ogs5py Documentation

Release 1.3.0

Sebastian Mueller

Apr 15, 2023

CONTENTS

CHAPTER 1

OGS5PY QUICKSTART

ogs5py is A python-API for the OpenGeoSys 5 scientific modeling package.

1.1 Installation

The package can be installed via pip. On Windows you can install WinPython to get Python and pip running.

```
pip install ogs5py
```

Or with conda:

```
conda install ogs5py
```

1.2 Citation

If you are using ogs5py in your publication please cite our paper:

> Müller, S., Zech, A. and Heße, F.: ogs5py: A Python-API for the OpenGeoSys 5 Scientific Modeling Package. Groundwater, 59: 117-122. https://doi.org/10.1111/gwat.13017, 2021.

You can cite the Zenodo code publication of ogs5py by:

> Sebastian Müller. GeoStat-Framework/ogs5py. Zenodo. https://doi.org/10.5281/zenodo.2546767

If you want to cite a specific version, have a look at the Zenodo site.

1.3 Further Information

- General homepage: https://www.opengeosys.org/ogs-5
- OGS5 Repository: https://github.com/ufz/ogs5
- Keyword documentation: https://ogs5-keywords.netlify.com
- OGS5 Benchmarks: https://github.com/ufz/ogs5-benchmarks
- ogs5py Benchmarks: https://github.com/GeoStat-Framework/ogs5py_benchmarks

1.4 Pumping Test Example

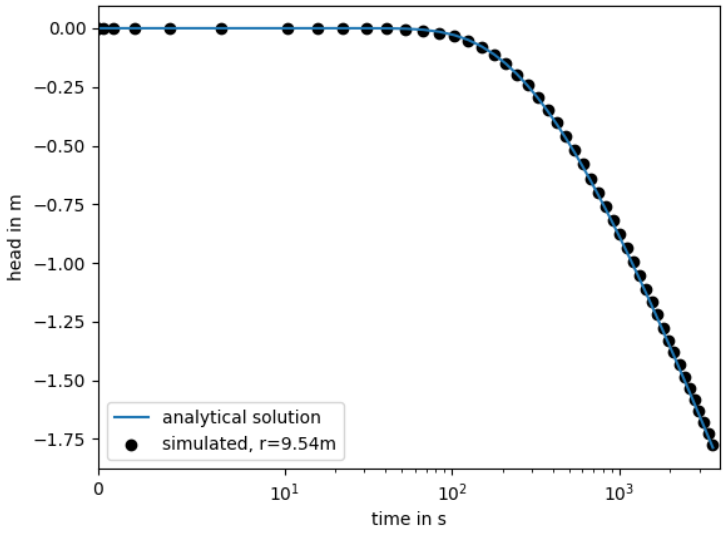
In the following a simple transient pumping test is simulated on a radial symmetric mesh. The point output at the observation well is plotted afterwards.

```
from ogs5py import OGS, specialrange, generate_time
from matplotlib import pyplot as plt

# discretization and parameters
time = specialrange(0, 3600, 50, typ="cub")
rad = specialrange(0, 1000, 100, typ="cub")
obs = rad[21]
angles = 32
storage = 1e-3
transmissivity = 1e-4
rate = -1e-3
# model setup
model = OGS(task_root="pump_test", task_id="model")
# generate a radial mesh and geometry ("boundary" polyline)
model.msh.generate("radial", dim=2, rad=rad, angles=angles)
model.gli.generate("radial", dim=2, rad_out=rad[-1], angles=angles)
model.gli.add_points([0.0, 0.0, 0.0], "pwell")
model.gli.add_points([obs, 0.0, 0.0], "owell")
model.bc.add_block(  # boundary condition
    PCS_TYPE="GROUNDWATER_FLOW",
    PRIMARY_VARIABLE="HEAD",
    GEO_TYPE=["POLYLINE", "boundary"],
    DIS_TYPE=["CONSTANT", 0.0],
)
model.st.add_block(  # source term
    PCS_TYPE="GROUNDWATER_FLOW",
    PRIMARY_VARIABLE="HEAD",
    GEO_TYPE=["POINT", "pwell"],
    DIS_TYPE=["CONSTANT_NEUMANN", rate],
)
model.ic.add_block(  # initial condition
    PCS_TYPE="GROUNDWATER_FLOW",
    PRIMARY_VARIABLE="HEAD",
    GEO_TYPE="DOMAIN",
    DIS_TYPE=["CONSTANT", 0.0],
)
model.mmp.add_block(  # medium properties
    GEOMETRY_DIMENSION=2,
    STORAGE=[1, storage],
    PERMEABILITY_TENSOR=["ISOTROPIC", transmissivity],
```

(continues on next page)

(continued from previous page)

```
)
model.num.add_block(  # numerical solver
    PCS_TYPE="GROUNDWATER_FLOW",
    LINEAR_SOLVER=[2, 5, 1e-14, 1000, 1.0, 100, 4],
)
model.out.add_block(  # point observation
    PCS_TYPE="GROUNDWATER_FLOW",
    NOD_VALUES="HEAD",
    GEO_TYPE=["POINT", "owell"],
    DAT_TYPE="TECPLOT",
)
model.pcs.add_block(  # set the process type
    PCS_TYPE="GROUNDWATER_FLOW", NUM_TYPE="NEW"
)
model.tim.add_block(  # set the timesteps
    PCS_TYPE="GROUNDWATER_FLOW",
    **generate_time(time)
)
model.write_input()
model.run_model()
```



1.5 OGS5 executable

To obtain an OGS5 executable, ogs5py brings a download routine *download_ogs*:

```
from ogs5py import download_ogs
download_ogs()
```

Then a executable is stored in the ogs5py config path and will be called when a model is run.

You can pass a `version` statement to the `download_ogs` routine, to obtain a specific version (5.7, 5.7.1 (win only) and 5.8). For OGS 5.7 there are executables for Windows/Linux and MacOS. For "5.8" there are no MacOS pre-builds.

If you have compiled your own OGS5 version, you can add your executable to the ogs5py config path with *add_exe*:

```
from ogs5py import add_exe
add_exe("path/to/your/ogs/exe")
```

Otherwise you need to specify the path to the executable within the run command:

```
model.run_model(ogs_exe="path/to/ogs")
```

1.6 Requirements

- NumPy >= 1.14.5
- Pandas >= 0.23.2
- meshio >= 4
- lxml >= 4
- pexpect >= 4
- vtk >= 9

1.7 License

MIT

CHAPTER 2

OGS5PY TUTORIALS

In the following you will find several Tutorials on how to use ogs5py to explore its whole beauty and power.

2.1 Tutorial 1: A pumping test

This is a minimal example on how to setup a pumping test with ogs5py. The result was plotted against the analytical solution.

In this example we use the *generate_time* function, to use an array of time points for the time stepping definition.

```
model.tim.add_block(**generate_time(time))
```

is equivalent to:

```
model.tim.add_block(
    TIME_START=0,
    TIME_END=time[-1],
    TIME_STEPS=[
        [1, time[0]],
        [1, time[1]],
        [1, time[2]],
        # ...
    ],
)
```

The script:

```
import anaflow as ana
from ogs5py import OGS, specialrange, generate_time
from matplotlib import pyplot as plt

# discretization and parameters
time = specialrange(0, 3600, 50, typ="cub")
rad = specialrange(0, 1000, 100, typ="cub")
obs = rad[21]
angles = 32
storage = 1e-3
transmissivity = 1e-4
rate = -1e-3
```

(continues on next page)

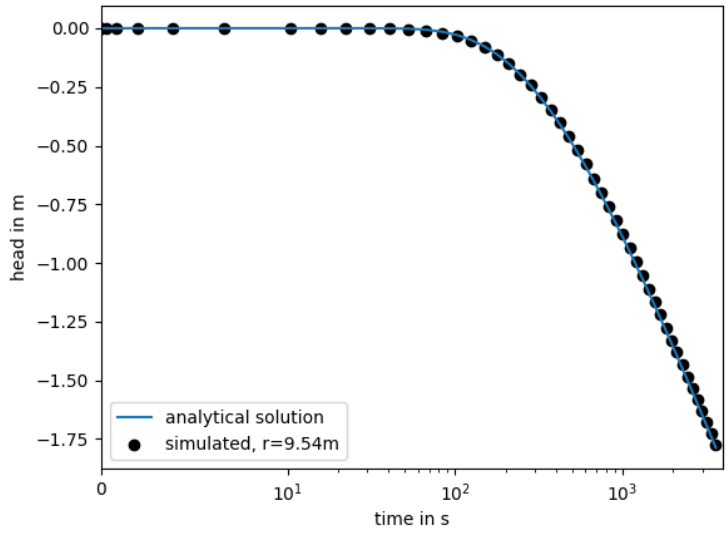
(continued from previous page)

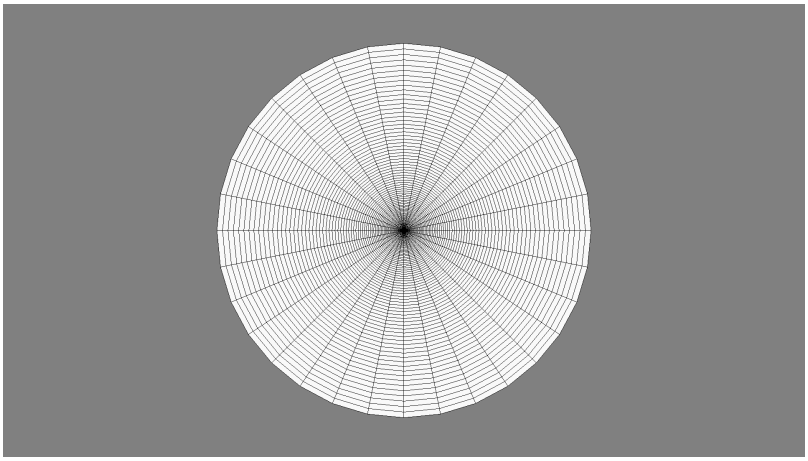
```
# model setup
model = OGS(task_root="pump_test", task_id="model")
model.pcs.add_block(  # set the process type
    PCS_TYPE="GROUNDWATER_FLOW", NUM_TYPE="NEW"
)
# generate a radial mesh and geometry ("boundary" polyline)
model.msh.generate("radial", dim=2, rad=rad, angles=angles)
model.gli.generate("radial", dim=2, rad_out=rad[-1], angles=angles)
model.gli.add_points([0.0, 0.0, 0.0], "pwell")
model.gli.add_points([obs, 0.0, 0.0], "owell")
model.bc.add_block(  # boundary condition
    PCS_TYPE="GROUNDWATER_FLOW",
    PRIMARY_VARIABLE="HEAD",
    GEO_TYPE=["POLYLINE", "boundary"],
    DIS_TYPE=["CONSTANT", 0.0],
)
model.ic.add_block(  # initial condition
    PCS_TYPE="GROUNDWATER_FLOW",
    PRIMARY_VARIABLE="HEAD",
    GEO_TYPE="DOMAIN",
    DIS_TYPE=["CONSTANT", 0.0],
)
model.st.add_block(  # source term
    PCS_TYPE="GROUNDWATER_FLOW",
    PRIMARY_VARIABLE="HEAD",
    GEO_TYPE=["POINT", "pwell"],
    DIS_TYPE=["CONSTANT_NEUMANN", rate],
)
model.mmp.add_block(  # medium properties
    GEOMETRY_DIMENSION=2,
    STORAGE=[1, storage],
    PERMEABILITY_TENSOR=["ISOTROPIC", transmissivity],
)
model.num.add_block(  # numerical solver
    PCS_TYPE="GROUNDWATER_FLOW",
    LINEAR_SOLVER=[2, 5, 1e-14, 1000, 1.0, 100, 4]
)
model.out.add_block(  # point observation
    PCS_TYPE="GROUNDWATER_FLOW",
    NOD_VALUES="HEAD",
    GEO_TYPE=["POINT", "owell"],
    DAT_TYPE="TECPLOT",
)
model.tim.add_block(  # set the timesteps
    PCS_TYPE="GROUNDWATER_FLOW",
    **generate_time(time)  # generate input from time-series
)
model.write_input()
success = model.run_model()
print("success:", success)
# observation
point = model.readtec_point(pcs="GROUNDWATER_FLOW")
time = point["owell"]["TIME"]
head = point["owell"]["HEAD"]
# analytical solution
head_ana = ana.theis(time, obs, storage, transmissivity, rate=rate)
```

(continues on next page)

(continued from previous page)

```
# comparisson plot
plt.scatter(time, head, color="k", label="simulated, r={:04.2f}m".format(obs))
plt.plot(time, head_ana, label="analytical solution")
plt.xscale("symlog", linthreshx=10, subsx=range(1, 10))
plt.xlim([0, 1.1 * time[-1]])
plt.xlabel("time in s")
plt.ylabel("head in m")
plt.legend()
plt.show()
# show mesh
model.msh.show()
```





2.2 Tutorial 2: Interaction with GSTools

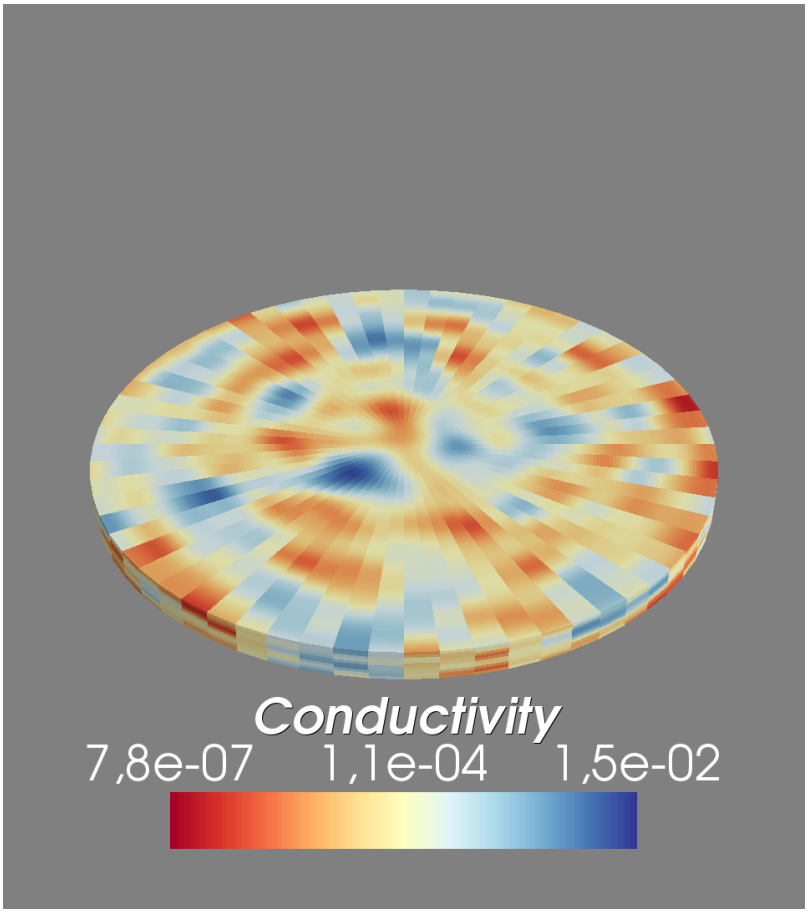
In this example we are generating a log-normal dirtributed conductivity field on a generated mesh with the aid of GSTools. and perform a steady pumping test.

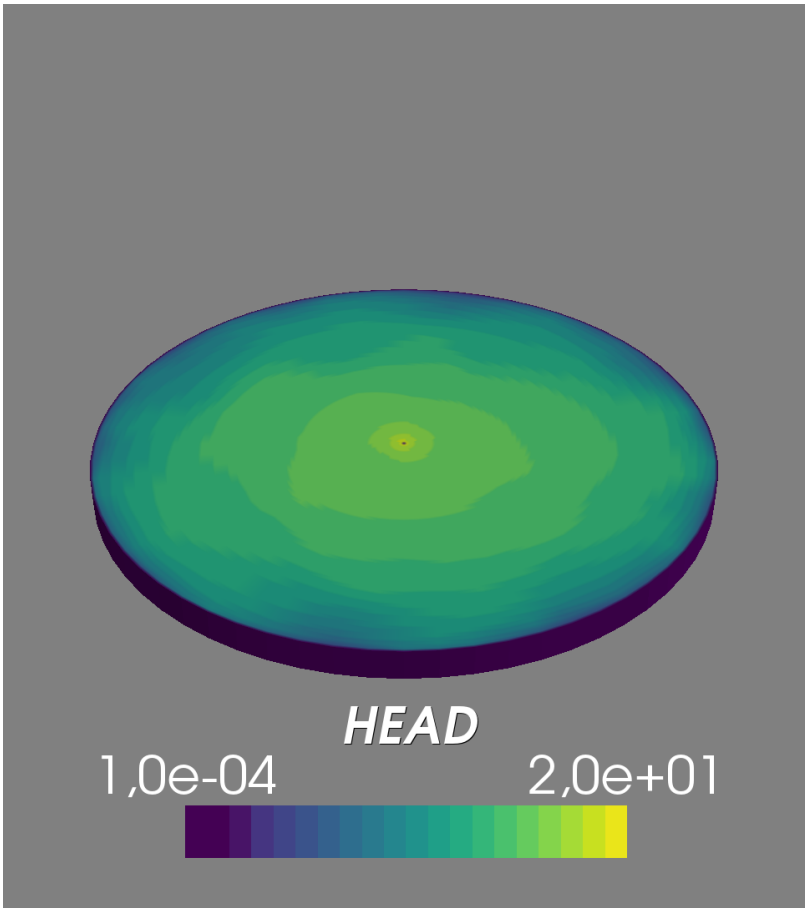
```
import numpy as np
from ogs5py import OGS, by_id, show_vtk
from gstools import SRF, Gaussian

# covariance model for conductivity field
cov_model = Gaussian(dim=3, var=2, len_scale=10, anis=[1, 0.2])
srf = SRF(model=cov_model, mean=-9, seed=1000)
# model setup
model = OGS(task_root="test_het_3D", task_id="model", output_dir="out")
model.pcs.add_block(  # set the process type
    PCS_TYPE="GROUNDWATER_FLOW", NUM_TYPE="NEW", TIM_TYPE="STEADY"
)
# generate a radial 3D mesh and conductivity field
model.msh.generate(
    "radial", dim=3, angles=64, rad=np.arange(101), z_arr=-np.arange(11)
)
cond = np.exp(srf.mesh(model.msh))
model.mpd.add(name="conductivity")
model.mpd.add_block(  # edit recent mpd file
    MSH_TYPE="GROUNDWATER_FLOW",
    MMP_TYPE="PERMEABILITY",
    DIS_TYPE="ELEMENT",
    DATA=by_id(cond),
)
model.mmp.add_block(  # permeability, storage and porosity
    GEOMETRY_DIMENSION=3, PERMEABILITY_DISTRIBUTION=model.mpd.file_name
)
model.gli.generate("radial", dim=3, angles=64, rad_out=100, z_size=-10)
model.gli.add_polyline("pwell", [[0, 0, 0], [0, 0, -10]])
for srf in model.gli.SURFACE_NAMES:  # set boundary condition
    model.bc.add_block(
        PCS_TYPE="GROUNDWATER_FLOW",
        PRIMARY_VARIABLE="HEAD",
        GEO_TYPE=["SURFACE", srf],
        DIS_TYPE=["CONSTANT", 0.0],
    )
model.st.add_block(  # set pumping condition at the pumpingwell
    PCS_TYPE="GROUNDWATER_FLOW",
    PRIMARY_VARIABLE="HEAD",
    GEO_TYPE=["POLYLINE", "pwell"],
    DIS_TYPE=["CONSTANT_NEUMANN", 1.0e-3],
)
model.num.add_block(  # numerical solver
    PCS_TYPE="GROUNDWATER_FLOW",
    LINEAR_SOLVER=[2, 5, 1.0e-14, 1000, 1.0, 100, 4],
)
model.out.add_block(  # set the outputformat
    PCS_TYPE="GROUNDWATER_FLOW",
    NOD_VALUES="HEAD",
    GEO_TYPE="DOMAIN",
    DAT_TYPE="VTK",
)
```

(continues on next page)

(continued from previous page)

```
model.write_input()
success = model.run_model()

model.msh.show(show_cell_data={"Conductivity": cond}, log_scale=True)
files = model.output_files(pcs="GROUNDWATER_FLOW", typ="VTK")
show_vtk(files[-1], log_scale=True)  # show the last time-step
```





2.3 Tutorial 2: Interaction with pygmsh

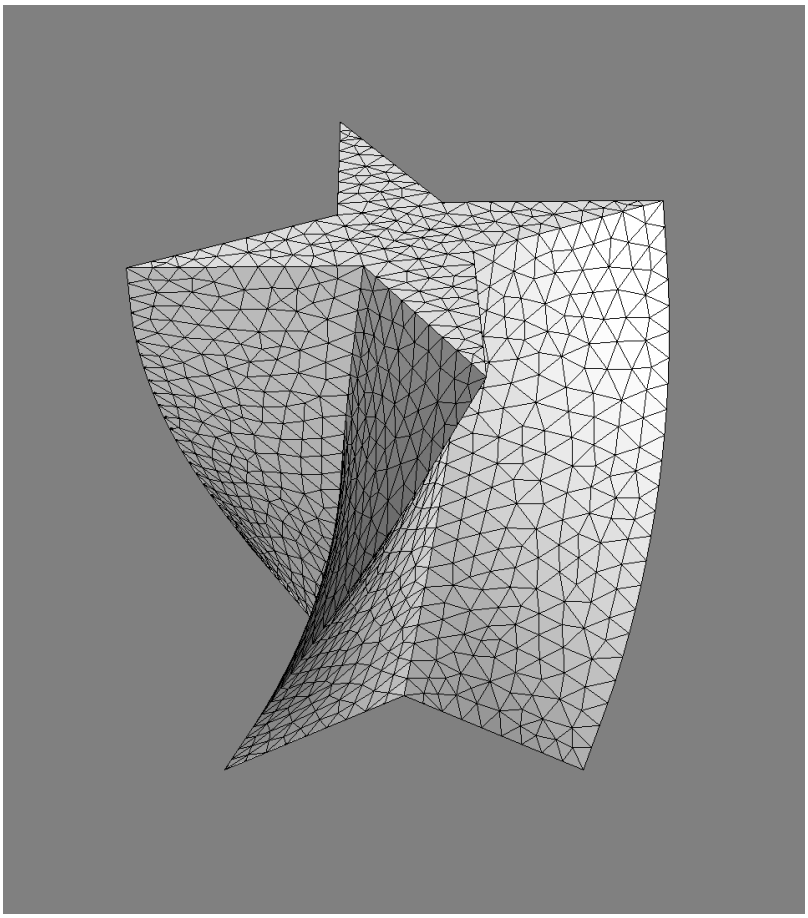
In this example we are generating different meshes with the aid of pygmsh and gmsh

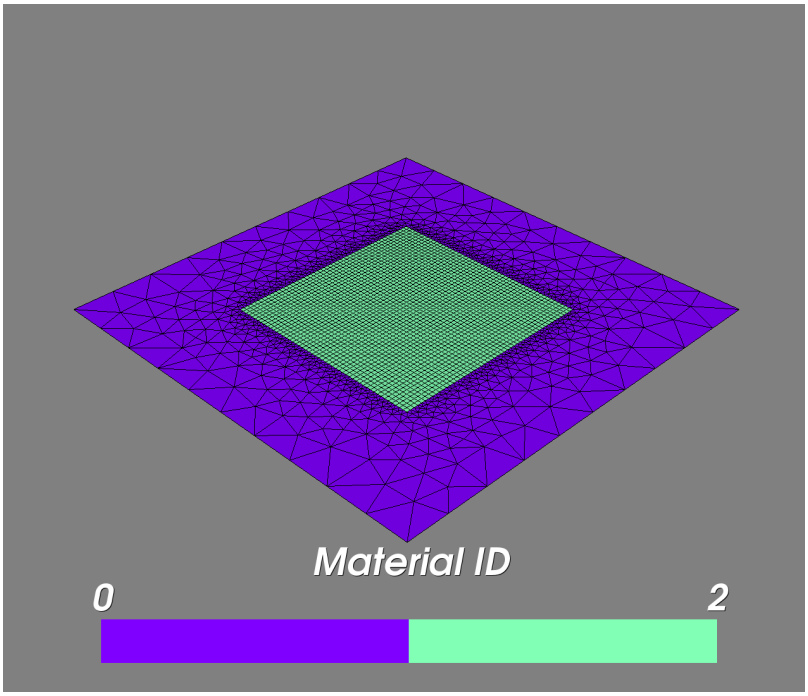
```
import numpy as np
import pygmsh

from ogs5py import OGS

with pygmsh.geo.Geometry() as geom:
    poly = geom.add_polygon(
        [
            [+0.0, +0.5],
            [-0.1, +0.1],
            [-0.5, +0.0],
            [-0.1, -0.1],
            [+0.0, -0.5],
            [+0.1, -0.1],
            [+0.5, +0.0],
            [+0.1, +0.1],
        ],
        mesh_size=0.05,
    )

    geom.twist(
        poly,
```

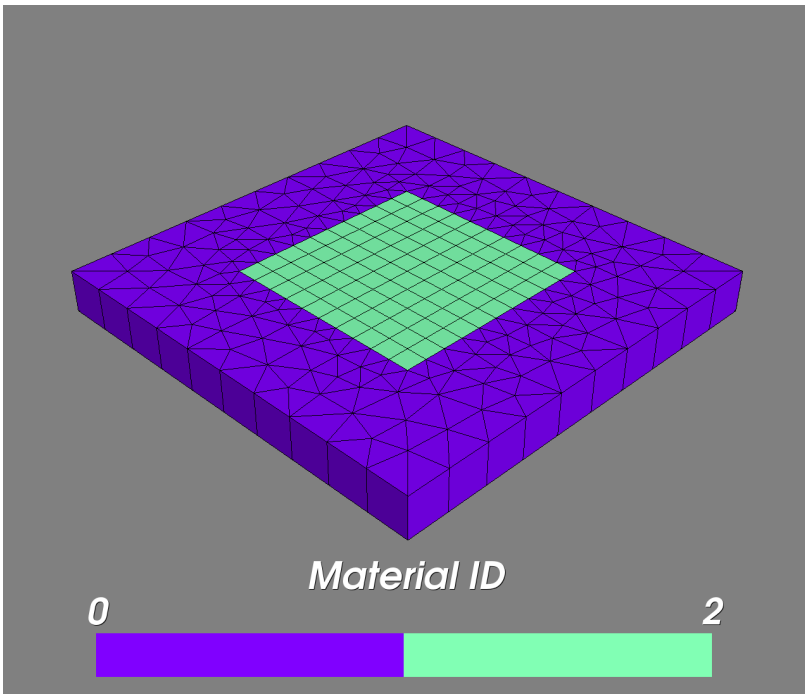
(continues on next page)

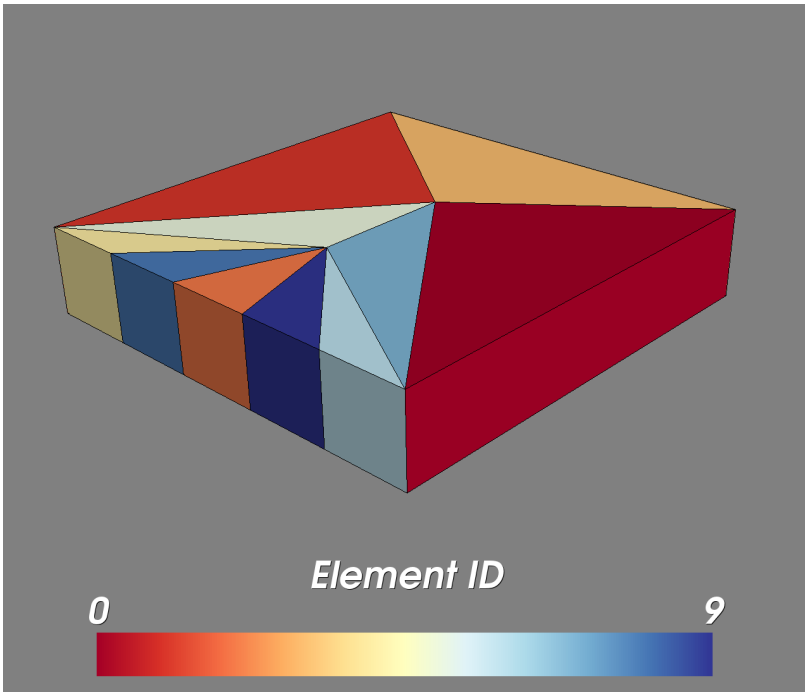
(continued from previous page)

```
        translation_axis=[0, 0, 1],
        rotation_axis=[0, 0, 1],
        point_on_axis=[0, 0, 0],
        angle=np.pi / 3,
    )

    mesh = geom.generate_mesh()

model = OGS()
# generate example above
model.msh.import_mesh(mesh, import_dim=3)
model.msh.show()
# generate a predefined grid adapter in 2D
model.msh.generate("grid_adapter2D", in_mat=1, out_mat=0, fill=True)
model.msh.show(show_material_id=True)
# generate a predefined grid adapter in 3D
model.msh.generate("grid_adapter3D", in_mat=1, out_mat=0, fill=True)
model.msh.show(show_material_id=True)
# generate a predefined block adapter in 3D
model.msh.generate("block_adapter3D", xy_dim=5.0, z_dim=1.0, in_res=1)
model.msh.show(show_element_id=True)
```




Material ID
0
2


Material ID
0
2


Element ID
0
9

CHAPTER 3

OGS5PY API

3.1 Purpose

ogs5py is A python-API for the OpenGeoSys 5 scientific modeling package.

The following functionalities are directly provided on module-level.

3.2 Subpackages

fileclasses	ogs5py subpackage providing the file classes.
reader	ogs5py subpackage providing reader for the ogs5 output.
tools	ogs5py subpackage providing tools.

ogs5py.fileclasses

ogs5py subpackage providing the file classes.

Subpackages

base	Base Classes for the OGS Files.
gli	Class for the ogs GEOMETRY file.
msh	Class for the ogs MESH file.

ogs5py.fileclasses.base

Base Classes for the OGS Files.

File Classes

File([task_root, task_id, file_ext])	File class with minimal functionality.
LineFile([lines, name, file_ext, task_root, ...])	OGS class to handle line-wise text files.
BlockFile([task_root, task_id, file_ext])	OGS Base class to derive all file formats.
MultiFile(base, **standard)	Class holding mulitple files of the same type.

ogs5py.fileclasses.base.File

class ogs5py.fileclasses.base.**File**(*task_root=None*, *task_id='model'*, *file_ext='.std'*)

Bases: object

File class with minimal functionality.

Parameters

- **task_root** (str, optional) – Path to the destiny folder. Default is cwd+"ogs5model"
- **task_id** (str, optional) – Name for the ogs task. Default: "model"
- **file_ext** (str, optional) – extension of the file (with leading dot ".std") Default: ".std"

Attributes

file_name
str: base name of the file with extension.

file_path
str: save path of the file.

force_writing
bool: state if the file is written even if empty.

is_empty
bool: state if the OGS file is empty.

name
str: name of the file without extension.

Methods

add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
check([verbose])	Check if the given file is valid.
del_copy_link()	Remove a former given link to an external file.
get_file_type()	Get the OGS file class name.
read_file(path[, encoding, verbose])	Read an existing file.
reset()	Delete every content.
save(path, **kwargs)	Save the actual file in the given path.
write_file()	Write the actual OGS input file to the given folder.

add_copy_link(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied
- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

`check`(*verbose=True*)

Check if the given file is valid.

Parameters
verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns
result – Validity of the given file.

Return type
bool

`del_copy_link`()

Remove a former given link to an external file.

`get_file_type`()

Get the OGS file class name.

`read_file`(*path*, *encoding=None*, *verbose=False*)

Read an existing file.

`reset`()

Delete every content.

`save`(*path*, ***kwargs*)

Save the actual file in the given path.

Parameters
path (*str*) – path to where to file should be saved

`write_file`()

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

`property file_name`

base name of the file with extension.

Type
str

`property file_path`

save path of the file.

Type
str

`property force_writing`

state if the file is written even if empty.

Type
bool

`property is_empty`

state if the OGS file is empty.

Type
bool

property name

name of the file without extension.

Type

str

ogs5py.fileclasses.base.LineFile

class ogs5py.fileclasses.base.**LineFile**(*lines=None*, *name=None*, *file_ext='.txt'*, *task_root=None*, *task_id='model'*)

Bases: *File*

OGS class to handle line-wise text files.

Parameters

- **lines** (*list of str, optional*) – content of the file as a list of lines Default: None
- **name** (*str, optional*) – name of the file without extension Default: "textfile"
- **file_ext** (*str, optional*) – extension of the file (with leading dot ".txt") Default: ".txt"
- **task_root** (*str, optional*) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (*str, optional*) – Name for the ogs task. (a place holder) Default: "model"

Attributes

file_name
str: base name of the file with extension.

file_path
str: save path of the file.

force_writing
bool: state if the file is written even if empty.

is_empty
bool: state if the file is empty.

name
str: name of the file without extension.

Methods

add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
check([verbose])	Check if the given text-file is valid.
del_copy_link()	Remove a former given link to an external file.
get_file_type()	Get the OGS file class name.
read_file(path[, encoding, verbose])	Read an existing OGS input file.
reset()	Delete every content.
save(path)	Save the actual line-wise file in the given path.
write_file()	Write the actual OGS input file to the given folder.

add_copy_link(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied
- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

`check`(*verbose=True*)

Check if the given text-file is valid.

Parameters
verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns
result – Validity of the given file.

Return type
bool

`del_copy_link`()

Remove a former given link to an external file.

`get_file_type`()

Get the OGS file class name.

`read_file`(*path*, *encoding=None*, *verbose=False*)

Read an existing OGS input file.

Parameters

- **path** (*str*) – path to the existing file that should be read
- **encoding** (*str or None, optional*) – encoding of the given file. If `None` is given, the system standard is used. Default: `None`
- **verbose** (*bool, optional*) – Print information of the reading process. Default: False

`reset`()

Delete every content.

`save`(*path*)

Save the actual line-wise file in the given path.

Parameters
path (*str*) – path to where to file should be saved

`write_file`()

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

`property file_name`

base name of the file with extension.

Type
`str`

`property file_path`

save path of the file.

Type
`str`

`property force_writing`

state if the file is written even if empty.

Type
`bool`

`property is_empty`

state if the file is empty.

Type
`bool`

property name

name of the file without extension.

Type

str

ogs5py.fileclasses.base.BlockFile

class ogs5py.fileclasses.base.**BlockFile**(*task_root=None*, *task_id='model'*, *file_ext='.std'*)

Bases: *File*

OGS Base class to derive all file formats.

Parameters

- **task_root** (*str, optional*) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (*str, optional*) – Name for the ogs task. Default: "model"
- **file_ext** (str, optional) – extension of the file (with leading dot ".std") Default: ".std"

Attributes

block_no
Number of blocks in the file.

file_name
str: base name of the file with extension.

file_path
str: save path of the file.

force_writing
bool: state if the file is written even if empty.

is_empty
State if the OGS file is empty.

name
str: name of the file without extension.

Methods

add_block([index, main_key])	Add a new Block to the actual file.
add_content(content[, main_index, ...])	Add single-line content to the actual file.
add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
add_main_keyword(key[, main_index])	Add a new main keyword (#key) to the actual file.
add_multi_content(content[, main_index, ...])	Add multiple content to the actual file.
add_sub_keyword(key[, main_index, sub_index])	Add a new sub keyword ($key) to the actual file.
append_to_block([index])	Append data to an existing Block in the actual file.
check([verbose])	Check if the given file is valid.
del_block([index, del_all])	Delete a block by its index.
del_content([main_index, sub_index, ...])	Delete content by its position.
del_copy_link()	Remove a former given link to an external file.
del_main_keyword([main_index, del_all])	Delete a main keyword (#key) by its position.
del_sub_keyword([main_index, sub_index, del_all])	Delete a sub keyword ($key) by its position.
get_block([index, as_dict])	Get a Block from the actual file.
get_block_no()	Get the number of blocks in the file.
get_file_type()	Get the OGS file class name.
get_multi_keys([index])	State if a block has a unique set of sub keywords.
is_block_unique([index])	State if a block has a unique set of sub keywords.
read_file(path[, encoding, verbose])	Read an existing OGS input file.
reset()	Delete every content.
save(path, **kwargs)	Save the actual OGS input file in the given path.
update_block([index, main_key])	Update a Block from the actual file.
write_file()	Write the actual OGS input file to the given folder.

add_block(*index=None*, *main_key=None*, ***block*)

Add a new Block to the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 …

$SUBKEY2
content2 …

which looks like the following:

```
FILE.add_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- **main_key** (*string, optional*) – Main keyword of the block that should be added (see: `MKEYS`) Default: the first main keyword of the file-type
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

add_content(*content*, *main_index=None*, *sub_index=None*, *line_index=None*)

Add single-line content to the actual file.

Parameters

- **content** (*list*) – list containing one line of content given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position, where the new line of content should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

If no sub keyword is present, a blank one (“”) will be added and the content is then directly connected to the actual main keyword.

---

add_copy_link(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied
- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

add_main_keyword(*key*, *main_index=None*)

Add a new main keyword (#key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – position, where the new main keyword should be added between the existing ones. As default, it is placed at the end.

add_multi_content(*content*, *main_index=None*, *sub_index=None*)

Add multiple content to the actual file.

Parameters

- **content** (*list*) – list containing lines of content, each given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

The content will be added at the end of the actual subkeyword.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

---

add_sub_keyword(*key*, *main_index=None*, *sub_index=None*)

Add a new sub keyword ($key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – position, where the new sub keyword should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the subkeyword is not added.

---

append_to_block(*index=None*, ***block*)

Append data to an existing Block in the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 …

$SUBKEY2
content2 …

which looks like the following:

```
FILE.append_to_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

`check`(*verbose=True*)

Check if the given file is valid.

