convert2ugrid

The Conversion of MOSSCO-output (Structured Grid) into DAVIT-acceptable NetCDF (Unstructured Grid)

Version 0.2

Introduction

Since MOSSCO is a huge ongoing project, that requires a lot of effort for testing, it would be reasonable to be able to visualize results and quickly assess them in a convenient way. One of the available softwares, that offers a wide range of analyzing tools is DAVIT(SmileConsult). Unfortunately MOSSCO output cannot be loaded directly into DAVIT, because it produces a structured-grid NetCDF, while DAVIT requires unstructured-grid. Therefore a data-conversion should be performed in order to create unstructured-grid-based "davit-friendly" NetCDF file. The format of the grid itself is well explained in BAW WIKI. This project describes the convertor written in Python programming language, the underlying ideas and implementations.

[0] Important information

[0.1] Basic Idea

The conversion can be broken into three main steps:

1. Generate unstructured grid from MOSSCO-structured grid (see section [1] Creating unstructured grid)
2. Describe MOSSCO-variables in a proper way, so they are accepted with DAVIT (see section [2] Describing the variables - "dictionary-cascade" approach)
3. Create NetCDF and store all information in it (see section [3] Creating "DAVIT-friendly" NetCDF file)

[0.2] MOSSCO-output

At the time of converter-development (03.2015-07.2015) MOSSCO produced output in a structured rectangular 2D grid, with values given at face-center. Land faces were masked with *fill_value*. Cell center X and Y coordinates were given as LON, LAT. Third dimension (Z) was implemented through sigma-layers. The file with simulation results contained both 2D and 3D variables, which mostly had dimensions as follows (time, y, x), (time, z, y, x).

[0.3] Python-script remarks

The script consist of an executable (currently has name ***convert2ugrid.py***), which runs a conversion routines (in future will be extended with a gui version), and of modules stored in ***./lib*** folder. Below is given a brief description of each module:

- ***__init__.py*** - is an empty file, indicating that this folder contains modules (see python documentation)
- ***Mesh2.py*** - file contains code relevant for generation of unstructured grid, describes objects *Node, Edge, Face, Grid2D* (see section [1] Creating unstructured grid)
- ***make_grid.py*** - convenient grid creation, uses module Mesh2.py
- ***process_mossco_netcdf.py*** - here collected routines relevant for processing information which should be gathered from MOSSCO-output files.

- ***process_davit_ncdf.py*** - here collected routines relevant for processing “DAVIT-friendly” NetCDF file (creating, appending, changing, removing). This module is dependent on module *process_mossco_netcdf.py*
- ***process_cdl.py*** - here collected routines relevant for processing ASCII “Dictionaries” (see section [2] Describing the variables - “dictionary-cascade” approach). Reading/writing is done in this module. Also has dependency on *process_mossco_netcdf.py*
- ***process_mixed_data.py*** - here collected routines that work with mixed information.. Dependency: *process_mossco_netcdf.py*
- ***create_uGrid_netcdf.py*** - is the module, that defines and executes consequently scripts “step 1”, “step 2”, “step 3” (see fig.4 for description of steps and inputs)

This convertor-script has been tested with following configuration:

- Python (2.7.6)
- netcdf4-python (1.0.8)
- numpy (1.8.0)

NetCDF related libraries:

- HDF5 (version: 1.8.11)
- NetCDF (version: 4.3.0)

[1] Creating unstructured grid

[1.1] Idea

The format of the desired unstructured grid was chosen taken in accordance with BAW WIKI.

Based on facts, that the MOSSCO-grid is rectangular and that the position of each valid face (polygon) is known (non fill_value cell in “bathymetry” variable in “topo.nc”), the following approach has been chosen (see fig. 1):

1. Create rectangular grid from *lon, lat* vector-variables (shape *lon*lat*)
2. (Optional) Apply mask based on *bathymetry* variable (to hide invalid elements)
3. Enumerate valid faces
4. Cycle over face-indexes and create objects of type *Node, Edge, Face*, determine connectivity, boundaries
5. Prepare all information about the grid in proper format by creating variables : nMaxMesh2_face_nodes, Mesh2_edge_nodes, Mesh2_edge_faces, Mesh2_face_nodes, Mesh2_face_edges, Mesh2_node_x, Mesh2_node_y, Mesh2_edge_x, Mesh2_edge_y, Mesh2_face_x, Mesh2_face_y, Mesh2_face_center_x, Mesh2_face_center_y, Mesh2_edge_x_bnd, Mesh2_edge_y_bnd, Mesh2_face_x_bnd, Mesh2_face_y_bnd

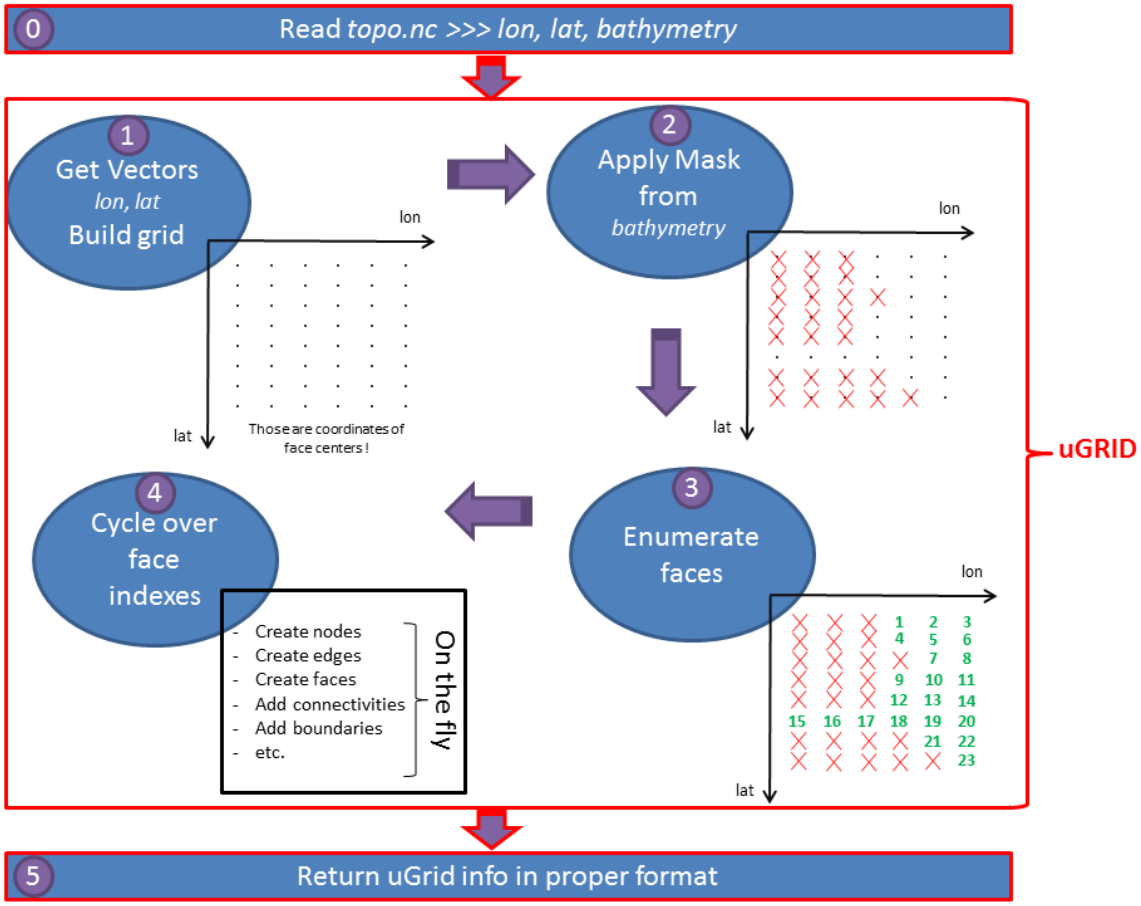


Figure 1. Scheme of creating uGrid from x- and y- vectors stored in MOSSCO-output files.

The example of punkt 4 of algorithm is shown in Appendix (see Example of block-4 of uGrid creation algorithm). In this example (at step 5), we can see, that face#1 and face#2 have two common nodes (#3, #4) and one common edge (#3). Since the nodes are generated for each encountered face index (algorithm punkt 4), one can expect “double” nodes and edges as shown on figure 2 (compare it with step 5 in example). There, node#3, node#4 and edge#3 were created while dealing with face#1. Here node#5, node#6 and edge#5 were created while dealing with face#2. To get rid of these additional indexes it is required to know neighbour-faces (to asses shared nodes and edges) at the time of face-generation. For example, following figure 2, if we would know (at the time of generating face#2) that there is a neighbour-face to the left, we would skip creating nodes #5, #6 and edge #5. Then the correct indexing would look like in example Example of block-4 of uGrid creation algorithm (step 5).

