netCDFs with stokes velocity drift

│ └── winds – place inside the netCDF files with wind speed at surface

4) **Inside each folder, place the netCDF required**. The name 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 analyzes the time dimension of the all netCDF files to provide a xml 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 files are names in the ncNamesLibrary_case1.xml described above.

The input data controls where your netCDF data 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 just want to perform a simulation just using the hydrodynamic currents to track the motion of water masses, we just have to set as input data:

```
<inputData>
   <inputDataDir name="nc_fields/currents" type="hydrodynamic"/>
</inputData>
```

1.7.2 Setup the simulation

The *<simulation>* controls some key parameters. The most important 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)"/>
</simulation>
```

The *<timestep>,* controls when the solution is computed. It is your choice and depend on the spatial and time resolution of your data and the hydrodinamic type. For a hydrodynamic currents from CMEMS with

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

The BoundingBox, can be smaller than the field domain inside your netCDF files. If a particle reaches the BoundingBox limits despite there is data outside of it, it will be delete from the domain.

The boundingBox can be bigger than your netCDF data, however, if the particles leave the data domain of your netcdf if will be also deleted because there is no data to integrate the particles.

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

1.7.3 Setup the sources

The source definition works in following way. Each source defined needs a block source. If you want to put two different sources: You require:

<source>

Source definition 1

Source rate

Source geometry

</source>

<source>

Source definition 2

Source rate

Source geometry

</source>

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

1.7.3.1 Setup the sources ID

ALL THE SOURCES HAVE A COMMON PROPERTY:

<source>

<setsource id="18" name="Spill_007" />

</source>

Each source require a id and name to identify it. This is very important to identify the origin of the particles and also, to set the type of material or particle that is going to be emitted and must be set later in the <sources type> definition.

IMPORTANT: EACH SOURCE, just can emit one type of particle. If you want to emit two types of particles from one sources, define two sources with two ids put a good name to identified it and then add the corresponding material in the sourcesTypes block.

MOHID Lagrangian support the following types of emissors:

- Box
- Sphere
- Point
- Line
- KMZ-polygon (example: area emissions)
- XY-Polygon (example: area emissions)
- Csv file (example: river emissions, time-variable point sources)
- Position time series (example: a boat emitting)
-

1.7.3.2 Setup the source rate

After defining the <setsource> block, then the important part is the emission rate. This rate of emission can be setted by two ways, from a csv file using the <rateTimeSeries> block :

USING A CSV FILE:

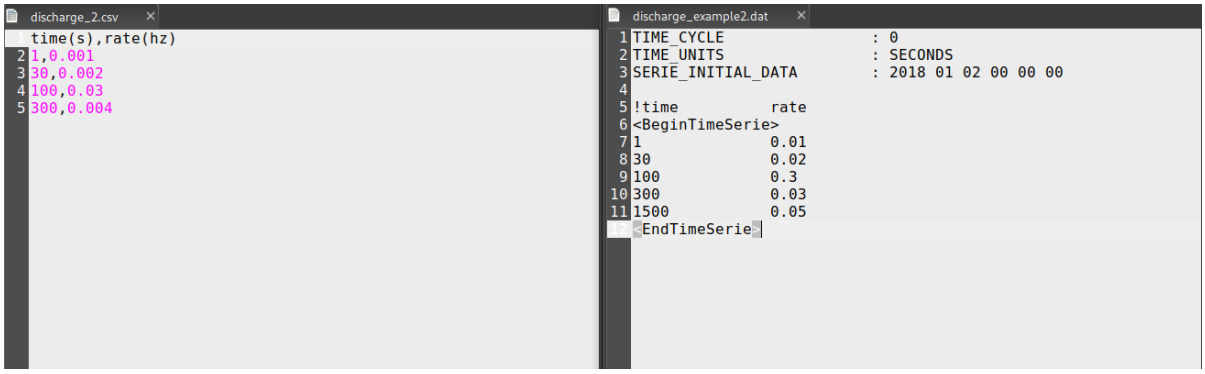
<rateTimeSeries>

<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 can handle this two types of files:



On the left, it is a pure csv file. In the left column, the time is in seconds from the begining of the simulation specified in <execution><parameters><parameter key=”Start” value=”2017 01 01 00 00 00”/>

In the right hand, the initial time for emission is at the SERIE_INITIAL_DATA attribute.

Also, we can 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" />

1.7.3.3 Setup the source geometry

This can be combined with a time interval (in seconds from Start time) to set where the source must be active:

<active start="15" end="end" comment="example: start='12.7' end='end'; start='0.0' end='95' " units_comment="seconds (s)" />

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

- Point

  <resolution dp="50" units_comment="metres (m)"/>

  <point x="2.5" y="5.5" z="0.75" units_comment="(deg,deg,m)"/>

- Box:

<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>

The point define the lower left corner.

- Line

  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 setup a sphere, you need a center 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

  To use a KMZ-polygon in <file name=”data/polygon.kmz”/> add the file path where your kmz file is stored. You can also specified 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 emisor point just use the following block:

<positionTimeSeries>

<file name="data/spill_trajectory.csv" />

</positionTimeSeries>

The format inside the *spill_trajectory.csv* must be:

```
TIME_CYCLE                : 0
TIME_UNITS                : SECONDS
SERIE_INITIAL_DATA        : 2005 08 31 01 00 00

!time   Lon Lat level
<BeginTimeSerie>
1   1.92    50.57   -3
2   1.925058457 50.5648824  -3
3   1.928727092 50.55476757 -3
4   1.929997942 50.55000823 -3
5   1.928521837 50.55547566 -3
6   1.924704348 50.56557384 -3
7   1.919594316 50.56996708 -3
8   1.914595755 50.56415883 -3
9   1.911082021 50.55409393 -3
10  1.910018521 50.55007401 -3
11  1.911697451 50.55621353 -3
12  1.915657523 50.56622858 -3
13  1.920810711 50.56986855 -3
14  1.925741135 50.56340787 -3
15  1.929094182 50.55345917 -3
16  1.929948585 50.55020513 -3
17  1.928069594 50.55697633 -3
18  1.923973465 50.56684231 -3
19  1.918785619 50.56970506 -3
20  1.913931427 50.56263448 -3
21  1.910744587 50.55286747 -3
22  1.910100691 50.55040074 -3
23  1.912176649 50.55775904 -3
<EndTimeSerie>
```

1.7.4 Setup the particle types

The <sourceTypes> contain the information about the particle type for each source. All the particle types must be placed in a xml file to act as a database for future plastics 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>

```
<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.xml"/>
</sourceTypes>
```

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

```
<?xml version="1.0" encoding="UTF-8" ?>
<materials>
    <plastic>
        <bag_1>
            <particulate value="false" />
            <density value="0.7" />
            <radius value="0.2" />
            <property key="condition" value="0.95" />
            <property key="degradation_rate" value="0.00000000158" />
        </bag_1>
        <expanded_polysryrene_1>
            <particulate value="false" />
            <density value="0.2" />
            <radius value="0.5" />
            <property key="condition" value="0.91" />
            <property key="degradation_rate" value="0.000000000317" />
        </expanded_polysryrene_1>
        <expanded_polysryrene_10>
            <particulate value="true" />
            <density value="0.2" />
            <radius value="2.4" />
            <property key="condition" value="0.97" />
            <property key="degradation_rate" value="0.000000000317" />
            <property key="intitial_concentration" value="0.05" />
            <property key="particle_radius" value="0.005" />
        </expanded_polysryrene_10>
        <expanded_polysryrene_12>
            <particulate value="true" />
            <density value="0.24" />
            <radius value="2.4" />
            <property key="condition" value="0.90" />
            <property key="degradation_rate" value="0.000000000317" />
            <property key="intitial_concentration" value="0.005" />
            <property key="particle_radius" value="0.0025" />
        </expanded_polysryrene_12>
    </plastic>
    <paper>
        <cardboard_1>
            <particulate value="false" />
            <density value="0.98" />
            <radius value="0.5" />
            <property key="condition" value="0.75" />
            <property key="degradation_rate" value="0.00000000634" />
        </cardboard_1>
    </paper>
</materials>
```

The different properties associated to 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 in your needs.

1.8 Setup the constants

The setup <constants> are global values affecting to the integration and should be changed should not be changed unless you need to adapt some values to your needs. The description in the comment section will guide you to make the 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:

<DiffusionCoeff value="0.75" comment="Horizontal diffusion coefficient. Default = 1.0" units_comment="m^2/s" />

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" />

Particles can be resuspended if they are on the floor (landIntMask = 1). At this moment the resuspension in the vertical is done by adding to the particle a fraction of the horizontal velocity module. This ResuspensionCoeff factor controls it. 0, means no resuspension and 1, means full horizontal module to push particles up in the vertical component.

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

Some datasets doesn'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 position on horizontal plane to compute the derivatives involved.

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