Parameters
verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns
result – Validity of the given file.

Return type
bool

`del_block`(*index=None*, *del_all=False*)

Delete a block by its index.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **del_all** (*bool, optional*) – State, if all blocks shall be deleted. Default: False

`del_content`(*main_index=-1*, *sub_index=-1*, *line_index=-1*, *del_all=False*)

Delete content by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be deleted. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position of the content line, that should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all content shall be deleted. Default: False

`del_copy_link`()

Remove a former given link to an external file.

`del_main_keyword`(*main_index=None*, *del_all=False*)

Delete a main keyword (#key) by its position.

Parameters

- **main_index** (*int, optional*) – position, which main keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all main keywords shall be deleted. Default: False

`del_sub_keyword`(*main_index=-1*, *sub_index=-1*, *del_all=False*)

Delete a sub keyword ($key) by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **pos** (*int, optional*) – position, which sub keyword should be deleted. Default: -1

- **del_all** (*bool, optional*) – State, if all sub keywords shall be deleted. Default: False

`get_block`(*index=None*, *as_dict=True*)

Get a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **as_dict** (*bool, optional*) – Here you can state of you want the output as a dictionary, which can be used as key-word-arguments for *add_block*. If False, you get the main-key, a list of sub-keys and a list of content. Default: True

`get_block_no`()

Get the number of blocks in the file.

`get_file_type`()

Get the OGS file class name.

`get_multi_keys`(*index=None*)

State if a block has a unique set of sub keywords.

`is_block_unique`(*index=None*)

State if a block has a unique set of sub keywords.

`read_file`(*path*, *encoding=None*, *verbose=False*)

Read an existing OGS input file.

Parameters

- **path** (*str*) – path to the existing file that should be read
- **encoding** (*str or None, optional*) – encoding of the given file. If `None` is given, the system standard is used. Default: `None`
- **verbose** (*bool, optional*) – Print information of the reading process. Default: False

`reset`()

Delete every content.

`save`(*path*, ***kwargs*)

Save the actual OGS input file in the given path.

Parameters

- **path** (*str*) – path to where to file should be saved
- **update** (*bool, optional*) – state if the content should be updated before saving. Default: True

`update_block`(*index=None*, *main_key=None*, ***block*)

Update a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is used. Default: None
- **main_key** (*string, optional*) – Main keyword of the block that should be updated (see: `MKEYS`) This shouldn't be done. Default: None
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

write_file()

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

MKEYS = []

Main Keywords of this OGS-BlockFile

Type
list

SKEYS = []

Sub Keywords of this OGS-BlockFile

Type
list

STD = {}

Standard Block OGS-BlockFile

Type
dict

property block_no

Number of blocks in the file.

property file_name

base name of the file with extension.

Type
str

property file_path

save path of the file.

Type
str

property force_writing

state if the file is written even if empty.

Type
bool

property is_empty

State if the OGS file is empty.

property name

name of the file without extension.

Type
str

ogs5py.fileclasses.base.MultiFile

class ogs5py.fileclasses.base.**MultiFile**(*base*, ***standard*)

Bases: object

Class holding mulitple files of the same type.

Parameters

- **base** (object) – Base class for the files
- ****standard** – Standard keyword arguments for new instances of the Base class.

Attributes

id
int: The current id to access in the file list.

Methods

add(*args, **kwargs)	Add a new instance of the base class with given args and kwargs.
append(file)	Append a new file to the list.
delete([file_id])	Delete a certain file.
reset_all()	Reset the Multi File.

add(**args*, ***kwargs*)

Add a new instance of the base class with given args and kwargs.

append(*file*)

Append a new file to the list.

delete(*file_id=-1*)

Delete a certain file.

reset_all()

Reset the Multi File.

property id

The current id to access in the file list.

Type
int

ogs5py.fileclasses.gli

Class for the ogs GEOMETRY file.

Subpackages

The generators can be called with *GLI.generate*

generator	Generators for the ogs GEOMETRY file.

ogs5py.fileclasses.gli.generator

Generators for the ogs GEOMETRY file.

Generators

These generators can be called with *GLI.generate*

rectangular([dim, ori, size, name])	Generate a rectangular boundary for a grid in 2D or 3D as gli.
radial([dim, ori, angles, rad_out, rad_in, ...])	Generate a radial boundary for a grid in 2D or 3D.

ogs5py.fileclasses.gli.generator.rectangular

ogs5py.fileclasses.gli.generator.**rectangular**(*dim=2*, *ori=(0.0, 0.0, 0.0)*, *size=(10.0, 10.0, 10.0)*, *name='boundary'*)

Generate a rectangular boundary for a grid in 2D or 3D as gli.

Parameters

- **dim** (*int, optional*) – Dimension of the resulting mesh, either 2 or 3. Default: 3
- **ori** (*list of float, optional*) – Origin of the mesh Default: [0.0, 0.0, 0.0]
- **size** (*list of float, optional*) – Size of the mesh Default: [10.0, 10.0, 10.0]
- **name** (*str, optional*) – Name of the boundary. In 3D there will be 4 surfaces where the names are generated by adding an ID: "_0", "_1", "_2", "_3" Default: "boundary"

Returns
result

Return type
gli

ogs5py.fileclasses.gli.generator.radial

ogs5py.fileclasses.gli.generator.**radial**(*dim=3*, *ori=(0.0, 0.0, 0.0)*, *angles=16*, *rad_out=10.0*, *rad_in=None*, *z_size=-1.0*, *name_out='boundary'*, *name_in='well'*)

Generate a radial boundary for a grid in 2D or 3D.

Parameters

- **dim** (*int, optional*) – Dimension of the resulting mesh, either 2 or 3. Default: 3
- **ori** (*list of float, optional*) – Origin of the mesh Default: [0.0, 0.0, 0.0]
- **angles** (*int, optional*) – Number of angles. Default: 16
- **rad_out** (*float, optional*) – Radius of the outer boundary, Default: 10.
- **rad_out** (*float or None, optional*) – Radius of the inner boundary if needed. (i.e. the well)
- **z_size** (*float, optional*) – size of the mesh in z-direction
- **name_out** (*str, optional*) – Name of the outer boundary. In 3D there will be as many surfaces as angles are given. Their names are generated by adding the angle number: "_0", "_1", … Default: "boundary"
- **name_in** (*str, optional*) – Name of the inner boundary. In 3D there will be as many surfaces as angles are given. Their names are generated by adding the angle number: "_0", "_1", … Default: "well"

Returns
result

Return type
gli

File Classes

GLI([gli_dict])	Class for the ogs GEOMETRY file.
GLIext([typ, data, name, file_ext, ...])	Class for an external definition for the ogs GEOMETRY file.

ogs5py.fileclasses.msh

Class for the ogs MESH file.

Subpackages

The generators can be called with *generate()*

generator	Generators for the ogs MESH file.

ogs5py.fileclasses.msh.generator

Generators for the ogs MESH file.

Generators

These generators can be called with *MSH.generate*

rectangular([dim, mesh_origin, element_no, ...])	Generate a rectangular grid in 2D or 3D.
radial([dim, mesh_origin, angles, rad, z_arr])	Generate a radial grid in 2D or 3D.
grid_adapter2D([out_dim, in_dim, out_res, ...])	Generate a grid adapter.
grid_adapter3D([out_dim, in_dim, z_dim, ...])	Generate a grid adapter.
block_adapter3D([xy_dim, z_dim, in_res])	Generate a block adapter.
gmsh(geo_object[, import_dim])	Generate mesh from gmsh code or gmsh .geo file.

ogs5py.fileclasses.msh.generator.rectangular

ogs5py.fileclasses.msh.generator.**rectangular**(*dim=2*, *mesh_origin=(0.0, 0.0, 0.0)*, *element_no=(10, 10, 10)*, *element_size=(1.0, 1.0, 1.0)*)

Generate a rectangular grid in 2D or 3D.

Parameters

- **dim** (*int, optional*) – Dimension of the resulting mesh, either 2 or 3. Default: 3
- **mesh_origin** (*list of float, optional*) – Origin of the mesh Default: [0.0, 0.0, 0.0]
- **element_no** (*list of int, optional*) – Number of elements in each direction. Default: [10, 10, 10]
- **element_size** (*list of float, optional*) – Size of an element in each direction. Default: [1.0 ,1.0 ,1.0]

Returns

result – Result contains one '#FEM_MSH' block of the OGS mesh file with the following information (sorted by keys):

mesh_data
[dict] dictionary containing information about

- AXISYMMETRY (bool)
- CROSS_SECTION (bool)
- PCS_TYPE (str)
- GEO_TYPE (str)

- GEO_NAME (str)
- LAYER (int)

nodes
[ndarray] Array with all node postions

elements
[dict] contains nodelists for elements sorted by element types

material_id
[dict] contains material ids for each element sorted by element types

element_id
[dict] contains element ids for each element sorted by element types

Return type
dictionary

ogs5py.fileclasses.msh.generator.radial

ogs5py.fileclasses.msh.generator.**radial**(*dim=3*, *mesh_origin=(0.0, 0.0, 0.0)*, *angles=16*, *rad=array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])*, *z_arr=array([0, -1])*)

Generate a radial grid in 2D or 3D.

Parameters

- **dim** (*int, optional*) – Dimension of the resulting mesh, either 2 or 3. Default: 3
- **mesh_origin** (*list of float, optional*) – Origin of the mesh Default: [0.0, 0.0, 0.0]
- **angles** (*int, optional*) – Number of elements in each direction. Default: [10, 10, 10]
- **rad** (*array, optional*) – array of radii to set in the mesh
- **z_arr** (*array, optional*) – array of z values to set the layers in the mesh (only needed for dim=3) needs to be sorted in negative z direction

Returns

result – Result contains one '#FEM_MSH' block of the OGS mesh file with the following information (sorted by keys):

mesh_data
[dict] dictionary containing information about

- AXISYMMETRY (bool)
- CROSS_SECTION (bool)
- PCS_TYPE (str)
- GEO_TYPE (str)
- GEO_NAME (str)
- LAYER (int)

nodes
[ndarray] Array with all node postions

elements
[dict] contains nodelists for elements sorted by element types

material_id
[dict] contains material ids for each element sorted by element types

element_id
[dict] contains element ids for each element sorted by element types

Return type
dictionary

ogs5py.fileclasses.msh.generator.grid_adapter2D

ogs5py.fileclasses.msh.generator.**grid_adapter2D**(*out_dim=(100.0, 100.0)*, *in_dim=(50.0, 50.0)*, *out_res=(10.0, 10.0)*, *in_res=(1.0, 1.0)*, *out_pos=(0.0, 0.0)*, *in_pos=(25.0, 25.0)*, *z_pos=0.0*, *in_mat=0*, *out_mat=0*, *fill=False*)

Generate a grid adapter.

2D adapter from an outer grid resolution to an inner grid resolution with gmsh.

Parameters

- **out_dim** (*list of 2 float*) – xy-Dimension of the outer block
- **in_dim** (*list of 2 float*) – xy-Dimension of the inner block
- **out_res** (*list of 2 float*) – Grid resolution of the outer block
- **in_res** (*list of 2 float*) – Grid resolution of the inner block
- **out_pos** (*list of 2 float*) – xy-Position of the origin of the outer block
- **in_pos** (*list of 2 float*) – xy-Position of the origin of the inner block
- **z_pos** (*float*) – z-Position of the origin of the whole block
- **in_mat** (*integer*) – Material-ID of the inner block
- **out_mat** (*integer*) – Material-ID of the outer block
- **fill** (*bool, optional*) – State if the inner block should be filled with a rectangular mesh. Default: False.

Returns

result – Result contains one '#FEM_MSH' block of the OGS mesh file with the following information (sorted by keys):

mesh_data
[dict] dictionary containing information about

- AXISYMMETRY (bool)
- CROSS_SECTION (bool)
- PCS_TYPE (str)
- GEO_TYPE (str)
- GEO_NAME (str)
- LAYER (int)

nodes
[ndarray] Array with all node postions

elements
[dict] contains nodelists for elements sorted by element types

material_id
[dict] contains material ids for each element sorted by element types

element_id
[dict] contains element ids for each element sorted by element types

Return type
dictionary

ogs5py.fileclasses.msh.generator.grid_adapter3D

ogs5py.fileclasses.msh.generator.**grid_adapter3D**(*out_dim=(100.0, 100.0)*, *in_dim=(50.0, 50.0)*, *z_dim=-10.0*, *out_res=(10.0, 10.0, 10.0)*, *in_res=(5.0, 5.0, 5.0)*, *out_pos=(0.0, 0.0)*, *in_pos=(25.0, 25.0)*, *z_pos=0.0*, *in_mat=0*, *out_mat=0*, *fill=False*)

Generate a grid adapter.

3D adapter from an outer grid resolution to an inner grid resolution with gmsh.

Parameters

- **out_dim** (*list of 2 float*) – xy-Dimension of the outer block
- **in_dim** (*list of 2 float*) – xy-Dimension of the inner block
- **z_dim** (*float*) – z-Dimension of the whole block
- **out_res** (*list of 3 float*) – Grid resolution of the outer block
- **in_res** (*list of 3 float*) – Grid resolution of the inner block
- **out_pos** (*list of 2 float*) – xy-Position of the origin of the outer block
- **in_pos** (*list of 2 float*) – xy-Position of the origin of the inner block
- **z_dim** (*float*) – z-Position of the origin of the whole block
- **in_mat** (*integer*) – Material-ID of the inner block
- **out_mat** (*integer*) – Material-ID of the outer block
- **fill** (*bool, optional*) – State if the inner block should be filled with a rectangular mesh. Default: False.

Returns

result – Result contains one '#FEM_MSH' block of the OGS mesh file with the following information (sorted by keys):

mesh_data
[dict] dictionary containing information about

- AXISYMMETRY (bool)
- CROSS_SECTION (bool)
- PCS_TYPE (str)
- GEO_TYPE (str)
- GEO_NAME (str)
- LAYER (int)

nodes
[ndarray] Array with all node postions

elements
[dict] contains nodelists for elements sorted by element types

material_id
[dict] contains material ids for each element sorted by element types

element_id
[dict] contains element ids for each element sorted by element types

Return type
dictionary

ogs5py.fileclasses.msh.generator.block_adapter3D

ogs5py.fileclasses.msh.generator.**block_adapter3D**(*xy_dim=10.0*, *z_dim=5.0*, *in_res=1.0*)

Generate a block adapter.

It has a given resolution at the southern side with gmsh.

Parameters

- **xy_dim** (*float*) – xy-Dimension of the whole block
- **z_dim** (*float*) – z-Dimension of the whole block
- **in_res** (*float*) – Grid resolution at the southern side of the block

Returns

result – Result contains one '#FEM_MSH' block of the OGS mesh file with the following information (sorted by keys):

mesh_data
[dict] dictionary containing information about

- AXISYMMETRY (bool)
- CROSS_SECTION (bool)
- PCS_TYPE (str)
- GEO_TYPE (str)
- GEO_NAME (str)
- LAYER (int)

nodes
[ndarray] Array with all node postions

elements
[dict] contains nodelists for elements sorted by element types

material_id
[dict] contains material ids for each element sorted by element types

element_id
[dict] contains element ids for each element sorted by element types

Return type
dictionary

ogs5py.fileclasses.msh.generator.gmsh

ogs5py.fileclasses.msh.generator.**gmsh**(*geo_object*, *import_dim=(1, 2, 3)*)

Generate mesh from gmsh code or gmsh .geo file.

Parameters

- **geo_object** (*str or list of str*) – Either path to the gmsh .geo file or list of codelines for a .geo file.
- **import_dim** (*iterable of int or single int, optional*) – State which elements should be imported by dimensionality. Default: (1, 2, 3)

Returns

result – Result contains one '#FEM_MSH' block of the OGS mesh file with the following information (sorted by keys):

mesh_data
[dict] dictionary containing information about

- AXISYMMETRY (bool)
- CROSS_SECTION (bool)
- PCS_TYPE (str)
- GEO_TYPE (str)
- GEO_NAME (str)
- LAYER (int)

nodes
[ndarray] Array with all node postions

elements
[dict] contains nodelists for elements sorted by element types

material_id
[dict] contains material ids for each element sorted by element types

element_id
[dict] contains element ids for each element sorted by element types

Return type
dictionary

File Class

MSH([mesh_list])	Class for a multi layer mesh file that contains multiple '#FEM_MSH' Blocks.

File Classes

Classes for all OGS5 Files

ASC(**OGS_Config)	Class for the ogs ASC file.
BC(**OGS_Config)	Class for the ogs BOUNDARY CONDITION file.
CCT(**OGS_Config)	Class for the ogs COMMUNICATION TABLE file.
DDC(**OGS_Config)	Class for the ogs MPI DOMAIN DECOMPOSITION file.
FCT(**OGS_Config)	Class for the ogs FUNCTION file.
GEM(**OGS_Config)	Class for the ogs GEOCHEMICAL THERMODYNAMIC MODELING COUPLING file.
GEMinit([lst_name, dch, ipm, dbr, ...])	Class for GEMS3K input file.
GLI([gli_dict])	Class for the ogs GEOMETRY file.
GLIext([typ, data, name, file_ext, ...])	Class for an external definition for the ogs GEOMETRY file.
IC(**OGS_Config)	Class for the ogs INITIAL_CONDITION file.
RFR([variables, data, units, headers, name, ...])	Class for the ogs RESTART file, if the DIS_TYPE in IC is set to RESTART.
KRC(**OGS_Config)	Class for the ogs KINETRIC REACTION file.
MCP(**OGS_Config)	Class for the ogs COMPONENT_PROPERTIES file.
MFP(**OGS_Config)	Class for the ogs FLUID PROPERTY file.
MMP(**OGS_Config)	Class for the ogs MEDIUM_PROPERTIES file.
MPD([name, file_ext])	Class for the ogs MEDIUM_PROPERTIES_DISTRIBUTED file.
MSH([mesh_list])	Class for a multi layer mesh file that contains multiple '#FEM_MSH' Blocks.
MSP(**OGS_Config)	Class for the ogs SOLID_PROPERTIES file.
NUM(**OGS_Config)	Class for the ogs NUMERICS file.
OUT(**OGS_Config)	Class for the ogs OUTPUT file.
PCS(**OGS_Config)	Class for the ogs PROCESS file.
PCT([data, s_flag, task_root, task_id])	Class for the ogs Particle file, if the PCS TYPE is RANDOM_WALK.
PQC(**OGS_Config)	Class for the ogs PHREEQC interface file.
PQCdat(**OGS_Config)	Class for the ogs PHREEQC dat file.
REI(**OGS_Config)	Class for the ogs REACTION_INTERFACE file.
RFD(**OGS_Config)	Class for the ogs USER DEFINED TIME CURVES file.
ST(**OGS_Config)	Class for the ogs SOURCE_TERM file.
TIM(**OGS_Config)	Class for the ogs TIME_STEPPING file.

ogs5py.fileclasses.ASC

class ogs5py.fileclasses.**ASC**(***OGS_Config*)

Bases: *LineFile*

Class for the ogs ASC file.

Parameters

- **lines** (*list of str, optional*) – content of the file as a list of lines Default: None
- **name** (*str, optional*) – name of the file without extension Default: "textfile"
- **task_root** (*str, optional*) – Path to the destiny folder. Default: cwd+"ogs5model"
- **task_id** (*str, optional*) – Name for the ogs task. (a place holder) Default: "model"

Notes

This is just handled as a line-wise file. You can access the data by line with:

ASC.lines

This file type comes either from .tim .pcs or .gem

Attributes

file_name
str: base name of the file with extension.

file_path
str: save path of the file.

force_writing
bool: state if the file is written even if empty.

is_empty
bool: state if the file is empty.

name
str: name of the file without extension.

Methods

add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
check([verbose])	Check if the given text-file is valid.
del_copy_link()	Remove a former given link to an external file.
get_file_type()	Get the OGS file class name.
read_file(path[, encoding, verbose])	Read an existing OGS input file.
reset()	Delete every content.
save(path)	Save the actual line-wise file in the given path.
write_file()	Write the actual OGS input file to the given folder.

add_copy_link(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied
- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

check(*verbose=True*)

Check if the given text-file is valid.

Parameters
verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns
result – Validity of the given file.

Return type
bool

del_copy_link()

Remove a former given link to an external file.

get_file_type()

Get the OGS file class name.

read_file(*path*, *encoding=None*, *verbose=False*)

Read an existing OGS input file.

Parameters

- **path** (*str*) – path to the existing file that should be read
- **encoding** (*str or None, optional*) – encoding of the given file. If `None` is given, the system standard is used. Default: `None`
- **verbose** (*bool, optional*) – Print information of the reading process. Default: False

reset()

Delete every content.

save(*path*)

Save the actual line-wise file in the given path.

Parameters

path (*str*) – path to where to file should be saved

write_file()

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

property file_name

base name of the file with extension.

Type

str

property file_path

save path of the file.

Type

str

property force_writing

state if the file is written even if empty.

Type

bool

property is_empty

state if the file is empty.

Type

bool

property name

name of the file without extension.

Type

str

ogs5py.fileclasses.BC

class ogs5py.fileclasses.**BC**(***OGS_Config*)

Bases: *BlockFile*

Class for the ogs BOUNDARY CONDITION file.

Parameters

- **task_root** (*str, optional*) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (*str, optional*) – Name for the ogs task. Default: "model"

Notes

Main-Keywords (#):

- BOUNDARY_CONDITION

Sub-Keywords ($) per Main-Keyword:

- BOUNDARY_CONDITION
  - COMP_NAME
  - CONSTRAINED
  - COPY_VALUE
  - DIS_TYPE
  - DIS_TYPE_CONDITION
  - EPSILON
  - EXCAVATION
  - FCT_TYPE
  - GEO_TYPE
  - MSH_TYPE
  - NO_DISP_INCREMENT
  - PCS_TYPE
  - PRESSURE_AS_HEAD
  - PRIMARY_VARIABLE
  - TIME_CONTROLLED_ACTIVE
  - TIME_INTERVAL
  - TIM_TYPE

Standard block:

PCS_TYPE
"GROUNDWATER_FLOW"

PRIMARY_VARIABLE
"HEAD"

DIS_TYPE
["CONSTANT", 0.0]

GEO_TYPE
["POLYLINE", "boundary"]

Keyword documentation:
https://ogs5-keywords.netlify.com/ogs/wiki/public/doc-auto/by_ext/bc

Reading routines:
https://github.com/ufz/ogs5/blob/master/FEM/rf_bc_new.cpp#L228

See also:

add_block

Attributes

block_no
Number of blocks in the file.

file_name
str: base name of the file with extension.

file_path
str: save path of the file.

force_writing
bool: state if the file is written even if empty.

is_empty
State if the OGS file is empty.

name
str: name of the file without extension.

Methods

add_block([index, main_key])	Add a new Block to the actual file.
add_content(content[, main_index, ...])	Add single-line content to the actual file.
add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
add_main_keyword(key[, main_index])	Add a new main keyword (#key) to the actual file.
add_multi_content(content[, main_index, ...])	Add multiple content to the actual file.
add_sub_keyword(key[, main_index, sub_index])	Add a new sub keyword ($key) to the actual file.
append_to_block([index])	Append data to an existing Block in the actual file.
check([verbose])	Check if the given file is valid.
del_block([index, del_all])	Delete a block by its index.
del_content([main_index, sub_index, ...])	Delete content by its position.
del_copy_link()	Remove a former given link to an external file.
del_main_keyword([main_index, del_all])	Delete a main keyword (#key) by its position.
del_sub_keyword([main_index, sub_index, del_all])	Delete a sub keyword ($key) by its position.
get_block([index, as_dict])	Get a Block from the actual file.
get_block_no()	Get the number of blocks in the file.
get_file_type()	Get the OGS file class name.
get_multi_keys([index])	State if a block has a unique set of sub keywords.
is_block_unique([index])	State if a block has a unique set of sub keywords.
read_file(path[, encoding, verbose])	Read an existing OGS input file.
reset()	Delete every content.
save(path, **kwargs)	Save the actual OGS input file in the given path.
update_block([index, main_key])	Update a Block from the actual file.
write_file()	Write the actual OGS input file to the given folder.

add_block(*index=None*, *main_key=None*, ***block*)

Add a new Block to the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 …

$SUBKEY2
content2 …

which looks like the following:

```
FILE.add_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- **main_key** (*string, optional*) – Main keyword of the block that should be added (see: `MKEYS`) Default: the first main keyword of the file-type
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

add_content(*content*, *main_index=None*, *sub_index=None*, *line_index=None*)

Add single-line content to the actual file.

Parameters

- **content** (*list*) – list containing one line of content given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position, where the new line of content should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

If no sub keyword is present, a blank one (“”) will be added and the content is then directly connected to the actual main keyword.

---

add_copy_link(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied
- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

add_main_keyword(*key*, *main_index=None*)

Add a new main keyword (#key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – position, where the new main keyword should be added between the existing ones. As default, it is placed at the end.

add_multi_content(*content*, *main_index=None*, *sub_index=None*)

Add multiple content to the actual file.

Parameters

- **content** (*list*) – list containing lines of content, each given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

The content will be added at the end of the actual subkeyword.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

---

add_sub_keyword(*key*, *main_index=None*, *sub_index=None*)

Add a new sub keyword ($key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – position, where the new sub keyword should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the subkeyword is not added.

---

append_to_block(*index=None*, ***block*)

Append data to an existing Block in the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 …

$SUBKEY2
content2 …

which looks like the following:

```
FILE.append_to_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

`check`(*verbose=True*)

Check if the given file is valid.

Parameters
verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns
result – Validity of the given file.

Return type
bool

`del_block`(*index=None*, *del_all=False*)

Delete a block by its index.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **del_all** (*bool, optional*) – State, if all blocks shall be deleted. Default: False

`del_content`(*main_index=-1*, *sub_index=-1*, *line_index=-1*, *del_all=False*)

Delete content by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be deleted. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position of the content line, that should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all content shall be deleted. Default: False

`del_copy_link`()

Remove a former given link to an external file.

`del_main_keyword`(*main_index=None*, *del_all=False*)

Delete a main keyword (#key) by its position.

Parameters

- **main_index** (*int, optional*) – position, which main keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all main keywords shall be deleted. Default: False

`del_sub_keyword`(*main_index=-1*, *sub_index=-1*, *del_all=False*)

Delete a sub keyword ($key) by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **pos** (*int, optional*) – position, which sub keyword should be deleted. Default: -1

- **del_all** (*bool, optional*) – State, if all sub keywords shall be deleted. Default: False

`get_block`(*index=None*, *as_dict=True*)

Get a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **as_dict** (*bool, optional*) – Here you can state of you want the output as a dictionary, which can be used as key-word-arguments for *add_block*. If False, you get the main-key, a list of sub-keys and a list of content. Default: True

`get_block_no`()

Get the number of blocks in the file.

`get_file_type`()

Get the OGS file class name.

`get_multi_keys`(*index=None*)

State if a block has a unique set of sub keywords.

`is_block_unique`(*index=None*)

State if a block has a unique set of sub keywords.

`read_file`(*path*, *encoding=None*, *verbose=False*)

Read an existing OGS input file.

Parameters

- **path** (*str*) – path to the existing file that should be read
- **encoding** (*str or None, optional*) – encoding of the given file. If `None` is given, the system standard is used. Default: `None`
- **verbose** (*bool, optional*) – Print information of the reading process. Default: False

`reset`()

Delete every content.

`save`(*path*, ***kwargs*)

Save the actual OGS input file in the given path.

Parameters

- **path** (*str*) – path to where to file should be saved
- **update** (*bool, optional*) – state if the content should be updated before saving. Default: True

`update_block`(*index=None*, *main_key=None*, ***block*)

Update a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is used. Default: None
- **main_key** (*string, optional*) – Main keyword of the block that should be updated (see: `MKEYS`) This shouldn't be done. Default: None
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

write_file()

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

MKEYS = ['BOUNDARY_CONDITION']

Main Keywords of this OGS-BlockFile

Type
list

SKEYS = [['PCS_TYPE', 'PRIMARY_VARIABLE', 'COMP_NAME', 'GEO_TYPE', 'DIS_TYPE', 'TIM_TYPE', 'FCT_TYPE', 'MSH_TYPE', 'DIS_TYPE_CONDITION', 'EPSILON', 'TIME_CONTROLLED_ACTIVE', 'TIME_INTERVAL', 'EXCAVATION', 'NO_DISP_INCREMENT', 'COPY_VALUE', 'PRESSURE_AS_HEAD', 'CONSTRAINED']]

Sub Keywords of this OGS-BlockFile

Type
list

STD = {'DIS_TYPE': ['CONSTANT', 0.0], 'GEO_TYPE': ['POLYLINE', 'boundary'], 'PCS_TYPE': 'GROUNDWATER_FLOW', 'PRIMARY_VARIABLE': 'HEAD'}

Standard Block OGS-BlockFile

Type
dict

property block_no

Number of blocks in the file.

property file_name

base name of the file with extension.

Type
str

property file_path

save path of the file.

Type
str

property force_writing

state if the file is written even if empty.

Type
bool

property is_empty

State if the OGS file is empty.

property name

name of the file without extension.

Type
str

ogs5py.fileclasses.CCT

class ogs5py.fileclasses.**CCT**(***OGS_Config*)

Bases: *BlockFile*

Class for the ogs COMMUNICATION TABLE file.

Parameters

- **task_root** (*str, optional*) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (*str, optional*) – Name for the ogs task. Default: "model"

Notes

Main-Keywords (#):

- COMMUNICATION_TABLE

Sub-Keywords ($) per Main-Keyword:

- COMMUNICATION_TABLE
  - MYRANK
  - NEIGHBOR
  - NNEIGHBORS

Standard block:
None

Keyword documentation:
https://ogs5-keywords.netlify.com/ogs/wiki/public/doc-auto/by_ext/cct

Reading routines:
https://github.com/ufz/ogs5/blob/master/FEM/fct_mpi.cpp#L27

See also:

add_block

Attributes

block_no
Number of blocks in the file.

file_name
str: base name of the file with extension.

file_path
str: save path of the file.

force_writing
bool: state if the file is written even if empty.

is_empty
State if the OGS file is empty.

name
str: name of the file without extension.

Methods

add_block([index, main_key])	Add a new Block to the actual file.
add_content(content[, main_index, ...])	Add single-line content to the actual file.
add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
add_main_keyword(key[, main_index])	Add a new main keyword (#key) to the actual file.
add_multi_content(content[, main_index, ...])	Add multiple content to the actual file.
add_sub_keyword(key[, main_index, sub_index])	Add a new sub keyword ($key) to the actual file.
append_to_block([index])	Append data to an existing Block in the actual file.
check([verbose])	Check if the given file is valid.
del_block([index, del_all])	Delete a block by its index.
del_content([main_index, sub_index, ...])	Delete content by its position.
del_copy_link()	Remove a former given link to an external file.
del_main_keyword([main_index, del_all])	Delete a main keyword (#key) by its position.
del_sub_keyword([main_index, sub_index, del_all])	Delete a sub keyword ($key) by its position.
get_block([index, as_dict])	Get a Block from the actual file.
get_block_no()	Get the number of blocks in the file.
get_file_type()	Get the OGS file class name.
get_multi_keys([index])	State if a block has a unique set of sub keywords.
is_block_unique([index])	State if a block has a unique set of sub keywords.
read_file(path[, encoding, verbose])	Read an existing OGS input file.
reset()	Delete every content.
save(path, **kwargs)	Save the actual OGS input file in the given path.
update_block([index, main_key])	Update a Block from the actual file.
write_file()	Write the actual OGS input file to the given folder.

add_block(*index=None*, *main_key=None*, ***block*)

Add a new Block to the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 …

$SUBKEY2
content2 …

which looks like the following:

```
FILE.add_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- **main_key** (*string, optional*) – Main keyword of the block that should be added (see: `MKEYS`) Default: the first main keyword of the file-type
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

add_content(*content*, *main_index=None*, *sub_index=None*, *line_index=None*)

Add single-line content to the actual file.

Parameters

- **content** (*list*) – list containing one line of content given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position, where the new line of content should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

`add_copy_link`(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied
- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

`add_main_keyword`(*key*, *main_index=None*)

Add a new main keyword (#key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – position, where the new main keyword should be added between the existing ones. As default, it is placed at the end.

`add_multi_content`(*content*, *main_index=None*, *sub_index=None*)

Add multiple content to the actual file.

Parameters

- **content** (*list*) – list containing lines of content, each given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

The content will be added at the end of the actual subkeyword.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

add_sub_keyword(*key*, *main_index=None*, *sub_index=None*)

Add a new sub keyword ($key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – position, where the new sub keyword should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the subkeyword is not added.

append_to_block(*index=None*, ***block*)

Append data to an existing Block in the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 …

$SUBKEY2
content2 …

which looks like the following:

```
FILE.append_to_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

check(*verbose=True*)

Check if the given file is valid.

Parameters
verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns
result – Validity of the given file.

Return type
bool

del_block(*index=None*, *del_all=False*)

Delete a block by its index.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **del_all** (*bool, optional*) – State, if all blocks shall be deleted. Default: False

del_content(*main_index=-1*, *sub_index=-1*, *line_index=-1*, *del_all=False*)

Delete content by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be deleted. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position of the content line, that should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all content shall be deleted. Default: False

del_copy_link()

Remove a former given link to an external file.

del_main_keyword(*main_index=None*, *del_all=False*)

Delete a main keyword (#key) by its position.

Parameters

- **main_index** (*int, optional*) – position, which main keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all main keywords shall be deleted. Default: False

del_sub_keyword(*main_index=-1*, *sub_index=-1*, *del_all=False*)

Delete a sub keyword ($key) by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **pos** (*int, optional*) – position, which sub keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all sub keywords shall be deleted. Default: False

get_block(*index=None*, *as_dict=True*)

Get a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **as_dict** (*bool, optional*) – Here you can state of you want the output as a dictionary, which can be used as key-word-arguments for *add_block*. If False, you get the main-key, a list of sub-keys and a list of content. Default: True

get_block_no()

Get the number of blocks in the file.

get_file_type()

Get the OGS file class name.

get_multi_keys(*index=None*)

State if a block has a unique set of sub keywords.

is_block_unique(*index=None*)

State if a block has a unique set of sub keywords.

read_file(*path*, *encoding=None*, *verbose=False*)

Read an existing OGS input file.

Parameters

- **path** (*str*) – path to the existing file that should be read
- **encoding** (*str or None, optional*) – encoding of the given file. If `None` is given, the system standard is used. Default: `None`
- **verbose** (*bool, optional*) – Print information of the reading process. Default: False

reset()

Delete every content.

save(*path*, ***kwargs*)

Save the actual OGS input file in the given path.

Parameters

- **path** (*str*) – path to where to file should be saved
- **update** (*bool, optional*) – state if the content should be updated before saving. Default: True

update_block(*index=None*, *main_key=None*, ***block*)

Update a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is used. Default: None
- **main_key** (*string, optional*) – Main keyword of the block that should be updated (see: `MKEYS`) This shouldn't be done. Default: None
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

write_file()

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

MKEYS = ['COMMUNICATION_TABLE']

Main Keywords of this OGS-BlockFile

Type
list

SKEYS = [['MYRANK', 'NNEIGHBORS', 'NEIGHBOR']]

Sub Keywords of this OGS-BlockFile

Type
list

STD = {}

Standard Block OGS-BlockFile

Type
dict

property block_no

Number of blocks in the file.

property file_name
: base name of the file with extension.

  Type
  : str

property file_path
: save path of the file.

  Type
  : str

property force_writing
: state if the file is written even if empty.

  Type
  : bool

property is_empty
: State if the OGS file is empty.

property name
: name of the file without extension.

  Type
  : str

ogs5py.fileclasses.DDC

class ogs5py.fileclasses.**DDC**(***OGS_Config*)

Bases: *BlockFile*

Class for the ogs MPI DOMAIN DECOMPOSITION file.

Parameters

- **task_root** (*str, optional*) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (*str, optional*) – Name for the ogs task. Default: "model"

Notes

Main-Keywords (#):

- DOMAIN

Sub-Keywords ($) per Main-Keyword:

- DOMAIN
  - ELEMENTS
  - NODES_INNER
  - NODES_BORDER

Standard block:
None

Keyword documentation:
None

Reading routines:
https://github.com/ufz/ogs5/blob/master/FEM/par_ddc.cpp

See also:

add_block

Attributes

block_no
Number of blocks in the file.

file_name
`str`: base name of the file with extension.

file_path
`str`: save path of the file.

force_writing
`bool`: state if the file is written even if empty.

is_empty
State if the OGS file is empty.

name
`str`: name of the file without extension.