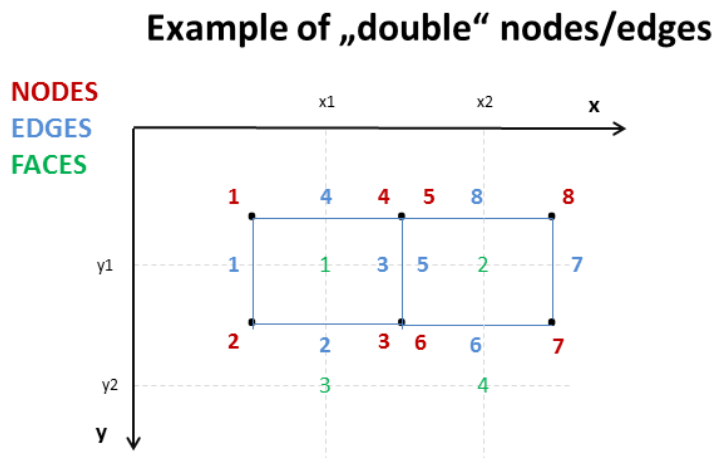


Figure 2. False, “double”-indexing of nodes/edges. Faces 1 and 2 have two shared nodes, which are indexed twice (node#4=node#5, node#3=node#6). Same applies for shared edge (edge#3=edge#5)

As explained above, the correct *node* and *edge* indexing require detection of neighbour polygons. Main idea for this detection is outlined below. While cycling through face indexes, it is only needed to check whether the current face has neighbours at a)top-right, b)top, c)top-left, d)left (these terms are related to cell i-,j- indexes). Lets assume that current face has indexes [i, j], then:

- Top-right neighbour will have index [i+1, j-1]
- Top neighbour will have index [i, j-1]
- Top-left neighbour will have index [i-1, j-1]
- Left neighbour will have index [i-1, j]

All of those possible neighbour-faces are already created, because i-direction (x-direction) is prefered to j-direction while cycling over the grid.

First of all, the whole grid is first divided into 5 zones, depending on possible neighbour-condition (see fig.3). Secondly, while cycling through face indexes in punkt 4 in algorithm above, the correct neighbour-zone is determined, and based on the chosen zone neighbours are found.

Examples for figure 3:

- for face#2 and face#3 the code would search for neighbor only at position left to it. For face#2 the left-neighbour would be found (face#1), oppositely for face#3 no left-neighbour would be found (masked values are ignored).
- for face#14 the code would search for neighbors only at positions a)top-right, b)top, c)top-left, d)left. It would find following neighbour faces: face#8 and face#13
- for face#24 the code would search for neighbors only at positions b)top, c)top-left, d)left. It would find following neighbour faces: face#16 and face#15.

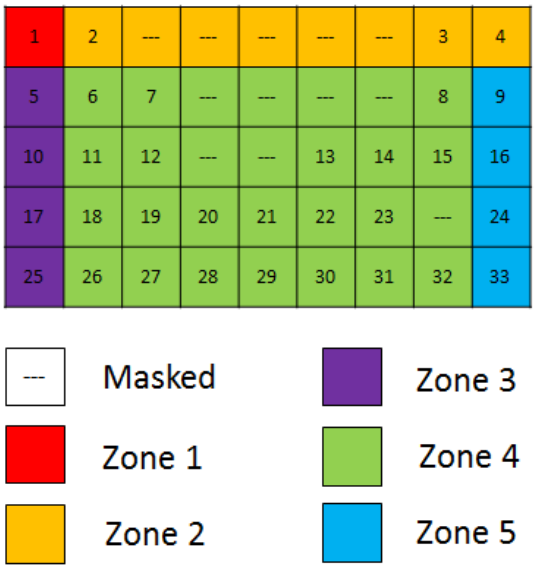


Figure 3. A rectangular 9x5 grid (numbers indicate face indexes, --- stands for masked cells) is colored based on cell i-,j- index. The color represents different neighbour-condition. As described in section [1.1] Idea, the only neighbours we are interested in are a)top-right, b)top, c)top-left, d)left. Element in Zone 1 do not have any neighbours. Elements in Zone 2 can have neighbors only at d)left. In Zone 3 - at a)top-right, b)top. In Zone 4 - at a)top-right, b)top, c)top-left, d)left. In Zone 5 - at b)top, c)top-left, d)left.

[1.2] Python implementation

The code describing this section can be found in modules *Mesh2.py, make_grid.py.* User access grid-creation routines at “step 3”-script (see [3.1] “STEP 3” script)

An OOP approach has been chosen to deal with grid conversion. Three basic classes are describing grid elements: *Node, Edge, Face*. *Node* is an object which has following 3 attributes: x-coordinate, y-coordinate, index. *Edge* is an object consisting of 2 nodes (objects of type *Node*) and has its own index as well. *Face* object consists of 4 nodes (the code has been written in a way to handle also 3-node faces , triangles) and unique index. Various properties such as:

- “edge center coordinates”, “edge length” for edges
- “center of mass”, “center of circumcircle”, “area” , etc. for faces

can be calculated within class methods (see code). The basic information about these 3 classes is summarized in table 1.

The fourth class *Grid2D* serves as a container for all *Node, Edge, Face* objects that are created during grid-conversion. Additionally, *Grid2D* provides an index map for easy face access (see fig.1 block-4). Within this class an unstructured grid is actually created following the algorithm described in section [1.1] Idea. It is important to understand, that all simplifications, which are currently in use (see section [1.3] Limitations) are introduced only within this class and not within classes *Node, Edge, Face* (thus allowing the independent usage of *Node, Edge, Face* objects for other grids).

Object type	Initialization	Important "Getters"
Node	x (*float*) y (*float*) index (*int*)	get_index() get_node_x() get_node_y()
Edge	node1 (*Node*) node2 (*Node*) index (*int*)	get_index() get_edge_x() get_edge_y() get_edge_length() get_node1() >>> object *Node* get_node2() >>> object *Node*
Face	node_list (list of 3 or 4 objects *Node*) index (*int*)	get_index() get_face_x() get_face_y() get_face_center_x() get_face_center_y() get_area() get_nNodes() >>> *int*, number of nodes get_nodes() >>> list of objects *Node* (3 or 4) get_edges() >>> list of objects *Edge* (3 or 4)
Grid2D	x_vector (1D array with *floats* of lenght m) y_vector (1D array with *floats* of lenght n) mask (2D array with *booleans* of shape m*n)	get_nMaxMesh2_face_nodes() get_Mesh2_node_x() get_Mesh2_node_y() get_Mesh2_edge_x() get_Mesh2_edge_y() get_Mesh2_face_x() get_Mesh2_face_y() get_Mesh2_face_center_x() get_Mesh2_face_center_y() get_Mesh2_edge_nodes() get_Mesh2_edge_faces() get_Mesh2_face_nodes() get_Mesh2_face_edges() get_Mesh2_face_edges() get_Mesh2_edge_bnd_x() get_Mesh2_edge_bnd_y() get_Mesh2_face_bnd_x() get_Mesh2_face_bnd_y()

Table 1. Four main classes Node, Edge, Face, Grid2D from which the grid is made. Column 3 shows the required inputs for initialization of the object of specific type. Column 3 shows the most important "getter"-methods.

[1.3] Limitations

In accordance with MOSSCO-output, for the ease of grid-generation following simplifications were used:

- 2D rectangular uniform grid (all polygons are rectangles of equal size)
- Vertical z-layers (

Remarks:

- it is easy to generalize current routines to work with non-uniform 2D rectangular grid, if the node-coordinates would be provided.

[1.4] Mapping data from MOSSCO into new grid

After the unstructured grid is created , next step is to map variables in a following way:

MOSSCO variable dimension		**uGrid variable dimension**
(time)	>>>	(nMesh2_data_time)
(time, y, x)	>>>	(nMesh2_data_time, nMesh2_face)
(time, y, x) depth averaged	>>>	(nMesh2_data_time, nMesh2_layer_2d, nMesh2_face)
(time, z, y, x)	>>>	(nMesh2_data_time, nMesh2_layer_3d, nMesh2_face)

The general idea for mapping is to flatten a 2D array (y, x) into a vector, so that the position-index of values in this vector would correspond to proper face in uGrid-topology variables (Mesh2_face_x, Mesh2_face_y, etc.). This is done via functions *read_mossco_nc_1d(), read_mossco_nc_2d(), read_mossco_nc_3d(), read_mossco_nc_4d()* located in module *process_mossco_netcdf.py.* Each of these functions reads a MOSSCO-variable, processes data (a mask can also be applied to data, see algorithm in section [1.1] Idea) and return already mapped array. The only step left - is to save received data into NetCDF file.