Methods

add_block([index, main_key])	Add a new Block to the actual file.
add_content(content[, main_index, ...])	Add single-line content to the actual file.
add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
add_main_keyword(key[, main_index])	Add a new main keyword (#key) to the actual file.
add_multi_content(content[, main_index, ...])	Add multiple content to the actual file.
add_sub_keyword(key[, main_index, sub_index])	Add a new sub keyword ($key) to the actual file.
append_to_block([index])	Append data to an existing Block in the actual file.
check([verbose])	Check if the given file is valid.
del_block([index, del_all])	Delete a block by its index.
del_content([main_index, sub_index, ...])	Delete content by its position.
del_copy_link()	Remove a former given link to an external file.
del_main_keyword([main_index, del_all])	Delete a main keyword (#key) by its position.
del_sub_keyword([main_index, sub_index, del_all])	Delete a sub keyword ($key) by its position.
get_block([index, as_dict])	Get a Block from the actual file.
get_block_no()	Get the number of blocks in the file.
get_file_type()	Get the OGS file class name.
get_multi_keys([index])	State if a block has a unique set of sub keywords.
is_block_unique([index])	State if a block has a unique set of sub keywords.
read_file(path[, encoding, verbose])	Read an existing OGS input file.
reset()	Delete every content.
save(path, **kwargs)	Save the actual DDC input file in the given path.
update_block([index, main_key])	Update a Block from the actual file.
write_file()	Write the actual OGS input file to the given folder.

add_block(*index=None*, *main_key=None*, ***block*)

Add a new Block to the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 …

$SUBKEY2
content2 …

which looks like the following:

```
FILE.add_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- **main_key** (*string, optional*) – Main keyword of the block that should be added (see: `MKEYS`) Default: the first main keyword of the file-type
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

add_content(*content*, *main_index=None*, *sub_index=None*, *line_index=None*)

Add single-line content to the actual file.

Parameters

- **content** (*list*) – list containing one line of content given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position, where the new line of content should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

`add_copy_link`(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied
- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

`add_main_keyword`(*key*, *main_index=None*)

Add a new main keyword (#key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – position, where the new main keyword should be added between the existing ones. As default, it is placed at the end.

`add_multi_content`(*content*, *main_index=None*, *sub_index=None*)

Add multiple content to the actual file.

Parameters

- **content** (*list*) – list containing lines of content, each given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

The content will be added at the end of the actual subkeyword.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

`add_sub_keyword`(*key*, *main_index=None*, *sub_index=None*)

Add a new sub keyword ($key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – position, where the new sub keyword should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the subkeyword is not added.

`append_to_block`(*index=None*, ***block*)

Append data to an existing Block in the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 …

$SUBKEY2
content2 …

which looks like the following:

```
FILE.append_to_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

`check`(*verbose=True*)

Check if the given file is valid.

Parameters
verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns
result – Validity of the given file.

Return type
bool

`del_block`(*index=None*, *del_all=False*)

Delete a block by its index.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **del_all** (*bool, optional*) – State, if all blocks shall be deleted. Default: False

del_content(*main_index=-1*, *sub_index=-1*, *line_index=-1*, *del_all=False*)

Delete content by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be deleted. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position of the content line, that should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all content shall be deleted. Default: False

del_copy_link()

Remove a former given link to an external file.

del_main_keyword(*main_index=None*, *del_all=False*)

Delete a main keyword (#key) by its position.

Parameters

- **main_index** (*int, optional*) – position, which main keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all main keywords shall be deleted. Default: False

del_sub_keyword(*main_index=-1*, *sub_index=-1*, *del_all=False*)

Delete a sub keyword ($key) by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **pos** (*int, optional*) – position, which sub keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all sub keywords shall be deleted. Default: False

get_block(*index=None*, *as_dict=True*)

Get a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **as_dict** (*bool, optional*) – Here you can state of you want the output as a dictionary, which can be used as key-word-arguments for *add_block*. If False, you get the main-key, a list of sub-keys and a list of content. Default: True

get_block_no()

Get the number of blocks in the file.

get_file_type()

Get the OGS file class name.

get_multi_keys(*index=None*)

State if a block has a unique set of sub keywords.

is_block_unique(*index=None*)

State if a block has a unique set of sub keywords.

`read_file`(*path*, *encoding=None*, *verbose=False*)

Read an existing OGS input file.

Parameters

- **path** (*str*) – path to the existing file that should be read
- **encoding** (*str or None, optional*) – encoding of the given file. If `None` is given, the system standard is used. Default: `None`
- **verbose** (*bool, optional*) – Print information of the reading process. Default: False

`reset`()

Delete every content.

`save`(*path*, ***kwargs*)

Save the actual DDC input file in the given path.

Parameters

- **path** (*str*) – path to where to file should be saved
- **update** (*bool, optional*) – state if the content should be updated before saving. Default: True

`update_block`(*index=None*, *main_key=None*, ***block*)

Update a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is used. Default: None
- **main_key** (*string, optional*) – Main keyword of the block that should be updated (see: `MKEYS`) This shouldn't be done. Default: None
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

`write_file`()

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

`MKEYS = ['DOMAIN']`

Main Keywords of this OGS-BlockFile

Type
list

`SKEYS = [['ELEMENTS', 'NODES_INNER', 'NODES_BORDER']]`

Sub Keywords of this OGS-BlockFile

Type
list

`STD = {}`

Standard Block OGS-BlockFile

Type
dict

`property block_no`

Number of blocks in the file.

property file_name

base name of the file with extension.

Type
str

property file_path

save path of the file.

Type
str

property force_writing

state if the file is written even if empty.

Type
bool

property is_empty

State if the OGS file is empty.

property name

name of the file without extension.

Type
str

ogs5py.fileclasses.FCT

class ogs5py.fileclasses.**FCT**(***OGS_Config*)

Bases: *BlockFile*

Class for the ogs FUNCTION file.

Parameters

- **task_root** (*str, optional*) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (*str, optional*) – Name for the ogs task. Default: "model"

Notes

Main-Keywords (#):

- FUNCTION

Sub-Keywords ($) per Main-Keyword:

- FUNCTION
  - DATA
  - DIMENSION
  - DIS_TYPE
  - GEO_TYPE
  - MATRIX
  - TYPE
  - VARIABLES

Standard block:
None

Keyword documentation:
https://ogs5-keywords.netlify.com/ogs/wiki/public/doc-auto/by_ext/fct

Reading routines:
https://github.com/ufz/ogs5/blob/master/FEM/rf_fct.cpp#L82

See also:

add_block

Attributes

block_no
Number of blocks in the file.

file_name
str: base name of the file with extension.

file_path
str: save path of the file.

force_writing
bool: state if the file is written even if empty.

is_empty
State if the OGS file is empty.

name
str: name of the file without extension.

Methods

add_block([index, main_key])	Add a new Block to the actual file.
add_content(content[, main_index, ...])	Add single-line content to the actual file.
add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
add_main_keyword(key[, main_index])	Add a new main keyword (#key) to the actual file.
add_multi_content(content[, main_index, ...])	Add multiple content to the actual file.
add_sub_keyword(key[, main_index, sub_index])	Add a new sub keyword ($key) to the actual file.
append_to_block([index])	Append data to an existing Block in the actual file.
check([verbose])	Check if the given file is valid.
del_block([index, del_all])	Delete a block by its index.
del_content([main_index, sub_index, ...])	Delete content by its position.
del_copy_link()	Remove a former given link to an external file.
del_main_keyword([main_index, del_all])	Delete a main keyword (#key) by its position.
del_sub_keyword([main_index, sub_index, del_all])	Delete a sub keyword ($key) by its position.
get_block([index, as_dict])	Get a Block from the actual file.
get_block_no()	Get the number of blocks in the file.
get_file_type()	Get the OGS file class name.
get_multi_keys([index])	State if a block has a unique set of sub keywords.
is_block_unique([index])	State if a block has a unique set of sub keywords.
read_file(path[, encoding, verbose])	Read an existing OGS input file.
reset()	Delete every content.
save(path, **kwargs)	Save the actual OGS input file in the given path.
update_block([index, main_key])	Update a Block from the actual file.
write_file()	Write the actual OGS input file to the given folder.

add_block(*index=None*, *main_key=None*, ***block*)

Add a new Block to the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 …

$SUBKEY2
content2 …

which looks like the following:

```
FILE.add_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- **main_key** (*string, optional*) – Main keyword of the block that should be added (see: `MKEYS`) Default: the first main keyword of the file-type
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

add_content(*content*, *main_index=None*, *sub_index=None*, *line_index=None*)

Add single-line content to the actual file.

Parameters

- **content** (*list*) – list containing one line of content given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position, where the new line of content should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

`add_copy_link`(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied
- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

`add_main_keyword`(*key*, *main_index=None*)

Add a new main keyword (#key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – position, where the new main keyword should be added between the existing ones. As default, it is placed at the end.

`add_multi_content`(*content*, *main_index=None*, *sub_index=None*)

Add multiple content to the actual file.

Parameters

- **content** (*list*) – list containing lines of content, each given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

The content will be added at the end of the actual subkeyword.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

`add_sub_keyword`(*key*, *main_index=None*, *sub_index=None*)

Add a new sub keyword ($key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – position, where the new sub keyword should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the subkeyword is not added.

`append_to_block`(*index=None*, ***block*)

Append data to an existing Block in the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 …

$SUBKEY2
content2 …

which looks like the following:

```
FILE.append_to_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

`check`(*verbose=True*)

Check if the given file is valid.

Parameters
verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns
result – Validity of the given file.

Return type
bool

`del_block`(*index=None*, *del_all=False*)

Delete a block by its index.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **del_all** (*bool, optional*) – State, if all blocks shall be deleted. Default: False

del_content(*main_index=-1*, *sub_index=-1*, *line_index=-1*, *del_all=False*)

Delete content by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be deleted. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position of the content line, that should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all content shall be deleted. Default: False

del_copy_link()

Remove a former given link to an external file.

del_main_keyword(*main_index=None*, *del_all=False*)

Delete a main keyword (#key) by its position.

Parameters

- **main_index** (*int, optional*) – position, which main keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all main keywords shall be deleted. Default: False

del_sub_keyword(*main_index=-1*, *sub_index=-1*, *del_all=False*)

Delete a sub keyword ($key) by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **pos** (*int, optional*) – position, which sub keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all sub keywords shall be deleted. Default: False

get_block(*index=None*, *as_dict=True*)

Get a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **as_dict** (*bool, optional*) – Here you can state of you want the output as a dictionary, which can be used as key-word-arguments for *add_block*. If False, you get the main-key, a list of sub-keys and a list of content. Default: True

get_block_no()

Get the number of blocks in the file.

get_file_type()

Get the OGS file class name.

get_multi_keys(*index=None*)

State if a block has a unique set of sub keywords.

is_block_unique(*index=None*)

State if a block has a unique set of sub keywords.

`read_file`(*path*, *encoding=None*, *verbose=False*)

Read an existing OGS input file.

Parameters

- **path** (*str*) – path to the existing file that should be read
- **encoding** (*str or None, optional*) – encoding of the given file. If `None` is given, the system standard is used. Default: `None`
- **verbose** (*bool, optional*) – Print information of the reading process. Default: False

`reset`()

Delete every content.

`save`(*path*, ***kwargs*)

Save the actual OGS input file in the given path.

Parameters

- **path** (*str*) – path to where to file should be saved
- **update** (*bool, optional*) – state if the content should be updated before saving. Default: True

`update_block`(*index=None*, *main_key=None*, ***block*)

Update a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is used. Default: None
- **main_key** (*string, optional*) – Main keyword of the block that should be updated (see: `MKEYS`) This shouldn't be done. Default: None
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

`write_file`()

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

`MKEYS = ['FUNCTION']`

Main Keywords of this OGS-BlockFile

Type
`list`

`SKEYS = [['TYPE', 'GEO_TYPE', 'DIS_TYPE', 'VARIABLES', 'DIMENSION', 'MATRIX', 'DATA']]`

Sub Keywords of this OGS-BlockFile

Type
`list`

`STD = {}`

Standard Block OGS-BlockFile

Type
`dict`

`property block_no`

Number of blocks in the file.

property file_name
: base name of the file with extension.

  Type
  : str

property file_path
: save path of the file.

  Type
  : str

property force_writing
: state if the file is written even if empty.

  Type
  : bool

property is_empty
: State if the OGS file is empty.

property name
: name of the file without extension.

  Type
  : str

ogs5py.fileclasses.GEM

class ogs5py.fileclasses.**GEM**(***OGS_Config*)

Bases: *BlockFile*

Class for the ogs GEOCHEMICAL THERMODYNAMIC MODELING COUPLING file.

Parameters

- **task_root** (*str, optional*) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (*str, optional*) – Name for the ogs task. Default: "model"

Notes

Main-Keywords (#):

- GEM_PROPERTIES

Sub-Keywords ($) per Main-Keyword:

- GEM_PROPERTIES
  - CALCULATE_BOUNDARY_NODES
  - DISABLE_GEMS
  - FLAG_COUPLING_HYDROLOGY
  - FLAG_DISABLE_GEM
  - FLAG_POROSITY_CHANGE
  - GEM_CALCULATE_BOUNDARY_NODES
  - GEM_INIT_FILE
  - GEM_THREADS
  - ITERATIVE_SCHEME
  - KINETIC_GEM
  - MAX_FAILED_NODES
  - MAX_POROSITY
  - MIN_POROSITY
  - MY_SMART_GEMS
  - PRESSURE_GEM
  - TEMPERATURE_GEM
  - TRANSPORT_B

Standard block:
None

Keyword documentation:
https://ogs5-keywords.netlify.com/ogs/wiki/public/doc-auto/by_ext/gem

Reading routines:
https://github.com/ufz/ogs5/blob/master/FEM/rf_REACT_GEM.cpp#L2644

See also:

add_block

Attributes

block_no
Number of blocks in the file.

file_name
str: base name of the file with extension.

file_path
str: save path of the file.

force_writing
bool: state if the file is written even if empty.

is_empty
State if the OGS file is empty.

name
str: name of the file without extension.

Methods

add_block([index, main_key])	Add a new Block to the actual file.
add_content(content[, main_index, ...])	Add single-line content to the actual file.
add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
add_main_keyword(key[, main_index])	Add a new main keyword (#key) to the actual file.
add_multi_content(content[, main_index, ...])	Add multiple content to the actual file.
add_sub_keyword(key[, main_index, sub_index])	Add a new sub keyword ($key) to the actual file.
append_to_block([index])	Append data to an existing Block in the actual file.
check([verbose])	Check if the given file is valid.
del_block([index, del_all])	Delete a block by its index.
del_content([main_index, sub_index, ...])	Delete content by its position.
del_copy_link()	Remove a former given link to an external file.
del_main_keyword([main_index, del_all])	Delete a main keyword (#key) by its position.
del_sub_keyword([main_index, sub_index, del_all])	Delete a sub keyword ($key) by its position.
get_block([index, as_dict])	Get a Block from the actual file.
get_block_no()	Get the number of blocks in the file.
get_file_type()	Get the OGS file class name.
get_multi_keys([index])	State if a block has a unique set of sub keywords.
is_block_unique([index])	State if a block has a unique set of sub keywords.
read_file(path[, encoding, verbose])	Read an existing OGS input file.
reset()	Delete every content.
save(path, **kwargs)	Save the actual OGS input file in the given path.
update_block([index, main_key])	Update a Block from the actual file.
write_file()	Write the actual OGS input file to the given folder.

add_block(*index=None*, *main_key=None*, ***block*)

Add a new Block to the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 ...

$SUBKEY2
content2 ...

which looks like the following:

`FILE.add_block(SUBKEY1=content1, SUBKEY2=content2)`

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- **main_key** (*string, optional*) – Main keyword of the block that should be added (see: `MKEYS`) Default: the first main keyword of the file-type
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

add_content(*content*, *main_index=None*, *sub_index=None*, *line_index=None*)

Add single-line content to the actual file.

Parameters

- **content** (*list*) – list containing one line of content given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position, where the new line of content should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

---

add_copy_link(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied
- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

add_main_keyword(*key*, *main_index=None*)

Add a new main keyword (#key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – position, where the new main keyword should be added between the existing ones. As default, it is placed at the end.

add_multi_content(*content*, *main_index=None*, *sub_index=None*)

Add multiple content to the actual file.

Parameters

- **content** (*list*) – list containing lines of content, each given as a list of single statements

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

The content will be added at the end of the actual subkeyword.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

`add_sub_keyword`(*key*, *main_index=None*, *sub_index=None*)

Add a new sub keyword ($key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – position, where the new sub keyword should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the subkeyword is not added.

`append_to_block`(*index=None*, ***block*)

Append data to an existing Block in the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 …

$SUBKEY2
content2 …

which looks like the following:

```
FILE.append_to_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

`check`(*verbose=True*)

Check if the given file is valid.

Parameters
verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns
result – Validity of the given file.

Return type
bool

`del_block`(*index=None*, *del_all=False*)

Delete a block by its index.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **del_all** (*bool, optional*) – State, if all blocks shall be deleted. Default: False

`del_content`(*main_index=-1*, *sub_index=-1*, *line_index=-1*, *del_all=False*)

Delete content by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be deleted. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position of the content line, that should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all content shall be deleted. Default: False

`del_copy_link`()

Remove a former given link to an external file.

`del_main_keyword`(*main_index=None*, *del_all=False*)

Delete a main keyword (#key) by its position.

Parameters

- **main_index** (*int, optional*) – position, which main keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all main keywords shall be deleted. Default: False

`del_sub_keyword`(*main_index=-1*, *sub_index=-1*, *del_all=False*)

Delete a sub keyword ($key) by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **pos** (*int, optional*) – position, which sub keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all sub keywords shall be deleted. Default: False

`get_block`(*index=None*, *as_dict=True*)

Get a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **as_dict** (*bool, optional*) – Here you can state of you want the output as a dictionary, which can be used as key-word-arguments for *add_block*. If False, you get the main-key, a list of sub-keys and a list of content. Default: True

`get_block_no()`

Get the number of blocks in the file.

`get_file_type()`

Get the OGS file class name.

`get_multi_keys(index=None)`

State if a block has a unique set of sub keywords.

`is_block_unique(index=None)`

State if a block has a unique set of sub keywords.

`read_file(path, encoding=None, verbose=False)`

Read an existing OGS input file.

Parameters

- **path** (*str*) – path to the existing file that should be read
- **encoding** (*str or None, optional*) – encoding of the given file. If `None` is given, the system standard is used. Default: `None`
- **verbose** (*bool, optional*) – Print information of the reading process. Default: False

`reset()`

Delete every content.

`save(path, **kwargs)`

Save the actual OGS input file in the given path.

Parameters

- **path** (*str*) – path to where to file should be saved
- **update** (*bool, optional*) – state if the content should be updated before saving. Default: True

`update_block(index=None, main_key=None, **block)`

Update a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is used. Default: None
- **main_key** (*string, optional*) – Main keyword of the block that should be updated (see: `MKEYS`) This shouldn't be done. Default: None
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

`write_file()`

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

`MKEYS = ['GEM_PROPERTIES']`

Main Keywords of this OGS-BlockFile

Type
`list`

SKEYS = [['GEM_INIT_FILE', 'GEM_THREADS', 'TRANSPORT_B', 'FLAG_POROSITY_CHANGE', 'MIN_POROSITY', 'MAX_POROSITY', 'FLAG_COUPLING_HYDROLOGY', 'ITERATIVE_SCHEME', 'CALCULATE_BOUNDARY_NODES', 'TEMPERATURE_GEM', 'PRESSURE_GEM', 'MAX_FAILED_NODES', 'MY_SMART_GEMS', 'FLAG_DISABLE_GEM', 'KINETIC_GEM', 'DISABLE_GEMS', 'GEM_CALCULATE_BOUNDARY_NODES', 'GEM_SMART', 'FLAG_NODE_ELEMENT', 'FLAG_CALCULATE_BOUNDARY_NODE']]

Sub Keywords of this OGS-BlockFile

Type
list

STD = {}

Standard Block OGS-BlockFile

Type
dict

property block_no

Number of blocks in the file.

property file_name

base name of the file with extension.

Type
str

property file_path

save path of the file.

Type
str

property force_writing

state if the file is written even if empty.

Type
bool

property is_empty

State if the OGS file is empty.

property name

name of the file without extension.

Type
str

ogs5py.fileclasses.GEMinit

class ogs5py.fileclasses.**GEMinit**(*lst_name='model-dat.lst'*, *dch=None*, *ipm=None*, *dbr=None*, *task_root=None*, *task_id='model'*)

Bases: object

Class for GEMS3K input file.

lst file that contains the names of

- the GEMS data file (dch file),
- the GEMS numerical settings (ipm file)
- the example setup (dbr file)

used to initialize the GEMS3K kernel.

Parameters

- **lst_name** (str or None, optional) – name of the lst file
- **dch** (*LineFile* or None) – the GEMS data file
- **ipm** (*LineFile* or None) – the GEMS data file
- **dbr** (*LineFile* or None) – the GEMS data file
- **task_root** (*str, optional*) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (*str, optional*) – Name for the ogs task. Default: "model"

Notes

http://gems.web.psi.ch/GEMS3/index.html

http://gems.web.psi.ch/GEMS3K/

http://gems.web.psi.ch/GEMS3K/doc/html/gems3k-iofiles.html

Attributes

file_ext
The extension of the lst file.

file_names
The names of the included files.

files
List of the included files: dch, ipm, dbr.

is_empty
State if the file is empty.

name
The name of the lst file.

task_root
Get and set the task_root path of the ogs model.

Methods

check([verbose])	Check if the GEM external file is valid.
get_file_type()	Get the OGS file class name.
read_file(path[, encoding, verbose])	Read a given GEM external input lst-file.
reset()	Delete every content.
save(path)	Save the actual GEM external file in the given path.
write_file()	Write the actual OGS input file to the given folder.

check(*verbose=True*)

Check if the GEM external file is valid.

Parameters
verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns
result – Validity.

Return type
bool

get_file_type()

Get the OGS file class name.

read_file(*path*, *encoding=None*, *verbose=False*)

Read a given GEM external input lst-file.

Parameters
path (*str*) – path to the file

Notes

This also reads the given files in the lst-file. (dch, ipm, dbr)

reset()

Delete every content.

save(*path*)

Save the actual GEM external file in the given path.

lst file containing: dch, ipm, dbr

Parameters
path (*str*) – path to where to file should be saved

write_file()

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

property file_ext

The extension of the lst file.

property file_names

The names of the included files.

property files

dch, ipm, dbr.

Type
List of the included files

property is_empty

State if the file is empty.

property name

The name of the lst file.

property task_root

Get and set the task_root path of the ogs model.

ogs5py.fileclasses.GLI

class ogs5py.fileclasses.**GLI**(*gli_dict=None*, ***OGS_Config*)

Bases: *File*

Class for the ogs GEOMETRY file.

Parameters

- **gli_dict** (dict or None, optional) – dictionary containing the gli file Includes the following information (sorted by keys):

  points
  [ndarray] Array with all point postions

  point_names
  [ndarray (of strings)] Array with all point names

  point_md
  [ndarray] Array with all Material-densities at the points if point_md should be undefined it takes the value -np.inf

  polylines
  [list of dict] each containing information about

  - ID (int or None)
  - NAME (str)
  - POINTS (ndarray)
  - EPSILON (float or None)
  - TYPE (int or None)
  - MAT_GROUP (int or None)
  - POINT_VECTOR (str or None)

  surfaces
  [list of dict] each containing information about

  - ID (int or None)
  - NAME (str)
  - POLYLINES (list of str)
  - EPSILON (float or None)
  - TYPE (int or None)
  - MAT_GROUP (int or None)
  - TIN (str or None)

  volumes
  [list of dict] each containing information about

  - NAME (str)
  - SURFACES (list of str)
  - TYPE (int or None)
  - MAT_GROUP (int or None)
  - LAYER (int or None)

  Default: None

- **task_root** (str, optional) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (str, optional) – Name for the ogs task. Default: "model"

Attributes

POINTS
ndarray: POINTS (n,3) of the gli, defined by xyz-coordinates.

POINT_MD
ndarray: material density values of POINTS of the gli.

POINT_NAMES
ndarray: names of POINTS of the gli.

POINT_NO
int: number of POINTS of the gli.

POLYLINES
List of dict: POLYLINES of the gli.

POLYLINE_NAMES
List of str: names of POLYLINES of the gli.

POLYLINE_NO
int: number of POLYLINES of the gli.

SURFACES
List of dict: SURFACES of the gli.

SURFACE_NAMES
List of str: names of SURFACES of the gli.

SURFACE_NO
int: number of SURFACES of the gli.

VOLUMES
List of dict: VOLUMES of the gli.

VOLUME_NAMES
List of str: names of VOLUMES of the gli.

VOLUME_NO
int: number of VOLUMES of the gli.

file_name
str: base name of the file with extension.

file_path
str: save path of the file.

force_writing
bool: state if the file is written even if empty.

is_empty
bool: State if the GLI File is empty.

name
str: name of the file without extension.

Methods

__call__()	Return a copy of the underlying dictionary of the gli.
add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
add_points(points[, names, md, decimals])	Add a list of points (ndarray with shape (n,3)).
add_polyline(name, points[, ply_id, ...])	Add a polyline to the gli.
add_surface(name, polylines[, srf_id, ...])	Add a new surface.
add_volume(name, surfaces[, vol_type, ...])	Add a new volume.
check([verbose])	Check if the gli is valid.
del_copy_link()	Remove a former given link to an external file.
generate([generator])	Use a gli-generator from the generator module.
get_file_type()	Get the OGS file class name.
load(filepath[, verbose, encoding])	Load an OGS5 gli from file.
pnt_coord([pnt_name, pnt_id])	Get Point coordinates either by name or ID.
read_file(path[, encoding, verbose])	Load an OGS5 gli from file.
remove_point(id_or_name)	Remove a point by its name or ID.
remove_polyline(names)	Remove a polyline by its name.
remove_surface(names)	Remove a surface by its name.
remove_volume(names)	Remove a volume by its name.
reset()	Delete every content.
rotate(angle[, rotation_axis, rotation_point])	Rotate points around a given rotation point and axis with given angle.
save(path, **kwargs)	Save the gli to an OGS5 gli file.
set_dict(gli_dict)	Set a gli dict as returned by tools methods or generators.
shift(vector)	Shift points with a given vector.
swap_axis([axis1, axis2])	Swap axis of the coordinate system.
write_file()	Write the actual OGS input file to the given folder.

__call__()

Return a copy of the underlying dictionary of the gli.

Returns
Mesh – dictionary representation of the mesh

Return type
dict

add_copy_link(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied
- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

add_points(*points*, *names=None*, *md=None*, *decimals=4*)

Add a list of points (ndarray with shape (n,3)).

Keeps the pointlist unique. If a named point is added, that was already present, it will be renamed with the new name. Same for md. The pointlists of the polylines will be updated.

Parameters

- **points** (*ndarray*) – Array with new points.

- **names** (*ndarray of str or None, optional*) – array containing the names. If None, all new points are unnamed. Default: None
- **md** (*ndarray of float or None, optional*) – array containing the material densitiy. If None, all new points will have unspecified md. Default: None
- **decimals** (*int, optional*) – Number of decimal places to round the added points to (default: 4). If decimals is negative, it specifies the number of positions to the left of the decimal point. This will not round the new points, it's just for comparison of the already present points to guarante uniqueness.

Returns
new_pos – array with the IDs of the added points in the pointlist of the gli.

Return type
ndarray

`add_polyline`(*name*, *points*, *ply_id=None*, *epsilon=None*, *ply_type=None*, *mat_group=None*, *point_vector=None*, *closed=False*, *decimals=4*)

Add a polyline to the gli.

Parameters

- **name** (*str*) – name of the new polyline
- **points** (*ndarray*) – Array with the points. Either array of point IDs or new coordinates.
- **ply_id** (*int or None, optional*) – Default: None
- **epsilon** (*float or None, optional*) – Default: None
- **ply_type** (*int or None, optional*) – Default: None
- **mat_group** (*int or None, optional*) – Default: None
- **point_vector** (*str or None, optional*) – Default: None
- **closed** (*bool, optional*) – If the polyline shall be closed, the first point will be added as last point again. Default: False
- **decimals** (*int, optional*) – Number of decimal places to round the added points to (default: 4). If decimals is negative, it specifies the number of positions to the left of the decimal point. This will not round the new points, it's just for comparison of the already present points to guarante uniqueness.

`add_surface`(*name*, *polylines*, *srf_id=None*, *epsilon=None*, *srf_type=0*, *mat_group=None*, *tin=None*)

Add a new surface.

Parameters

- **name** (*str*) – name of the new surface
- **polylines** (*list of str*) – List of the surface-defining polyline-names
- **srf_id** (*int or None, optional*) – Default: None
- **epsilon** (*float or None, optional*) – Default: None
- **srf_type** (*int or None, optional*) – Default: None
- **mat_group** (*int or None, optional*) – Default: None
- **tin** (*str or None, optional*) – Default: None

`add_volume`(*name*, *surfaces*, *vol_type=None*, *mat_group=None*, *layer=None*)

Add a new volume.

Parameters

- **name** (*str*) – name of the new surface

- **surfaces** (*list of str*) – List of the volume-defining surface-names
- **vol_type** (*int or None, optional*) – Default: None
- **mat_group** (*int or None, optional*) – Default: None
- **layer** (*int or None, optional*) – Default: None

check(*verbose=True*)

Check if the gli is valid.

In the sence, that the contained data is consistent.

Parameters
verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns
result – Validity of the given gli.

Return type
bool

del_copy_link()

Remove a former given link to an external file.

generate(*generator='rectangular'*, ***kwargs*)

Use a gli-generator from the generator module.

See: *ogs5py.fileclasses.gli.generator*

Parameters

- **generator** (*str*) – set the generator from the generator module
- ****kwargs** – kwargs will be forwarded to the generator in use

Notes

The following generators are available:

rectangular([dim, ori, size, name])	Generate a rectangular boundary for a grid in 2D or 3D as gli.
radial([dim, ori, angles, rad_out, rad_in, ...])	Generate a radial boundary for a grid in 2D or 3D.

get_file_type()

Get the OGS file class name.

load(*filepath*, *verbose=False*, *encoding=None*, ***kwargs*)

Load an OGS5 gli from file.

kwargs will be forwarded to "tools.load_ogs5gli"

Parameters

- **filepath** (*string*) – path to the '*.gli' OGS5 gli file to load
- **verbose** (*bool, optional*) – Print information of the reading process. Default: True

pnt_coord(*pnt_name=None*, *pnt_id=None*)

Get Point coordinates either by name or ID.

Parameters

- **pnt_name** (*str*) – Point name.
- **pnt_id** (*int*) – Point ID.

`read_file`(*path*, *encoding=None*, *verbose=False*)

Load an OGS5 gli from file.

Parameters

- **path** (*string*) – path to the '*.gli' OGS5 gli file to load
- **encoding** (*str or None, optional*) – encoding of the given file. If `None` is given, the system standard is used. Default: `None`
- **verbose** (*bool, optional*) – Print information of the reading process. Default: False

`remove_point`(*id_or_name*)

Remove a point by its name or ID.

If Points are removed, that define polylines, they will be removed. Same for surfaces and volumes.

Parameters

id_or_name (*int or str or list of int or list of str*) – Points to be removed. Unknown names or IDs are ignored.

`remove_polyline`(*names*)

Remove a polyline by its name.

If Polylines are removed, that define surfaces, they will be removed. Same for volumes.

Parameters

names (*str or list of str*) – Polylines to be removed. Unknown names are ignored.

`remove_surface`(*names*)

Remove a surface by its name.

If Surfaces are removed, that define Volumes, they will be removed.

Parameters

names (*str or list of str*) – Surfaces to be removed. Unknown names are ignored.

`remove_volume`(*names*)

Remove a volume by its name.

Parameters

names (*str or list of str*) – Volumes to be removed. Unknown names are ignored.

`reset`()

Delete every content.

`rotate`(*angle*, *rotation_axis=(0.0, 0.0, 1.0)*, *rotation_point=(0.0, 0.0, 0.0)*)

Rotate points around a given rotation point and axis with given angle.

Parameters

- **angle** (*float*) – rotation angle given in radial length
- **rotation_axis** (*array_like, optional*) – Array containing the vector for ratation axis. Default: (0,0,1)
- **rotation_point** (*array_like, optional*) – Vector of the ratation base point. Default:(0,0,0)

`save`(*path*, ***kwargs*)

Save the gli to an OGS5 gli file.

kwargs will be forwarded to "tools.save_ogs5gli"

Parameters

- **path** (*string*) – path to the '*.gli' OGS5 gli file to save
- **verbose** (*bool, optional*) – Print information of the writing process. Default: True

set_dict(*gli_dict*)

Set a gli dict as returned by tools methods or generators.

Gli will be checked for validity.

Parameters

gli_dict (dict) – dictionary containing the gli file Includes the following information (sorted by keys):

points
[ndarray] Array with all point postions

point_names
[ndarray (of strings)] Array with all point names

point_md
[ndarray] Array with all Material-densities at the points if point_md should be undefined it takes the value -np.inf

polylines
[list of dict] each containing information about

- ID (int or None)
- NAME (str)
- POINTS (ndarray)
- EPSILON (float or None)
- TYPE (int or None)
- MAT_GROUP (int or None)
- POINT_VECTOR (str or None)

surfaces
[list of dict] each containing information about

- ID (int or None)
- NAME (str)
- POLYLINES (list of str)
- EPSILON (float or None)
- TYPE (int or None)
- MAT_GROUP (int or None)
- TIN (str or None)

volumes
[list of dict] each containing information about

- NAME (str)
- SURFACES (list of str)
- TYPE (int or None)
- MAT_GROUP (int or None)
- LAYER (int or None)

shift(*vector*)

Shift points with a given vector.

Parameters

vector (*ndarray*) – array containing the shifting vector

swap_axis(*axis1='y'*, *axis2='z'*)

Swap axis of the coordinate system.

Parameters

- **axis1** (str or int, optional) – First selected Axis. Either in ["x", "y", "z"] or in [0, 1, 2]. Default: "y"
- **axis2** (str or int, optional) – Second selected Axis. Either in ["x", "y", "z"] or in [0, 1, 2]. Default: "z"

write_file()

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

property POINTS

POINTS (n,3) of the gli, defined by xyz-coordinates.

Type
ndarray

property POINT_MD

material density values of POINTS of the gli.

Type
ndarray

property POINT_NAMES

names of POINTS of the gli.

Type
ndarray

property POINT_NO

number of POINTS of the gli.

Type
int

property POLYLINES

POLYLINES of the gli.

Type
List of dict

property POLYLINE_NAMES

names of POLYLINES of the gli.

Type
List of str

property POLYLINE_NO

number of POLYLINES of the gli.

Type
int

property SURFACES

SURFACES of the gli.

Type
List of dict

property SURFACE_NAMES

names of SURFACES of the gli.

Type
List of str

property SURFACE_NO

number of SURFACES of the gli.

Type
int

property VOLUMES

VOLUMES of the gli.

Type
List of dict

property VOLUME_NAMES

names of VOLUMES of the gli.

Type
List of str

property VOLUME_NO

number of VOLUMES of the gli.

Type
int

property file_name

base name of the file with extension.

Type
str

property file_path

save path of the file.

Type
str

property force_writing

state if the file is written even if empty.

Type
bool

property is_empty

State if the GLI File is empty.

Type
bool

property name

name of the file without extension.

Type
str

ogs5py.fileclasses.GLIext

class ogs5py.fileclasses.**GLIext**(*typ='TIN'*, *data=None*, *name=None*, *file_ext=None*, *task_root=None*, *task_id='model'*)

Bases: *File*

Class for an external definition for the ogs GEOMETRY file.

Parameters

- **typ** (str, optional) – Type of the extermal geometry definition. Either TIN for a triangulated surface or POINT_VECTOR for a polyline. Default: "TIN"
- **data** (numpy.ndarray, optional) – Data for the external geometry definition. Default: None
- **name** (*str, optional*) – File name for the RFR file. If None, the task_id is used. Default: None
- **file_ext** (str, optional) – extension of the file (with leading dot ".rfr") Default: ".rfr"
- **task_root** (*str, optional*) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (*str, optional*) – Name for the ogs task. Default: "model"

Attributes

file_name
str: base name of the file with extension.

file_path
str: save path of the file.

force_writing
bool: state if the file is written even if empty.

is_empty
State if the OGS file is empty.

name
str: name of the file without extension.

Methods

add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
check([verbose])	Check if the external geometry definition is valid.
del_copy_link()	Remove a former given link to an external file.
get_file_type()	Get the OGS file class name.
read_file(path, **kwargs)	Read a given GLI_EXT input file.
reset()	Delete every content.
save(path)	Save the actual GLI external file in the given path.
write_file()	Write the actual OGS input file to the given folder.

add_copy_link(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied

- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

`check`(*verbose=True*)

Check if the external geometry definition is valid.

In the sence, that the contained data is consistent.

Parameters
verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns
result – Validity of the given gli.

Return type
bool

`del_copy_link`()

Remove a former given link to an external file.

`get_file_type`()

Get the OGS file class name.

`read_file`(*path*, ***kwargs*)

Read a given GLI_EXT input file.

Parameters
path (*str*) – path to the file

`reset`()

Delete every content.

`save`(*path*)

Save the actual GLI external file in the given path.

Parameters
path (*str*) – path to where to file should be saved

`write_file`()

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

`property file_name`

base name of the file with extension.

Type
str

`property file_path`

save path of the file.

Type
str

`property force_writing`

state if the file is written even if empty.

Type
bool

`property is_empty`

State if the OGS file is empty.

property name

name of the file without extension.

Type

str

ogs5py.fileclasses.IC

class ogs5py.fileclasses.**IC**(***OGS_Config*)

Bases: *BlockFile*

Class for the ogs INITIAL_CONDITION file.

Parameters

- **task_root** (*str, optional*) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (*str, optional*) – Name for the ogs task. Default: "model"

Notes

Main-Keywords (#):

- INITIAL_CONDITION

Sub-Keywords ($) per Main-Keyword:

- INITIAL_CONDITION
  - PCS_TYPE
  - PRIMARY_VARIABLE
  - COMP_NAME
  - STORE_VALUES
  - DIS_TYPE
  - GEO_TYPE

Standard block:

PCS_TYPE
"GROUNDWATER_FLOW"

PRIMARY_VARIABLE
"HEAD"

GEO_TYPE
"DOMAIN"

DIS_TYPE
["CONSTANT", 0.0]

Keyword documentation:
https://ogs5-keywords.netlify.com/ogs/wiki/public/doc-auto/by_ext/ic

Reading routines:
https://github.com/ufz/ogs5/blob/master/FEM/rf_ic_new.cpp#L222

See also:

add_block

Attributes

block_no
Number of blocks in the file.

file_name
str: base name of the file with extension.

file_path
str: save path of the file.

force_writing
bool: state if the file is written even if empty.

is_empty
State if the OGS file is empty.

name
str: name of the file without extension.

Methods

add_block([index, main_key])	Add a new Block to the actual file.
add_content(content[, main_index, ...])	Add single-line content to the actual file.
add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
add_main_keyword(key[, main_index])	Add a new main keyword (#key) to the actual file.
add_multi_content(content[, main_index, ...])	Add multiple content to the actual file.
add_sub_keyword(key[, main_index, sub_index])	Add a new sub keyword ($key) to the actual file.
append_to_block([index])	Append data to an existing Block in the actual file.
check([verbose])	Check if the given file is valid.
del_block([index, del_all])	Delete a block by its index.
del_content([main_index, sub_index, ...])	Delete content by its position.
del_copy_link()	Remove a former given link to an external file.
del_main_keyword([main_index, del_all])	Delete a main keyword (#key) by its position.
del_sub_keyword([main_index, sub_index, del_all])	Delete a sub keyword ($key) by its position.
get_block([index, as_dict])	Get a Block from the actual file.
get_block_no()	Get the number of blocks in the file.
get_file_type()	Get the OGS file class name.
get_multi_keys([index])	State if a block has a unique set of sub keywords.
is_block_unique([index])	State if a block has a unique set of sub keywords.
read_file(path[, encoding, verbose])	Read an existing OGS input file.
reset()	Delete every content.
save(path, **kwargs)	Save the actual OGS input file in the given path.
update_block([index, main_key])	Update a Block from the actual file.
write_file()	Write the actual OGS input file to the given folder.

add_block(*index=None*, *main_key=None*, ***block*)

Add a new Block to the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 ...

$SUBKEY2
content2 ...

which looks like the following:

```
FILE.add_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.

- **main_key** (*string, optional*) – Main keyword of the block that should be added (see: `MKEYS`) Default: the first main keyword of the file-type
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

add_content(*content*, *main_index=None*, *sub_index=None*, *line_index=None*)

Add single-line content to the actual file.

Parameters

- **content** (*list*) – list containing one line of content given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position, where the new line of content should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

---

add_copy_link(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied
- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

add_main_keyword(*key*, *main_index=None*)

Add a new main keyword (#key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – position, where the new main keyword should be added between the existing ones. As default, it is placed at the end.

add_multi_content(*content*, *main_index=None*, *sub_index=None*)

Add multiple content to the actual file.

Parameters

- **content** (*list*) – list containing lines of content, each given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.

- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

The content will be added at the end of the actual subkeyword.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

`add_sub_keyword`(*key*, *main_index=None*, *sub_index=None*)

Add a new sub keyword ($key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – position, where the new sub keyword should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the subkeyword is not added.

`append_to_block`(*index=None*, ***block*)

Append data to an existing Block in the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 …

$SUBKEY2
content2 …

which looks like the following:

```
FILE.append_to_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

`check`(*verbose=True*)

Check if the given file is valid.

Parameters
verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns
result – Validity of the given file.

Return type
bool

`del_block`(*index=None*, *del_all=False*)

Delete a block by its index.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **del_all** (*bool, optional*) – State, if all blocks shall be deleted. Default: False

`del_content`(*main_index=-1*, *sub_index=-1*, *line_index=-1*, *del_all=False*)

Delete content by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be deleted. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position of the content line, that should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all content shall be deleted. Default: False

`del_copy_link`()

Remove a former given link to an external file.

`del_main_keyword`(*main_index=None*, *del_all=False*)

Delete a main keyword (#key) by its position.

Parameters

- **main_index** (*int, optional*) – position, which main keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all main keywords shall be deleted. Default: False

`del_sub_keyword`(*main_index=-1*, *sub_index=-1*, *del_all=False*)

Delete a sub keyword ($key) by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **pos** (*int, optional*) – position, which sub keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all sub keywords shall be deleted. Default: False

`get_block`(*index=None*, *as_dict=True*)

Get a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **as_dict** (*bool, optional*) – Here you can state of you want the output as a dictionary, which can be used as key-word-arguments for *add_block*. If False, you get the main-key, a list of sub-keys and a list of content. Default: True

`get_block_no`()

Get the number of blocks in the file.

`get_file_type`()

Get the OGS file class name.

`get_multi_keys`(*index=None*)

State if a block has a unique set of sub keywords.

`is_block_unique`(*index=None*)

State if a block has a unique set of sub keywords.

`read_file`(*path*, *encoding=None*, *verbose=False*)

Read an existing OGS input file.

Parameters

- **path** (*str*) – path to the existing file that should be read
- **encoding** (*str or None, optional*) – encoding of the given file. If `None` is given, the system standard is used. Default: `None`
- **verbose** (*bool, optional*) – Print information of the reading process. Default: False

`reset`()

Delete every content.

`save`(*path*, ***kwargs*)

Save the actual OGS input file in the given path.

Parameters

- **path** (*str*) – path to where to file should be saved
- **update** (*bool, optional*) – state if the content should be updated before saving. Default: True

`update_block`(*index=None*, *main_key=None*, ***block*)

Update a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is used. Default: None
- **main_key** (*string, optional*) – Main keyword of the block that should be updated (see: `MKEYS`) This shouldn't be done. Default: None
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

`write_file`()

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

`MKEYS = ['INITIAL_CONDITION']`

Main Keywords of this OGS-BlockFile

Type
`list`

`SKEYS = [['PCS_TYPE', 'PRIMARY_VARIABLE', 'COMP_NAME', 'STORE_VALUES', 'DIS_TYPE', 'GEO_TYPE']]`

Sub Keywords of this OGS-BlockFile

Type
`list`

STD = {'DIS_TYPE': ['CONSTANT', 0.0], 'GEO_TYPE': 'DOMAIN', 'PCS_TYPE': 'GROUNDWATER_FLOW', 'PRIMARY_VARIABLE': 'HEAD'}

Standard Block OGS-BlockFile

Type
dict

property block_no

Number of blocks in the file.

property file_name

base name of the file with extension.

Type
str

property file_path

save path of the file.

Type
str

property force_writing

state if the file is written even if empty.

Type
bool

property is_empty

State if the OGS file is empty.

property name

name of the file without extension.

Type
str

ogs5py.fileclasses.RFR

class ogs5py.fileclasses.**RFR**(*variables=None*, *data=None*, *units=None*, *headers=None*, *name=None*, *file_ext='.rfr'*, *task_root=None*, *task_id='model'*)

Bases: *File*

Class for the ogs RESTART file, if the DIS_TYPE in IC is set to RESTART.

Parameters

- **variables** (list of str, optional) – List of variable names. Default: None
- **data** (numpy.ndarray, optional) – RFR data. 2D array, where the first dimension is the number of variables. Default: None
- **units** (list of str, optional) – List of units for the occurring variables. Can be None. OGS5 ignores them anyway. Default: None
- **headers** (*str or None, optional*) – First four lines of the RFR file. If None, a standard header is written. Default: None
- **name** (*str, optional*) – File name for the RFR file. If None, the task_id is used. Default: None
- **file_ext** (str, optional) – extension of the file (with leading dot ".rfr") Default: ".rfr"
- **task_root** (*str, optional*) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (*str, optional*) – Name for the ogs task. Default: "model"

Notes

First line (ignored):

- #0#0#0#1#100000#0… (don't ask why)

Second line (ignored):

- 1 1 4 (don't ask why)

Third line (information about Variables):

- (No. of Var.) (No of data of 1. Var) (No of data of 2. Var) …
- 1 1 (example: 1 Variable with 1 component)
- 2 1 1 (example: 2 Variables with 1 component each)
- only 1 scalar per Variable allowed (bug in OGS5). See: https://github.com/ufz/ogs5/issues/151

Fourth line (Variable names and units):

- (Name1), (Unit1), (Name2), (Unit2), …
- units are ignored

Data (multiple lines):

- (index) (Var1data1) .. (Var1dataN1) (Var2data1) .. (Var2dataN2) …

Keyword documentation:
https://ogs5-keywords.netlify.com/ogs/wiki/public/doc-auto/by_ext/ic

Reading routines:
https://github.com/ufz/ogs5/blob/master/FEM/rf_ic_new.cpp#L932

Attributes

data
Data in the RFR file.

file_name
str: base name of the file with extension.

file_path
str: save path of the file.

force_writing
bool: state if the file is written even if empty.

is_empty
State if the OGS file is empty.

name
str: name of the file without extension.

units
List of variable-units in the RFR file.

var_count
Count of variables in the RFR file (line 3).

var_info
Infos about variables and units in the RFR file (line 4).

variables
List of variables in the RFR file.

Methods

add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
check([verbose])	Check if the external geometry definition is valid.
del_copy_link()	Remove a former given link to an external file.
get_file_type()	Get the OGS file class name.
read_file(path[, encoding, verbose])	Write the actual RFR input file to the given folder.
reset()	Delete every content.
save(path, **kwargs)	Save the actual RFR external file in the given path.
write_file()	Write the actual OGS input file to the given folder.

add_copy_link(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied
- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

check(*verbose=True*)

Check if the external geometry definition is valid.

In the sence, that the contained data is consistent.

Parameters
verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns
result – Validity of the given gli.

Return type
bool

`del_copy_link`()
Remove a former given link to an external file.

`get_file_type`()
Get the OGS file class name.

`read_file`(*path*, *encoding=None*, *verbose=False*)
Write the actual RFR input file to the given folder.

`reset`()
Delete every content.

`save`(*path*, ***kwargs*)
Save the actual RFR external file in the given path.

Parameters
path (*str*) – path to where to file should be saved

`write_file`()
Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

`property data`
Data in the RFR file.

`property file_name`
base name of the file with extension.

Type
str

`property file_path`
save path of the file.

Type
str

`property force_writing`
state if the file is written even if empty.

Type
bool

`property is_empty`
State if the OGS file is empty.

`property name`
name of the file without extension.

Type
str

`property units`
List of variable-units in the RFR file.

`property var_count`
Count of variables in the RFR file (line 3).

`property var_info`
Infos about variables and units in the RFR file (line 4).

property variables

List of variables in the RFR file.

ogs5py.fileclasses.KRC

class ogs5py.fileclasses.**KRC**(***OGS_Config*)

Bases: *BlockFile*

Class for the ogs KINETRIC REACTION file.

Parameters

- **task_root** (*str, optional*) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (*str, optional*) – Name for the ogs task. Default: "model"

Notes

Main-Keywords (#):

- MICROBE_PROPERTIES
- REACTION
- BLOB_PROPERTIES
- KINREACTIONDATA

Sub-Keywords ($) per Main-Keyword:

- MICROBE_PROPERTIES
  - MICROBENAME
  - _drmc__PARAMETERS
  - MONOD_REACTION_NAME
- REACTION
  - NAME
  - TYPE
  - BACTERIANAME
  - EQUATION
  - RATECONSTANT
  - GROWTH
  - MONODTERMS
  - THRESHHOLDTERMS
  - INHIBITIONTERMS
  - PRODUCTIONTERMS
  - PRODUCTIONSTOCH
  - BACTERIAL_YIELD
  - ISOTOPE_FRACTIONATION
  - BACTERIA_SPECIFIC_CAPACITY
  - TEMPERATURE_DEPENDENCE
  - _drmc_
  - STANDARD_GIBBS_ENERGY
  - EXCHANGE_PARAMETERS

  - SORPTION_TYPE
  - NAPL_PROPERTIES
  - REACTION_ORDER
  - MINERALNAME
  - CHEMAPPNAME
  - EQUILIBRIUM_CONSTANT
  - RATE_EXPONENTS
  - REACTIVE_SURFACE_AREA
  - PRECIPITATION_BY_BASETERM_ONLY
  - PRECIPITATION_FACTOR
  - PRECIPITATION_EXPONENT
  - BASETERM
  - MECHANISMTERM
  - SWITCH_OFF_GEOMETRY
- BLOB_PROPERTIES
  - NAME
  - D50
  - DM
  - DS
  - UI
  - NAPL_CONTENT_INI
  - NAPL_CONTENT_RES
  - GRAIN_SPHERE_RATIO
  - TORTUOSITY
  - LENGTH
  - CALC_SHERWOOD
  - CALC_SHERWOOD_MODIFIED
  - SHERWOOD_MODEL
  - GEOMETRY
  - GAS_DISSOLUTION
  - INTERFACIAL_AREA
- KINREACTIONDATA
  - SOLVER_TYPE
  - RELATIVE_ERROR
  - MIN_TIMESTEP
  - INITIAL_TIMESTEP
  - BACTERIACAPACITY
  - MIN_BACTERIACONC
  - MIN_CONCENTRATION_REPLACE

- SURFACES
- ALLOW_REACTIONS
- NO_REACTIONS
- COPY_CONCENTRATIONS
- LAGNEAU_BENCHMARK
- SCALE_DCDT
- SORT_NODES
- OMEGA_THRESHOLD
- REACTION_DEACTIVATION
- DEBUG_OUTPUT
- ACTIVITY_MODEL

Standard block:
None

Keyword documentation:
https://ogs5-keywords.netlify.com/ogs/wiki/public/doc-auto/by_ext/krc

Reading routines:
https://github.com/ufz/ogs5/blob/master/FEM/rf_kinreact.cpp

MICROBE_PROPERTIES :
https://github.com/ufz/ogs5/blob/master/FEM/rf_kinreact.cpp#L232

REACTION :
https://github.com/ufz/ogs5/blob/master/FEM/rf_kinreact.cpp#L1549

BLOB_PROPERTIES :
https://github.com/ufz/ogs5/blob/master/FEM/rf_kinreact.cpp#L2622

KINREACTIONDATA :
https://github.com/ufz/ogs5/blob/master/FEM/rf_kinreact.cpp#L3185

See also:

add_block

Attributes

block_no
Number of blocks in the file.

file_name
`str`: base name of the file with extension.

file_path
`str`: save path of the file.

force_writing
`bool`: state if the file is written even if empty.

is_empty
State if the OGS file is empty.

name
`str`: name of the file without extension.

Methods

add_block([index, main_key])	Add a new Block to the actual file.
add_content(content[, main_index, ...])	Add single-line content to the actual file.
add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
add_main_keyword(key[, main_index])	Add a new main keyword (#key) to the actual file.
add_multi_content(content[, main_index, ...])	Add multiple content to the actual file.
add_sub_keyword(key[, main_index, sub_index])	Add a new sub keyword ($key) to the actual file.
append_to_block([index])	Append data to an existing Block in the actual file.
check([verbose])	Check if the given file is valid.
del_block([index, del_all])	Delete a block by its index.
del_content([main_index, sub_index, ...])	Delete content by its position.
del_copy_link()	Remove a former given link to an external file.
del_main_keyword([main_index, del_all])	Delete a main keyword (#key) by its position.
del_sub_keyword([main_index, sub_index, del_all])	Delete a sub keyword ($key) by its position.
get_block([index, as_dict])	Get a Block from the actual file.
get_block_no()	Get the number of blocks in the file.
get_file_type()	Get the OGS file class name.
get_multi_keys([index])	State if a block has a unique set of sub keywords.
is_block_unique([index])	State if a block has a unique set of sub keywords.
read_file(path[, encoding, verbose])	Read an existing OGS input file.
reset()	Delete every content.
save(path, **kwargs)	Save the actual OGS input file in the given path.
update_block([index, main_key])	Update a Block from the actual file.
write_file()	Write the actual OGS input file to the given folder.

add_block(*index=None*, *main_key=None*, ***block*)

Add a new Block to the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 ...

$SUBKEY2
content2 ...

which looks like the following:

```
FILE.add_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- **main_key** (*string, optional*) – Main keyword of the block that should be added (see: `MKEYS`) Default: the first main keyword of the file-type
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

add_content(*content*, *main_index=None*, *sub_index=None*, *line_index=None*)

Add single-line content to the actual file.

Parameters

- **content** (*list*) – list containing one line of content given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position, where the new line of content should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

`add_copy_link`(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied
- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

`add_main_keyword`(*key*, *main_index=None*)

Add a new main keyword (#key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – position, where the new main keyword should be added between the existing ones. As default, it is placed at the end.

`add_multi_content`(*content*, *main_index=None*, *sub_index=None*)

Add multiple content to the actual file.

Parameters

- **content** (*list*) – list containing lines of content, each given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

The content will be added at the end of the actual subkeyword.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

add_sub_keyword(*key*, *main_index=None*, *sub_index=None*)

Add a new sub keyword ($key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – position, where the new sub keyword should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the subkeyword is not added.

---

append_to_block(*index=None*, ***block*)

Append data to an existing Block in the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 …

$SUBKEY2
content2 …

which looks like the following:

```
FILE.append_to_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

check(*verbose=True*)

Check if the given file is valid.

Parameters
verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns
result – Validity of the given file.

Return type
bool

del_block(*index=None*, *del_all=False*)

Delete a block by its index.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **del_all** (*bool, optional*) – State, if all blocks shall be deleted. Default: False

del_content(*main_index=-1*, *sub_index=-1*, *line_index=-1*, *del_all=False*)

Delete content by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be deleted. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position of the content line, that should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all content shall be deleted. Default: False

del_copy_link()

Remove a former given link to an external file.

del_main_keyword(*main_index=None*, *del_all=False*)

Delete a main keyword (#key) by its position.

Parameters

- **main_index** (*int, optional*) – position, which main keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all main keywords shall be deleted. Default: False

del_sub_keyword(*main_index=-1*, *sub_index=-1*, *del_all=False*)

Delete a sub keyword ($key) by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **pos** (*int, optional*) – position, which sub keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all sub keywords shall be deleted. Default: False

get_block(*index=None*, *as_dict=True*)

Get a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **as_dict** (*bool, optional*) – Here you can state of you want the output as a dictionary, which can be used as key-word-arguments for *add_block*. If False, you get the main-key, a list of sub-keys and a list of content. Default: True

get_block_no()

Get the number of blocks in the file.

get_file_type()

Get the OGS file class name.

get_multi_keys(*index=None*)

State if a block has a unique set of sub keywords.

is_block_unique(*index=None*)

State if a block has a unique set of sub keywords.

`read_file`(*path*, *encoding=None*, *verbose=False*)

Read an existing OGS input file.

Parameters

- **path** (*str*) – path to the existing file that should be read
- **encoding** (*str or None, optional*) – encoding of the given file. If `None` is given, the system standard is used. Default: `None`
- **verbose** (*bool, optional*) – Print information of the reading process. Default: False

`reset`()

Delete every content.

`save`(*path*, ***kwargs*)

Save the actual OGS input file in the given path.

Parameters

- **path** (*str*) – path to where to file should be saved
- **update** (*bool, optional*) – state if the content should be updated before saving. Default: True

`update_block`(*index=None*, *main_key=None*, ***block*)

Update a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is used. Default: None
- **main_key** (*string, optional*) – Main keyword of the block that should be updated (see: `MKEYS`) This shouldn't be done. Default: None
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

`write_file`()

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

`MKEYS = ['MICROBE_PROPERTIES', 'REACTION', 'BLOB_PROPERTIES', 'KINREACTIONDATA']`

Main Keywords of this OGS-BlockFile

Type
`list`

SKEYS = [['MICROBENAME', '_drmc__PARAMETERS', 'MONOD_REACTION_NAME'], ['NAME', 'TYPE', 'BACTERIANAME', 'EQUATION', 'RATECONSTANT', 'GROWTH', 'MONODTERMS', 'THRESHHOLDTERMS', 'INHIBITIONTERMS', 'PRODUCTIONTERMS', 'PRODUCTIONSTOCH', 'BACTERIAL_YIELD', 'ISOTOPE_FRACTIONATION', 'BACTERIA_SPECIFIC_CAPACITY', 'TEMPERATURE_DEPENDENCE', '_drmc_', 'STANDARD_GIBBS_ENERGY', 'EXCHANGE_PARAMETERS', 'SORPTION_TYPE', 'NAPL_PROPERTIES', 'REACTION_ORDER', 'MINERALNAME', 'CHEMAPPNAME', 'EQUILIBRIUM_CONSTANT', 'RATE_EXPONENTS', 'REACTIVE_SURFACE_AREA', 'PRECIPITATION_BY_BASETERM_ONLY', 'PRECIPITATION_FACTOR', 'PRECIPITATION_EXPONENT', 'BASETERM', 'MECHANISMTERM', 'SWITCH_OFF_GEOMETRY'], ['NAME', 'D50', 'DM', 'DS', 'UI', 'NAPL_CONTENT_INI', 'NAPL_CONTENT_RES', 'GRAIN_SPHERE_RATIO', 'TORTUOSITY', 'LENGTH', 'CALC_SHERWOOD', 'CALC_SHERWOOD_MODIFIED', 'SHERWOOD_MODEL', 'GEOMETRY', 'GAS_DISSOLUTION', 'INTERFACIAL_AREA'], ['SOLVER_TYPE', 'RELATIVE_ERROR', 'MIN_TIMESTEP', 'INITIAL_TIMESTEP', 'BACTERIACAPACITY', 'MIN_BACTERIACONC', 'MIN_CONCENTRATION_REPLACE', 'SURFACES', 'ALLOW_REACTIONS', 'NO_REACTIONS', 'COPY_CONCENTRATIONS', 'LAGNEAU_BENCHMARK', 'SCALE_DCDT', 'SORT_NODES', 'OMEGA_THRESHOLD', 'REACTION_DEACTIVATION', 'DEBUG_OUTPUT', 'ACTIVITY_MODEL', 'REALATIVE_ERROR', 'MAX_TIMESTEP']]

Sub Keywords of this OGS-BlockFile

Type
list

STD = {}

Standard Block OGS-BlockFile

Type
dict

property block_no

Number of blocks in the file.

property file_name

base name of the file with extension.

Type
str

property file_path

save path of the file.

Type
str

property force_writing

state if the file is written even if empty.

Type
bool

property is_empty

State if the OGS file is empty.

property name

name of the file without extension.

Type
str

ogs5py.fileclasses.MCP

class ogs5py.fileclasses.**MCP**(***OGS_Config*)

Bases: *BlockFile*

Class for the ogs COMPONENT_PROPERTIES file.

Parameters

- **task_root** (*str, optional*) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (*str, optional*) – Name for the ogs task. Default: "model"

Notes

Main-Keywords (#):

- COMPONENT_PROPERTIES

Sub-Keywords ($) per Main-Keyword:

- COMPONENT_PROPERTIES
  - ACENTRIC_FACTOR
  - A_ZERO
  - BUBBLE_VELOCITY
  - CRITICAL_PRESSURE
  - CRITICAL_TEMPERATURE
  - DECAY
  - DIFFUSION
  - FLUID_ID
  - FLUID_PHASE
  - FORMULA
  - ISOTHERM
  - MAXIMUM_AQUEOUS_SOLUBILITY
  - MINERAL_DENSITY
  - MOBILE
  - MOLAR_DENSITY
  - MOLAR_VOLUME
  - MOLAR_WEIGHT
  - MOL_MASS
  - NAME
  - OutputMassOfComponentInModel
  - TRANSPORT_PHASE
  - VALENCE
  - VOLUME_DIFFUSION

Standard block:

None

Keyword documentation:
https://ogs5-keywords.netlify.com/ogs/wiki/public/doc-auto/by_ext/mcp

Reading routines:
https://github.com/ufz/ogs5/blob/master/FEM/rfmat_cp.cpp#L269

See also:

add_block

Attributes

block_no
Number of blocks in the file.

file_name
str: base name of the file with extension.

file_path
str: save path of the file.

force_writing
bool: state if the file is written even if empty.

is_empty
State if the OGS file is empty.

name
str: name of the file without extension.

Methods

add_block([index, main_key])	Add a new Block to the actual file.
add_content(content[, main_index, ...])	Add single-line content to the actual file.
add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
add_main_keyword(key[, main_index])	Add a new main keyword (#key) to the actual file.
add_multi_content(content[, main_index, ...])	Add multiple content to the actual file.
add_sub_keyword(key[, main_index, sub_index])	Add a new sub keyword ($key) to the actual file.
append_to_block([index])	Append data to an existing Block in the actual file.
check([verbose])	Check if the given file is valid.
del_block([index, del_all])	Delete a block by its index.
del_content([main_index, sub_index, ...])	Delete content by its position.
del_copy_link()	Remove a former given link to an external file.
del_main_keyword([main_index, del_all])	Delete a main keyword (#key) by its position.
del_sub_keyword([main_index, sub_index, del_all])	Delete a sub keyword ($key) by its position.
get_block([index, as_dict])	Get a Block from the actual file.
get_block_no()	Get the number of blocks in the file.
get_file_type()	Get the OGS file class name.
get_multi_keys([index])	State if a block has a unique set of sub keywords.
is_block_unique([index])	State if a block has a unique set of sub keywords.
read_file(path[, encoding, verbose])	Read an existing OGS input file.
reset()	Delete every content.
save(path, **kwargs)	Save the actual OGS input file in the given path.
update_block([index, main_key])	Update a Block from the actual file.
write_file()	Write the actual OGS input file to the given folder.

`add_block`(*index=None*, *main_key=None*, ***block*)

Add a new Block to the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 ...

$SUBKEY2
content2 ...

which looks like the following:

```
FILE.add_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- **main_key** (*string, optional*) – Main keyword of the block that should be added (see: `MKEYS`) Default: the first main keyword of the file-type
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

`add_content`(*content*, *main_index=None*, *sub_index=None*, *line_index=None*)

Add single-line content to the actual file.

Parameters

- **content** (*list*) – list containing one line of content given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position, where the new line of content should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

`add_copy_link`(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied
- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

add_main_keyword(*key*, *main_index=None*)

Add a new main keyword (#key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – position, where the new main keyword should be added between the existing ones. As default, it is placed at the end.

add_multi_content(*content*, *main_index=None*, *sub_index=None*)

Add multiple content to the actual file.

Parameters

- **content** (*list*) – list containing lines of content, each given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

The content will be added at the end of the actual subkeyword.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

---

add_sub_keyword(*key*, *main_index=None*, *sub_index=None*)

Add a new sub keyword ($key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – position, where the new sub keyword should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the subkeyword is not added.

---

append_to_block(*index=None*, ***block*)

Append data to an existing Block in the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 …

$SUBKEY2
content2 …

which looks like the following:

```
FILE.append_to_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

`check`(*verbose=True*)

Check if the given file is valid.

Parameters

verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns

result – Validity of the given file.

Return type

bool

`del_block`(*index=None*, *del_all=False*)

Delete a block by its index.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **del_all** (*bool, optional*) – State, if all blocks shall be deleted. Default: False

`del_content`(*main_index=-1*, *sub_index=-1*, *line_index=-1*, *del_all=False*)

Delete content by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be deleted. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position of the content line, that should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all content shall be deleted. Default: False

`del_copy_link`()

Remove a former given link to an external file.

`del_main_keyword`(*main_index=None*, *del_all=False*)

Delete a main keyword (#key) by its position.

Parameters

- **main_index** (*int, optional*) – position, which main keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all main keywords shall be deleted. Default: False

`del_sub_keyword`(*main_index=-1*, *sub_index=-1*, *del_all=False*)

Delete a sub keyword ($key) by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.

- **pos** (*int, optional*) – position, which sub keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all sub keywords shall be deleted. Default: False

`get_block`(*index=None*, *as_dict=True*)

Get a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **as_dict** (*bool, optional*) – Here you can state of you want the output as a dictionary, which can be used as key-word-arguments for *add_block*. If False, you get the main-key, a list of sub-keys and a list of content. Default: True

`get_block_no`()

Get the number of blocks in the file.

`get_file_type`()

Get the OGS file class name.

`get_multi_keys`(*index=None*)

State if a block has a unique set of sub keywords.

`is_block_unique`(*index=None*)

State if a block has a unique set of sub keywords.

`read_file`(*path*, *encoding=None*, *verbose=False*)

Read an existing OGS input file.

Parameters

- **path** (*str*) – path to the existing file that should be read
- **encoding** (*str or None, optional*) – encoding of the given file. If `None` is given, the system standard is used. Default: `None`
- **verbose** (*bool, optional*) – Print information of the reading process. Default: False

`reset`()

Delete every content.

`save`(*path*, ***kwargs*)

Save the actual OGS input file in the given path.

Parameters

- **path** (*str*) – path to where to file should be saved
- **update** (*bool, optional*) – state if the content should be updated before saving. Default: True

`update_block`(*index=None*, *main_key=None*, ***block*)

Update a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is used. Default: None
- **main_key** (*string, optional*) – Main keyword of the block that should be updated (see: `MKEYS`) This shouldn't be done. Default: None
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

write_file()

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

MKEYS = ['COMPONENT_PROPERTIES']

Main Keywords of this OGS-BlockFile

Type
list

SKEYS = [['NAME', 'FORMULA', 'MOBILE', 'TRANSPORT_PHASE', 'FLUID_PHASE', 'MOL_MASS', 'CRITICAL_PRESSURE', 'CRITICAL_TEMPERATURE', 'ACENTRIC_FACTOR', 'FLUID_ID', 'MOLAR_VOLUME', 'VOLUME_DIFFUSION', 'MINERAL_DENSITY', 'DIFFUSION', 'DECAY', 'ISOTHERM', 'BUBBLE_VELOCITY', 'MOLAR_DENSITY', 'MOLAR_WEIGHT', 'MAXIMUM_AQUEOUS_SOLUBILITY', 'OutputMassOfComponentInModel', 'VALENCE', 'A_ZERO', 'CRITICAL_VOLUME', 'CRITICAL_DENSITY', 'COMP_CAPACITY', 'COMP_CONDUCTIVITY', 'SOLUTE', 'MOLECULAR_WEIGHT']]

Sub Keywords of this OGS-BlockFile

Type
list

STD = {}

Standard Block OGS-BlockFile

Type
dict

property block_no

Number of blocks in the file.

property file_name

base name of the file with extension.

Type
str

property file_path

save path of the file.

Type
str

property force_writing

state if the file is written even if empty.

Type
bool

property is_empty

State if the OGS file is empty.

property name

name of the file without extension.

Type
str

ogs5py.fileclasses.MFP

class ogs5py.fileclasses.**MFP**(***OGS_Config*)

Bases: *BlockFile*

Class for the ogs FLUID PROPERTY file.

Parameters

- **task_root** (*str, optional*) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (*str, optional*) – Name for the ogs task. Default: "model"

Notes

Main-Keywords (#):

- FLUID_PROPERTIES

Sub-Keywords ($) per Main-Keyword:

- FLUID_PROPERTIES
  - COMPONENTS
  - COMPRESSIBILITY
  - DAT_TYPE
  - DECAY
  - DENSITY
  - DIFFUSION
  - DRHO_DT_UNSATURATED
  - EOS_TYPE
  - FLUID_NAME
  - FLUID_TYPE
  - GRAVITY
  - HEAT_CONDUCTIVITY
  - ISOTHERM
  - JTC
  - NON_GRAVITY
  - PHASE_DIFFUSION
  - SPECIFIC_HEAT_CAPACITY
  - SPECIFIC_HEAT_SOURCE
  - TEMPERATURE
  - VISCOSITY

Standard block:

FLUID_TYPE
"LIQUID"

DENSITY
[1, 1.0e+03]

VISCOSITY
[1, 1.0e-03]

Keyword documentation:
https://ogs5-keywords.netlify.com/ogs/wiki/public/doc-auto/by_ext/mfp

Reading routines:
https://github.com/ufz/ogs5/blob/master/FEM/rf_mfp_new.cpp#L140

See also:

add_block

Attributes

block_no
Number of blocks in the file.

file_name
str: base name of the file with extension.

file_path
str: save path of the file.

force_writing
bool: state if the file is written even if empty.

is_empty
State if the OGS file is empty.

name
str: name of the file without extension.

Methods

add_block([index, main_key])	Add a new Block to the actual file.
add_content(content[, main_index, ...])	Add single-line content to the actual file.
add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
add_main_keyword(key[, main_index])	Add a new main keyword (#key) to the actual file.
add_multi_content(content[, main_index, ...])	Add multiple content to the actual file.
add_sub_keyword(key[, main_index, sub_index])	Add a new sub keyword ($key) to the actual file.
append_to_block([index])	Append data to an existing Block in the actual file.
check([verbose])	Check if the given file is valid.
del_block([index, del_all])	Delete a block by its index.
del_content([main_index, sub_index, ...])	Delete content by its position.
del_copy_link()	Remove a former given link to an external file.
del_main_keyword([main_index, del_all])	Delete a main keyword (#key) by its position.
del_sub_keyword([main_index, sub_index, del_all])	Delete a sub keyword ($key) by its position.
get_block([index, as_dict])	Get a Block from the actual file.
get_block_no()	Get the number of blocks in the file.
get_file_type()	Get the OGS file class name.
get_multi_keys([index])	State if a block has a unique set of sub keywords.
is_block_unique([index])	State if a block has a unique set of sub keywords.
read_file(path[, encoding, verbose])	Read an existing OGS input file.
reset()	Delete every content.
save(path, **kwargs)	Save the actual OGS input file in the given path.
update_block([index, main_key])	Update a Block from the actual file.
write_file()	Write the actual OGS input file to the given folder.

add_block(*index=None*, *main_key=None*, ***block*)

Add a new Block to the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 ...

$SUBKEY2
content2 ...

which looks like the following:

```
FILE.add_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- **main_key** (*string, optional*) – Main keyword of the block that should be added (see: `MKEYS`) Default: the first main keyword of the file-type
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

add_content(*content*, *main_index=None*, *sub_index=None*, *line_index=None*)

Add single-line content to the actual file.

Parameters

- **content** (*list*) – list containing one line of content given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position, where the new line of content should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

---

add_copy_link(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied
- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

add_main_keyword(*key*, *main_index=None*)

Add a new main keyword (#key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – position, where the new main keyword should be added between the existing ones. As default, it is placed at the end.

add_multi_content(*content*, *main_index=None*, *sub_index=None*)

Add multiple content to the actual file.

Parameters

- **content** (*list*) – list containing lines of content, each given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

The content will be added at the end of the actual subkeyword.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

---

add_sub_keyword(*key*, *main_index=None*, *sub_index=None*)

Add a new sub keyword ($key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – position, where the new sub keyword should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the subkeyword is not added.