Remarks:

- it is quite challenging to implement logic for correct dimension-determination, since MOSSCO-output netcdf file has a number of different dimensions: some of them are duplicated (getmGrid2D_getm_2 == getmGrid3D_getm_2 == pelagic_horizontal_grid_2, getmGrid2D_getm_1 == getmGrid3D_getm_1 == pelagic_horizontal_grid_1 and others), some - are changing their name depending on setup parameters (lat/lon >>> y/x).
- MOSSCO-variables are located in different grids (soil grid, hydrodynamic grid)
- the dimension names are hardcoded in module *process_mossco_netcdf.py*, within functions *read_mossco_nc_1(/2/3/4)d()*

[2] Describing the variables - “dictionary-cascade” approach

At the time of script-development MOSSCO produced more than 80 variables within its output-netcdf. Some general remarks about these variables:

1. Some of them are not in the area of our interest (*“dissolved_reduced_substances_upward_flux_at_soil_surface”, “total_custom_light_extinction_in_water”,* etc.)
2. Some of them are located not in hydrodynamic grid , rather in soil grid(*“slow_detritus_C_sources-sinks_in_soil”, “temperature_in_soil”,* etc.)
3. The data location of some variables is given via variable name and not via dimensions (i.e. variables “*depth_averaged_x_velocity_in_water*” and “*x_velocity_at_soil_surface*” share dimensions (*time*, *getmGrid2D_getm_2*, *getmGrid2D_getm_1*))
4. Some of the dimensions have equal size (i.e. *getmGrid2D_getm_2 = getmGrid3D_getm_2 = pelagic_horizontal_grid_2,* etc.)
5. Some variables did not have correct standard names, units and other attributes (i.e. variable “*time”* had an attribute “*units: seconds since 2009-01-02 00:00:00”*, which is not correct, due to lack of time-zone information. Correct attribute for GMT+1 would look like “*units: seconds since 2009-01-02 00:00:00 01:00”*)
6. MOSSCO itself is under development, and the output format may change
7. DAVIT software requires specific format of NetCDF files (see BAW WIKI)

One of the big questions of this project was “how to pick needed variables from MOSSCO and describe them correctly?” Taking into account remarks above, obviously this kind of task requires a lot of user input. A user interaction through ASCII files was chosen as a method of choice. The scheme below introduces the main idea of this method (see fig. 4).

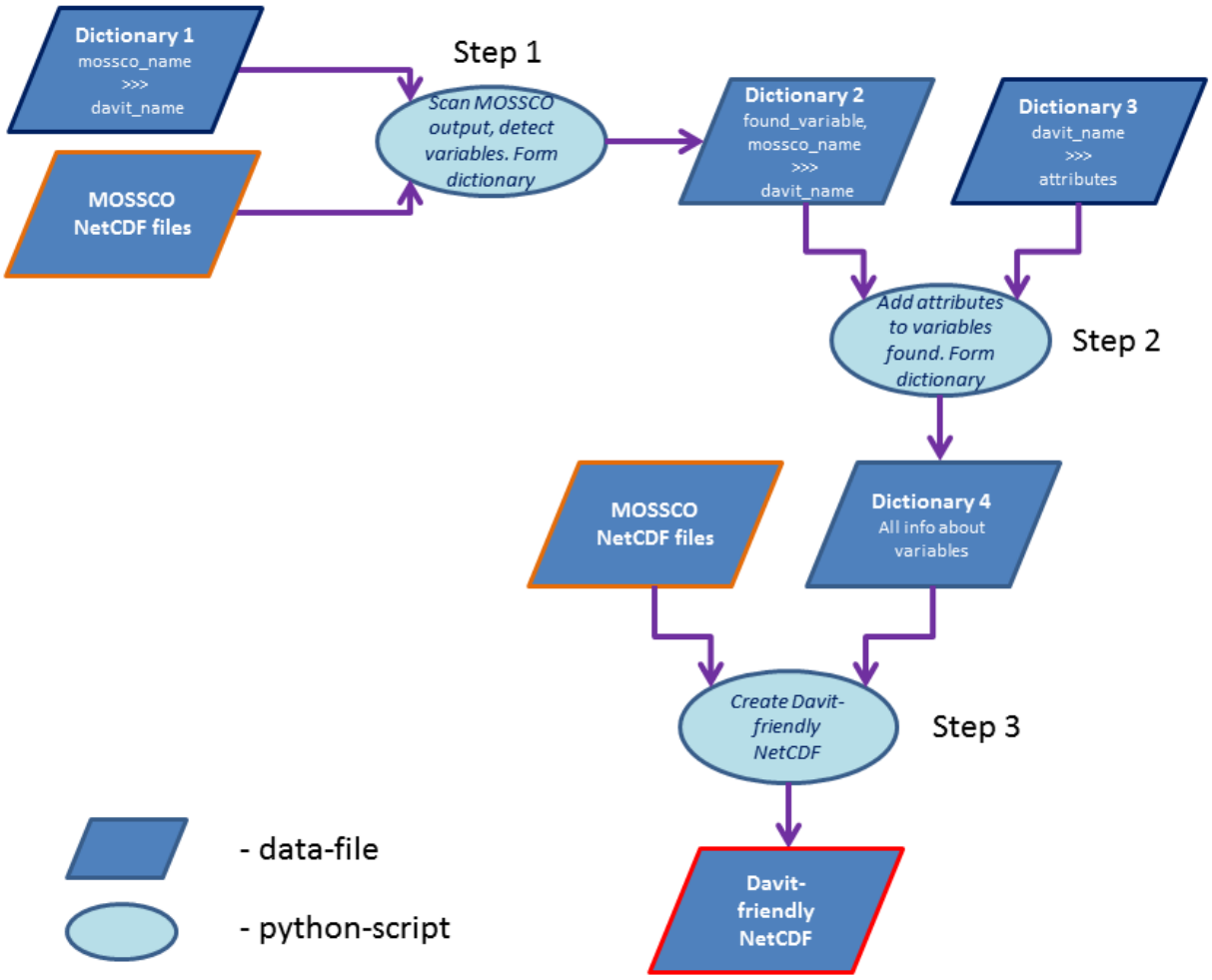


Figure 4. Scheme shows the "dictionary-cascade" approach used to transform variables from MOSSCO format into DAVIT-friendly format.

[2.1] "STEP 1" script

At first within "Dictionary 1" (see Example of "Dictionary 1"), user connects MOSSCO-variables names with required variable names (see fig.4). Basically it can be explained as virtual "renaming". Then the "Step 1"-script looks through passed MOSSCO netcdf files and searches for those variables that can be theoretically converted (must conform to valid dimensions); then it cycles over the found variables and tries to match them with names from "Dictionary 1". Finally, output of this script is placed in "Dictionary 2" (see Example of "Dictionary 2").

Note:

- If the variable is not shown in 2nd column of "Dictionary 2", most likely it has dimensions that were not recognised. If you want to convert a variable that was not detected, try including it manually in this dictionary before proceeding to "Step 2"-script

- 3rd column of “Dictionary 2” can be either a name taken from “Dictionary 1” or NOT_INCLUDED, meaning that the variable described in current row will be completely ignored by further processing.

[2.2] “STEP 2” script

Now (thanks to “Dictionary 2” created via [2.1] “STEP 1” script) we know which variables we want to convert (see fig.4), and we know what would be the new names. But we still do not know the required format for these variables (see p.7 in remarks above). Therefore “Dictionary 3” (see Example of “Dictionary 3”) is needed, which would describe the new format of each variable. The “Step 2”-script goes through “Dictionary 2” and for each variable looks up description in “Dictionary 3” by matching variable name. At the end, joined information is stored in “Dictionary 4” (see Example of “Dictionary 4”).

Note:

- “Dictionary 3” is a CDL format ASCII file, absolutely independent from MOSSCO.
- “Dictionary 4” is also a CDL format ASCII file. Additional attributes that point to the data location are added for each variable.
- Only those variables, that are listed in “Dictionary 4” will be appended to NetCDF (variables needed for mesh-generation are not listed in “Dictionary 4”, see [3.2] Writing uGrid information into NetCDF)
- Before proceeding to “Step 3”-script, user can modify “Dictionary 4”.