---

append_to_block(*index=None*, ***block*)

Append data to an existing Block in the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 …

$SUBKEY2
content2 …

which looks like the following:

```
FILE.append_to_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

`check`(*verbose=True*)

Check if the given file is valid.

Parameters
verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns
result – Validity of the given file.

Return type
bool

`del_block`(*index=None*, *del_all=False*)

Delete a block by its index.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **del_all** (*bool, optional*) – State, if all blocks shall be deleted. Default: False

`del_content`(*main_index=-1*, *sub_index=-1*, *line_index=-1*, *del_all=False*)

Delete content by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be deleted. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position of the content line, that should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all content shall be deleted. Default: False

`del_copy_link`()

Remove a former given link to an external file.

`del_main_keyword`(*main_index=None*, *del_all=False*)

Delete a main keyword (#key) by its position.

Parameters

- **main_index** (*int, optional*) – position, which main keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all main keywords shall be deleted. Default: False

`del_sub_keyword`(*main_index=-1*, *sub_index=-1*, *del_all=False*)

Delete a sub keyword ($key) by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.

- **pos** (*int, optional*) – position, which sub keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all sub keywords shall be deleted. Default: False

`get_block`(*index=None*, *as_dict=True*)

Get a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **as_dict** (*bool, optional*) – Here you can state of you want the output as a dictionary, which can be used as key-word-arguments for *add_block*. If False, you get the main-key, a list of sub-keys and a list of content. Default: True

`get_block_no`()

Get the number of blocks in the file.

`get_file_type`()

Get the OGS file class name.

`get_multi_keys`(*index=None*)

State if a block has a unique set of sub keywords.

`is_block_unique`(*index=None*)

State if a block has a unique set of sub keywords.

`read_file`(*path*, *encoding=None*, *verbose=False*)

Read an existing OGS input file.

Parameters

- **path** (*str*) – path to the existing file that should be read
- **encoding** (*str or None, optional*) – encoding of the given file. If `None` is given, the system standard is used. Default: `None`
- **verbose** (*bool, optional*) – Print information of the reading process. Default: False

`reset`()

Delete every content.

`save`(*path*, ***kwargs*)

Save the actual OGS input file in the given path.

Parameters

- **path** (*str*) – path to where to file should be saved
- **update** (*bool, optional*) – state if the content should be updated before saving. Default: True

`update_block`(*index=None*, *main_key=None*, ***block*)

Update a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is used. Default: None
- **main_key** (*string, optional*) – Main keyword of the block that should be updated (see: `MKEYS`) This shouldn't be done. Default: None
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

write_file()

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

MKEYS = ['FLUID_PROPERTIES']

Main Keywords of this OGS-BlockFile

Type
list

SKEYS = [['FLUID_TYPE', 'COMPONENTS', 'FLUID_NAME', 'EOS_TYPE', 'COMPRESSIBILITY', 'JTC', 'DAT_TYPE', 'NON_GRAVITY', 'DRHO_DT_UNSATURATED', 'DENSITY', 'TEMPERATURE', 'VISCOSITY', 'SPECIFIC_HEAT_CAPACITY', 'SPECIFIC_HEAT_CONDUCTIVITY', 'HEAT_CAPACITY', 'HEAT_CONDUCTIVITY', 'PHASE_DIFFUSION', 'DIFFUSION', 'DECAY', 'ISOTHERM', 'GRAVITY', 'SPECIFIC_HEAT_SOURCE', 'PCS_TYPE', 'THERMAL']]

Sub Keywords of this OGS-BlockFile

Type
list

STD = {'DENSITY': [1, 1000.0], 'FLUID_TYPE': 'LIQUID', 'VISCOSITY': [1, 0.001]}

Standard Block OGS-BlockFile

Type
dict

property block_no

Number of blocks in the file.

property file_name

base name of the file with extension.

Type
str

property file_path

save path of the file.

Type
str

property force_writing

state if the file is written even if empty.

Type
bool

property is_empty

State if the OGS file is empty.

property name

name of the file without extension.

Type
str

ogs5py.fileclasses.MMP

class ogs5py.fileclasses.**MMP**(***OGS_Config*)

Bases: *BlockFile*

Class for the ogs MEDIUM_PROPERTIES file.

Parameters

- **task_root** (*str, optional*) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (*str, optional*) – Name for the ogs task. Default: "model"

Notes

Main-Keywords (#):

- MEDIUM_PROPERTIES

Sub-Keywords ($) per Main-Keyword:

- MEDIUM_PROPERTIES
  - CAPILLARY_PRESSURE
  - CHANNEL
  - COMPOUND_DEPENDENT_DT
  - CONDUCTIVITY_MODEL
  - CONVERSION_FACTOR
  - DATA
  - DIFFUSION
  - DIS_TYPE
  - ELEMENT_VOLUME_MULTIPLYER
  - EVAPORATION
  - FLOWLINEARITY
  - GEOMETRY_AREA
  - GEOMETRY_DIMENSION
  - GEOMETRY_INCLINATION
  - GEO_TYPE
  - HEAT_DISPERSION
  - HEAT_TRANSFER
  - INTERPHASE_FRICTION
  - MASS_DISPERSION
  - MMP_TYPE
  - MSH_TYPE
  - NAME
  - ORGANIC_CARBON
  - PARTICLE_DIAMETER
  - PCS_TYPE

- PERMEABILITY_FUNCTION_DEFORMATION
- PERMEABILITY_FUNCTION_EFFSTRESS
- PERMEABILITY_FUNCTION_POROSITY
- PERMEABILITY_FUNCTION_PRESSURE
- PERMEABILITY_FUNCTION_STRAIN
- PERMEABILITY_FUNCTION_STRESS
- PERMEABILITY_FUNCTION_VELOCITY
- PERMEABILITY_SATURATION
- PERMEABILITY_TENSOR
- PERMEABILITY_DISTRIBUTION
- POROSITY
- POROSITY_DISTRIBUTION
- RILL
- SPECIFIC_STORAGE
- STORAGE
- STORAGE_FUNCTION_EFFSTRESS
- SURFACE_FRICTION
- TORTUOSITY
- TRANSFER_COEFFICIENT
- UNCONFINED
- VOL_BIO
- VOL_MAT
- WIDTH

Standard block:

GEOMETRY_DIMENSION
2,

STORAGE
[1, 1.0e-4],

PERMEABILITY_TENSOR
["ISOTROPIC", 1.0e-4],

POROSITY
[1, 0.2]

Keyword documentation:
https://ogs5-keywords.netlify.com/ogs/wiki/public/doc-auto/by_ext/mmp

Reading routines:
https://github.com/ufz/ogs5/blob/master/FEM/rf_mmp_new.cpp#L281

See also:

add_block

Attributes

block_no
Number of blocks in the file.

file_name
str: base name of the file with extension.

file_path
str: save path of the file.

force_writing
bool: state if the file is written even if empty.

is_empty
State if the OGS file is empty.

name
str: name of the file without extension.

Methods

add_block([index, main_key])	Add a new Block to the actual file.
add_content(content[, main_index, ...])	Add single-line content to the actual file.
add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
add_main_keyword(key[, main_index])	Add a new main keyword (#key) to the actual file.
add_multi_content(content[, main_index, ...])	Add multiple content to the actual file.
add_sub_keyword(key[, main_index, sub_index])	Add a new sub keyword ($key) to the actual file.
append_to_block([index])	Append data to an existing Block in the actual file.
check([verbose])	Check if the given file is valid.
del_block([index, del_all])	Delete a block by its index.
del_content([main_index, sub_index, ...])	Delete content by its position.
del_copy_link()	Remove a former given link to an external file.
del_main_keyword([main_index, del_all])	Delete a main keyword (#key) by its position.
del_sub_keyword([main_index, sub_index, del_all])	Delete a sub keyword ($key) by its position.
get_block([index, as_dict])	Get a Block from the actual file.
get_block_no()	Get the number of blocks in the file.
get_file_type()	Get the OGS file class name.
get_multi_keys([index])	State if a block has a unique set of sub keywords.
is_block_unique([index])	State if a block has a unique set of sub keywords.
read_file(path[, encoding, verbose])	Read an existing OGS input file.
reset()	Delete every content.
save(path, **kwargs)	Save the actual OGS input file in the given path.
update_block([index, main_key])	Update a Block from the actual file.
write_file()	Write the actual OGS input file to the given folder.

add_block(*index=None*, *main_key=None*, ***block*)

Add a new Block to the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 ...

$SUBKEY2
content2 ...

which looks like the following:

`FILE.add_block(SUBKEY1=content1, SUBKEY2=content2)`

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- **main_key** (*string, optional*) – Main keyword of the block that should be added (see: `MKEYS`) Default: the first main keyword of the file-type
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

add_content(*content*, *main_index=None*, *sub_index=None*, *line_index=None*)

Add single-line content to the actual file.

Parameters

- **content** (*list*) – list containing one line of content given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position, where the new line of content should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

---

add_copy_link(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied
- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

add_main_keyword(*key*, *main_index=None*)

Add a new main keyword (#key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – position, where the new main keyword should be added between the existing ones. As default, it is placed at the end.

add_multi_content(*content*, *main_index=None*, *sub_index=None*)

Add multiple content to the actual file.

Parameters

- **content** (*list*) – list containing lines of content, each given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

The content will be added at the end of the actual subkeyword.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

`add_sub_keyword`(*key*, *main_index=None*, *sub_index=None*)

Add a new sub keyword ($key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – position, where the new sub keyword should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the subkeyword is not added.

`append_to_block`(*index=None*, ***block*)

Append data to an existing Block in the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 …

$SUBKEY2
content2 …

which looks like the following:

```
FILE.append_to_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

`check`(*verbose=True*)

Check if the given file is valid.

Parameters
verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns
result – Validity of the given file.

Return type
bool

`del_block`(*index=None*, *del_all=False*)

Delete a block by its index.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **del_all** (*bool, optional*) – State, if all blocks shall be deleted. Default: False

`del_content`(*main_index=-1*, *sub_index=-1*, *line_index=-1*, *del_all=False*)

Delete content by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be deleted. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position of the content line, that should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all content shall be deleted. Default: False

`del_copy_link`()

Remove a former given link to an external file.

`del_main_keyword`(*main_index=None*, *del_all=False*)

Delete a main keyword (#key) by its position.

Parameters

- **main_index** (*int, optional*) – position, which main keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all main keywords shall be deleted. Default: False

`del_sub_keyword`(*main_index=-1*, *sub_index=-1*, *del_all=False*)

Delete a sub keyword ($key) by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **pos** (*int, optional*) – position, which sub keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all sub keywords shall be deleted. Default: False

`get_block`(*index=None*, *as_dict=True*)

Get a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None

- **as_dict** (*bool, optional*) – Here you can state of you want the output as a dictionary, which can be used as key-word-arguments for *add_block*. If False, you get the main-key, a list of sub-keys and a list of content. Default: True

`get_block_no()`

Get the number of blocks in the file.

`get_file_type()`

Get the OGS file class name.

`get_multi_keys(index=None)`

State if a block has a unique set of sub keywords.

`is_block_unique(index=None)`

State if a block has a unique set of sub keywords.

`read_file(path, encoding=None, verbose=False)`

Read an existing OGS input file.

Parameters

- **path** (*str*) – path to the existing file that should be read
- **encoding** (*str or None, optional*) – encoding of the given file. If `None` is given, the system standard is used. Default: `None`
- **verbose** (*bool, optional*) – Print information of the reading process. Default: False

`reset()`

Delete every content.

`save(path, **kwargs)`

Save the actual OGS input file in the given path.

Parameters

- **path** (*str*) – path to where to file should be saved
- **update** (*bool, optional*) – state if the content should be updated before saving. Default: True

`update_block(index=None, main_key=None, **block)`

Update a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is used. Default: None
- **main_key** (*string, optional*) – Main keyword of the block that should be updated (see: `MKEYS`) This shouldn't be done. Default: None
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

`write_file()`

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

`MKEYS = ['MEDIUM_PROPERTIES']`

Main Keywords of this OGS-BlockFile

Type
`list`

SKEYS = [['PCS_TYPE', 'NAME', 'GEO_TYPE', 'GEOMETRY_DIMENSION', 'GEOMETRY_INCLINATION', 'GEOMETRY_AREA', 'POROSITY', 'VOL_MAT', 'VOL_BIO', 'TORTUOSITY', 'FLOWLINEARITY', 'ORGANIC_CARBON', 'STORAGE', 'CONDUCTIVITY_MODEL', 'UNCONFINED', 'PERMEABILITY_TENSOR', 'PERMEABILITY_FUNCTION_DEFORMATION', 'PERMEABILITY_FUNCTION_STRAIN', 'PERMEABILITY_FUNCTION_PRESSURE', 'PERMEABILITY_SATURATION', 'PERMEABILITY_FUNCTION_STRESS', 'PERMEABILITY_FUNCTION_EFFSTRESS', 'PERMEABILITY_FUNCTION_VELOCITY', 'PERMEABILITY_FUNCTION_POROSITY', 'CAPILLARY_PRESSURE', 'TRANSFER_COEFFICIENT', 'SPECIFIC_STORAGE', 'STORAGE_FUNCTION_EFFSTRESS', 'MASS_DISPERSION', 'COMPOUND_DEPENDENT_DT', 'HEAT_DISPERSION', 'DIFFUSION', 'EVAPORATION', 'SURFACE_FRICTION', 'WIDTH', 'RILL', 'CHANNEL', 'PERMEABILITY_DISTRIBUTION', 'POROSITY_DISTRIBUTION', 'HEAT_TRANSFER', 'PARTICLE_DIAMETER', 'INTERPHASE_FRICTION', 'ELEMENT_VOLUME_MULTIPLYER', 'MEDIUM_TYPE', 'DENSITY']]

Sub Keywords of this OGS-BlockFile

Type
list

STD = {'GEOMETRY_DIMENSION': 2, 'PERMEABILITY_TENSOR': ['ISOTROPIC', 0.0001], 'POROSITY': [1, 0.2], 'STORAGE': [1, 0.0001]}

Standard Block OGS-BlockFile

Type
dict

property block_no

Number of blocks in the file.

property file_name

base name of the file with extension.

Type
str

property file_path

save path of the file.

Type
str

property force_writing

state if the file is written even if empty.

Type
bool

property is_empty

State if the OGS file is empty.

property name

name of the file without extension.

Type
str

ogs5py.fileclasses.MPD

class ogs5py.fileclasses.**MPD**(*name=None*, *file_ext='.mpd'*, ***OGS_Config*)

Bases: *BlockFile*

Class for the ogs MEDIUM_PROPERTIES_DISTRIBUTED file.

Parameters

- **name** (*str, optional*) – File name for the MPD file. If `None`, the task_id is used. Default: `None`
- **file_ext** (`str`, optional) – extension of the file (with leading dot ".mpd") Default: ".mpd"
- **task_root** (*str, optional*) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (*str, optional*) – Name for the ogs task. Default: "model"

Notes

Main-Keywords (#):

- MEDIUM_PROPERTIES_DISTRIBUTED

Sub-Keywords ($) per Main-Keyword:

- MEDIUM_PROPERTIES_DISTRIBUTED
  - MSH_TYPE
  - MMP_TYPE
  - DIS_TYPE
  - CONVERSION_FACTOR
  - DATA

Standard block:
None

Keyword documentation:
https://ogs5-keywords.netlify.com/ogs/wiki/public/doc-auto/by_ext/mmp

Reading routines:
https://github.com/ufz/ogs5/blob/master/FEM/rf_mmp_new.cpp#L5706

See also:

add_block

Attributes

block_no
Number of blocks in the file.

file_name
`str`: base name of the file with extension.

file_path
`str`: save path of the file.

force_writing
`bool`: state if the file is written even if empty.

is_empty
State if the OGS file is empty.

name
str: name of the file without extension.

top_com
Top comment is 'None' for the MPD file.

Methods

add_block([index, main_key])	Add a new Block to the actual file.
add_content(content[, main_index, ...])	Add single-line content to the actual file.
add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
add_main_keyword(key[, main_index])	Add a new main keyword (#key) to the actual file.
add_multi_content(content[, main_index, ...])	Add multiple content to the actual file.
add_sub_keyword(key[, main_index, sub_index])	Add a new sub keyword ($key) to the actual file.
append_to_block([index])	Append data to an existing Block in the actual file.
check([verbose])	Check if the given file is valid.
del_block([index, del_all])	Delete a block by its index.
del_content([main_index, sub_index, ...])	Delete content by its position.
del_copy_link()	Remove a former given link to an external file.
del_main_keyword([main_index, del_all])	Delete a main keyword (#key) by its position.
del_sub_keyword([main_index, sub_index, del_all])	Delete a sub keyword ($key) by its position.
get_block([index, as_dict])	Get a Block from the actual file.
get_block_no()	Get the number of blocks in the file.
get_file_type()	Get the OGS file class name.
get_multi_keys([index])	State if a block has a unique set of sub keywords.
is_block_unique([index])	State if a block has a unique set of sub keywords.
read_file(path[, encoding, verbose])	Read an existing OGS input file.
reset()	Delete every content.
save(path, **kwargs)	Save the actual OGS input file in the given path.
update_block([index, main_key])	Update a Block from the actual file.
write_file()	Write the actual OGS input file to the given folder.

add_block(*index=None*, *main_key=None*, ***block*)

Add a new Block to the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 ...

$SUBKEY2
content2 ...

which looks like the following:

```
FILE.add_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- **main_key** (*string, optional*) – Main keyword of the block that should be added (see: `MKEYS`) Default: the first main keyword of the file-type
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block

should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

`add_content`(*content*, *main_index=None*, *sub_index=None*, *line_index=None*)

Add single-line content to the actual file.

Parameters

- **content** (*list*) – list containing one line of content given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position, where the new line of content should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

`add_copy_link`(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied
- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

`add_main_keyword`(*key*, *main_index=None*)

Add a new main keyword (#key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – position, where the new main keyword should be added between the existing ones. As default, it is placed at the end.

`add_multi_content`(*content*, *main_index=None*, *sub_index=None*)

Add multiple content to the actual file.

Parameters

- **content** (*list*) – list containing lines of content, each given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

The content will be added at the end of the actual subkeyword.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

---

add_sub_keyword(*key*, *main_index=None*, *sub_index=None*)

Add a new sub keyword ($key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – position, where the new sub keyword should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the subkeyword is not added.

---

append_to_block(*index=None*, ***block*)

Append data to an existing Block in the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 ...

$SUBKEY2
content2 ...

which looks like the following:

```
FILE.append_to_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

check(*verbose=True*)

Check if the given file is valid.

Parameters
verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns
result – Validity of the given file.

Return type
bool

del_block(*index=None*, *del_all=False*)

Delete a block by its index.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **del_all** (*bool, optional*) – State, if all blocks shall be deleted. Default: False

`del_content`(*main_index=-1*, *sub_index=-1*, *line_index=-1*, *del_all=False*)

Delete content by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be deleted. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position of the content line, that should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all content shall be deleted. Default: False

`del_copy_link`()

Remove a former given link to an external file.

`del_main_keyword`(*main_index=None*, *del_all=False*)

Delete a main keyword (#key) by its position.

Parameters

- **main_index** (*int, optional*) – position, which main keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all main keywords shall be deleted. Default: False

`del_sub_keyword`(*main_index=-1*, *sub_index=-1*, *del_all=False*)

Delete a sub keyword ($key) by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **pos** (*int, optional*) – position, which sub keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all sub keywords shall be deleted. Default: False

`get_block`(*index=None*, *as_dict=True*)

Get a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **as_dict** (*bool, optional*) – Here you can state of you want the output as a dictionary, which can be used as key-word-arguments for *add_block*. If False, you get the main-key, a list of sub-keys and a list of content. Default: True

`get_block_no`()

Get the number of blocks in the file.

`get_file_type`()

Get the OGS file class name.

`get_multi_keys`(*index=None*)

State if a block has a unique set of sub keywords.

`is_block_unique`(*index=None*)

State if a block has a unique set of sub keywords.

`read_file`(*path*, *encoding=None*, *verbose=False*)

Read an existing OGS input file.

Parameters

- **path** (*str*) – path to the existing file that should be read
- **encoding** (*str or None, optional*) – encoding of the given file. If `None` is given, the system standard is used. Default: `None`
- **verbose** (*bool, optional*) – Print information of the reading process. Default: False

`reset`()

Delete every content.

`save`(*path*, ***kwargs*)

Save the actual OGS input file in the given path.

Parameters

- **path** (*str*) – path to where to file should be saved
- **update** (*bool, optional*) – state if the content should be updated before saving. Default: True

`update_block`(*index=None*, *main_key=None*, ***block*)

Update a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is used. Default: None
- **main_key** (*string, optional*) – Main keyword of the block that should be updated (see: `MKEYS`) This shouldn't be done. Default: None
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

`write_file`()

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

`MKEYS = ['MEDIUM_PROPERTIES_DISTRIBUTED']`

Main Keywords of this OGS-BlockFile

Type
`list`

`SKEYS = [['MSH_TYPE', 'MMP_TYPE', 'DIS_TYPE', 'CONVERSION_FACTOR', 'DATA']]`

Sub Keywords of this OGS-BlockFile

Type
`list`

`STD = {}`

Standard Block OGS-BlockFile

Type
`dict`

property block_no

Number of blocks in the file.

property file_name

base name of the file with extension.

Type
str

property file_path

save path of the file.

Type
str

property force_writing

state if the file is written even if empty.

Type
bool

property is_empty

State if the OGS file is empty.

property name

name of the file without extension.

Type
str

property top_com

Top comment is 'None' for the MPD file.

ogs5py.fileclasses.MSH

class ogs5py.fileclasses.**MSH**(*mesh_list=None*, ***OGS_Config*)

Bases: MSHsgl

Class for a multi layer mesh file that contains multiple '#FEM_MSH' Blocks.

Parameters

- **mesh_list** (*list of dict or None, optional*) – each dictionary contains one '#FEM_MSH' block of the mesh file with with the following information (sorted by keys):

  mesh_data
  [dict] dictionary containing information about

  - AXISYMMETRY (bool)
  - CROSS_SECTION (bool)
  - PCS_TYPE (str)
  - GEO_TYPE (str)
  - GEO_NAME (str)
  - LAYER (int)

  nodes
  [ndarray] Array with all node postions

  elements
  [dict] contains nodelists for elements sorted by element types

  material_id
  [dict] contains material ids for each element sorted by element types

  element_id
  [dict] contains element ids for each element sorted by element types

- **task_root** (*str, optional*) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (*str, optional*) – Name for the ogs task. Default: "model"

Attributes

AXISYMMETRY
bool: AXISYMMETRY attribute.

CROSS_SECTION
bool: CROSS_SECTION attribute.

ELEMENTS
Get and set the ELEMENTS of the mesh.

ELEMENT_ID
Get and set the ELEMENT_IDs of the mesh.

ELEMENT_NO
int: number of ELEMENTS.

ELEMENT_TYPES
set: ELEMENT types of the mesh.

GEO_NAME
str: GEO_NAME.

GEO_TYPE
str: GEO_TYPE.

LAYER
int: LAYER.

MATERIAL_ID
Get and set the MATERIAL_IDs of the mesh.

MATERIAL_ID_flat
Get flat version of the MATERIAL_IDs of the mesh.

NODES
ndarray: (n,3) NODES of the mesh by its xyz-coordinates.

NODE_NO
int: number of NODES.

PCS_TYPE
str: PCS_TYPE.

block
int: The actual block to access in the file.

block_no
int: The number of blocks in the file.

center
Get the mesh center.

centroids
Get the centroids of the mesh.

centroids_flat
Get flat version of the centroids of the mesh.

file_name
str: base name of the file with extension.

file_path
str: save path of the file.

force_writing
bool: state if the file is written even if empty.

is_empty
State if file is empty.

name
str: name of the file without extension.

node_centroids
Get the node centroids of the mesh.

node_centroids_flat
Get flat version of the node centroids of the mesh.

volumes
Get the volumes of the mesh-elements.

volumes_flat
Get flat version of the volumes of the mesh-elements.

Methods

__call__()	Get a copy of the underlying dictionary.
add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
check([verbose])	Check if the mesh is valid.
combine_mesh(ext_mesh, **kwargs)	Combine this mesh with an external mesh.
del_copy_link()	Remove a former given link to an external file.
export_mesh(filepath[, verbose])	Export the mesh to an unstructured mesh in diffrent file-formats.
generate([generator])	Use a mesh-generator from the generator module.
get_file_type()	Get the OGS file class name.
import_mesh(filepath, **kwargs)	Import an external unstructured mesh from diffrent file-formats.
load(filepath, **kwargs)	Load an OGS5 mesh from file.
read_file(path[, encoding, verbose])	Load an OGS5 mesh from file.
remove_dim(remove)	Remove elements by given dimensions from a mesh.
reset()	Delete every content.
rotate(angle[, rotation_axis, rotation_point])	Rotate a given mesh around a given rotation axis with a given angle.
save(path, **kwargs)	Save the mesh to an OGS5 mesh file.
set_dict(mesh_dict)	Set an mesh as returned by tools methods or generators.
set_material_id([material_id, element_id, ...])	Set material IDs by the corresponding element IDs.
shift(vector)	Shift a given mesh with a given vector.
show([show_cell_data, show_material_id, ...])	Display the mesh colored by its material ID.
swap_axis([axis1, axis2])	Swap axis of the coordinate system.
transform(xyz_func, **kwargs)	Transform a given mesh with a given function "xyz_func".
write_file()	Write the actual OGS input file to the given folder.

__call__()

Get a copy of the underlying dictionary.

add_copy_link(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied
- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

check(*verbose=True*)

Check if the mesh is valid.

Checked in the sence, that the contained data is consistent. Checks for correct element definitions or Node duplicates are not carried out.

Parameters

verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns

result – Validity of the given mesh.

Return type

bool

combine_mesh(*ext_mesh*, ***kwargs*)

Combine this mesh with an external mesh.

The node list will be updated to eliminate duplicates. Element intersections are not checked. kwargs will be forwarded to "tools.combine"

Parameters

- **ext_mesh** (*mesh, dict or file*) – This is the mesh that should be added to the existing one.
- **decimals** (*int, optional*) – Number of decimal places to round the nodes to (default: 3). This will not round the output, it is just for comparison of the node vectors.
- **fast** (*bool, optional*) – If fast is True, the vector comparison is executed by a decimal comparison. If fast is False, all pairwise distances are calculated. Default: False

del_copy_link()

Remove a former given link to an external file.

export_mesh(*filepath*, *verbose=False*, ***kwargs*)

Export the mesh to an unstructured mesh in diffrent file-formats.

kwargs will be forwarded to "tools.export_mesh"

Parameters

- **filepath** (*string*) – path to the file to export
- **file_format** (*str, optional*) – Here you can specify the fileformat. If 'None' it will be determined by file extension. Default: None
- **export_material_id** (*bool, optional*) – Here you can specify if the material_id should be exported. Default: True
- **export_element_id** (*bool, optional*) – Here you can specify if the element_id should be exported. Default: True
- **cell_data_by_id** (*ndarray or dict, optional*) – Here you can specify additional element data sorted by their IDs. It can be a dictionary with data-name as key and the ndarray as value. Default: None
- **point_data** (*ndarray or dict, optional*) – Here you can specify additional point data sorted by their IDs. It can be a dictionary with data-name as key and the ndarray as value. Default: None
- **field_data** (*ndarray or dict, optional*) – Here you can specify additional field data of the mesh. It can be a dictionary with data-name as key and the ndarray as value. Default: None

---

Notes

This routine calls the 'write' function from the meshio package and converts the input (see here: https://github.com/nschloe/meshio)

---

generate(*generator='rectangular'*, ***kwargs*)

Use a mesh-generator from the generator module.

See: *ogs5py.fileclasses.msh.generator*

Parameters

- **generator** (*str*) – set the generator from the generator module
- ****kwargs** – kwargs will be forwarded to the generator in use

Notes

The following generators are available:

rectangular([dim, mesh_origin, element_no, ...])	Generate a rectangular grid in 2D or 3D.
radial([dim, mesh_origin, angles, rad, z_arr])	Generate a radial grid in 2D or 3D.
grid_adapter2D([out_dim, in_dim, out_res, ...])	Generate a grid adapter.
grid_adapter3D([out_dim, in_dim, z_dim, ...])	Generate a grid adapter.
block_adapter3D([xy_dim, z_dim, in_res])	Generate a block adapter.
gmsh(geo_object[, import_dim])	Generate mesh from gmsh code or gmsh .geo file.

`get_file_type()`

Get the OGS file class name.

`import_mesh(`*filepath*, ***kwargs*`)`

Import an external unstructured mesh from diffrent file-formats.

kwargs will be forwarded to "tools.import_mesh"

Parameters

- **filepath** (*string or meshio.Mesh instance*) – mesh to import
- **file_format** (*str, optional*) – Here you can specify the fileformat. If 'None' it will be determined by file extension. Default: None
- **ignore_unknown** (*bool, optional*) – Unknown data in the file will be ignored. Default: False
- **import_dim** (*iterable of int, optional*) – State which elements should be imported by dimensionality. Can be used to sort out unneeded elements for example from gmsh. Default: (1, 2, 3)

Notes

This routine calls the 'read' function from the meshio package and converts the output (see here: https://github.com/nschloe/meshio) If there is any "vertex" (0D element) in the element data, it will be removed.

`load(`*filepath*, ***kwargs*`)`

Load an OGS5 mesh from file.

kwargs will be forwarded to "tools.load_ogs5msh"

Parameters

- **filepath** (*string*) – path to the '*.msh' OGS5 mesh file to load
- **verbose** (*bool, optional*) – Print information of the reading process. Default: True
- **ignore_unknown** (*bool, optional*) – Unknown data in the file will be ignored. Default: False
- **max_node_no** (*int, optional*) – If you know the maximal node number per elements in the mesh file, you can optimise the reading a bit. By default the algorithm will assume hexahedrons as 'largest' elements in the mesh. Default: 8

- **encoding** (*str or None, optional*) – encoding of the given file. If `None` is given, the system standard is used. Default: `None`

Notes

The $AREA keyword within the Nodes definition is NOT supported and will violate the read data if present.

`read_file`(*path*, *encoding=None*, *verbose=False*)

Load an OGS5 mesh from file.

Parameters

- **path** (*str*) – path to the '*.msh' OGS5 mesh file to load
- **encoding** (*str or None, optional*) – encoding of the given file. If `None` is given, the system standard is used. Default: `None`
- **verbose** (*bool, optional*) – Print information of the reading process. Default: True

`remove_dim`(*remove*)

Remove elements by given dimensions from a mesh.

Parameters

remove (*iterable of int or single int*) – State which elements should be removed by dimensionality (1, 2, 3).

`reset`()

Delete every content.

`rotate`(*angle*, *rotation_axis=(0.0, 0.0, 1.0)*, *rotation_point=(0.0, 0.0, 0.0)*)

Rotate a given mesh around a given rotation axis with a given angle.

Parameters

- **angle** (*float*) – rotation angle given in radial length
- **rotation_axis** (*array_like, optional*) – Array containing the vector for ratation axis. Default: (0,0,1)
- **rotation_point** (*array_like, optional*) – Vector of the ratation base point. Default:(0,0,0)

`save`(*path*, ***kwargs*)

Save the mesh to an OGS5 mesh file.

kwargs will be forwarded to "tools.save_ogs5msh"

Parameters

- **path** (*string*) – path to the '*.msh' OGS5 mesh file to save
- **verbose** (*bool, optional*) – Print information of the writing process. Default: True

`set_dict`(*mesh_dict*)

Set an mesh as returned by tools methods or generators.

Mesh will be checked for validity.

Parameters

mesh_dict (*dict or None, optional*) – Contains one '#FEM_MSH' block of an OGS5 mesh file with the following information (sorted by keys):

mesh_data

[dict] dictionary containing information about

- AXISYMMETRY (bool)

- CROSS_SECTION (bool)
- PCS_TYPE (str)
- GEO_TYPE (str)
- GEO_NAME (str)
- LAYER (int)

nodes
[ndarray] Array with all node postions

elements
[dict] contains nodelists for elements sorted by element types

material_id
[dict] contains material ids for each element sorted by types

element_id
[dict] contains element ids for each element sorted by types

`set_material_id`(*material_id=0*, *element_id=None*, *element_mask=None*)

Set material IDs by the corresponding element IDs.

Parameters

- **material_id** (`int` or ndarray, optional) – The new material IDs. Either one value or an array. Default: 0
- **element_id** (*ndarray or None, optional*) – The corresponding element IDs, where to set the material IDs. If None, all elements are assumed and the material IDs are added by their index. Default: None
- **element_mask** (*ndarray or None, optional*) – Instead of the element IDs, one can specify a mask to select the element IDs. Default: None

`shift`(*vector*)

Shift a given mesh with a given vector.

Parameters
vector (*ndarray*) – array containing the shifting vector

`show`(*show_cell_data=None*, *show_material_id=False*, *show_element_id=False*, *log_scale=False*)

Display the mesh colored by its material ID.

Parameters

- **show_cell_data** (*ndarray or dict, optional*) – Here you can specify additional element/cell data sorted by their IDs. It can be a dictionary with data-name as key and the ndarray as value. Default: None
- **show_material_id** (*bool, optional*) – Here you can specify if the material_id should be shown. Default: False
- **show_element_id** (*bool, optional*) – Here you can specify if the element_id should be shown. Default: False
- **log_scale** (*bool, optional*) – State if the cell_data should be shown in log scale. Default: False

---

Notes

This routine needs "mayavi" to display the mesh. (see here: https://github.com/enthought/mayavi)

`swap_axis`(*axis1='y'*, *axis2='z'*)

Swap axis of the coordinate system.

Parameters

- **axis1** (`str` or `int`, optional) – First selected Axis. Either in ["x", "y", "z"] or in [0, 1, 2]. Default: "y"
- **axis2** (`str` or `int`, optional) – Second selected Axis. Either in ["x", "y", "z"] or in [0, 1, 2]. Default: "z"

`transform`(*xyz_func*, ***kwargs*)

Transform a given mesh with a given function "xyz_func".

kwargs will be forwarded to "xyz_func".

Parameters

xyz_func (*function*) – the function transforming the points: `x_new, y_new, z_new = f(x_old, y_old, z_old, **kwargs)`

`write_file`()

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

`property AXISYMMETRY`

AXISYMMETRY attribute.

Type

bool

`property CROSS_SECTION`

CROSS_SECTION attribute.

Type

bool

`property ELEMENTS`

Get and set the ELEMENTS of the mesh.

Notes

Type

[dict of ndarrays] The elements are a dictionary sorted by their element-type

"line"

[ndarray of shape (n_line,2)] 1D element with 2 nodes

"tri"

[ndarray of shape (n_tri,3)] 2D element with 3 nodes

"quad"

[ndarray of shape (n_quad,4)] 2D element with 4 nodes

"tet"

[ndarray of shape (n_tet,4)] 3D element with 4 nodes

"pyra"

[ndarray of shape (n_pyra,5)] 3D element with 5 nodes

"pris"

[ndarray of shape (n_pris,6)] 3D element with 6 nodes

"hex"

[ndarray of shape (n_hex,8)] 3D element with 8 nodes

---

property ELEMENT_ID

Get and set the ELEMENT_IDs of the mesh.

Standard element id order is given by:

"line" "tri" "quad" "tet" "pyra" "pris" "hex"

Notes

Type

[dict of ndarrays] The element IDs are a dictionary containing ints sorted by their element-type

"line"
[ndarray of shape (n_line,)] 1D element with 2 nodes

"tri"
[ndarray of shape (n_tri,)] 2D element with 3 nodes

"quad"
[ndarray of shape (n_quad,)] 2D element with 4 nodes

"tet"
[ndarray of shape (n_tet,)] 3D element with 4 nodes

"pyra"
[ndarray of shape (n_pyra,)] 3D element with 5 nodes

"pris"
[ndarray of shape (n_pris,)] 3D element with 6 nodes

"hex"
[ndarray of shape (n_hex,)] 3D element with 8 nodes

---

property ELEMENT_NO

number of ELEMENTS.

Type
int

property ELEMENT_TYPES

ELEMENT types of the mesh.

Type
set

property GEO_NAME

GEO_NAME.

Type
str

property GEO_TYPE

GEO_TYPE.

Type
str

property LAYER

LAYER.

Type
int

property MATERIAL_ID

Get and set the MATERIAL_IDs of the mesh.

Notes

Type

[dict of ndarrays] The material IDs are a dictionary containing ints sorted by their element-type

"line"

[ndarray of shape (n_line,)] 1D element with 2 nodes

"tri"

[ndarray of shape (n_tri,)] 2D element with 3 nodes

"quad"

[ndarray of shape (n_quad,)] 2D element with 4 nodes

"tet"

[ndarray of shape (n_tet,)] 3D element with 4 nodes

"pyra"

[ndarray of shape (n_pyra,)] 3D element with 5 nodes

"pris"

[ndarray of shape (n_pris,)] 3D element with 6 nodes

"hex"

[ndarray of shape (n_hex,)] 3D element with 8 nodes

---

property MATERIAL_ID_flat

Get flat version of the MATERIAL_IDs of the mesh.

See "mesh.MATERIAL_ID" This flattend MATERIAL_IDs are a stacked version of MATERIAL_ID, to get one continous array. They are stacked in order of the ELEMENT_IDs. Standard stack order is given by:

"line" "tri" "quad" "tet" "pyra" "pris" "hex"

Notes

Type

[ndarray] The centroids are a list containing xyz-coordiantes

---

property NODES

(n,3) NODES of the mesh by its xyz-coordinates.

Type

ndarray

property NODE_NO

number of NODES.

Type

int

property PCS_TYPE

PCS_TYPE.

Type

str

property block

The actual block to access in the file.

Type
int

property block_no

The number of blocks in the file.

Type
int

property center

Get the mesh center.

property centroids

Get the centroids of the mesh.

Notes

Type
[dict of ndarrays] The centroids are a dictionary containing xyz-coordiantes sorted by their element-type

"line"
[ndarray of shape (n_line,3)] 1D element with 2 nodes

"tri"
[ndarray of shape (n_tri,3)] 2D element with 3 nodes

"quad"
[ndarray of shape (n_quad,3)] 2D element with 4 nodes

"tet"
[ndarray of shape (n_tet,3)] 3D element with 4 nodes

"pyra"
[ndarray of shape (n_pyra,3)] 3D element with 5 nodes

"pris"
[ndarray of shape (n_pris,3)] 3D element with 6 nodes

"hex"
[ndarray of shape (n_hex,3)] 3D element with 8 nodes

---

property centroids_flat

Get flat version of the centroids of the mesh.

See the "mesh.get_centroids" method. This flattend centroids are a stacked version of centroids, to get one continous array. They are stacked in order of the element ids. Standard stack order is given by:

"line" "tri" "quad" "tet" "pyra" "pris" "hex"

Notes

Type
[ndarray] The centroids are a list containing xyz-coordiantes

---

property file_name

base name of the file with extension.

Type
str

property file_path

save path of the file.

Type
str

property force_writing

state if the file is written even if empty.

Type
bool

property is_empty

State if file is empty.

property name

name of the file without extension.

Type
str

property node_centroids

Get the node centroids of the mesh.

Notes

Type
[dict of ndarrays] The centroids are a dictionary containing xyz-coordiantes sorted by their element-type

"line"
[ndarray of shape (n_line,3)] 1D element with 2 nodes

"tri"
[ndarray of shape (n_tri,3)] 2D element with 3 nodes

"quad"
[ndarray of shape (n_quad,3)] 2D element with 4 nodes

"tet"
[ndarray of shape (n_tet,3)] 3D element with 4 nodes

"pyra"
[ndarray of shape (n_pyra,3)] 3D element with 5 nodes

"pris"
[ndarray of shape (n_pris,3)] 3D element with 6 nodes

"hex"
[ndarray of shape (n_hex,3)] 3D element with 8 nodes

---

property node_centroids_flat

Get flat version of the node centroids of the mesh.