[3] Creating “DAVIT-friendly” NetCDF file

[3.1] “STEP 3” script

At this point (see fig.4), all preliminary information is gathered, and the actual conversion is ready to be done. Here is the algorithm of the “Step 3”-script:

- Creating uGrid (see section [1] Creating unstructured grid)
    - read lon, lat vectors
    - read mask
    - create grid
    - prepare grid-information in correct format
- Get number of z-layers
- Creating DAVIT-friendly NetCDF
    - choosing coordinate system (local/geographic)
    - writing “*Mesh2_crs*” variable
    - writing variables related to grid (see [3.2] Writing uGrid information into NetCDF)
    - writing *Mesh2_data_time* variable
    - writing variables related to vertical-coordinates
    - writing synoptic data variables specified in *“Dictionary 4”* (see also [1.4] Mapping data from MOSSCO into new grid)
        - appending normal variables
        - appending “auto-variables” (see [3.1.3] Auto-variables)

[3.1.1] Writing uGrid information into NetCDF

The already created grid is saved into NetCDF file. It is done with a function *create_uGrid_ncdf()* within module *process_davit_ncdf.py.*

Function appends following dimensions to NetCDF file:

- *nMesh2_node*
- *nMesh2_edge*
- *nMesh2_face*
- *nMaxMesh2_face_nodes*
- *two*
- *three*
- *nMesh2_time*
- *nMesh2_data_time*
- *nMesh2_layer_2d*
- *nMesh2_layer_3d*
- *nMesh2_class_names_strlen*
- *nMesh2_suspension_classes*

Function appends following variables to NetCDF file depending on coordinate_mode ("geographic"/"local"):

- *Mesh2_node_lon / Mesh2_node_x*
- *Mesh2_node_lat / Mesh2_node_y*
- *Mesh2_edge_lon / Mesh2_edge_x*
- *Mesh2_edge_lat / Mesh2_edge_y*
- *Mesh2_face_lon / Mesh2_face_x*
- *Mesh2_face_lat / Mesh2_face_y*
- *Mesh2_face_center_lon / Mesh2_face_center_x*
- *Mesh2_face_center_lat / Mesh2_face_center_y*
- *(OPTIONAL) Mesh2_edge_lon_bnd / Mesh2_edge_x_bnd*
- *(OPTIONAL) Mesh2_edge_lat_bnd / Mesh2_edge_y_bnd*
- *(OPTIONAL) Mesh2_face_lon_bnd / Mesh2_face_x_bnd*
- *(OPTIONAL) Mesh2_face_lat_bnd / Mesh2_face_y_bnd*
- *Mesh2_edge_nodes / Mesh2_edge_nodes*
- *Mesh2_edge_faces / Mesh2_edge_faces*
- *Mesh2_face_nodes / Mesh2_face_nodes*
- *Mesh2_face_edges / Mesh2_face_edges*
- *(OPTIONAL) --- / Mesh2_face_area*
- *Mesh2 / Mesh2*
- *Mesh2_crs / ---*

[3.1.2] Vertical layers

Currently MOSSCO is generating output in sigma-layers, but the DAVIT is limited to z-layers. In order to visualize these layers in DAVIT, the MOSSCO data is stored in **artificial** z-layers.

These layers have a thickness of 1m each; first layer starts from MSL. In example, if data has 5 vertical layers in z-direction, then their actual elevation values will be overwritten with 0.5, 1.5, 2.5, 3.5, 4.5 meters below MSL as cell center elevation. More information can be found in code located in the module *process_davit_ncdf.py* within the function *append_test_Mesh2_face_z_3d_and_Mesh2_face_z_3d_bnd()*

[3.1.3] Auto-variables

Auto-variables are those generated by the code automatically without user interaction. For example this could be vectors. If we have x- and y- velocity we might also be interested in saving the magnitude as separate variable. User should predefine those variables before "STEP 2". Currently it is done in a hard-coded way: dictionary descrbing names of desired auto-variables is located in the module *process_cdl.py* within the function *create_cdl_file()*.

[4] How to use the script

Note: this section will refer to files located on BAW internal servers. You may want to skip this section if you do not have a proper access.

Here is the step-by-step example of using current script.

1) Make a backup of folder, since its content will be modified:
   a. */net/themis/system/akprog/python/qad/convert2ugrid/info/tutorial/output*
2) Here are the mossco NetCDF files we will be using:
   a. */net/themis/system/akprog/python/qad/convert2ugrid/info/tutorial/data/netcdf_reference_3d.nc*
   b. */net/themis/system/akprog/python/qad/convert2ugrid/info/tutorial/data/topo.nc*

   Where (a) contains simulation data and (b) – topology data. To take a closer look into the files you may use the *ncdump* tool (linux only)

3) The default dictionaries (dictionary#1 and dictionary#3) are located as follows:
   a. */net/themis/system/akprog/python/qad/convert2ugrid/info/tutorial/user_input/dictionary1.txt*
   b. */net/themis/system/akprog/python/qad/convert2ugrid/info/tutorial/user_input/dictionary3.cdl*

4) Let us open the script…

```
if __name__ == '__main__':
    setup_path = './info/tutorial'
    dict1 = os.path.join(setup_path, 'user_input/dictionary1.txt')
    dict3 = os.path.join(setup_path, 'user_input/dictionary3.cdl')

    topo_nc     = os.path.join(setup_path, 'data/topo.nc')
    synoptic_nc = os.path.join(setup_path, 'data/netcdf_reference_3d.nc')
    list_with_synoptic_nc = [synoptic_nc, topo_nc]

    # setting paths: OUTPUT FILES....
    dict2  = os.path.join(setup_path,  'output/dictionary2.txt')
    dict4  = os.path.join(setup_path, 'output/dictionary4.cdl')
    nc_out = os.path.join(setup_path, 'output/nsbs_davit.nc')

    # running script...
    create_uGrid_netcdf.create_davit_friendly_netcdf(topo_nc=topo_nc, list_with_synoptic_nc=list_with_synoptic_nc, nc_out=nc_out,
                dictionary_1=dict1, dictionary_2=dict2, dictionary_3=dict3, dictionary_4=dict4,
                start_from_step=1, create_davit_netcdf=True, log=True)
```

First we specify variable <setup_path>. Then, our inputs: dictionaries #1, #3 and NetCDF files (variables <dict1>, <dict3>, <topo_nc>, <synoptic_nc>). In this case we have only one file with simulation data, but we can specify more by adding additional variable (i.e. <synoptic_nc1>, <synoptic_nc2>, …). In that case, do not forget to add those variables to list <list_with_synoptic_nc> in a similar manner it is done in current example. The order of the files in a list matters! See code in function <create_davit_friendly_netcdf()> in module <create_uGrid_netcdf.py> for details. Finally we prescribe names for output files in <dict2>, <dict4>, <nc_out>.

5) Parameters of the main function…
   - We pass the parameters with filenames to the function.
   - Then, we decide from which step we want to start the script. In this case it is <*start_from_step=1*>, since we want to start from the very beginning (see section [2] Describing the variables - "dictionary-cascade" approach for details). If dictionaries #2 is already existing, you may start with <*start_from_step=2* >, and respectively if dictionaries #4 is present, you may start with <*start_from_step=3*>.
   - Leave flag < *create_davit_netcdf* > to True, if you want a netcdf-file to be created.
   - You can switch < *log* > to False, if you don't want to see any logs in console.

6) Running the script…
   After all necessary inputs have been set, execute the script by typing in console:
   $ python convert2ugrid.py

7) Output
   With the parameters specified above, 3 files are created during the run:

   a. */net/themis/system/akprog/python/qad/convert2ugrid/info/tutorial/output/dictionary2.txt*
   b. */net/themis/system/akprog/python/qad/convert2ugrid/info/tutorial/output/dictionary4.cdl*
   c. */net/themis/system/akprog/python/qad/convert2ugrid/info/tutorial/output/nsbs_davit.nc*

   First two – dictionaries #2 and #4. And a NetCDF file, that should be compatible with DAVIT.

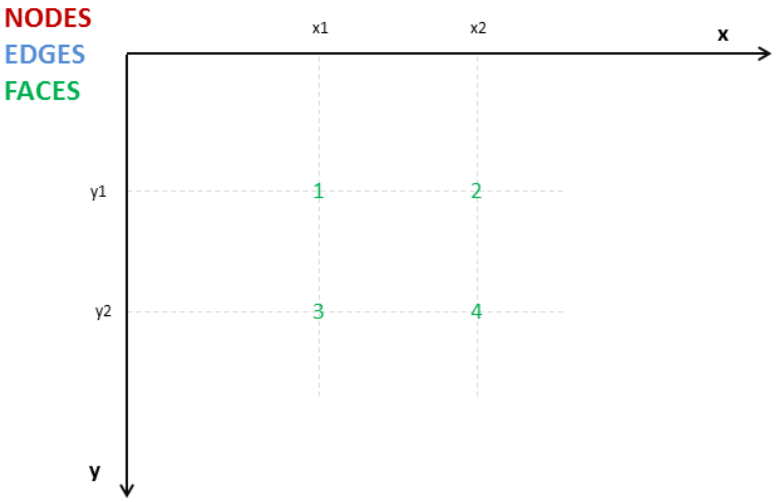
   In case you are not happy with the NetCDF file, you can edit its content by changing dictionary#4 manually and rerunning the script with parameter <*start_from_step=3*>.

8) Deflation
   Afterwards the file <nsbs_davit.nc> has been deflated with following command:
   $ nccopy –d 5 <oldname> <newname>

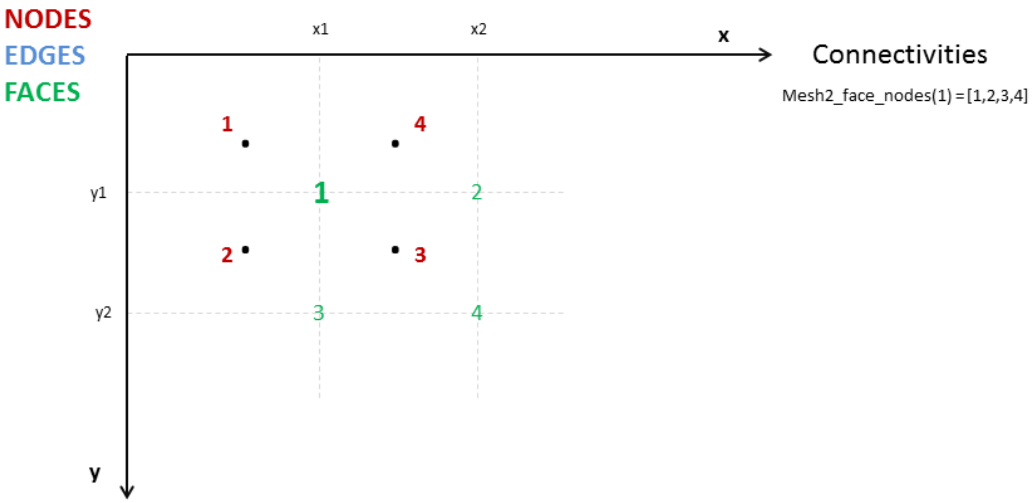
Appendix

1) Example of block-4 of uGrid creation algorithm

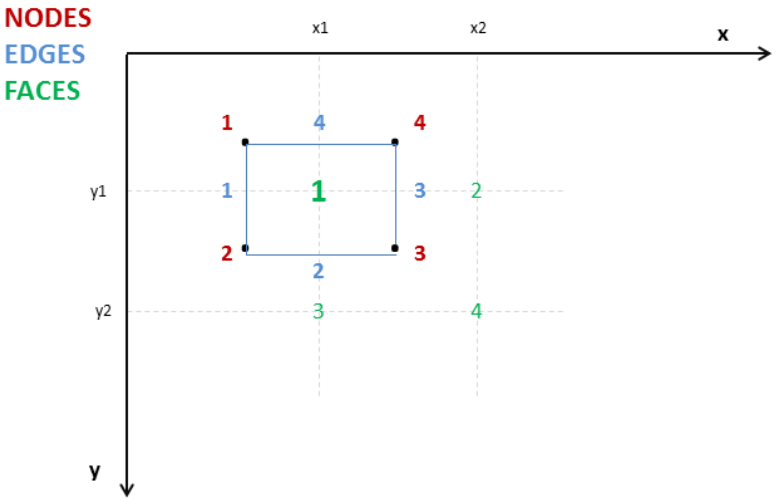
Example of block 4: „Cycle over face indexes“ (step 1)



Example of block 4: „Cycle over face indexes“ (step 2)



Example of block 4: „Cycle over face indexes“ (step 3)



Connectivities

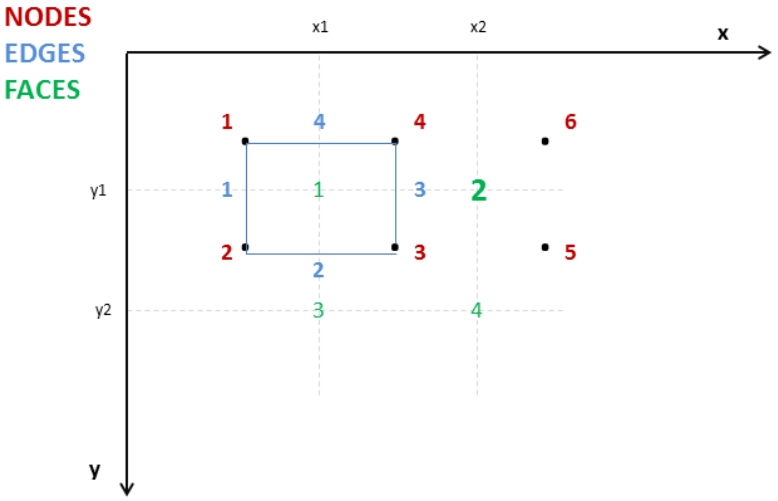
Mesh2_face_nodes(1) = [1,2,3,4]

Mesh2_face_edges(1) = [1,2,3,4]

Mesh2_edge_nodes(1) = [1,2]
Mesh2_edge_nodes(2) = [2,3]
Mesh2_edge_nodes(3) = [3,4]
Mesh2_edge_nodes(4) = [4,1]

Mesh2_edge_faces(1) = [1,-999]
Mesh2_edge_faces(2) = [1,3]
Mesh2_edge_faces(3) = [1,2]
Mesh2_edge_faces(4) = [1,-999]

Example of block 4: „Cycle over face indexes“ (step 4)



Connectivities

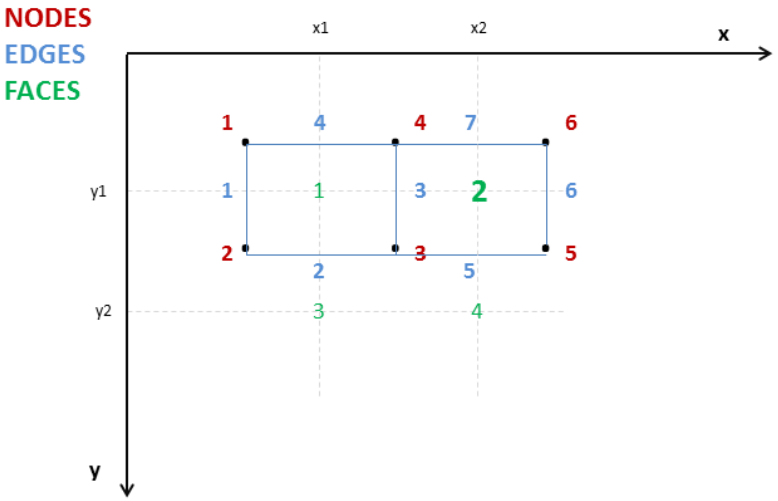
Mesh2_face_nodes(1) = [1,2,3,4]
Mesh2_face_nodes(2) = [4,3,5,6]

Mesh2_face_edges(1) = [1,2,3,4]

Mesh2_edge_nodes(1) = [1,2]
Mesh2_edge_nodes(2) = [2,3]
Mesh2_edge_nodes(3) = [3,4]
Mesh2_edge_nodes(4) = [4,1]

Mesh2_edge_faces(1) = [1,-999]
Mesh2_edge_faces(2) = [1,3]
Mesh2_edge_faces(3) = [1,2]
Mesh2_edge_faces(4) = [1,-999]

Example of block 4: „Cycle over face indexes“ (step 5)



Connectivities

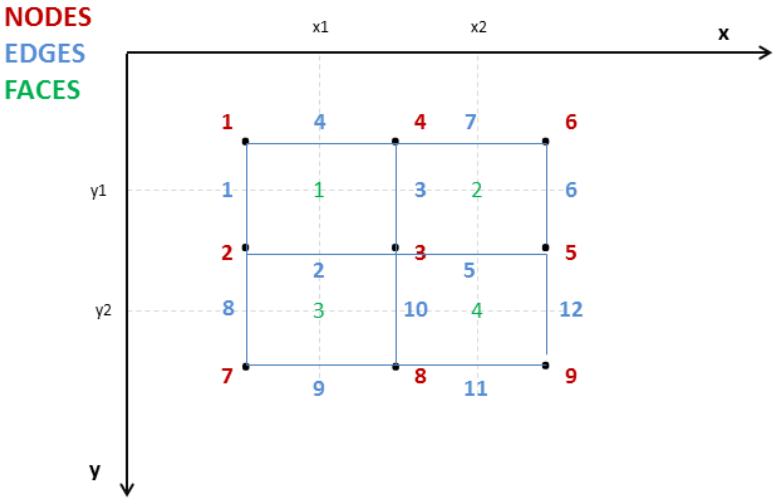
Mesh2_face_nodes(1) = [1,2,3,4]
Mesh2_face_nodes(2) = [4,3,5,6]

Mesh2_face_edges(1) = [1,2,3,4]
Mesh2_face_edges(2) = [3,5,6,7]

Mesh2_edge_nodes(1) = [1,2]
Mesh2_edge_nodes(2) = [2,3]
Mesh2_edge_nodes(3) = [3,4]
Mesh2_edge_nodes(4) = [4,1]
Mesh2_edge_nodes(5) = [3,5]
…

Mesh2_edge_faces(1) = [1,-999]
Mesh2_edge_faces(2) = [1,3]
Mesh2_edge_faces(3) = [1,2]
Mesh2_edge_faces(4) = [1,-999]
Mesh2_edge_faces(5) = [2,4]
…

Example of block 4: „Cycle over face indexes“ (step 6)



Connectivities

Mesh2_face_nodes(1) = [1,2,3,4]
Mesh2_face_nodes(2) = [4,3,5,6]
Mesh2_face_nodes(3) = [2,7,8,3]
Mesh2_face_nodes(4) = [3,8,9,4]

Mesh2_face_edges(1) = [1,2,3,4]
Mesh2_face_edges(2) = [3,5,6,7]
Mesh2_face_edges(3) = [8,9,10,2]
Mesh2_face_edges(4) = [10,11,12,5]

Mesh2_edge_nodes(1) = [1,2]
Mesh2_edge_nodes(2) = [2,3]
Mesh2_edge_nodes(3) = [3,4]
Mesh2_edge_nodes(4) = [4,1]
Mesh2_edge_nodes(5) = [3,5]
…
Mesh2_edge_nodes(12) = [9,5]