See the "mesh.get_centroids" method. This flattend centroids are a stacked version of centroids, to get one continous array. They are stacked in order of the element ids. Standard stack order is given by:

"line" "tri" "quad" "tet" "pyra" "pris" "hex"

Notes

Type
[ndarray] The centroids are a list containing xyz-coordiantes

---

`property volumes`

Get the volumes of the mesh-elements.

Notes

Type

[dict of ndarrays] The volumes are a dictionary containing the n-dimension volumes sorted by their element-type

"line"
[ndarray of shape (n_line,3)] 1D element with 2 nodes

"tri"
[ndarray of shape (n_tri,3)] 2D element with 3 nodes

"quad"
[ndarray of shape (n_quad,3)] 2D element with 4 nodes

"tet"
[ndarray of shape (n_tet,3)] 3D element with 4 nodes

"pyra"
[ndarray of shape (n_pyra,3)] 3D element with 5 nodes

"pris"
[ndarray of shape (n_pris,3)] 3D element with 6 nodes

"hex"
[ndarray of shape (n_hex,3)] 3D element with 8 nodes

`property volumes_flat`

Get flat version of the volumes of the mesh-elements.

This flattend volumes are a stacked version of centroids, to get one continous array. They are stacked in order of the element ids. Standard stack order is given by:

"line" "tri" "quad" "tet" "pyra" "pris" "hex"

Notes

Type

[ndarray] The volumes are a list containing the n-dimensional element volume

---

ogs5py.fileclasses.MSP

class ogs5py.fileclasses.**MSP**(***OGS_Config*)

Bases: *BlockFile*

Class for the ogs SOLID_PROPERTIES file.

Parameters

- **task_root** (*str, optional*) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (*str, optional*) – Name for the ogs task. Default: "model"

Notes

Main-Keywords (#):

- SOLID_PROPERTIES

Sub-Keywords ($) per Main-Keyword:

- SOLID_PROPERTIES
  - BIOT_CONSTANT
  - CREEP
  - DENSITY
  - ELASTICITY
  - EXCAVATION
  - E_Function
  - GRAVITY_CONSTANT
  - MICRO_STRUCTURE_PLAS
  - NAME
  - NON_REACTIVE_FRACTION
  - PLASTICITY
  - REACTIVE_SYSTEM
  - SOLID_BULK_MODULUS
  - SPECIFIC_HEAT_SOURCE
  - STRESS_INTEGRATION_TOLERANCE
  - STRESS_UNIT
  - SWELLING_PRESSURE_TYPE
  - THERMAL
  - THRESHOLD_DEV_STR
  - TIME_DEPENDENT_YOUNGS_POISSON

Standard block:
None

Keyword documentation:
https://ogs5-keywords.netlify.com/ogs/wiki/public/doc-auto/by_ext/msp

Reading routines:
https://github.com/ufz/ogs5/blob/master/FEM/rf_msp_new.cpp#L65

See also:

add_block

Attributes

block_no
Number of blocks in the file.

file_name
str: base name of the file with extension.

file_path
str: save path of the file.

force_writing
bool: state if the file is written even if empty.

is_empty
State if the OGS file is empty.

name
str: name of the file without extension.

Methods

add_block([index, main_key])	Add a new Block to the actual file.
add_content(content[, main_index, ...])	Add single-line content to the actual file.
add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
add_main_keyword(key[, main_index])	Add a new main keyword (#key) to the actual file.
add_multi_content(content[, main_index, ...])	Add multiple content to the actual file.
add_sub_keyword(key[, main_index, sub_index])	Add a new sub keyword ($key) to the actual file.
append_to_block([index])	Append data to an existing Block in the actual file.
check([verbose])	Check if the given file is valid.
del_block([index, del_all])	Delete a block by its index.
del_content([main_index, sub_index, ...])	Delete content by its position.
del_copy_link()	Remove a former given link to an external file.
del_main_keyword([main_index, del_all])	Delete a main keyword (#key) by its position.
del_sub_keyword([main_index, sub_index, del_all])	Delete a sub keyword ($key) by its position.
get_block([index, as_dict])	Get a Block from the actual file.
get_block_no()	Get the number of blocks in the file.
get_file_type()	Get the OGS file class name.
get_multi_keys([index])	State if a block has a unique set of sub keywords.
is_block_unique([index])	State if a block has a unique set of sub keywords.
read_file(path[, encoding, verbose])	Read an existing OGS input file.
reset()	Delete every content.
save(path, **kwargs)	Save the actual OGS input file in the given path.
update_block([index, main_key])	Update a Block from the actual file.
write_file()	Write the actual OGS input file to the given folder.

add_block(*index=None*, *main_key=None*, ***block*)

Add a new Block to the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 ...

$SUBKEY2
content2 ...

which looks like the following:

```
FILE.add_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- **main_key** (*string, optional*) – Main keyword of the block that should be added (see: `MKEYS`) Default: the first main keyword of the file-type
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

add_content(*content*, *main_index=None*, *sub_index=None*, *line_index=None*)

Add single-line content to the actual file.

Parameters

- **content** (*list*) – list containing one line of content given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position, where the new line of content should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

---

add_copy_link(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied
- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

add_main_keyword(*key*, *main_index=None*)

Add a new main keyword (#key) to the actual file.

Parameters

- **key** (*string*) – key name

- **main_index** (*int, optional*) – position, where the new main keyword should be added between the existing ones. As default, it is placed at the end.

add_multi_content(*content*, *main_index=None*, *sub_index=None*)

Add multiple content to the actual file.

Parameters

- **content** (*list*) – list containing lines of content, each given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

The content will be added at the end of the actual subkeyword.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

---

add_sub_keyword(*key*, *main_index=None*, *sub_index=None*)

Add a new sub keyword ($key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – position, where the new sub keyword should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the subkeyword is not added.

---

append_to_block(*index=None*, ***block*)

Append data to an existing Block in the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 …

$SUBKEY2
content2 …

which looks like the following:

```
FILE.append_to_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.

- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

`check`(*verbose=True*)

Check if the given file is valid.

Parameters
verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns
result – Validity of the given file.

Return type
bool

`del_block`(*index=None*, *del_all=False*)

Delete a block by its index.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **del_all** (*bool, optional*) – State, if all blocks shall be deleted. Default: False

`del_content`(*main_index=-1*, *sub_index=-1*, *line_index=-1*, *del_all=False*)

Delete content by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be deleted. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position of the content line, that should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all content shall be deleted. Default: False

`del_copy_link`()

Remove a former given link to an external file.

`del_main_keyword`(*main_index=None*, *del_all=False*)

Delete a main keyword (#key) by its position.

Parameters

- **main_index** (*int, optional*) – position, which main keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all main keywords shall be deleted. Default: False

`del_sub_keyword`(*main_index=-1*, *sub_index=-1*, *del_all=False*)

Delete a sub keyword ($key) by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **pos** (*int, optional*) – position, which sub keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all sub keywords shall be deleted. Default: False

`get_block`(*index=None*, *as_dict=True*)

Get a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **as_dict** (*bool, optional*) – Here you can state of you want the output as a dictionary, which can be used as key-word-arguments for *add_block*. If False, you get the main-key, a list of sub-keys and a list of content. Default: True

`get_block_no`()

Get the number of blocks in the file.

`get_file_type`()

Get the OGS file class name.

`get_multi_keys`(*index=None*)

State if a block has a unique set of sub keywords.

`is_block_unique`(*index=None*)

State if a block has a unique set of sub keywords.

`read_file`(*path*, *encoding=None*, *verbose=False*)

Read an existing OGS input file.

Parameters

- **path** (*str*) – path to the existing file that should be read
- **encoding** (*str or None, optional*) – encoding of the given file. If `None` is given, the system standard is used. Default: `None`
- **verbose** (*bool, optional*) – Print information of the reading process. Default: False

`reset`()

Delete every content.

`save`(*path*, ***kwargs*)

Save the actual OGS input file in the given path.

Parameters

- **path** (*str*) – path to where to file should be saved
- **update** (*bool, optional*) – state if the content should be updated before saving. Default: True

`update_block`(*index=None*, *main_key=None*, ***block*)

Update a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is used. Default: None
- **main_key** (*string, optional*) – Main keyword of the block that should be updated (see: `MKEYS`) This shouldn't be done. Default: None
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

write_file()

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

MKEYS = ['SOLID_PROPERTIES']

Main Keywords of this OGS-BlockFile

Type
list

SKEYS = [['NAME', 'SWELLING_PRESSURE_TYPE', 'DENSITY', 'THERMAL', 'ELASTICITY', 'EXCAVATION', 'E_Function', 'TIME_DEPENDENT_YOUNGS_POISSON', 'CREEP', 'THRESHOLD_DEV_STR', 'BIOT_CONSTANT', 'SOLID_BULK_MODULUS', 'STRESS_INTEGRATION_TOLERANCE', 'STRESS_UNIT', 'GRAVITY_CONSTANT', 'GRAVITY_RAMP', 'PLASTICITY', 'REACTIVE_SYSTEM', 'NON_REACTIVE_FRACTION', 'SPECIFIC_HEAT_SOURCE', 'ENTHALPY_CORRECTION_REFERENCE_TEMPERATURE', 'MICRO_STRUCTURE_PLAS']]

Sub Keywords of this OGS-BlockFile

Type
list

STD = {}

Standard Block OGS-BlockFile

Type
dict

property block_no

Number of blocks in the file.

property file_name

base name of the file with extension.

Type
str

property file_path

save path of the file.

Type
str

property force_writing

state if the file is written even if empty.

Type
bool

property is_empty

State if the OGS file is empty.

property name

name of the file without extension.

Type
str

ogs5py.fileclasses.NUM

class ogs5py.fileclasses.**NUM**(***OGS_Config*)

Bases: *BlockFile*

Class for the ogs NUMERICS file.

Parameters

- **task_root** (*str, optional*) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (*str, optional*) – Name for the ogs task. Default: "model"

Notes

Main-Keywords (#):

- NUMERICS

Sub-Keywords ($) per Main-Keyword:

- NUMERICS
  - COUPLED_PROCESS
  - COUPLING_CONTROL
  - COUPLING_ITERATIONS
  - DYNAMIC_DAMPING
  - ELE_GAUSS_POINTS
  - ELE_MASS_LUMPING
  - ELE_SUPG
  - ELE_UPWINDING
  - EXTERNAL_SOLVER_OPTION
  - FEM_FCT
  - GRAVITY_PROFILE
  - LINEAR_SOLVER
  - LOCAL_PICARD
  - NON_LINEAR_ITERATION
  - NON_LINEAR_SOLVER
  - NON_LINEAR_UPDATE_VELOCITY
  - OVERALL_COUPLING
  - PCS_TYPE
  - PLASTICITY_TOLERANCE
  - RENUMBER

Standard block:

PCS_TYPE
"GROUNDWATER_FLOW"

LINEAR_SOLVER
[2, 5, 1.0e-14, 1000, 1.0, 100, 4]

Keyword documentation:
https://ogs5-keywords.netlify.com/ogs/wiki/public/doc-auto/by_ext/num

Reading routines:
https://github.com/ufz/ogs5/blob/master/FEM/rf_num_new.cpp#L346

See also:

add_block

Attributes

block_no
Number of blocks in the file.

file_name
str: base name of the file with extension.

file_path
str: save path of the file.

force_writing
bool: state if the file is written even if empty.

is_empty
State if the OGS file is empty.

name
str: name of the file without extension.

Methods

add_block([index, main_key])	Add a new Block to the actual file.
add_content(content[, main_index, ...])	Add single-line content to the actual file.
add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
add_main_keyword(key[, main_index])	Add a new main keyword (#key) to the actual file.
add_multi_content(content[, main_index, ...])	Add multiple content to the actual file.
add_sub_keyword(key[, main_index, sub_index])	Add a new sub keyword ($key) to the actual file.
append_to_block([index])	Append data to an existing Block in the actual file.
check([verbose])	Check if the given file is valid.
del_block([index, del_all])	Delete a block by its index.
del_content([main_index, sub_index, ...])	Delete content by its position.
del_copy_link()	Remove a former given link to an external file.
del_main_keyword([main_index, del_all])	Delete a main keyword (#key) by its position.
del_sub_keyword([main_index, sub_index, del_all])	Delete a sub keyword ($key) by its position.
get_block([index, as_dict])	Get a Block from the actual file.
get_block_no()	Get the number of blocks in the file.
get_file_type()	Get the OGS file class name.
get_multi_keys([index])	State if a block has a unique set of sub keywords.
is_block_unique([index])	State if a block has a unique set of sub keywords.
read_file(path[, encoding, verbose])	Read an existing OGS input file.
reset()	Delete every content.
save(path, **kwargs)	Save the actual OGS input file in the given path.
update_block([index, main_key])	Update a Block from the actual file.
write_file()	Write the actual OGS input file to the given folder.

add_block(*index=None*, *main_key=None*, ***block*)

Add a new Block to the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 ...

$SUBKEY2
content2 ...

which looks like the following:

```
FILE.add_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- **main_key** (*string, optional*) – Main keyword of the block that should be added (see: `MKEYS`) Default: the first main keyword of the file-type
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

add_content(*content*, *main_index=None*, *sub_index=None*, *line_index=None*)

Add single-line content to the actual file.

Parameters

- **content** (*list*) – list containing one line of content given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position, where the new line of content should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

---

add_copy_link(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied
- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

add_main_keyword(*key*, *main_index=None*)

Add a new main keyword (#key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – position, where the new main keyword should be added between the existing ones. As default, it is placed at the end.

add_multi_content(*content*, *main_index=None*, *sub_index=None*)

Add multiple content to the actual file.

Parameters

- **content** (*list*) – list containing lines of content, each given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

The content will be added at the end of the actual subkeyword.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

---

add_sub_keyword(*key*, *main_index=None*, *sub_index=None*)

Add a new sub keyword ($key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – position, where the new sub keyword should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the subkeyword is not added.

---

append_to_block(*index=None*, ***block*)

Append data to an existing Block in the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 …

$SUBKEY2
content2 …

which looks like the following:

```
FILE.append_to_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

`check`(*verbose=True*)

Check if the given file is valid.

Parameters
verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns
result – Validity of the given file.

Return type
bool

`del_block`(*index=None*, *del_all=False*)

Delete a block by its index.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **del_all** (*bool, optional*) – State, if all blocks shall be deleted. Default: False

`del_content`(*main_index=-1*, *sub_index=-1*, *line_index=-1*, *del_all=False*)

Delete content by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be deleted. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position of the content line, that should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all content shall be deleted. Default: False

`del_copy_link`()

Remove a former given link to an external file.

`del_main_keyword`(*main_index=None*, *del_all=False*)

Delete a main keyword (#key) by its position.

Parameters

- **main_index** (*int, optional*) – position, which main keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all main keywords shall be deleted. Default: False

`del_sub_keyword`(*main_index=-1*, *sub_index=-1*, *del_all=False*)

Delete a sub keyword ($key) by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.

- **pos** (*int, optional*) – position, which sub keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all sub keywords shall be deleted. Default: False

`get_block`(*index=None*, *as_dict=True*)

Get a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **as_dict** (*bool, optional*) – Here you can state of you want the output as a dictionary, which can be used as key-word-arguments for *add_block*. If False, you get the main-key, a list of sub-keys and a list of content. Default: True

`get_block_no`()

Get the number of blocks in the file.

`get_file_type`()

Get the OGS file class name.

`get_multi_keys`(*index=None*)

State if a block has a unique set of sub keywords.

`is_block_unique`(*index=None*)

State if a block has a unique set of sub keywords.

`read_file`(*path*, *encoding=None*, *verbose=False*)

Read an existing OGS input file.

Parameters

- **path** (*str*) – path to the existing file that should be read
- **encoding** (*str or None, optional*) – encoding of the given file. If `None` is given, the system standard is used. Default: `None`
- **verbose** (*bool, optional*) – Print information of the reading process. Default: False

`reset`()

Delete every content.

`save`(*path*, ***kwargs*)

Save the actual OGS input file in the given path.

Parameters

- **path** (*str*) – path to where to file should be saved
- **update** (*bool, optional*) – state if the content should be updated before saving. Default: True

`update_block`(*index=None*, *main_key=None*, ***block*)

Update a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is used. Default: None
- **main_key** (*string, optional*) – Main keyword of the block that should be updated (see: `MKEYS`) This shouldn't be done. Default: None
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

write_file()

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

MKEYS = ['NUMERICS']

Main Keywords of this OGS-BlockFile

Type
list

SKEYS = [['PCS_TYPE', 'RENUMBER', 'PLASTICITY_TOLERANCE', 'NON_LINEAR_ITERATION', 'NON_LINEAR_SOLVER', 'LINEAR_SOLVER', 'OVERALL_COUPLING', 'COUPLING_ITERATIONS', 'COUPLING_CONTROL', 'COUPLED_PROCESS', 'EXTERNAL_SOLVER_OPTION', 'ELE_GAUSS_POINTS', 'ELE_MASS_LUMPING', 'ELE_UPWINDING', 'ELE_SUPG', 'GRAVITY_PROFILE', 'DYNAMIC_DAMPING', 'LOCAL_PICARD1', 'NON_LINEAR_UPDATE_VELOCITY', 'FEM_FCT', 'NEWTON_DAMPING', 'ADDITIONAL_NEWTON_TOLERANCES', 'REACTION_SCALING', 'METHOD']]

Sub Keywords of this OGS-BlockFile

Type
list

STD = {'LINEAR_SOLVER': [2, 5, 1e-14, 1000, 1.0, 100, 4], 'PCS_TYPE': 'GROUNDWATER_FLOW'}

Standard Block OGS-BlockFile

Type
dict

property block_no

Number of blocks in the file.

property file_name

base name of the file with extension.

Type
str

property file_path

save path of the file.

Type
str

property force_writing

state if the file is written even if empty.

Type
bool

property is_empty

State if the OGS file is empty.

property name

name of the file without extension.

Type
str

ogs5py.fileclasses.OUT

class ogs5py.fileclasses.**OUT**(***OGS_Config*)

Bases: *BlockFile*

Class for the ogs OUTPUT file.

Parameters

- **task_root** (*str, optional*) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (*str, optional*) – Name for the ogs task. Default: "model"

Notes

Main-Keywords (#):

- OUTPUT
- VERSION

Sub-Keywords ($) per Main-Keyword:

- OUTPUT
  - NOD_VALUES
  - PCON_VALUES
  - ELE_VALUES
  - RWPT_VALUES
  - GEO_TYPE
  - TIM_TYPE
  - DAT_TYPE
  - VARIABLESHARING
  - AMPLIFIER
  - PCS_TYPE
  - DIS_TYPE
  - MSH_TYPE
  - MMP_VALUES
  - MFP_VALUES
  - TECPLOT_ZONE_SHARE
  - TECPLOT_ELEMENT_OUTPUT_CELL_CENTERED
  - TECPLOT_ZONES_FOR_MG
- VERSION

  (content directly related to the main-keyword)

Standard block:

NOD_VALUES
"HEAD"

GEO_TYPE
"DOMAIN"

DAT_TYPE
"PVD"

TIM_TYPE
["STEPS", 1]

Keyword documentation:
https://ogs5-keywords.netlify.com/ogs/wiki/public/doc-auto/by_ext/out

Reading routines:

https://github.com/ufz/ogs5/blob/master/FEM/Output.cpp#L194

https://github.com/ufz/ogs5/blob/master/FEM/rf_out_new.cpp

See also:

add_block

Attributes

block_no
Number of blocks in the file.

file_name
`str`: base name of the file with extension.

file_path
`str`: save path of the file.

force_writing
`bool`: state if the file is written even if empty.

is_empty
State if the OGS file is empty.

name
`str`: name of the file without extension.

Methods

add_block([index, main_key])	Add a new Block to the actual file.
add_content(content[, main_index, ...])	Add single-line content to the actual file.
add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
add_main_keyword(key[, main_index])	Add a new main keyword (#key) to the actual file.
add_multi_content(content[, main_index, ...])	Add multiple content to the actual file.
add_sub_keyword(key[, main_index, sub_index])	Add a new sub keyword ($key) to the actual file.
append_to_block([index])	Append data to an existing Block in the actual file.
check([verbose])	Check if the given file is valid.
del_block([index, del_all])	Delete a block by its index.
del_content([main_index, sub_index, ...])	Delete content by its position.
del_copy_link()	Remove a former given link to an external file.
del_main_keyword([main_index, del_all])	Delete a main keyword (#key) by its position.
del_sub_keyword([main_index, sub_index, del_all])	Delete a sub keyword ($key) by its position.
get_block([index, as_dict])	Get a Block from the actual file.
get_block_no()	Get the number of blocks in the file.
get_file_type()	Get the OGS file class name.
get_multi_keys([index])	State if a block has a unique set of sub keywords.
is_block_unique([index])	State if a block has a unique set of sub keywords.
read_file(path[, encoding, verbose])	Read an existing OGS input file.
reset()	Delete every content.
save(path, **kwargs)	Save the actual OGS input file in the given path.
update_block([index, main_key])	Update a Block from the actual file.
write_file()	Write the actual OGS input file to the given folder.

add_block(*index=None*, *main_key=None*, ***block*)

Add a new Block to the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 ...

$SUBKEY2
content2 ...

which looks like the following:

```
FILE.add_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- **main_key** (*string, optional*) – Main keyword of the block that should be added (see: `MKEYS`) Default: the first main keyword of the file-type
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

add_content(*content*, *main_index=None*, *sub_index=None*, *line_index=None*)

Add single-line content to the actual file.

Parameters

- **content** (*list*) – list containing one line of content given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position, where the new line of content should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

`add_copy_link`(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied
- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

`add_main_keyword`(*key*, *main_index=None*)

Add a new main keyword (#key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – position, where the new main keyword should be added between the existing ones. As default, it is placed at the end.

`add_multi_content`(*content*, *main_index=None*, *sub_index=None*)

Add multiple content to the actual file.

Parameters

- **content** (*list*) – list containing lines of content, each given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

The content will be added at the end of the actual subkeyword.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

add_sub_keyword(*key*, *main_index=None*, *sub_index=None*)

Add a new sub keyword ($key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – position, where the new sub keyword should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the subkeyword is not added.

---

append_to_block(*index=None*, ***block*)

Append data to an existing Block in the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 …

$SUBKEY2
content2 …

which looks like the following:

```
FILE.append_to_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

check(*verbose=True*)

Check if the given file is valid.

Parameters
verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns
result – Validity of the given file.

Return type
bool

del_block(*index=None*, *del_all=False*)

Delete a block by its index.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **del_all** (*bool, optional*) – State, if all blocks shall be deleted. Default: False

del_content(*main_index=-1*, *sub_index=-1*, *line_index=-1*, *del_all=False*)

Delete content by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be deleted. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position of the content line, that should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all content shall be deleted. Default: False

del_copy_link()

Remove a former given link to an external file.

del_main_keyword(*main_index=None*, *del_all=False*)

Delete a main keyword (#key) by its position.

Parameters

- **main_index** (*int, optional*) – position, which main keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all main keywords shall be deleted. Default: False

del_sub_keyword(*main_index=-1*, *sub_index=-1*, *del_all=False*)

Delete a sub keyword ($key) by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **pos** (*int, optional*) – position, which sub keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all sub keywords shall be deleted. Default: False

get_block(*index=None*, *as_dict=True*)

Get a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **as_dict** (*bool, optional*) – Here you can state of you want the output as a dictionary, which can be used as key-word-arguments for *add_block*. If False, you get the main-key, a list of sub-keys and a list of content. Default: True

get_block_no()

Get the number of blocks in the file.

get_file_type()

Get the OGS file class name.

get_multi_keys(*index=None*)

State if a block has a unique set of sub keywords.

is_block_unique(*index=None*)

State if a block has a unique set of sub keywords.

`read_file`(*path*, *encoding=None*, *verbose=False*)

Read an existing OGS input file.

Parameters

- **path** (*str*) – path to the existing file that should be read
- **encoding** (*str or None, optional*) – encoding of the given file. If `None` is given, the system standard is used. Default: `None`
- **verbose** (*bool, optional*) – Print information of the reading process. Default: False

`reset`()

Delete every content.

`save`(*path*, ***kwargs*)

Save the actual OGS input file in the given path.

Parameters

- **path** (*str*) – path to where to file should be saved
- **update** (*bool, optional*) – state if the content should be updated before saving. Default: True

`update_block`(*index=None*, *main_key=None*, ***block*)

Update a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is used. Default: None
- **main_key** (*string, optional*) – Main keyword of the block that should be updated (see: `MKEYS`) This shouldn't be done. Default: None
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

`write_file`()

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

`MKEYS = ['OUTPUT', 'VERSION']`

Main Keywords of this OGS-BlockFile

Type
`list`

`SKEYS = [['NOD_VALUES', 'PCON_VALUES', 'ELE_VALUES', 'RWPT_VALUES', 'GEO_TYPE', 'TIM_TYPE', 'DAT_TYPE', 'VARIABLESHARING', 'AMPLIFIER', 'PCS_TYPE', 'DIS_TYPE', 'MSH_TYPE', 'MMP_VALUES', 'MFP_VALUES', 'TECPLOT_ZONE_SHARE', 'TECPLOT_ELEMENT_OUTPUT_CELL_CENTERED', 'TECPLOT_ZONES_FOR_MG'], ['']]`

Sub Keywords of this OGS-BlockFile

Type
`list`

`STD = {'DAT_TYPE': 'PVD', 'GEO_TYPE': 'DOMAIN', 'NOD_VALUES': 'HEAD', 'TIM_TYPE': ['STEPS', 1]}`

Standard Block OGS-BlockFile

Type
`dict`

property block_no

Number of blocks in the file.

property file_name

base name of the file with extension.

Type
str

property file_path

save path of the file.

Type
str

property force_writing

state if the file is written even if empty.

Type
bool

property is_empty

State if the OGS file is empty.

property name

name of the file without extension.

Type
str

ogs5py.fileclasses.PCS

class ogs5py.fileclasses.**PCS**(***OGS_Config*)

Bases: *BlockFile*

Class for the ogs PROCESS file.

Parameters

- **task_root** (*str, optional*) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (*str, optional*) – Name for the ogs task. Default: "model"

Notes

Main-Keywords (#):

- PROCESS

Sub-Keywords ($) per Main-Keyword:

- PROCESS
  - APP_TYPE
  - BOUNDARY_CONDITION_OUTPUT
  - COUNT
  - CPL_TYPE
  - DEACTIVATED_SUBDOMAIN
  - DISSOLVED_CO2_INGAS_PCS_NAME
  - DISSOLVED_CO2_PCS_NAME
  - TEMPERATURE_UNIT
  - ELEMENT_MATRIX_OUTPUT
  - GEO_TYPE
  - MEDIUM_TYPE
  - MEMORY_TYPE
  - MSH_TYPE
  - NEGLECT_H_INI_EFFECT
  - NUM_TYPE
  - OutputMassOfGasInModel
  - PCS_TYPE
  - PHASE_TRANSITION
  - PRIMARY_VARIABLE
  - PROCESSED_BC
  - RELOAD
  - SATURATION_SWITCH
  - SAVE_ECLIPSE_DATA_FILES
  - SIMULATOR
  - SIMULATOR_MODEL_PATH

- SIMULATOR_PATH
- SIMULATOR_WELL_PATH
- ST_RHS
- TIME_CONTROLLED_EXCAVATION
- TIM_TYPE
- UPDATE_INI_STATE
- USE_PRECALCULATED_FILES
- USE_VELOCITIES_FOR_TRANSPORT

Standard block:

PCS_TYPE
"GROUNDWATER_FLOW"

NUM_TYPE
"NEW"

Keyword documentation:
https://ogs5-keywords.netlify.com/ogs/wiki/public/doc-auto/by_ext/pcs

Reading routines:
https://github.com/ufz/ogs5/blob/master/FEM/rf_pcs.cpp#L1803

See also:

add_block

Attributes

block_no
Number of blocks in the file.

file_name
`str`: base name of the file with extension.

file_path
`str`: save path of the file.

force_writing
`bool`: state if the file is written even if empty.

is_empty
State if the OGS file is empty.

name
`str`: name of the file without extension.

Methods

add_block([index, main_key])	Add a new Block to the actual file.
add_content(content[, main_index, ...])	Add single-line content to the actual file.
add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
add_main_keyword(key[, main_index])	Add a new main keyword (#key) to the actual file.
add_multi_content(content[, main_index, ...])	Add multiple content to the actual file.
add_sub_keyword(key[, main_index, sub_index])	Add a new sub keyword ($key) to the actual file.
append_to_block([index])	Append data to an existing Block in the actual file.
check([verbose])	Check if the given file is valid.
del_block([index, del_all])	Delete a block by its index.
del_content([main_index, sub_index, ...])	Delete content by its position.
del_copy_link()	Remove a former given link to an external file.
del_main_keyword([main_index, del_all])	Delete a main keyword (#key) by its position.
del_sub_keyword([main_index, sub_index, del_all])	Delete a sub keyword ($key) by its position.
get_block([index, as_dict])	Get a Block from the actual file.
get_block_no()	Get the number of blocks in the file.
get_file_type()	Get the OGS file class name.
get_multi_keys([index])	State if a block has a unique set of sub keywords.
is_block_unique([index])	State if a block has a unique set of sub keywords.
read_file(path[, encoding, verbose])	Read an existing OGS input file.
reset()	Delete every content.
save(path, **kwargs)	Save the actual OGS input file in the given path.
update_block([index, main_key])	Update a Block from the actual file.
write_file()	Write the actual OGS input file to the given folder.

add_block(*index=None*, *main_key=None*, ***block*)

Add a new Block to the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 ...

$SUBKEY2
content2 ...

which looks like the following:

```
FILE.add_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- **main_key** (*string, optional*) – Main keyword of the block that should be added (see: `MKEYS`) Default: the first main keyword of the file-type
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

add_content(*content*, *main_index=None*, *sub_index=None*, *line_index=None*)

Add single-line content to the actual file.

Parameters

- **content** (*list*) – list containing one line of content given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position, where the new line of content should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

`add_copy_link`(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied
- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

`add_main_keyword`(*key*, *main_index=None*)

Add a new main keyword (#key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – position, where the new main keyword should be added between the existing ones. As default, it is placed at the end.

`add_multi_content`(*content*, *main_index=None*, *sub_index=None*)

Add multiple content to the actual file.

Parameters

- **content** (*list*) – list containing lines of content, each given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

The content will be added at the end of the actual subkeyword.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

add_sub_keyword(*key*, *main_index=None*, *sub_index=None*)

Add a new sub keyword ($key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – position, where the new sub keyword should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the subkeyword is not added.

---

append_to_block(*index=None*, ***block*)

Append data to an existing Block in the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 …

$SUBKEY2
content2 …

which looks like the following:

```
FILE.append_to_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

check(*verbose=True*)

Check if the given file is valid.

Parameters
verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns
result – Validity of the given file.

Return type
bool

del_block(*index=None*, *del_all=False*)

Delete a block by its index.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **del_all** (*bool, optional*) – State, if all blocks shall be deleted. Default: False

del_content(*main_index=-1*, *sub_index=-1*, *line_index=-1*, *del_all=False*)

Delete content by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be deleted. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position of the content line, that should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all content shall be deleted. Default: False

del_copy_link()

Remove a former given link to an external file.

del_main_keyword(*main_index=None*, *del_all=False*)

Delete a main keyword (#key) by its position.

Parameters

- **main_index** (*int, optional*) – position, which main keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all main keywords shall be deleted. Default: False

del_sub_keyword(*main_index=-1*, *sub_index=-1*, *del_all=False*)

Delete a sub keyword ($key) by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **pos** (*int, optional*) – position, which sub keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all sub keywords shall be deleted. Default: False

get_block(*index=None*, *as_dict=True*)

Get a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **as_dict** (*bool, optional*) – Here you can state of you want the output as a dictionary, which can be used as key-word-arguments for *add_block*. If False, you get the main-key, a list of sub-keys and a list of content. Default: True

get_block_no()

Get the number of blocks in the file.

get_file_type()

Get the OGS file class name.

get_multi_keys(*index=None*)

State if a block has a unique set of sub keywords.

is_block_unique(*index=None*)

State if a block has a unique set of sub keywords.

`read_file`(*path*, *encoding=None*, *verbose=False*)

Read an existing OGS input file.

Parameters

- **path** (*str*) – path to the existing file that should be read
- **encoding** (*str or None, optional*) – encoding of the given file. If `None` is given, the system standard is used. Default: `None`
- **verbose** (*bool, optional*) – Print information of the reading process. Default: False

`reset`()

Delete every content.

`save`(*path*, ***kwargs*)

Save the actual OGS input file in the given path.

Parameters

- **path** (*str*) – path to where to file should be saved
- **update** (*bool, optional*) – state if the content should be updated before saving. Default: True

`update_block`(*index=None*, *main_key=None*, ***block*)

Update a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is used. Default: None
- **main_key** (*string, optional*) – Main keyword of the block that should be updated (see: `MKEYS`) This shouldn't be done. Default: None
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

`write_file`()

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

`MKEYS = ['PROCESS']`

Main Keywords of this OGS-BlockFile

Type
`list`

```
SKEYS = [['PCS_TYPE', 'NUM_TYPE', 'CPL_TYPE', 'TIM_TYPE', 'APP_TYPE', 'COUNT',
'PRIMARY_VARIABLE', 'TEMPERATURE_UNIT', 'ELEMENT_MATRIX_OUTPUT',
'BOUNDARY_CONDITION_OUTPUT', 'OutputMassOfGasInModel', 'ST_RHS', 'PROCESSED_BC',
'MEMORY_TYPE', 'RELOAD', 'DEACTIVATED_SUBDOMAIN', 'MSH_TYPE', 'MEDIUM_TYPE',
'SATURATION_SWITCH', 'USE_VELOCITIES_FOR_TRANSPORT', 'SIMULATOR',
'SIMULATOR_PATH', 'SIMULATOR_MODEL_PATH', 'USE_PRECALCULATED_FILES',
'SAVE_ECLIPSE_DATA_FILES', 'SIMULATOR_WELL_PATH', 'PHASE_TRANSITION',
'DISSOLVED_CO2_PCS_NAME', 'DISSOLVED_CO2_INGAS_PCS_NAME',
'TIME_CONTROLLED_EXCAVATION', 'NEGLECT_H_INI_EFFECT', 'UPDATE_INI_STATE',
'CONSTANT']]
```

Sub Keywords of this OGS-BlockFile

Type
`list`

STD = {'NUM_TYPE': 'NEW', 'PCS_TYPE': 'GROUNDWATER_FLOW'}

Standard Block OGS-BlockFile

Type
dict

property block_no

Number of blocks in the file.

property file_name

base name of the file with extension.

Type
str

property file_path

save path of the file.

Type
str

property force_writing

state if the file is written even if empty.

Type
bool

property is_empty

State if the OGS file is empty.

property name

name of the file without extension.

Type
str

ogs5py.fileclasses.PCT

class ogs5py.fileclasses.**PCT**(*data=None*, *s_flag=1*, *task_root=None*, *task_id='model'*)

Bases: *File*

Class for the ogs Particle file, if the PCS TYPE is RANDOM_WALK.

Parameters

- **data** (*np.array or None*) – particle data. Default: None
- **s_flag** (*int, optional*) – 1 for same pseudo-random series, 0 for different pseudo-random series. Default: 1
- **task_root** (*str, optional*) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (*str, optional*) – Name for the ogs task. Default: "model"

Attributes

file_name
str: base name of the file with extension.

file_path
str: save path of the file.

force_writing
bool: state if the file is written even if empty.

is_empty
State if the OGS file is empty.

name
str: name of the file without extension.

Methods

add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
check([verbose])	Check if the external geometry definition is valid.
del_copy_link()	Remove a former given link to an external file.
get_file_type()	Get the OGS file class name.
read_file(path, **kwargs)	Write the actual OGS input file to the given folder.
reset()	Delete every content.
save(path)	Save the actual PCT external file in the given path.
write_file()	Write the actual OGS input file to the given folder.

add_copy_link(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied
- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

check(*verbose=True*)

Check if the external geometry definition is valid.

In the sence, that the contained data is consistent.

Parameters
verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns
result – Validity of the given gli.

Return type
bool

`del_copy_link()`

Remove a former given link to an external file.

`get_file_type()`

Get the OGS file class name.

`read_file(path, **kwargs)`

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

`reset()`

Delete every content.

`save(path)`

Save the actual PCT external file in the given path.

Parameters
path (*str*) – path to where to file should be saved

`write_file()`

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

`property file_name`

base name of the file with extension.

Type
str

`property file_path`

save path of the file.

Type
str

`property force_writing`

state if the file is written even if empty.

Type
bool

`property is_empty`

State if the OGS file is empty.

`property name`

name of the file without extension.

Type
str

ogs5py.fileclasses.PQC

class ogs5py.fileclasses.**PQC**(***OGS_Config*)

Bases: *LineFile*

Class for the ogs PHREEQC interface file.

Parameters

- **task_root** (*str, optional*) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (*str, optional*) – Name for the ogs task. Default: "model"

Notes

This is just handled as a line-wise file. You can access the data by line with:

PQC.lines

Keyword documentation:
https://ogs5-keywords.netlify.com/ogs/wiki/public/doc-auto/by_ext/pqc

Reading routines:
https://github.com/ufz/ogs5/blob/master/FEM/rf_react.cpp#L2136

Attributes

file_name
str: base name of the file with extension.

file_path
str: save path of the file.

force_writing
bool: state if the file is written even if empty.

is_empty
bool: state if the file is empty.

name
str: name of the file without extension.