Mesh2_edge_faces(1) = [1,-999]
Mesh2_edge_faces(2) = [1,3]
Mesh2_edge_faces(3) = [1,2]
Mesh2_edge_faces(4) = [1,-999]
Mesh2_edge_faces(5) = [2,4]
…
Mesh2_edge_faces(12) = [4,-999]

1) Basic description of *Node* python-object

NODES

Init-Setters

Node.set_index()
Node.set_x()
Node.set_y()

Getters

Node.get_node_index()
Node.get_node_x()
Node.get_node_y()

1) Basic description of *Edge* python-object

EDGES

Init-Setters

Edge.set_index()
Edge.set_node1() <<< object *Node*
Edge.set_node2() <<< object *Node*

Getters

Edge.get_index()
Edge.get_edge_x()
Edge.get_edge_y()
Edge.get_edge_length()
Edge.get_node1() >>> object *Node*
Edge.get_node2() >>> object *Node*

1) Basic description of *Face* python-object

FACES

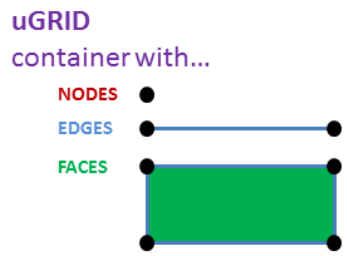
Init-Setters

Face.set_index()
Face.set_nodes() <<< list of objects *Node* (3 or 4)

Getters

Face.get_index()
Face.get_face_x()
Face.get_face_y()
Face.get_face_center_x()
Face.get_face_center_y()
Face.get_area()
Face.get_nNodes() >>> int, number of nodes
Face.get_nodes() >>> list of objects *Node* (3 or 4)
Face.get_edges() >>> list of objects *Edge* (3 or 4)

1) Basic description of *uGrid* python-object



Init-Setters

Grid2D.set_x_vector()
Grid2D.set_y_vector()
Grid2D.set_mask()

Getters

Grid2D.get_nMaxMesh2_face_nodes()

Grid2D.get_Mesh2_node_x()
Grid2D.get_Mesh2_node_y()
Grid2D.get_Mesh2_edge_x()
Grid2D.get_Mesh2_edge_y()
Grid2D.get_Mesh2_face_x()
Grid2D.get_Mesh2_face_y()
Grid2D.get_Mesh2_face_center_x()
Grid2D.get_Mesh2_face_center_y()

Grid2D.get_Mesh2_edge_nodes()
Grid2D.get_Mesh2_edge_faces()
Grid2D.get_Mesh2_face_nodes()
Grid2D.get_Mesh2_face_edges()
Grid2D.get_Mesh2_face_edges()

Grid2D.get_Mesh2_edge_bnd_x()
Grid2D.get_Mesh2_edge_bnd_y()
Grid2D.get_Mesh2_face_bnd_x()
Grid2D.get_Mesh2_face_bnd_y()

1) Example of “Dictionary 1”

```
// This file is a dictionary describing default connections between the variable names
// in BAW-format and mossco-format. The names are given in '' or in "". First stays
// the BAW name and after >>> follows the MOSSCO variable name. Comments may follow
// after //, spaces, newlines and tabs may be used freely for readibility
// -------------------------------------------------------------------------------

'Mesh2_face_Stroemungsgeschwindigkeit_x_2d'      >>> 'depth_averaged_x_velocity_in_water'
'Mesh2_face_Stroemungsgeschwindigkeit_y_2d'      >>> 'depth_averaged_y_velocity_in_water'
'Mesh2_face_Temperatur_3d'                       >>> 'temperature_in_water'
'Mesh2_face_depth_2d'                            >>> 'bathymetry'
'Mesh2_face_Wasserstand_2d'                      >>> 'water_depth_at_soil_surface'
'Mesh2_face_WaveHeight_2d'                       >>> 'wave_height'
'Mesh2_face_SedimentFlux_at_Soil_2D'     >>> 'concentration_of_SPM_upward_flux_at_soil_surface_001'
'Mesh2_face_RadiactiveFlux_at_Surface_2D'        >>> 'surface_downwelling_photosynthetic_radiative_flux'
'Mesh2_face_Temperature_at_Soil_2D'      >>> 'temperature_at_soil_surface'
'Mesh2_face_WindVelocity_at_10m_x_2d'            >>> 'wind_x_velocity_at_10m'
'Mesh2_face_WindVelocity_at_10m_y_2d'            >>> 'wind_y_velocity_at_10m'
'Mesh2_face_WaterVelocity_at_Soil_x_2d'          >>> 'x_velocity_at_soil_surface'
'Mesh2_face_WaterVelocity_at_Soil_y_2d'          >>> 'y_velocity_at_soil_surface'
```

1) Example of “Dictionary 2”

```
// file describes corelation between mossco-output-netcdf variable names and
// baw-format variable names
//
// format: "filename", "mossco variable name" >>> "corresponding baw format variable name"
```

```
// format: "filename", "mossco variable name" >>> NOT_INCLUDED
// spaces and tabs may be used freely for readability. Comments may follow after "//"
//
// Displaying Davit-friendly variables found in MOSSCO output netcdf file
// Let Davit-friendly variables be those, which have dimensions:
// 2D: (ydim, xdim)
// 3D: (tdim, ydim, xdim)
// 4D: (tdim, zdim, ydim, xdim)
// ---------------------------------------------------------------------------------------------

//   2D
  "\\widar\home\mossco.nc",   "pet_getmGrid2D_getm"                                          >>> NOT_INCLUDED
//   3D
  "\\widar\home\mossco.nc",   "Effect_of_MPB_on_critical_bed_shearstress_at_soil_surface"        >>> NOT_INCLUDED
  "\\widar\home\mossco.nc",   "Effect_of_MPB_on_sediment_erodibility_at_soil_surface"            >>> NOT_INCLUDED
  "\\widar\home\mossco.nc",   "Effect_of_Mbalthica_on_critical_bed_shearstress_at_soil_surface"  >>> NOT_INCLUDED
  "\\widar\home\mossco.nc",   "Effect_of_Mbalthica_on_sediment_erodibility_at_soil_surface"      >>> NOT_INCLUDED
  "\\widar\home\mossco.nc",   "depth_averaged_x_velocity_in_water"                               >>>
"Mesh2_face_Stroemungsgeschwindigkeit_x_2d"
  "\\widar\home\mossco.nc",   "depth_averaged_y_velocity_in_water"                               >>>
"Mesh2_face_Stroemungsgeschwindigkeit_y_2d"
  "\\widar\home\mossco.nc",   "detritus-P_upward_flux_at_soil_surface"                           >>> NOT_INCLUDED
  "\\widar\home\mossco.nc",   "dissolved_oxygen_upward_flux_at_soil_surface"                     >>> NOT_INCLUDED
  "\\widar\home\mossco.nc",   "dissolved_reduced_substances_upward_flux_at_soil_surface"     >>> NOT_INCLUDED
  "\\widar\home\mossco.nc",   "fast_detritus_C_upward_flux_at_soil_surface"                      >>> NOT_INCLUDED
  "\\widar\home\mossco.nc",   "layer_height_at_soil_surface"                                     >>> NOT_INCLUDED
  "\\widar\home\mossco.nc",   "slow_detritus_C_upward_flux_at_soil_surface"                      >>> NOT_INCLUDED
  "\\widar\home\mossco.nc",   "temperature_at_soil_surface"                          >>> "Mesh2_face_Temperature_at_Soil_2D"
  "\\widar\home\mossco.nc",   "water_depth_at_soil_surface"                          >>> "Mesh2_face_Wasserstand_2d"
  "\\widar\home\mossco.nc",   "wave_height"                                          >>> "Mesh2_face_WaveHeight_2d"
  "\\widar\home\mossco.nc",   "wave_number"                                          >>> "Mesh2_face_WaveNumber_2D"
  "\\widar\home\mossco.nc",   "wave_period"                                          >>> "Mesh2_face_MeanWavePeriod_2D"
  "\\widar\home\mossco.nc",   "wind_x_velocity_at_10m"                               >>> "Mesh2_face_WindVelocity_at_10m_x_2d"
  "\\widar\home\mossco.nc",   "wind_y_velocity_at_10m"                               >>> "Mesh2_face_WindVelocity_at_10m_y_2d"
  "\\widar\home\mossco.nc",   "x_velocity_at_soil_surface"                           >>> "Mesh2_face_WaterVelocity_at_Soil_x_2d"
  "\\widar\home\mossco.nc",   "y_velocity_at_soil_surface"                           >>> "Mesh2_face_WaterVelocity_at_Soil_y_2d"
//   4D
  "\\widar\home\mossco.nc",   "concentration_of_SPM_in_water_001"                  >>> NOT_INCLUDED
  "\\widar\home\mossco.nc",   "concentration_of_SPM_in_water_002"                  >>> NOT_INCLUDED
  "\\widar\home\mossco.nc",   "concentration_of_SPM_sources-sinks_in_water"        >>> NOT_INCLUDED
  "\\widar\home\mossco.nc",   "concentration_of_SPM_z_velocity_in_water_001"     >>> NOT_INCLUDED
  "\\widar\home\mossco.nc",   "concentration_of_SPM_z_velocity_in_water_002"     >>> NOT_INCLUDED
  "\\widar\home\mossco.nc",   "temperature_in_water"                                 >>> "Mesh2_face_Temperatur_3d"
  "\\widar\home\mossco.nc",   "total_custom_light_extinction_in_water"                       >>> NOT_INCLUDED
  "\\widar\home\mossco.nc",   "total_dissolved_inorganic_carbon_sources-sinks_in_water"          >>> NOT_INCLUDED
  "\\widar\home\mossco.nc",   "zooplankton_in_water"                                 >>> NOT_INCLUDED
  "\\widar\home\mossco.nc",   "zooplankton_sources-sinks_in_water"                           >>> NOT_INCLUDED
  "\\widar\home\mossco.nc",   "zooplankton_z_velocity_in_water"                      >>> NOT_INCLUDED

// ---------------------------------------------------------------------------------------------
  "\\widar\home\topo.nc",   "bathymetry"                                        >>> "Mesh2_face_depth_2d"
```

1) Example of “Dictionary 3”

```
float Mesh2_face_Stroemungsgeschwindigkeit_x_2d(nMesh2_data_time, nMesh2_layer_2d, nMesh2_face) ;
        Mesh2_face_Stroemungsgeschwindigkeit_x_2d:long_name = "Stroemungsgeschwindigkeit (x-Komponente), Face (Polygon)" ;
        Mesh2_face_Stroemungsgeschwindigkeit_x_2d:units = "m s-1" ;
        Mesh2_face_Stroemungsgeschwindigkeit_x_2d:name_id = 2 ;
        Mesh2_face_Stroemungsgeschwindigkeit_x_2d:_FillValue = 1.e+31f ;
        Mesh2_face_Stroemungsgeschwindigkeit_x_2d:cell_methods = "nMesh2_data_time: point nMesh2_layer_2d: mean area: point"
;
        Mesh2_face_Stroemungsgeschwindigkeit_x_2d:coordinates = "Mesh2_face_x Mesh2_face_y Mesh2_face_lon Mesh2_face_lat
Mesh2_face_z_face_2d" ;
        Mesh2_face_Stroemungsgeschwindigkeit_x_2d:grid_mapping = "Mesh2_crs" ;
```

Mesh2_face_Stroemungsgeschwindigkeit_x_2d:standard_name = "sea_water_x_velocity" ;
Mesh2_face_Stroemungsgeschwindigkeit_x_2d:mesh = "Mesh2" ;
Mesh2_face_Stroemungsgeschwindigkeit_x_2d:location = "face" ;
Mesh2_face_Stroemungsgeschwindigkeit_x_2d:davit_role = "visualization_variable" ;

float Mesh2_face_Stroemungsgeschwindigkeit_y_2d(nMesh2_data_time, nMesh2_layer_2d, nMesh2_face) ;
Mesh2_face_Stroemungsgeschwindigkeit_y_2d:long_name = "Stroemungsgeschwindigkeit (y-Komponente), Face (Polygon)" ;
Mesh2_face_Stroemungsgeschwindigkeit_y_2d:units = "m s-1" ;
Mesh2_face_Stroemungsgeschwindigkeit_y_2d:name_id = 2 ;
Mesh2_face_Stroemungsgeschwindigkeit_y_2d:_FillValue = 1.e+31f ;
Mesh2_face_Stroemungsgeschwindigkeit_y_2d:cell_methods = "nMesh2_data_time: point nMesh2_layer_2d: mean area: point"
;
Mesh2_face_Stroemungsgeschwindigkeit_y_2d:coordinates = "Mesh2_face_x Mesh2_face_y Mesh2_face_lon Mesh2_face_lat
Mesh2_face_z_face_2d" ;
Mesh2_face_Stroemungsgeschwindigkeit_y_2d:grid_mapping = "Mesh2_crs" ;
Mesh2_face_Stroemungsgeschwindigkeit_y_2d:standard_name = "sea_water_y_velocity" ;
Mesh2_face_Stroemungsgeschwindigkeit_y_2d:mesh = "Mesh2" ;
Mesh2_face_Stroemungsgeschwindigkeit_y_2d:location = "face" ;
Mesh2_face_Stroemungsgeschwindigkeit_y_2d:davit_role = "visualization_variable" ;

float Mesh2_face_Stroemungsgeschwindigkeit_m_2d(nMesh2_data_time, nMesh2_layer_2d, nMesh2_face) ;
Mesh2_face_Stroemungsgeschwindigkeit_m_2d:long_name = "Stroemungsgeschwindigkeit (Betrag), Face (Polygon)" ;
Mesh2_face_Stroemungsgeschwindigkeit_m_2d:units = "m s-1" ;
Mesh2_face_Stroemungsgeschwindigkeit_m_2d:name_id = 2 ;
Mesh2_face_Stroemungsgeschwindigkeit_m_2d:_FillValue = 1.e+31f ;
Mesh2_face_Stroemungsgeschwindigkeit_m_2d:cell_methods = "nMesh2_data_time: point nMesh2_layer_2d: mean area: point"
;
Mesh2_face_Stroemungsgeschwindigkeit_m_2d:coordinates = "Mesh2_face_x Mesh2_face_y Mesh2_face_lon Mesh2_face_lat
Mesh2_face_z_face_2d" ;
Mesh2_face_Stroemungsgeschwindigkeit_m_2d:grid_mapping = "Mesh2_crs" ;
Mesh2_face_Stroemungsgeschwindigkeit_m_2d:standard_name = "magnitude_of_sea_water_velocity" ;
Mesh2_face_Stroemungsgeschwindigkeit_m_2d:mesh = "Mesh2" ;
Mesh2_face_Stroemungsgeschwindigkeit_m_2d:location = "face" ;
Mesh2_face_Stroemungsgeschwindigkeit_m_2d:davit_role = "visualization_variable" ;

float Mesh2_face_Temperatur_3d(nMesh2_data_time, nMesh2_layer_3d, nMesh2_face) ;
Mesh2_face_Temperatur_3d:long_name = "Temperatur, Face (Polygon)" ;
Mesh2_face_Temperatur_3d:units = "degC" ;
Mesh2_face_Temperatur_3d:name_id = 6 ;
Mesh2_face_Temperatur_3d:_FillValue = 1.e+31f ;
Mesh2_face_Temperatur_3d:cell_measures = "area: Mesh2_face_wet_area" ;
Mesh2_face_Temperatur_3d:cell_methods = "nMesh2_data_time: point nMesh2_layer_3d: mean area: point" ;
Mesh2_face_Temperatur_3d:coordinates = "Mesh2_face_x Mesh2_face_y Mesh2_face_lon Mesh2_face_lat
Mesh2_face_z_face_3d" ;
Mesh2_face_Temperatur_3d:grid_mapping = "Mesh2_crs" ;
Mesh2_face_Temperatur_3d:standard_name = "temperature" ;
Mesh2_face_Temperatur_3d:mesh = "Mesh2" ;
Mesh2_face_Temperatur_3d:location = "face" ;

double Mesh2_face_depth_2d(nMesh2_time, nMesh2_face) ;
Mesh2_face_depth_2d:long_name = "Topographie" ;
Mesh2_face_depth_2d:units = "m" ;
Mesh2_face_depth_2d:name_id = 17 ;
Mesh2_face_depth_2d:_FillValue = 1.e+31 ;
Mesh2_face_depth_2d:cell_measures = "area: Mesh2_face_area" ;
Mesh2_face_depth_2d:cell_methods = "nMesh2_time: mean area: mean" ;
Mesh2_face_depth_2d:coordinates = "Mesh2_face_x Mesh2_face_y Mesh2_face_lon Mesh2_face_lat" ;
Mesh2_face_depth_2d:grid_mapping = "Mesh2_crs" ;
Mesh2_face_depth_2d:standard_name = "sea_floor_depth_below_geoid" ;
Mesh2_face_depth_2d:mesh = "Mesh2" ;
Mesh2_face_depth_2d:location = "face" ;
Mesh2_face_depth_2d:davit_role = "visualization_variable" ;

```
float Mesh2_face_Wasserstand_2d(nMesh2_data_time, nMesh2_face) ;
        Mesh2_face_Wasserstand_2d:long_name = "Wasserstand, Face (Polygon)" ;
        Mesh2_face_Wasserstand_2d:units = "m" ;
        Mesh2_face_Wasserstand_2d:name_id = 3 ;
        Mesh2_face_Wasserstand_2d:_FillValue = 1.e+31f ;
        Mesh2_face_Wasserstand_2d:cell_measures = "area: Mesh2_face_wet_area" ;
        Mesh2_face_Wasserstand_2d:cell_methods = "nMesh2_data_time: point area: mean" ;
        Mesh2_face_Wasserstand_2d:coordinates = "Mesh2_face_x Mesh2_face_y Mesh2_face_lon Mesh2_face_lat" ;
        Mesh2_face_Wasserstand_2d:grid_mapping = "Mesh2_crs" ;
        Mesh2_face_Wasserstand_2d:standard_name = "sea_surface_height" ;
        Mesh2_face_Wasserstand_2d:mesh = "Mesh2" ;
        Mesh2_face_Wasserstand_2d:location = "face" ;
```