Methods

add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
check([verbose])	Check if the given text-file is valid.
del_copy_link()	Remove a former given link to an external file.
get_file_type()	Get the OGS file class name.
read_file(path[, encoding, verbose])	Read an existing OGS input file.
reset()	Delete every content.
save(path)	Save the actual line-wise file in the given path.
write_file()	Write the actual OGS input file to the given folder.

add_copy_link(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied
- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

`check`(*verbose=True*)

Check if the given text-file is valid.

Parameters
verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns
result – Validity of the given file.

Return type
bool

`del_copy_link`()

Remove a former given link to an external file.

`get_file_type`()

Get the OGS file class name.

`read_file`(*path*, *encoding=None*, *verbose=False*)

Read an existing OGS input file.

Parameters

- **path** (*str*) – path to the existing file that should be read
- **encoding** (*str or None, optional*) – encoding of the given file. If `None` is given, the system standard is used. Default: `None`
- **verbose** (*bool, optional*) – Print information of the reading process. Default: False

`reset`()

Delete every content.

`save`(*path*)

Save the actual line-wise file in the given path.

Parameters
path (*str*) – path to where to file should be saved

`write_file`()

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

`property file_name`

base name of the file with extension.

Type
`str`

`property file_path`

save path of the file.

Type
`str`

`property force_writing`

state if the file is written even if empty.

Type
`bool`

property is_empty

state if the file is empty.

Type
bool

property name

name of the file without extension.

Type
str

ogs5py.fileclasses.PQCdat

class ogs5py.fileclasses.**PQCdat**(***OGS_Config*)

Bases: *LineFile*

Class for the ogs PHREEQC dat file.

Parameters

- **task_root** (*str, optional*) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (*str, optional*) – Name for the ogs task. Default: "model"

Notes

This is just handled as a line-wise file. You can access the data by line with:

PQCdat.lines

Keyword documentation:
https://ogs5-keywords.netlify.com/ogs/wiki/public/doc-auto/by_ext/pqc

Reading routines:
https://github.com/ufz/ogs5/blob/master/FEM/rf_react.cpp#L2136

Attributes

file_name
str: base name of the file with extension.

file_path
str: save path of the file.

force_writing
bool: state if the file is written even if empty.

is_empty
bool: state if the file is empty.

name
str: name of the file without extension.

Methods

add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
check([verbose])	Check if the given text-file is valid.
del_copy_link()	Remove a former given link to an external file.
get_file_type()	Get the OGS file class name.
read_file(path[, encoding, verbose])	Read an existing OGS input file.
reset()	Delete every content.
save(path)	Save the actual line-wise file in the given path.
write_file()	Write the actual OGS input file to the given folder.

add_copy_link(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied
- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

`check`(*verbose=True*)

Check if the given text-file is valid.

Parameters
verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns
result – Validity of the given file.

Return type
bool

`del_copy_link`()

Remove a former given link to an external file.

`get_file_type`()

Get the OGS file class name.

`read_file`(*path*, *encoding=None*, *verbose=False*)

Read an existing OGS input file.

Parameters

- **path** (*str*) – path to the existing file that should be read
- **encoding** (*str or None, optional*) – encoding of the given file. If `None` is given, the system standard is used. Default: `None`
- **verbose** (*bool, optional*) – Print information of the reading process. Default: False

`reset`()

Delete every content.

`save`(*path*)

Save the actual line-wise file in the given path.

Parameters
path (*str*) – path to where to file should be saved

`write_file`()

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

`property file_name`

base name of the file with extension.

Type
str

`property file_path`

save path of the file.

Type
str

`property force_writing`

state if the file is written even if empty.

Type
bool

property is_empty

state if the file is empty.

Type
bool

property name

name of the file without extension.

Type
str

ogs5py.fileclasses.REI

class ogs5py.fileclasses.**REI**(***OGS_Config*)

Bases: *BlockFile*

Class for the ogs REACTION_INTERFACE file.

Parameters

- **task_root** (*str, optional*) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (*str, optional*) – Name for the ogs task. Default: "model"

Notes

Main-Keywords (#):

- REACTION_INTERFACE

Sub-Keywords ($) per Main-Keyword:

- REACTION_INTERFACE
  - ALL_PCS_DUMP
  - DISSOLVED_NEUTRAL_CO2_SPECIES_NAME
  - HEATPUMP_2DH_TO_2DV
  - INITIAL_CONDITION_OUTPUT
  - MOL_PER
  - PCS_RENAME_INIT
  - PCS_RENAME_POST
  - PCS_RENAME_PRE
  - POROSITY_RESTART
  - PRESSURE
  - P_VLE
  - RESIDUAL
  - SODIUM_SPECIES_NAME
  - SOLID_SPECIES_DUMP_MOLES
  - TEMPERATURE
  - UPDATE_INITIAL_SOLID_COMPOSITION
  - VLE
  - WATER_CONCENTRATION
  - WATER_SATURATION_LIMIT
  - WATER_SPECIES_NAME

Standard block:
None

Keyword documentation:
https://ogs5-keywords.netlify.com/ogs/wiki/public/doc-auto/by_ext/rei

Reading routines:
https://github.com/ufz/ogs5/blob/master/FEM/rf_react_int.cpp#L173

See also:

add_block

Attributes

block_no
Number of blocks in the file.

file_name
str: base name of the file with extension.

file_path
str: save path of the file.

force_writing
bool: state if the file is written even if empty.

is_empty
State if the OGS file is empty.

name
str: name of the file without extension.

Methods

add_block([index, main_key])	Add a new Block to the actual file.
add_content(content[, main_index, ...])	Add single-line content to the actual file.
add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
add_main_keyword(key[, main_index])	Add a new main keyword (#key) to the actual file.
add_multi_content(content[, main_index, ...])	Add multiple content to the actual file.
add_sub_keyword(key[, main_index, sub_index])	Add a new sub keyword ($key) to the actual file.
append_to_block([index])	Append data to an existing Block in the actual file.
check([verbose])	Check if the given file is valid.
del_block([index, del_all])	Delete a block by its index.
del_content([main_index, sub_index, ...])	Delete content by its position.
del_copy_link()	Remove a former given link to an external file.
del_main_keyword([main_index, del_all])	Delete a main keyword (#key) by its position.
del_sub_keyword([main_index, sub_index, del_all])	Delete a sub keyword ($key) by its position.
get_block([index, as_dict])	Get a Block from the actual file.
get_block_no()	Get the number of blocks in the file.
get_file_type()	Get the OGS file class name.
get_multi_keys([index])	State if a block has a unique set of sub keywords.
is_block_unique([index])	State if a block has a unique set of sub keywords.
read_file(path[, encoding, verbose])	Read an existing OGS input file.
reset()	Delete every content.
save(path, **kwargs)	Save the actual OGS input file in the given path.
update_block([index, main_key])	Update a Block from the actual file.
write_file()	Write the actual OGS input file to the given folder.

add_block(*index=None*, *main_key=None*, ***block*)

Add a new Block to the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 ...

$SUBKEY2
content2 ...

which looks like the following:

```
FILE.add_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- **main_key** (*string, optional*) – Main keyword of the block that should be added (see: `MKEYS`) Default: the first main keyword of the file-type
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

add_content(*content*, *main_index=None*, *sub_index=None*, *line_index=None*)

Add single-line content to the actual file.

Parameters

- **content** (*list*) – list containing one line of content given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position, where the new line of content should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

---

add_copy_link(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied
- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

add_main_keyword(*key*, *main_index=None*)

Add a new main keyword (#key) to the actual file.

Parameters

- **key** (*string*) – key name

- **main_index** (*int, optional*) – position, where the new main keyword should be added between the existing ones. As default, it is placed at the end.

add_multi_content(*content*, *main_index=None*, *sub_index=None*)

Add multiple content to the actual file.

Parameters

- **content** (*list*) – list containing lines of content, each given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

The content will be added at the end of the actual subkeyword.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

---

add_sub_keyword(*key*, *main_index=None*, *sub_index=None*)

Add a new sub keyword ($key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – position, where the new sub keyword should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the subkeyword is not added.

---

append_to_block(*index=None*, ***block*)

Append data to an existing Block in the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 …

$SUBKEY2
content2 …

which looks like the following:

```
FILE.append_to_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.

- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

`check`(*verbose=True*)

Check if the given file is valid.

Parameters
verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns
result – Validity of the given file.

Return type
bool

`del_block`(*index=None*, *del_all=False*)

Delete a block by its index.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **del_all** (*bool, optional*) – State, if all blocks shall be deleted. Default: False

`del_content`(*main_index=-1*, *sub_index=-1*, *line_index=-1*, *del_all=False*)

Delete content by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be deleted. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position of the content line, that should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all content shall be deleted. Default: False

`del_copy_link`()

Remove a former given link to an external file.

`del_main_keyword`(*main_index=None*, *del_all=False*)

Delete a main keyword (#key) by its position.

Parameters

- **main_index** (*int, optional*) – position, which main keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all main keywords shall be deleted. Default: False

`del_sub_keyword`(*main_index=-1*, *sub_index=-1*, *del_all=False*)

Delete a sub keyword ($key) by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **pos** (*int, optional*) – position, which sub keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all sub keywords shall be deleted. Default: False

`get_block`(*index=None*, *as_dict=True*)

Get a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **as_dict** (*bool, optional*) – Here you can state of you want the output as a dictionary, which can be used as key-word-arguments for *add_block*. If False, you get the main-key, a list of sub-keys and a list of content. Default: True

`get_block_no`()

Get the number of blocks in the file.

`get_file_type`()

Get the OGS file class name.

`get_multi_keys`(*index=None*)

State if a block has a unique set of sub keywords.

`is_block_unique`(*index=None*)

State if a block has a unique set of sub keywords.

`read_file`(*path*, *encoding=None*, *verbose=False*)

Read an existing OGS input file.

Parameters

- **path** (*str*) – path to the existing file that should be read
- **encoding** (*str or None, optional*) – encoding of the given file. If `None` is given, the system standard is used. Default: `None`
- **verbose** (*bool, optional*) – Print information of the reading process. Default: False

`reset`()

Delete every content.

`save`(*path*, ***kwargs*)

Save the actual OGS input file in the given path.

Parameters

- **path** (*str*) – path to where to file should be saved
- **update** (*bool, optional*) – state if the content should be updated before saving. Default: True

`update_block`(*index=None*, *main_key=None*, ***block*)

Update a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is used. Default: None
- **main_key** (*string, optional*) – Main keyword of the block that should be updated (see: `MKEYS`) This shouldn't be done. Default: None
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

write_file()

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

MKEYS = ['REACTION_INTERFACE']

Main Keywords of this OGS-BlockFile

Type
list

SKEYS = [['MOL_PER', 'WATER_CONCENTRATION', 'WATER_SPECIES_NAME', 'DISSOLVED_NEUTRAL_CO2_SPECIES_NAME', 'SODIUM_SPECIES_NAME', 'PRESSURE', 'TEMPERATURE', 'WATER_SATURATION_LIMIT', 'RESIDUAL', 'SOLID_SPECIES_DUMP_MOLES', 'ALL_PCS_DUMP', 'INITIAL_CONDITION_OUTPUT', 'UPDATE_INITIAL_SOLID_COMPOSITION', 'VLE', 'P_VLE', 'POROSITY_RESTART', 'HEATPUMP_2DH_TO_2DV', 'PCS_RENAME_INIT', 'PCS_RENAME_PRE', 'PCS_RENAME_POST', 'CONSTANT_PRESSURE', 'CONSTANT_TEMPERATURE']]

Sub Keywords of this OGS-BlockFile

Type
list

STD = {}

Standard Block OGS-BlockFile

Type
dict

property block_no

Number of blocks in the file.

property file_name

base name of the file with extension.

Type
str

property file_path

save path of the file.

Type
str

property force_writing

state if the file is written even if empty.

Type
bool

property is_empty

State if the OGS file is empty.

property name

name of the file without extension.

Type
str

ogs5py.fileclasses.RFD

class ogs5py.fileclasses.**RFD**(***OGS_Config*)

Bases: *BlockFile*

Class for the ogs USER DEFINED TIME CURVES file.

Parameters

- **task_root** (*str, optional*) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (*str, optional*) – Name for the ogs task. Default: "model"

Notes

Main-Keywords (#):

- PROJECT
- CURVE
- CURVES

Sub-Keywords ($) per Main-Keyword:
(no sub-keywords)

Standard block:
None

Keyword documentation:
https://ogs5-keywords.netlify.com/ogs/wiki/public/doc-auto/by_ext/rfd

Reading routines:
https://github.com/ufz/ogs5/blob/master/FEM/files0.cpp#L370

See also:

add_block

Attributes

block_no
Number of blocks in the file.

file_name
str: base name of the file with extension.

file_path
str: save path of the file.

force_writing
bool: state if the file is written even if empty.

is_empty
State if the OGS file is empty.

name
str: name of the file without extension.

Methods

add_block([index, main_key])	Add a new Block to the actual file.
add_content(content[, main_index, ...])	Add single-line content to the actual file.
add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
add_main_keyword(key[, main_index])	Add a new main keyword (#key) to the actual file.
add_multi_content(content[, main_index, ...])	Add multiple content to the actual file.
add_sub_keyword(key[, main_index, sub_index])	Add a new sub keyword ($key) to the actual file.
append_to_block([index])	Append data to an existing Block in the actual file.
check([verbose])	Check if the given file is valid.
del_block([index, del_all])	Delete a block by its index.
del_content([main_index, sub_index, ...])	Delete content by its position.
del_copy_link()	Remove a former given link to an external file.
del_main_keyword([main_index, del_all])	Delete a main keyword (#key) by its position.
del_sub_keyword([main_index, sub_index, del_all])	Delete a sub keyword ($key) by its position.
get_block([index, as_dict])	Get a Block from the actual file.
get_block_no()	Get the number of blocks in the file.
get_file_type()	Get the OGS file class name.
get_multi_keys([index])	State if a block has a unique set of sub keywords.
is_block_unique([index])	State if a block has a unique set of sub keywords.
read_file(path[, encoding, verbose])	Read an existing OGS input file.
reset()	Delete every content.
save(path, **kwargs)	Save the actual OGS input file in the given path.
update_block([index, main_key])	Update a Block from the actual file.
write_file()	Write the actual OGS input file to the given folder.

add_block(*index=None*, *main_key=None*, ***block*)

Add a new Block to the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 ...

$SUBKEY2
content2 ...

which looks like the following:

```
FILE.add_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- **main_key** (*string, optional*) – Main keyword of the block that should be added (see: `MKEYS`) Default: the first main keyword of the file-type
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

add_content(*content*, *main_index=None*, *sub_index=None*, *line_index=None*)

Add single-line content to the actual file.

Parameters

- **content** (*list*) – list containing one line of content given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position, where the new line of content should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

`add_copy_link`(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied
- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

`add_main_keyword`(*key*, *main_index=None*)

Add a new main keyword (#key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – position, where the new main keyword should be added between the existing ones. As default, it is placed at the end.

`add_multi_content`(*content*, *main_index=None*, *sub_index=None*)

Add multiple content to the actual file.

Parameters

- **content** (*list*) – list containing lines of content, each given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

The content will be added at the end of the actual subkeyword.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

add_sub_keyword(*key*, *main_index=None*, *sub_index=None*)

Add a new sub keyword ($key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – position, where the new sub keyword should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the subkeyword is not added.

---

append_to_block(*index=None*, ***block*)

Append data to an existing Block in the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 …

$SUBKEY2
content2 …

which looks like the following:

```
FILE.append_to_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

check(*verbose=True*)

Check if the given file is valid.

Parameters
verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns
result – Validity of the given file.

Return type
bool

del_block(*index=None*, *del_all=False*)

Delete a block by its index.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **del_all** (*bool, optional*) – State, if all blocks shall be deleted. Default: False

del_content(*main_index=-1*, *sub_index=-1*, *line_index=-1*, *del_all=False*)

Delete content by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be deleted. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position of the content line, that should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all content shall be deleted. Default: False

del_copy_link()

Remove a former given link to an external file.

del_main_keyword(*main_index=None*, *del_all=False*)

Delete a main keyword (#key) by its position.

Parameters

- **main_index** (*int, optional*) – position, which main keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all main keywords shall be deleted. Default: False

del_sub_keyword(*main_index=-1*, *sub_index=-1*, *del_all=False*)

Delete a sub keyword ($key) by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **pos** (*int, optional*) – position, which sub keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all sub keywords shall be deleted. Default: False

get_block(*index=None*, *as_dict=True*)

Get a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **as_dict** (*bool, optional*) – Here you can state of you want the output as a dictionary, which can be used as key-word-arguments for *add_block*. If False, you get the main-key, a list of sub-keys and a list of content. Default: True

get_block_no()

Get the number of blocks in the file.

get_file_type()

Get the OGS file class name.

get_multi_keys(*index=None*)

State if a block has a unique set of sub keywords.

is_block_unique(*index=None*)

State if a block has a unique set of sub keywords.

read_file(*path*, *encoding=None*, *verbose=False*)

Read an existing OGS input file.

Parameters

- **path** (*str*) – path to the existing file that should be read
- **encoding** (*str or None, optional*) – encoding of the given file. If `None` is given, the system standard is used. Default: `None`
- **verbose** (*bool, optional*) – Print information of the reading process. Default: False

reset()

Delete every content.

save(*path*, ***kwargs*)

Save the actual OGS input file in the given path.

Parameters

- **path** (*str*) – path to where to file should be saved
- **update** (*bool, optional*) – state if the content should be updated before saving. Default: True

update_block(*index=None*, *main_key=None*, ***block*)

Update a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is used. Default: None
- **main_key** (*string, optional*) – Main keyword of the block that should be updated (see: `MKEYS`) This shouldn't be done. Default: None
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

write_file()

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

MKEYS = ['PROJECT', 'CURVE', 'CURVES', 'RENUMBER', 'ITERATION_PROPERTIES_CONCENTRATION', 'REFERENCE_CONDITIONS', 'APRIORI_REFINE_ELEMENT']

Main Keywords of this OGS-BlockFile

Type
list

SKEYS = [[''], [''], [''], [''], [''], [''], ['']]

Sub Keywords of this OGS-BlockFile

Type
list

STD = {}

Standard Block OGS-BlockFile

Type
dict

property block_no

Number of blocks in the file.

property file_name

base name of the file with extension.

Type
str

property file_path

save path of the file.

Type
str

property force_writing

state if the file is written even if empty.

Type
bool

property is_empty

State if the OGS file is empty.

property name

name of the file without extension.

Type
str

ogs5py.fileclasses.ST

class ogs5py.fileclasses.**ST**(***OGS_Config*)

Bases: *BlockFile*

Class for the ogs SOURCE_TERM file.

Parameters

- **task_root** (*str, optional*) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (*str, optional*) – Name for the ogs task. Default: "model"

Notes

Main-Keywords (#):

- SOURCE_TERM

Sub-Keywords ($) per Main-Keyword:

- SOURCE_TERM
  - AIR_BREAKING
  - CHANNEL
  - COMP_NAME
  - CONSTRAINED
  - DISTRIBUTE_VOLUME_FLUX
  - EPSILON
  - DIS_TYPE
  - EXPLICIT_SURFACE_WATER_PRESSURE
  - FCT_TYPE
  - GEO_TYPE
  - MSH_TYPE
  - NEGLECT_SURFACE_WATER_PRESSURE
  - NODE_AVERAGING
  - PCS_TYPE
  - PRIMARY_VARIABLE
  - TIME_INTERPOLATION
  - TIM_TYPE

Standard block:

PCS_TYPE
"GROUNDWATER_FLOW"

PRIMARY_VARIABLE
"HEAD"

GEO_TYPE
["POINT", "WELL"]

DIS_TYPE
["CONSTANT_NEUMANN", -1.0e-03]

Keyword documentation:
https://ogs5-keywords.netlify.com/ogs/wiki/public/doc-auto/by_ext/st

Reading routines:
https://github.com/ufz/ogs5/blob/master/FEM/rf_st_new.cpp#L221

See also:

add_block

Attributes

block_no
Number of blocks in the file.

file_name
str: base name of the file with extension.

file_path
str: save path of the file.

force_writing
bool: state if the file is written even if empty.

is_empty
State if the OGS file is empty.

name
str: name of the file without extension.

Methods

add_block([index, main_key])	Add a new Block to the actual file.
add_content(content[, main_index, ...])	Add single-line content to the actual file.
add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
add_main_keyword(key[, main_index])	Add a new main keyword (#key) to the actual file.
add_multi_content(content[, main_index, ...])	Add multiple content to the actual file.
add_sub_keyword(key[, main_index, sub_index])	Add a new sub keyword ($key) to the actual file.
append_to_block([index])	Append data to an existing Block in the actual file.
check([verbose])	Check if the given file is valid.
del_block([index, del_all])	Delete a block by its index.
del_content([main_index, sub_index, ...])	Delete content by its position.
del_copy_link()	Remove a former given link to an external file.
del_main_keyword([main_index, del_all])	Delete a main keyword (#key) by its position.
del_sub_keyword([main_index, sub_index, del_all])	Delete a sub keyword ($key) by its position.
get_block([index, as_dict])	Get a Block from the actual file.
get_block_no()	Get the number of blocks in the file.
get_file_type()	Get the OGS file class name.
get_multi_keys([index])	State if a block has a unique set of sub keywords.
is_block_unique([index])	State if a block has a unique set of sub keywords.
read_file(path[, encoding, verbose])	Read an existing OGS input file.
reset()	Delete every content.
save(path, **kwargs)	Save the actual OGS input file in the given path.
update_block([index, main_key])	Update a Block from the actual file.
write_file()	Write the actual OGS input file to the given folder.

add_block(*index=None*, *main_key=None*, ***block*)

Add a new Block to the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 ...

$SUBKEY2
content2 ...

which looks like the following:

```
FILE.add_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- **main_key** (*string, optional*) – Main keyword of the block that should be added (see: `MKEYS`) Default: the first main keyword of the file-type
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

add_content(*content*, *main_index=None*, *sub_index=None*, *line_index=None*)

Add single-line content to the actual file.

Parameters

- **content** (*list*) – list containing one line of content given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position, where the new line of content should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

---

add_copy_link(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied
- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

add_main_keyword(*key*, *main_index=None*)

Add a new main keyword (#key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – position, where the new main keyword should be added between the existing ones. As default, it is placed at the end.

add_multi_content(*content*, *main_index=None*, *sub_index=None*)

Add multiple content to the actual file.

Parameters

- **content** (*list*) – list containing lines of content, each given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

The content will be added at the end of the actual subkeyword.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

---

add_sub_keyword(*key*, *main_index=None*, *sub_index=None*)

Add a new sub keyword ($key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – position, where the new sub keyword should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the subkeyword is not added.

---

append_to_block(*index=None*, ***block*)

Append data to an existing Block in the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 …

$SUBKEY2
content2 …

which looks like the following:

```
FILE.append_to_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

`check`(*verbose=True*)

Check if the given file is valid.

Parameters

verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns

result – Validity of the given file.

Return type

bool

`del_block`(*index=None*, *del_all=False*)

Delete a block by its index.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **del_all** (*bool, optional*) – State, if all blocks shall be deleted. Default: False

`del_content`(*main_index=-1*, *sub_index=-1*, *line_index=-1*, *del_all=False*)

Delete content by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be deleted. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position of the content line, that should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all content shall be deleted. Default: False

`del_copy_link`()

Remove a former given link to an external file.

`del_main_keyword`(*main_index=None*, *del_all=False*)

Delete a main keyword (#key) by its position.

Parameters

- **main_index** (*int, optional*) – position, which main keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all main keywords shall be deleted. Default: False

`del_sub_keyword`(*main_index=-1*, *sub_index=-1*, *del_all=False*)

Delete a sub keyword ($key) by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.

- **pos** (*int, optional*) – position, which sub keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all sub keywords shall be deleted. Default: False

`get_block`(*index=None*, *as_dict=True*)

Get a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **as_dict** (*bool, optional*) – Here you can state of you want the output as a dictionary, which can be used as key-word-arguments for *add_block*. If False, you get the main-key, a list of sub-keys and a list of content. Default: True

`get_block_no`()

Get the number of blocks in the file.

`get_file_type`()

Get the OGS file class name.

`get_multi_keys`(*index=None*)

State if a block has a unique set of sub keywords.

`is_block_unique`(*index=None*)

State if a block has a unique set of sub keywords.

`read_file`(*path*, *encoding=None*, *verbose=False*)

Read an existing OGS input file.

Parameters

- **path** (*str*) – path to the existing file that should be read
- **encoding** (*str or None, optional*) – encoding of the given file. If `None` is given, the system standard is used. Default: `None`
- **verbose** (*bool, optional*) – Print information of the reading process. Default: False

`reset`()

Delete every content.

`save`(*path*, ***kwargs*)

Save the actual OGS input file in the given path.

Parameters

- **path** (*str*) – path to where to file should be saved
- **update** (*bool, optional*) – state if the content should be updated before saving. Default: True

`update_block`(*index=None*, *main_key=None*, ***block*)

Update a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is used. Default: None
- **main_key** (*string, optional*) – Main keyword of the block that should be updated (see: `MKEYS`) This shouldn't be done. Default: None
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

write_file()

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

MKEYS = ['SOURCE_TERM']

Main Keywords of this OGS-BlockFile

Type
list

SKEYS = [['PCS_TYPE', 'PRIMARY_VARIABLE', 'COMP_NAME', 'GEO_TYPE', 'EPSILON', 'DIS_TYPE', 'NODE_AVERAGING', 'DISTRIBUTE_VOLUME_FLUX', 'NEGLECT_SURFACE_WATER_PRESSURE', 'EXPLICIT_SURFACE_WATER_PRESSURE', 'CHANNEL', 'AIR_BREAKING', 'TIM_TYPE', 'TIME_INTERPOLATION', 'FCT_TYPE', 'MSH_TYPE', 'CONSTRAINED']]

Sub Keywords of this OGS-BlockFile

Type
list

STD = {'DIS_TYPE': [['CONSTANT_NEUMANN', -0.001]], 'GEO_TYPE': [['POINT', 'WELL']], 'PCS_TYPE': 'GROUNDWATER_FLOW', 'PRIMARY_VARIABLE': 'HEAD'}

Standard Block OGS-BlockFile

Type
dict

property block_no

Number of blocks in the file.

property file_name

base name of the file with extension.

Type
str

property file_path

save path of the file.

Type
str

property force_writing

state if the file is written even if empty.

Type
bool

property is_empty

State if the OGS file is empty.

property name

name of the file without extension.

Type
str

ogs5py.fileclasses.TIM

class ogs5py.fileclasses.**TIM**(***OGS_Config*)

Bases: *BlockFile*

Class for the ogs TIME_STEPPING file.

Parameters

- **task_root** (*str, optional*) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (*str, optional*) – Name for the ogs task. Default: "model"

Notes

Main-Keywords (#):

- TIME_STEPPING

Sub-Keywords ($) per Main-Keyword:

- TIME_STEPPING
  - CRITICAL_TIME
  - INDEPENDENT
  - PCS_TYPE
  - SUBSTEPS
  - TIME_CONTROL
  - TIME_END
  - TIME_FIXED_POINTS
  - TIME_SPLITS
  - TIME_START
  - TIME_STEPS
  - TIME_UNIT

Standard block:

PCS_TYPE
"GROUNDWATER_FLOW"

TIME_START
0

TIME_END
1000

TIME_STEPS
[10, 100]

Keyword documentation:
https://ogs5-keywords.netlify.com/ogs/wiki/public/doc-auto/by_ext/tim

Reading routines:
https://github.com/ufz/ogs5/blob/master/FEM/rf_tim_new.cpp#L161

See also:

add_block

Attributes

block_no
Number of blocks in the file.

file_name
str: base name of the file with extension.

file_path
str: save path of the file.

force_writing
bool: state if the file is written even if empty.

is_empty
State if the OGS file is empty.

name
str: name of the file without extension.

Methods

add_block([index, main_key])	Add a new Block to the actual file.
add_content(content[, main_index, ...])	Add single-line content to the actual file.
add_copy_link(path[, symlink])	Add a link to copy a file instead of writing.
add_main_keyword(key[, main_index])	Add a new main keyword (#key) to the actual file.
add_multi_content(content[, main_index, ...])	Add multiple content to the actual file.
add_sub_keyword(key[, main_index, sub_index])	Add a new sub keyword ($key) to the actual file.
append_to_block([index])	Append data to an existing Block in the actual file.
check([verbose])	Check if the given file is valid.
del_block([index, del_all])	Delete a block by its index.
del_content([main_index, sub_index, ...])	Delete content by its position.
del_copy_link()	Remove a former given link to an external file.
del_main_keyword([main_index, del_all])	Delete a main keyword (#key) by its position.
del_sub_keyword([main_index, sub_index, del_all])	Delete a sub keyword ($key) by its position.
get_block([index, as_dict])	Get a Block from the actual file.
get_block_no()	Get the number of blocks in the file.
get_file_type()	Get the OGS file class name.
get_multi_keys([index])	State if a block has a unique set of sub keywords.
is_block_unique([index])	State if a block has a unique set of sub keywords.
read_file(path[, encoding, verbose])	Read an existing OGS input file.
reset()	Delete every content.
save(path, **kwargs)	Save the actual OGS input file in the given path.
update_block([index, main_key])	Update a Block from the actual file.
write_file()	Write the actual OGS input file to the given folder.

add_block(*index=None*, *main_key=None*, ***block*)

Add a new Block to the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 ...

$SUBKEY2
content2 ...

which looks like the following:

`FILE.add_block(SUBKEY1=content1, SUBKEY2=content2)`

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- **main_key** (*string, optional*) – Main keyword of the block that should be added (see: `MKEYS`) Default: the first main keyword of the file-type
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

add_content(*content*, *main_index=None*, *sub_index=None*, *line_index=None*)

Add single-line content to the actual file.

Parameters

- **content** (*list*) – list containing one line of content given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position, where the new line of content should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

---

add_copy_link(*path*, *symlink=False*)

Add a link to copy a file instead of writing.

Instead of writing a file, you can give a path to an existing file, that will be copied/linked to the target folder.

Parameters

- **path** (*str*) – path to the existing file that should be copied
- **symlink** (*bool, optional*) – on UNIX systems it is possible to use a symbolic link to save time if the file is big. Default: False

add_main_keyword(*key*, *main_index=None*)

Add a new main keyword (#key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – position, where the new main keyword should be added between the existing ones. As default, it is placed at the end.

add_multi_content(*content*, *main_index=None*, *sub_index=None*)

Add multiple content to the actual file.

Parameters

- **content** (*list*) – list containing lines of content, each given as a list of single statements
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be added. As default, the last sub keyword is taken.

Notes

There needs to be at least one main keyword, otherwise the content is not added.

The content will be added at the end of the actual subkeyword.

If no sub keyword is present, a blank one ("") will be added and the content is then directly connected to the actual main keyword.

`add_sub_keyword`(*key*, *main_index=None*, *sub_index=None*)

Add a new sub keyword ($key) to the actual file.

Parameters

- **key** (*string*) – key name
- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be added. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – position, where the new sub keyword should be added between the existing ones. As default, it is placed at the end.

Notes

There needs to be at least one main keyword, otherwise the subkeyword is not added.

`append_to_block`(*index=None*, ***block*)

Append data to an existing Block in the actual file.

Keywords are the sub keywords of the actual file type:

#MAIN_KEY

$SUBKEY1
content1 …

$SUBKEY2
content2 …

which looks like the following:

```
FILE.append_to_block(SUBKEY1=content1, SUBKEY2=content2)
```

Parameters

- **index** (*int or None, optional*) – Positional index, where to insert the given Block. As default, it will be added at the end. Default: None.
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

`check`(*verbose=True*)

Check if the given file is valid.

Parameters
verbose (*bool, optional*) – Print information for the executed checks. Default: True

Returns
result – Validity of the given file.

Return type
bool

`del_block`(*index=None*, *del_all=False*)

Delete a block by its index.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None
- **del_all** (*bool, optional*) – State, if all blocks shall be deleted. Default: False

`del_content`(*main_index=-1*, *sub_index=-1*, *line_index=-1*, *del_all=False*)

Delete content by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **sub_index** (*int, optional*) – index of the corresponding sub keyword where the content should be deleted. As default, the last sub keyword is taken.
- **line_index** (*int, optional*) – position of the content line, that should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all content shall be deleted. Default: False

`del_copy_link`()

Remove a former given link to an external file.

`del_main_keyword`(*main_index=None*, *del_all=False*)

Delete a main keyword (#key) by its position.

Parameters

- **main_index** (*int, optional*) – position, which main keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all main keywords shall be deleted. Default: False

`del_sub_keyword`(*main_index=-1*, *sub_index=-1*, *del_all=False*)

Delete a sub keyword ($key) by its position.

Parameters

- **main_index** (*int, optional*) – index of the corresponding main keyword where the sub keyword should be deleted. As default, the last main keyword is taken.
- **pos** (*int, optional*) – position, which sub keyword should be deleted. Default: -1
- **del_all** (*bool, optional*) – State, if all sub keywords shall be deleted. Default: False

`get_block`(*index=None*, *as_dict=True*)

Get a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is returned. Default: None

- **as_dict** (*bool, optional*) – Here you can state of you want the output as a dictionary, which can be used as key-word-arguments for *add_block*. If False, you get the main-key, a list of sub-keys and a list of content. Default: True

`get_block_no()`

Get the number of blocks in the file.

`get_file_type()`

Get the OGS file class name.

`get_multi_keys(index=None)`

State if a block has a unique set of sub keywords.

`is_block_unique(index=None)`

State if a block has a unique set of sub keywords.

`read_file(path, encoding=None, verbose=False)`

Read an existing OGS input file.

Parameters

- **path** (*str*) – path to the existing file that should be read
- **encoding** (*str or None, optional*) – encoding of the given file. If `None` is given, the system standard is used. Default: `None`
- **verbose** (*bool, optional*) – Print information of the reading process. Default: False

`reset()`

Delete every content.

`save(path, **kwargs)`

Save the actual OGS input file in the given path.

Parameters

- **path** (*str*) – path to where to file should be saved
- **update** (*bool, optional*) – state if the content should be updated before saving. Default: True

`update_block(index=None, main_key=None, **block)`

Update a Block from the actual file.

Parameters

- **index** (*int or None, optional*) – Positional index of the block of interest. As default, the last one is used. Default: None
- **main_key** (*string, optional*) – Main keyword of the block that should be updated (see: `MKEYS`) This shouldn't be done. Default: None
- ****block** (*keyword dict*) – here the dict-keywords are the ogs-subkeywords and the value is the content that should be added with this ogs-subkeyword If a block should contain content directly connected to a main keyword, use this main keyword as input-keyword and the content as value: `SUBKEY=content`

`write_file()`

Write the actual OGS input file to the given folder.

Its path is given by "task_root+task_id+file_ext".

`MKEYS = ['TIME_STEPPING']`

Main Keywords of this OGS-BlockFile

Type
`list`

SKEYS = [['PCS_TYPE', 'TIME_START', 'TIME_END', 'TIME_UNIT', 'INDEPENDENT', 'TIME_STEPS', 'TIME_SPLITS', 'CRITICAL_TIME', 'TIME_CONTROL']]

Sub Keywords of this OGS-BlockFile

Type
list

STD = {'PCS_TYPE': 'GROUNDWATER_FLOW', 'TIME_END': 1000, 'TIME_START': 0, 'TIME_STEPS': [10, 100]}

Standard Block OGS-BlockFile

Type
dict

property block_no

Number of blocks in the file.

property file_name

base name of the file with extension.

Type
str

property file_path

save path of the file.

Type
str

property force_writing

state if the file is written even if empty.

Type
bool

property is_empty

State if the OGS file is empty.

property name

name of the file without extension.

Type
str

ogs5py.reader

ogs5py subpackage providing reader for the ogs5 output.

Reader

readvtk([task_root, task_id, pcs, single_file])	A genearal reader for OGS vtk outputfiles.
readpvd([task_root, task_id, pcs, single_file])	Read a paraview pvd file.
readtec_point([task_root, task_id, pcs, ...])	Collect TECPLOT point output from OGS5.
readtec_polyline([task_root, task_id, pcs, ...])	Collect TECPLOT polyline output from OGS5.
VTK_ERR	vtkStringOutputWindow - File Specific output window class

ogs5py.reader.readvtk

ogs5py.reader.**readvtk**(*task_root='.'*, *task_id=None*, *pcs='ALL'*, *single_file=None*)

A genearal reader for OGS vtk outputfiles.

give a dictionary containing their data

the Filename of the pvd is structured the following way: {task_id}[_{PCS}]xxxx.vtk thereby the "_{PCS}" is optional and just present if a PCS_TYPE is specified in the *.out file

Parameters

- **task_root** (*string, optional*) – string containing the path to the directory containing the ogs output Default : "."
- **task_id** (*string, optional*) – string containing the file name of the ogs task without extension Default : None
- **pcs** (*string or None, optional*) – specify the PCS type that should be collected Possible values are:
  - None/"" (no PCS_TYPE specified in *.out)
  - "NO_PCS"
  - "GROUNDWATER_FLOW"
  - "LIQUID_FLOW"
  - "RICHARDS_FLOW"
  - "AIR_FLOW"
  - "MULTI_PHASE_FLOW"
  - "PS_GLOBAL"
  - "HEAT_TRANSPORT"
  - "DEFORMATION"
  - "MASS_TRANSPORT"
  - "OVERLAND_FLOW"
  - "FLUID_MOMENTUM"
  - "RANDOM_WALK"

  You can get a list with all known PCS-types by setting PCS="ALL" Default : None
- **single_file** (*string or None, optional*) – If you want to read just a single file, you can set the path here. Default : None

Returns

result – keys are the point names and the items are the data from the corresponding files if pcs="ALL", the output is a dictionary with the PCS-types as keys

Return type

dict

ogs5py.reader.readpvd

ogs5py.reader.**readpvd**(*task_root='.'*, *task_id=None*, *pcs='ALL'*, *single_file=None*)

Read a paraview pvd file.