1) Example of “Dictionary 4”

```
// This file has been generated by script, which couples BAW-synoptic data standart format with MOSSCO variable names
// This file is needed for further conversion of MOSSCO-netcdf file to DAVIT-friendly-netcdf file, and serves as an
// input for another processing script
//
// Below follows the description of synoptic data, presented in a form of NETCDF variables.
// Only those variables, listed below will be included in DAVIT-friendly netcdf.
// Note, that every variable has two additional non-standart string attributes: "_mossco_filename", "_mossco_varname".
// These attributes will not be included in DAVIT-friendly-netcdf, while only indicating the exact location of
// data for processing script. For example:
//          _mossco_filename = "D:\data\mossco_output.nc"
//          _mossco_varname = "depth_averaged_x_velocity_in_water"
// The standart BAW attributes, dimensions, and names are well described here:
//    http://www.baw.de/methoden/index.php5/NetCDF_Synoptische_Daten_im_Dreiecksgitter
// ------------------------------------------------------------------------------------------------

float Mesh2_face_Temperatur_3d (nMesh2_data_time, nMesh2_layer_3d, nMesh2_face) ;
   Mesh2_face_Temperatur_3d: _FillValue = 1.e+31 ;
   Mesh2_face_Temperatur_3d: grid_mapping = "Mesh2_crs" ;
   Mesh2_face_Temperatur_3d: location = "face" ;
   Mesh2_face_Temperatur_3d: coordinates = "Mesh2_face_x Mesh2_face_y Mesh2_face_lon Mesh2_face_lat Mesh2_face_z_3d" ;
   Mesh2_face_Temperatur_3d: long_name = "Temperatur, Face (Polygon)" ;
   Mesh2_face_Temperatur_3d: standard_name = "temperature" ;
   Mesh2_face_Temperatur_3d: cell_methods = "nMesh2_data_time: point nMesh2_layer_3d: mean area: point" ;
   Mesh2_face_Temperatur_3d: name_id = 6;
   Mesh2_face_Temperatur_3d: cell_measures = "area: Mesh2_face_wet_area" ;
   Mesh2_face_Temperatur_3d: units = "degC" ;
   Mesh2_face_Temperatur_3d: mesh = "Mesh2" ;
   Mesh2_face_Temperatur_3d: _mossco_filename = "\\Widar\home\mossco.nc" ;
   Mesh2_face_Temperatur_3d: _mossco_varname = "temperature_in_water" ;


double Mesh2_face_depth_2d (nMesh2_time, nMesh2_face) ;
   Mesh2_face_depth_2d: _FillValue = 1.e+31 ;
   Mesh2_face_depth_2d: grid_mapping = "Mesh2_crs" ;
   Mesh2_face_depth_2d: location = "face" ;
   Mesh2_face_depth_2d: coordinates = "Mesh2_face_x Mesh2_face_y Mesh2_face_lon Mesh2_face_lat" ;
   Mesh2_face_depth_2d: long_name = "Topographie" ;
   Mesh2_face_depth_2d: standard_name = "sea_floor_depth_below_geoid" ;
   Mesh2_face_depth_2d: cell_methods = "nMesh2_time: mean area: mean" ;
   Mesh2_face_depth_2d: name_id = 17 ;
   Mesh2_face_depth_2d: cell_measures = "area: Mesh2_face_area" ;
   Mesh2_face_depth_2d: units = "m" ;
   Mesh2_face_depth_2d: mesh = "Mesh2" ;
   Mesh2_face_depth_2d: _mossco_filename = "\\Widar\home\ak2stud\Nick\python_scripts\dev\uGrid\data\NSBS\topo.nc" ;
   Mesh2_face_depth_2d: _mossco_varname = "bathymetry" ;

float Mesh2_face_Wasserstand_2d (nMesh2_data_time, nMesh2_face) ;
   Mesh2_face_Wasserstand_2d: _FillValue = 1.e+31 ;
```

Mesh2_face_Wasserstand_2d: grid_mapping = "Mesh2_crs" ;
Mesh2_face_Wasserstand_2d: location = "face" ;
Mesh2_face_Wasserstand_2d: coordinates = "Mesh2_face_x Mesh2_face_y Mesh2_face_lon Mesh2_face_lat" ;
Mesh2_face_Wasserstand_2d: long_name = "Wasserstand, Face (Polygon)" ;
Mesh2_face_Wasserstand_2d: standard_name = "sea_surface_height" ;
Mesh2_face_Wasserstand_2d: cell_methods = "nMesh2_data_time: point area: mean" ;
Mesh2_face_Wasserstand_2d: name_id = 3 ;
Mesh2_face_Wasserstand_2d: cell_measures = "area: Mesh2_face_wet_area" ;
Mesh2_face_Wasserstand_2d: units = "m" ;
Mesh2_face_Wasserstand_2d: mesh = "Mesh2" ;
Mesh2_face_Wasserstand_2d: _mossco_filename = "\\Widar\home\mossco.nc" ;
Mesh2_face_Wasserstand_2d: _mossco_varname = "water_depth_at_soil_surface" ;

float Mesh2_face_Stroemungsgeschwindigkeit_x_2d (nMesh2_data_time, nMesh2_layer_2d, nMesh2_face) ;
Mesh2_face_Stroemungsgeschwindigkeit_x_2d: _FillValue = 1.e+31 ;
Mesh2_face_Stroemungsgeschwindigkeit_x_2d: grid_mapping = "Mesh2_crs" ;
Mesh2_face_Stroemungsgeschwindigkeit_x_2d: coordinates = "Mesh2_face_x Mesh2_face_y Mesh2_face_lon Mesh2_face_lat Mesh2_face_z_2d" ;
Mesh2_face_Stroemungsgeschwindigkeit_x_2d: long_name = "Stroemungsgeschwindigkeit (x-Komponente), Face (Polygon)" ;
Mesh2_face_Stroemungsgeschwindigkeit_x_2d: standard_name = "sea_water_x_velocity" ;
Mesh2_face_Stroemungsgeschwindigkeit_x_2d: cell_methods = "nMesh2_data_time: point nMesh2_layer_2d: mean area: point" ;
Mesh2_face_Stroemungsgeschwindigkeit_x_2d: name_id = 2 ;
Mesh2_face_Stroemungsgeschwindigkeit_x_2d: location = "face" ;
Mesh2_face_Stroemungsgeschwindigkeit_x_2d: units = "m s-1" ;
Mesh2_face_Stroemungsgeschwindigkeit_x_2d: mesh = "Mesh2" ;
Mesh2_face_Stroemungsgeschwindigkeit_x_2d: _mossco_filename = "\\Widar\home\mossco.nc" ;
Mesh2_face_Stroemungsgeschwindigkeit_x_2d: _mossco_varname = "depth_averaged_x_velocity_in_water" ;

float Mesh2_face_Stroemungsgeschwindigkeit_y_2d (nMesh2_data_time, nMesh2_layer_2d, nMesh2_face) ;
Mesh2_face_Stroemungsgeschwindigkeit_y_2d: _FillValue = 1.e+31 ;
Mesh2_face_Stroemungsgeschwindigkeit_y_2d: grid_mapping = "Mesh2_crs" ;
Mesh2_face_Stroemungsgeschwindigkeit_y_2d: coordinates = "Mesh2_face_x Mesh2_face_y Mesh2_face_lon Mesh2_face_lat Mesh2_face_z_2d" ;
Mesh2_face_Stroemungsgeschwindigkeit_y_2d: long_name = "Stroemungsgeschwindigkeit (y-Komponente), Face (Polygon)" ;
Mesh2_face_Stroemungsgeschwindigkeit_y_2d: standard_name = "sea_water_y_velocity" ;
Mesh2_face_Stroemungsgeschwindigkeit_y_2d: cell_methods = "nMesh2_data_time: point nMesh2_layer_2d: mean area: point" ;
Mesh2_face_Stroemungsgeschwindigkeit_y_2d: name_id = 2 ;
Mesh2_face_Stroemungsgeschwindigkeit_y_2d: location = "face" ;
Mesh2_face_Stroemungsgeschwindigkeit_y_2d: units = "m s-1" ;
Mesh2_face_Stroemungsgeschwindigkeit_y_2d: mesh = "Mesh2" ;
Mesh2_face_Stroemungsgeschwindigkeit_y_2d: _mossco_filename = "\\Widar\home\mossco.nc" ;
Mesh2_face_Stroemungsgeschwindigkeit_y_2d: _mossco_varname = "depth_averaged_y_velocity_in_water" ;