Convert all concerned files to a dictionary containing their data.

the Filename of the pvd is structured the following way: {task_id}[_{PCS}].pvd thereby the "_{PCS}" is optional and just present if a PCS_TYPE is specified in the *.out file

Parameters

- **task_root** (*string, optional*) – string containing the path to the directory containing the ogs output Default : "."
- **task_id** (*string, optional*) – string containing the file name of the ogs task without extension Default : None
- **pcs** (*string or None, optional*) – specify the PCS type that should be collected Possible values are:
  - None/"" (no PCS_TYPE specified in *.out)
  - "NO_PCS"
  - "GROUNDWATER_FLOW"
  - "LIQUID_FLOW"
  - "RICHARDS_FLOW"
  - "AIR_FLOW"
  - "MULTI_PHASE_FLOW"
  - "PS_GLOBAL"
  - "HEAT_TRANSPORT"
  - "DEFORMATION"
  - "MASS_TRANSPORT"
  - "OVERLAND_FLOW"
  - "FLUID_MOMENTUM"
  - "RANDOM_WALK"

  You can get a list with all known PCS-types by setting PCS="ALL" Default : None
- **single_file** (*string or None, optional*) – If you want to read just a single file, you can set the path here. Default : None

Returns

result – keys are the point names and the items are the data from the corresponding files if pcs="ALL", the output is a dictionary with the PCS-types as keys

Return type

dict

ogs5py.reader.readtec_point

ogs5py.reader.**readtec_point**(*task_root='.'*, *task_id=None*, *pcs='ALL'*, *single_file=None*)

Collect TECPLOT point output from OGS5.

the Filenames are structured the following way: {task_id}_time_{NAME}[_{PCS+extra}].tec thereby the "_{PCS}" is optional and just present if a PCS_TYPE is specified in the *.out file the "extra" will not be recognized and will destroy the search-process

Parameters

- **task_root** (*string, optional*) – string containing the path to the directory containing the ogs output Default : "."
- **task_id** (*string, optional*) – string containing the file name of the ogs task without extension Default : None
- **pcs** (*string or None, optional*) – specify the PCS type that should be collected Possible values are:
  - None/"" (no PCS_TYPE specified in *.out)
  - "NO_PCS"
  - "GROUNDWATER_FLOW"
  - "LIQUID_FLOW"
  - "RICHARDS_FLOW"
  - "AIR_FLOW"
  - "MULTI_PHASE_FLOW"
  - "PS_GLOBAL"
  - "HEAT_TRANSPORT"
  - "DEFORMATION"
  - "MASS_TRANSPORT"
  - "OVERLAND_FLOW"
  - "FLUID_MOMENTUM"
  - "RANDOM_WALK"

  You can get a list with all known PCS-types by setting PCS="ALL" Default : None
- **single_file** (*string or None, optional*) – If you want to read just a single file, you can set the path here. Default : None

Returns

result – keys are the point names and the items are the data from the corresponding files if pcs="ALL", the output is a dictionary with the PCS-types as keys

Return type

dict

ogs5py.reader.readtec_polyline

ogs5py.reader.**readtec_polyline**(*task_root='.'*, *task_id=None*, *pcs='ALL'*, *single_file=None*, *trim=True*)

Collect TECPLOT polyline output from OGS5.

the Filenames are structured the following way: {task_id}_ply_{NAME}_t{ply_id}[_{PCS}].tec thereby the "_{PCS}" is optional and just present if a PCS_TYPE is specified in the *.out file

Parameters

- **task_root** (*string, optional*) – string containing the path to the directory containing the ogs output Default : "."
- **task_id** (*string, optional*) – string containing the file name of the ogs task without extension Default : None
- **pcs** (*string or None, optional*) – specify the PCS type that should be collected Possible values are:
  - None/"" (no PCS_TYPE specified in *.out)
  - "NO_PCS"
  - "GROUNDWATER_FLOW"
  - "LIQUID_FLOW"
  - "RICHARDS_FLOW"
  - "AIR_FLOW"
  - "MULTI_PHASE_FLOW"
  - "PS_GLOBAL"
  - "HEAT_TRANSPORT"
  - "DEFORMATION"
  - "MASS_TRANSPORT"
  - "OVERLAND_FLOW"
  - "FLUID_MOMENTUM"
  - "RANDOM_WALK"

  You can get a list with all known PCS-types by setting pcs="ALL" Default : None
- **single_file** (*string or None, optional*) – If you want to read just a single file, you can set the path here. Default : None
- **trim** (*Bool, optional*) – if the ply_ids are not continuous, there will be "None" values in the output list. If trim is "True" these values will be eliminated. If there is just one output for a polyline, the list will be eliminated and the output will be the single dict. Default : True

Returns

result – keys are the Polyline names and the items are lists sorted by the ply_id (it is assumed, that the ply_ids are continuous, if not, the corresponding list entries are "None") if pcs="ALL", the output is a dictionary with the PCS-types as keys

Return type

dict

ogs5py.reader.VTK_ERR

ogs5py.reader.**VTK_ERR** =
<vtkmodules.vtkCommonCore.vtkStringOutputWindow(0x55d60a980dd0)>

vtkStringOutputWindow - File Specific output window class

Superclass: vtkOutputWindow

Writes debug/warning/error output to a log file instead of the console. To use this class, instantiate it and then call SetInstance(this).

ogs5py.tools

ogs5py subpackage providing tools.

Subpackages

tools	Tools for the ogs5py package.
script	Script generator for ogs5py.
download	Downloader for ogs5.
output	Tools for ogs5py output files (independent from VTK package).
vtk_viewer	Viewer for a vtk file.
types	type definitions for ogs5

ogs5py.tools.tools

Tools for the ogs5py package.

Classes

Output(file_or_name[, print_log])	A class to duplicate an output stream to stdout.

ogs5py.tools.tools.Output

class ogs5py.tools.tools.**Output**(*file_or_name*, *print_log=True*)

Bases: object

A class to duplicate an output stream to stdout.

Parameters

- **file_or_name** (*filename or open filehandle (writable)*) – File that will be duplicated
- **print_log** (*bool, optional*) – State if log should be printed. Default: True

Methods

close()	Close the file and restore the channel.
flush()	Flush both channels.
write(data)	Write data to both channels.

close()

Close the file and restore the channel.

flush()

Flush both channels.

write(*data*)

Write data to both channels.

File related

search_mkey(fin)	Search for the first main keyword in a given file-stream.
uncomment(line)	Remove OGS comments from a given line of an OGS file.
is_key(sline)	Check if the given splitted line is an OGS key.
is_mkey(sline)	Check if the given splitted line is a main key.
is_skey(sline)	Check if the given splitted line is a sub key.
get_key(sline)	Get the key of a splitted line if there is any, else return "".
find_key_in_list(key, key_list)	Look for the right corresponding key in a list.
format_dict(dict_in)	Format the dictionary to use upper-case keys.
format_content(content)	Format the content to be added to a 2D linewise array.
format_content_line(content)	Format a line of content to be a list of values.
guess_type(string)	Guess the type of a value given as string and return it accordingly.
search_task_id(task_root[, search_ext])	Search for OGS model names in the given path.
split_file_path(path[, abs_path])	Decompose a path to a file.
is_str_array(array)	A routine to check if an array contains strings.

ogs5py.tools.tools.search_mkey

ogs5py.tools.tools.**search_mkey**(*fin*)

Search for the first main keyword in a given file-stream.

Parameters
fin (*stream*) – given opened file

ogs5py.tools.tools.uncomment

ogs5py.tools.tools.**uncomment**(*line*)

Remove OGS comments from a given line of an OGS file.

Comments are indicated by ";". The line is then splitted by whitespaces.

Parameters
line (*str*) – given line

ogs5py.tools.tools.is_key

ogs5py.tools.tools.**is_key**(*sline*)

Check if the given splitted line is an OGS key.

Parameters
sline (*list of str*) – given splitted line

ogs5py.tools.tools.is_mkey

ogs5py.tools.tools.**is_mkey**(*sline*)

Check if the given splitted line is a main key.

Parameters
sline (*list of str*) – given splitted line

ogs5py.tools.tools.is_skey

ogs5py.tools.tools.**is_skey**(*sline*)

Check if the given splitted line is a sub key.

Parameters
sline (*list of str*) – given splitted line

ogs5py.tools.tools.get_key

ogs5py.tools.tools.**get_key**(*sline*)

Get the key of a splitted line if there is any, else return "".

Parameters
sline (*list of str*) – given splitted line

ogs5py.tools.tools.find_key_in_list

ogs5py.tools.tools.**find_key_in_list**(*key*, *key_list*)

Look for the right corresponding key in a list.

key has to start with an given key from the list and the longest key will be returned.

Parameters

- **key** (*str*) – Given key.
- **key_list** (*list of str*) – Valid keys to be checked against.

Returns
found_key – The best match. None if nothing was found.

Return type
str or None

ogs5py.tools.tools.format_dict

ogs5py.tools.tools.**format_dict**(*dict_in*)

Format the dictionary to use upper-case keys.

Parameters
dict_in (*dict*) – input dictionary

ogs5py.tools.tools.format_content

ogs5py.tools.tools.**format_content**(*content*)

Format the content to be added to a 2D linewise array.

Parameters
content (*anything*) – Single object, or list of objects, or list of lists of objects.

ogs5py.tools.tools.format_content_line

ogs5py.tools.tools.**format_content_line**(*content*)

Format a line of content to be a list of values.

Parameters
content (*anything*) – Single object, or list of objects

ogs5py.tools.tools.guess_type

ogs5py.tools.tools.**guess_type**(*string*)

Guess the type of a value given as string and return it accordingly.

Parameters
string (*str*) – given string containing the value

ogs5py.tools.tools.search_task_id

ogs5py.tools.tools.**search_task_id**(*task_root*, *search_ext=None*)

Search for OGS model names in the given path.

Parameters

- **task_root** (*str*) – Path to the destiny folder.
- **search_ext** (*str*) – OGS extension that should be searched for. Default: All known.

Returns
found_ids – List of all found task_ids.

Return type
list of str

ogs5py.tools.tools.split_file_path

ogs5py.tools.tools.**split_file_path**(*path*, *abs_path=False*)

Decompose a path to a file.

Decompose into the dir-path, the basename and the file-extension.

Parameters

- **path** (*string*) – string containing the path to a file
- **abs_path** (*bool, optional*) – convert the path to an absolut path. Default: False

Returns
result – tuple containing the dir-path, basename and file-extension

Return type
tuple of strings

ogs5py.tools.tools.is_str_array

ogs5py.tools.tools.**is_str_array**(*array*)

A routine to check if an array contains strings.

Parameters
array (*iterable*) – array to check

Return type
bool

Geometric tools

rotate_points(points, angle[, ...])	Rotate points around a given rotation point and axis with a given angle.
shift_points(points, vector)	Shift points with a given vector.
transform_points(points, xyz_func, **kwargs)	Transform points with a given function "xyz_func".
hull_deform(x_in, y_in, z_in[, niv_top, ...])	Providing a transformation function to deform a given mesh.
rotation_matrix(vector, angle)	Create a rotation matrix.
volume(typ, *pnt)	Volume of a OGS5 Meshelement.
centroid(typ, *pnt)	Centroid of a OGS5 Meshelement.

ogs5py.tools.tools.rotate_points

ogs5py.tools.tools.**rotate_points**(*points*, *angle*, *rotation_axis=(0.0, 0.0, 1.0)*, *rotation_point=(0.0, 0.0, 0.0)*)

Rotate points around a given rotation point and axis with a given angle.

Parameters

- **points** (*ndarray*) – Array with all points postions.
- **angle** (*float*) – rotation angle given in radial length
- **rotation_axis** (*array_like, optional*) – Array containing the vector for ratation axis. Default: (0,0,1)
- **rotation_point** (*array_like, optional*) – Array containing the vector for ratation base point. Default: (0,0,0)

Returns
new_array – rotated array

Return type
ndarray

ogs5py.tools.tools.shift_points

ogs5py.tools.tools.**shift_points**(*points*, *vector*)

Shift points with a given vector.

Parameters

- **points** (*ndarray*) – Array with all points postions.
- **vector** (*ndarray*) – array containing the shifting vector

Returns
new_array – shifted array

Return type
ndarray

ogs5py.tools.tools.transform_points

ogs5py.tools.tools.**transform_points**(*points*, *xyz_func*, ***kwargs*)

Transform points with a given function "xyz_func".

kwargs will be forwarded to "xyz_func".

Parameters

- **points** (*ndarray*) – Array with all points postions.
- **xyz_func** (*function*) – the function transforming the points x_new, y_new, z_new = f(x_old, y_old, z_old, **kwargs)

Returns
new_array – transformed array

Return type
ndarray

ogs5py.tools.tools.hull_deform

ogs5py.tools.tools.**hull_deform**(*x_in*, *y_in*, *z_in*, *niv_top=10.0*, *niv_bot=0.0*, *func_top=None*, *func_bot=None*, *direction='z'*)

Providing a transformation function to deform a given mesh.

Transformation is in a given direction by a self defined hull-functions `z = func(x, y)`. Could be used with `transform_mesh` and `transform_points`.

Parameters

- **x_in** (*ndarray*) – Array of the x-positions
- **y_in** (*ndarray*) – Array of the y-positions
- **z_in** (*ndarray*) – Array of the z-positions
- **niv_top** (*float*) – height of the top niveau to be deformed by func_top
- **niv_bot** (*float*) – height of the bottom niveau to be deformed by func_bot
- **func_top** (*function or float*) – function deforming the top niveau: `z_top = func_top(x, y)`
- **func_bot** (*function or float*) – function deforming the bottom niveau: `z_bot = func_bot(x, y)`
- **direction** (*string, optional*) – defining the direction of deforming. This direction will be used as z-value. Default: "z"

Returns
x_out, y_out, z_out – transformed arrays

Return type
ndarray

ogs5py.tools.tools.rotation_matrix

ogs5py.tools.tools.**rotation_matrix**(*vector*, *angle*)

Create a rotation matrix.

For rotation around a given vector with a given angle.

Parameters

- **vector** (*ndarray*) – array containing the vector for ratation axis
- **angle** (*float*) – rotation angle given in radial length

Returns

result – matrix to be used for matrix multiplication with vectors to be rotated.

Return type

ndarray

ogs5py.tools.tools.volume

ogs5py.tools.tools.**volume**(*typ*, **pnt*)

Volume of a OGS5 Meshelement.

Parameters

- **typ** (*string*) –

  OGS5 Meshelement type. Should be one of the following:

  - "line" : 1D element with 2 nodes
  - "tri" : 2D element with 3 nodes
  - "quad" : 2D element with 4 nodes
  - "tet" : 3D element with 4 nodes
  - "pyra" : 3D element with 5 nodes
  - "pris" : 3D element with 6 nodes
  - "hex" : 3D element with 8 nodes

- ***pnt** (Node Choordinates `pnt = (x_0, x_1, ...)`) – List of points defining the Meshelement. A point is given as an (x,y,z) tuple and for each point, there can be a stack of points, if the volume should be calculated for multiple elements of the same type.

Returns

Volume – Array containing the volumes of the give elements.

Return type

ndarray

ogs5py.tools.tools.centroid

ogs5py.tools.tools.**centroid**(*typ*, **pnt*)

Centroid of a OGS5 Meshelement.

Parameters

- **typ** (*string*) –

  OGS5 Meshelement type. Should be one of the following:

  – "line" : 1D element with 2 nodes
  – "tri" : 2D element with 3 nodes
  – "quad" : 2D element with 4 nodes
  – "tet" : 3D element with 4 nodes
  – "pyra" : 3D element with 5 nodes
  – "pris" : 3D element with 6 nodes
  – "hex" : 3D element with 8 nodes

- ***pnt** (Node Choordinates `pnt = (x_0, x_1, ...)`) – List of points defining the Meshelement. A point is given as an (x,y,z) tuple and for each point, there can be a stack of points, if the volume should be calculated for multiple elements of the same type.

Returns
centroid – Array containing the Centroids of the give elements.

Return type
ndarray

Notes

The calculation is performed by geometric decomposition of the elements.

https://en.wikipedia.org/wiki/Centroid#By_geometric_decomposition

Array tools

unique_rows(data[, decimals, fast])	Unique made row-data with respect to given precision.
replace(arr, inval, outval)	Replace values of 'arr'.
by_id(array[, ids])	Return a flattend array side-by-side with the array-element ids.
specialrange(val_min, val_max, steps[, typ])	Calculation of special point ranges.
generate_time(time_array[, time_start, ...])	Return a dictionary for the ".tim" file.

ogs5py.tools.tools.unique_rows

ogs5py.tools.tools.**unique_rows**(*data*, *decimals=4*, *fast=True*)

Unique made row-data with respect to given precision.

this is constructed to work best if point-pairs appear. The output is sorted like the input data. data needs to be 2D

Parameters

- **data** (*ndarray*) – 2D array containing the list of vectors that should be made unique
- **decimals** (*int, optional*) – Number of decimal places to round the 'data' to (default: 3). If decimals is negative, it specifies the number of positions to the left of the decimal point. This will not round the output, it is just for comparison of the vectors.
- **fast** (*bool, optional*) – If fast is True, the vector comparison is executed by a decimal comparison. If fast is False, all pairwise distances are calculated. Default: True

Returns

- **result** (*ndarray*) – 2D array of unique rows of data
- **ix** (*ndarray*) – index positions of output in input data (data[ix] = result) len(ix) = result.shape[0]
- **ixr** (*ndarray*) – reversed index positions of input in output data (result[ixr] = data) len(ixr) = data.shape[0]

Notes

This routine will preserve the order within the given array as effectively as possible. If you use it with a stack of 2 arrays and the first one is already unique, the resulting array will still have the first array at the beginning.

---

ogs5py.tools.tools.replace

ogs5py.tools.tools.**replace**(*arr*, *inval*, *outval*)

Replace values of 'arr'.

Replace values defined in 'inval' with values defined in 'outval'.

Parameters

- **arr** (*ndarray*) – array containing the input data
- **inval** (*ndarray*) – values appearing in 'arr' that should be replaced
- **outval** (*ndarray*) – values that should be written in 'arr' instead of values in 'inval'

Returns

result – array of the same shape as 'arr' containing the new data

Return type

ndarray

ogs5py.tools.tools.by_id

ogs5py.tools.tools.**by_id**(*array*, *ids=None*)

Return a flattend array side-by-side with the array-element ids.

Parameters

- **array** (*array-like*) – Input data. will be flattened.
- **ids** (*None or array-like*) – You can provide specific ids if needed. As default, the array-ids are used. Default: None

Return type

zipped (id, array) object

ogs5py.tools.tools.specialrange

ogs5py.tools.tools.**specialrange**(*val_min*, *val_max*, *steps*, *typ='exp'*)

Calculation of special point ranges.

Parameters

- **val_min** (`float`) – Starting value.
- **val_max** (`float`) – Ending value
- **steps** (`int`) – Number of steps.
- **typ** (`str` or `float`, optional) – Setting the kind of range-distribution. One can choose between
  - `"exp"`: for exponential behavior
  - `"log"`: for logarithmic behavior
  - `"geo"`: for geometric behavior
  - `"lin"`: for linear behavior
  - `"quad"`: for quadratic behavior
  - `"cub"`: for cubic behavior
  - `float`: here you can specifi any exponent (“quad” would be equivalent to 2)

  Default: `"exp"`

Returns

Array containing the special range

Return type

`numpy.ndarray`

Examples

```
>>> specialrange(1,10,4)
array([ 1.        ,  2.53034834,  5.23167968, 10.        ])
```

ogs5py.tools.tools.generate_time

ogs5py.tools.tools.**generate_time**(*time_array*, *time_start=0*, *factors=1*, *is_diff=False*)

Return a dictionary for the ".tim" file.

Parameters

- **time_array** (*array-like*) – Input time. will be flattened. Either time step sizes for each step, (is_diff=True) or an array of time-points.
- **time_start** (*float, optional*) – Starting point for time stepping. Default: 0
- **factors** (*int or array-like, optional*) – Repeating factors for each time step. Default: 1
- **is_diff** (*bool, optional*) – State if the given time array contains only the step size for each step. Default: False

Returns

dict – keys: {"TIME_START", "TIME_END", "TIME_STEPS"}

Return type

input dict for ".tim".

ogs5py.tools.script

Script generator for ogs5py.

Generator

gen_script(ogs_class[, script_dir, ...])	Generate a python script for the given model.

ogs5py.tools.script.gen_script

ogs5py.tools.script.**gen_script**(*ogs_class*, *script_dir='/home/docs/checkouts/readthedocs.org/user_builds/ogs5py/checkouts/stable/do*, *script_name='model.py'*, *ogs_cls_name='model'*, *task_root=None*, *task_id=None*, *output_dir=None*, *separate_files=None*)

Generate a python script for the given model.

Parameters

- **ogs_class** (*OGS*) – model class to be converted to a script
- **script_dir** (*str*) – target directory for the script
- **script_name** (*str*) – name for the script file (including .py ending)
- **ogs_cls_name** (*str*) – name of the model in the script
- **task_root** (*str*) – used task_root in the script
- **task_id** (*str*) – used task_id in the script
- **output_dir** (*str*) – used output_dir in the script
- **separate_files** (*list of str or None*) – list of files, that should be written to separate files and then loaded from the script

Notes

This will only create BlockFiles from the script. GLI and MSH files as well as every listed or line-wise file will be stored separately.

Helpers

formater(val)	Format values as string.
get_line(cont_line)	Create content line for the script.
tab(num)	Get tab indentation.
add_block_file(block_file, script[, ...])	Add block-file creation to script.
add_load_file(load_file, script[, ogs_cls_name])	Add a file to be loaded from a script.
add_list_file(list_file, script, typ[, ...])	Add a listed file to be loaded from a script.

ogs5py.tools.script.formater

ogs5py.tools.script.**formater**(*val*)

Format values as string.

Parameters
val (*value*) – input value to be formatted

ogs5py.tools.script.get_line

ogs5py.tools.script.**get_line**(*cont_line*)

Create content line for the script.

Parameters
cont_line (*list of values*) – content line from a BlockFile

ogs5py.tools.script.tab

ogs5py.tools.script.**tab**(*num*)

Get tab indentation.

Parameters
num (*int*) – indentation depth

ogs5py.tools.script.add_block_file

ogs5py.tools.script.**add_block_file**(*block_file*, *script*, *ogs_cls_name='model'*)

Add block-file creation to script.

Parameters

- **block_file** (*BlockFile*) – BlockFile class to be added to the script
- **script** (*stream*) – given opened file for the script
- **ogs_cls_name** (*str*) – name of the model within the script

ogs5py.tools.script.add_load_file

ogs5py.tools.script.**add_load_file**(*load_file*, *script*, *ogs_cls_name='model'*)

Add a file to be loaded from a script.

Parameters

- **load_file** (*OGSFile*) – file that should be saved and then loaded from the script
- **script** (*stream*) – given opened file for the script
- **ogs_cls_name** (*str*) – name of the model within the script

ogs5py.tools.script.add_list_file

ogs5py.tools.script.**add_list_file**(*list_file*, *script*, *typ*, *ogs_cls_name='model'*)

Add a listed file to be loaded from a script.

Parameters

- **list_file** (*File*) – listed file that should be saved and then loaded from the script
- **script** (*stream*) – given opened file for the script
- **typ** (*str*) – typ of the list file
- **ogs_cls_name** (*str*) – name of the model within the script

ogs5py.tools.download

Downloader for ogs5.

Downloader

A downloading routine to get the OSG5 executable.

download_ogs([version, system, path, name, ...])	Download the OGS5 executable.
add_exe(ogs_exe[, dest_name])	Add an OGS5 exe to *OGS5PY_CONFIG*.
reset_download()	Reset all downloads in *OGS5PY_CONFIG*.
OGS5PY_CONFIG	Standard config path for ogs5py.

ogs5py.tools.download.download_ogs

ogs5py.tools.download.**download_ogs**(*version='5.7'*, *system=None*, *path='/home/docs/.config/ogs5py'*, *name=None*, *build=None*)

Download the OGS5 executable.

Parameters

- **version** (str, optional) – Version to download ("5.7", "5.8", "5.7.1"). Default: "5.7"
- **system** (str, optional) – Target system (Linux, Windows, Darwin). Default: platform.system()
- **path** (str, optional) – Destination path. Default: *OGS5PY_CONFIG*
- **name** (str, optional) – Destination file name. Default "ogs[.exe]"
- **build** (str, optional) – Only None and "FEM" supported.

Returns

dest – If an OGS5 executable was successfully downloaded, the file-path is returned.

Return type

str

Notes

There is only an executable on "Darwin" for version "5.7".

Taken from:

- https://www.opengeosys.org/ogs-5/

---

ogs5py.tools.download.add_exe

ogs5py.tools.download.**add_exe**(*ogs_exe*, *dest_name=None*)

Add an OGS5 exe to *OGS5PY_CONFIG*.

Parameters

- **ogs_exe** (str) – Path to the ogs executable to be copied.
- **dest_name** (str, optional) – Destination file name. Default: basename of ogs_exe

Returns

dest – If an OGS5 executable was successfully copied, the file-path is returned.

Return type
str

ogs5py.tools.download.reset_download

ogs5py.tools.download.**reset_download**()
Reset all downloads in *OGS5PY_CONFIG*.

ogs5py.tools.download.OGS5PY_CONFIG

ogs5py.tools.download.**OGS5PY_CONFIG = '/home/docs/.config/ogs5py'**
Standard config path for ogs5py.

Type
str

ogs5py.tools.output

Tools for ogs5py output files (independent from VTK package).

Helpers

get_output_files(task_root, task_id[, pcs, ...])	Get a list of output file paths.
readpvd_single(infile)	Read a paraview pvd file.
split_ply_path(infile[, task_id, line_name, ...])	Retrive ogs-infos from filename for tecplot-polyline output.
split_pnt_path(infile[, task_id, pnt_name, ...])	Retrive ogs-infos from filename for tecplot-polyline output.

ogs5py.tools.output.get_output_files

ogs5py.tools.output.**get_output_files**(*task_root*, *task_id*, *pcs=None*, *typ='VTK'*, *element=None*)

Get a list of output file paths.

Parameters

- **task_root** (*string*) – string containing the path to the directory containing the ogs output
- **task_id** (*string*) – string containing the file name of the ogs task without extension
- **pcs** (*string or None, optional*) – specify the PCS type that should be collected Possible values are:
  - None/"" (no PCS_TYPE specified in *.out)
  - "NO_PCS"
  - "GROUNDWATER_FLOW"
  - "LIQUID_FLOW"
  - "RICHARDS_FLOW"
  - "AIR_FLOW"
  - "MULTI_PHASE_FLOW"
  - "PS_GLOBAL"
  - "HEAT_TRANSPORT"
  - "DEFORMATION"
  - "MASS_TRANSPORT"
  - "OVERLAND_FLOW"
  - "FLUID_MOMENTUM"
  - "RANDOM_WALK"

  Default : None
- **typ** (*string, optional*) – Type of the output ("VTK", "PVD", "TEC_POINT" or "TEC_POLYLINE"). Default : "VTK"
- **element** (*string or None, optional*) – For tecplot output you can specify the name of the output element. (Point-name of Line-name from GLI file) Default: None

ogs5py.tools.output.readpvd_single

ogs5py.tools.output.**readpvd_single**(*infile*)

Read a paraview pvd file.

Convert all concerned files to a dictionary containing their data.

ogs5py.tools.output.split_ply_path

ogs5py.tools.output.**split_ply_path**(*infile*, *task_id=None*, *line_name=None*, *PCS_name=None*, *split_extra=False*)

Retrive ogs-infos from filename for tecplot-polyline output.

{id}_ply_{line}_t{n}[_{PCS+extra}].tec

ogs5py.tools.output.split_pnt_path

ogs5py.tools.output.**split_pnt_path**(*infile*, *task_id=None*, *pnt_name=None*, *PCS_name=None*, *split_extra=False*, *guess_PCS=False*)

Retrive ogs-infos from filename for tecplot-polyline output.

{id}_time_{pnt}[_{PCS+extra}].tec

ogs5py.tools.vtk_viewer

Viewer for a vtk file.

Viewer

show_vtk(vtkfile[, log_scale])	Display a given mesh colored by its material ID.

ogs5py.tools.vtk_viewer.show_vtk

ogs5py.tools.vtk_viewer.**show_vtk**(*vtkfile*, *log_scale=False*)

Display a given mesh colored by its material ID.

Parameters

- **vtkfile** (str) – Path to the vtk/vtu file to show.
- **log_scale** (*bool, optional*) – State if the data should be shown in log scale. Default: False

Notes

This routine needs "mayavi" to display the mesh. (see here: https://github.com/enthought/mayavi)

ogs5py.tools.types

type definitions for ogs5

GLI related Constants

EMPTY_GLI	empty gli dict
GLI_KEYS	ogs gli dict keys
GLI_KEY_LIST	gli main keys
EMPTY_PLY	empty ogs gli polyline dict
PLY_KEYS	ogs gli polyline keys
PLY_KEY_LIST	gli polyline keys
PLY_TYPES	gli polyline key types
EMPTY_SRF	empty ogs gli surface dict
SRF_KEYS	ogs gli surface keys
SRF_KEY_LIST	gli surface keys
SRF_TYPES	gli surface key types
EMPTY_VOL	empty ogs gli volume dict
VOL_KEYS	ogs gli volume keys
VOL_KEY_LIST	gli volume keys
VOL_TYPES	gli volume key types

MSH related Constants

EMPTY_MSH	empty mesh dict
MESH_KEYS	ogs mesh dict-keys
MESH_DATA_KEYS	ogs mesh data keys
ELEM_1D	ogs element names
ELEM_2D	ogs element names
ELEM_3D	ogs element names
ELEM_DIM	ogs element names
ELEM_NAMES	ogs element names
ELEM_TYP	type code per element name
ELEM_TYP1D	type code per element name
ELEM_TYP2D	type code per element name
ELEM_TYP3D	type code per element name
VTK_TYP	vtk type codes per element name
MESHIO_NAMES	coresponding element names in meshio
NODE_NO	Node numbers per element name

General Constants

STRTYPE	type: base string type
PCS_TYP	PCS types
PCS_EXT	PCS file extensions with _
PRIM_VAR	primary variables
PRIM_VAR_BY_PCS	primary variables per PCS
OGS_EXT	all ogs file extensions
MULTI_FILES	all ogs files that can occure multiple times

ogs5py.tools.types.**EMPTY_GLI** = {'point_md': None, 'point_names': None, 'points': None, 'polylines': [], 'surfaces': [], 'volumes': []}

empty gli dict

Type
dict

ogs5py.tools.types.**GLI_KEYS** = {'point_md', 'point_names', 'points', 'polylines', 'surfaces', 'volumes'}

ogs gli dict keys

Type
set

ogs5py.tools.types.**GLI_KEY_LIST** = ['#POINTS', '#POLYLINE', '#SURFACE', '#VOLUME', '#STOP']

gli main keys

Type
list

ogs5py.tools.types.**EMPTY_PLY** = {'EPSILON': None, 'ID': None, 'MAT_GROUP': None, 'NAME': None, 'POINTS': None, 'POINT_VECTOR': None, 'TYPE': None}

empty ogs gli polyline dict

Type
dict

ogs5py.tools.types.**PLY_KEYS** = {'EPSILON', 'ID', 'MAT_GROUP', 'NAME', 'POINTS', 'POINT_VECTOR', 'TYPE'}

ogs gli polyline keys

Type
set

ogs5py.tools.types.**PLY_KEY_LIST** = ['ID', 'NAME', 'TYPE', 'EPSILON', 'MAT_GROUP', 'POINTS', 'POINT_VECTOR']

gli polyline keys

Type
list

ogs5py.tools.types.**PLY_TYPES** = [<class 'int'>, <class 'str'>, <class 'int'>, <class 'float'>, <class 'int'>, <class 'list'>, <class 'str'>]

gli polyline key types

Type
list

ogs5py.tools.types.**EMPTY_SRF** = {'EPSILON': None, 'ID': None, 'MAT_GROUP': None, 'NAME': None, 'POLYLINES': None, 'TIN': None, 'TYPE': None}

empty ogs gli surface dict

Type
dict

ogs5py.tools.types.**SRF_KEYS** = {'EPSILON', 'ID', 'MAT_GROUP', 'NAME', 'POLYLINES', 'TIN', 'TYPE'}

ogs gli surface keys

Type
set

ogs5py.tools.types.SRF_KEY_LIST = ['ID', 'NAME', 'EPSILON', 'TYPE', 'TIN', 'MAT_GROUP', 'POLYLINES']

gli surface keys

Type
list

ogs5py.tools.types.SRF_TYPES = [<class 'int'>, <class 'str'>, <class 'float'>, <class 'int'>, <class 'str'>, <class 'int'>, <class 'list'>]

gli surface key types

Type
list

ogs5py.tools.types.EMPTY_VOL = {'LAYER': None, 'MAT_GROUP': None, 'NAME': None, 'SURFACES': None, 'TYPE': None}

empty ogs gli volume dict

Type
dict

ogs5py.tools.types.VOL_KEYS = {'LAYER', 'MAT_GROUP', 'NAME', 'SURFACES', 'TYPE'}

ogs gli volume keys

Type
set

ogs5py.tools.types.VOL_KEY_LIST = ['NAME', 'TYPE', 'SURFACES', 'MAT_GROUP', 'LAYER']

gli volume keys

Type
list

ogs5py.tools.types.VOL_TYPES = [<class 'str'>, <class 'str'>, <class 'list'>, <class 'str'>, <class 'int'>]

gli volume key types

Type
list

ogs5py.tools.types.EMPTY_MSH = {'element_id': {}, 'elements': {}, 'material_id': {}, 'mesh_data': {}, 'nodes': array([], shape=(0, 3), dtype=float64)}

empty mesh dict

Type
dict

ogs5py.tools.types.MESH_KEYS = {'element_id', 'elements', 'material_id', 'mesh_data', 'nodes'}

ogs mesh dict-keys

Type
set

ogs5py.tools.types.MESH_DATA_KEYS = {'AXISYMMETRY', 'CROSS_SECTION', 'GEO_NAME', 'GEO_TYPE', 'LAYER', 'PCS_TYPE'}

ogs mesh data keys

Type
set

ogs5py.tools.types.ELEM_1D = ['line']

ogs element names

Type
set

ogs5py.tools.types.**ELEM_2D = ['tri', 'quad']**

ogs element names

Type
set

ogs5py.tools.types.**ELEM_3D = ['tet', 'pyra', 'pris', 'hex']**

ogs element names

Type
set

ogs5py.tools.types.**ELEM_DIM = [['line'], ['tri', 'quad'], ['tet', 'pyra', 'pris', 'hex']]**

ogs element names

Type
set

ogs5py.tools.types.**ELEM_NAMES = ['line', 'tri', 'quad', 'tet', 'pyra', 'pris', 'hex']**

ogs element names

Type
list

ogs5py.tools.types.**ELEM_TYP = {0: 'line', 1: 'tri', 2: 'quad', 3: 'tet', 4: 'pyra', 5: 'pris', 6: 'hex', 'line': 0, 'tri': 1, 'quad': 2, 'tet': 3, 'pyra': 4, 'pris': 5, 'hex': 6}**

type code per element name

Type
dict

ogs5py.tools.types.**ELEM_TYP1D = {0: 'line', 'line': 0}**

type code per element name

Type
dict

ogs5py.tools.types.**ELEM_TYP2D = {1: 'tri', 2: 'quad', 'tri': 1, 'quad': 2}**

type code per element name

Type
dict

ogs5py.tools.types.**ELEM_TYP3D = {3: 'tet', 4: 'pyra', 5: 'pris', 6: 'hex', 'tet': 3, 'pyra': 4, 'pris': 5, 'hex': 6}**

type code per element name

Type
dict

ogs5py.tools.types.**VTK_TYP = {3: 'line', 5: 'tri', 9: 'quad', 10: 'tet', 14: 'pyra', 13: 'pris', 12: 'hex', 'line': 3, 'tri': 5, 'quad': 9, 'tet': 10, 'pyra': 14, 'pris': 13, 'hex': 12}**

vtk type codes per element name

Type
dict

ogs5py.tools.types.**MESHIO_NAMES = ['line', 'triangle', 'quad', 'tetra', 'pyramid', 'wedge', 'hexahedron']**

coresponding element names in meshio

Type
list

ogs5py.tools.types.**NODE_NO = {0: 2, 1: 3, 2: 4, 3: 4, 4: 5, 5: 6, 6: 8, 'line': 2, 'tri': 3, 'quad': 4, 'tet': 4, 'pyra': 5, 'pris': 6, 'hex': 8}**

Node numbers per element name

Type
dict

ogs5py.tools.types.**STRTYPE = <class 'str'>**

base string type

Type
type

ogs5py.tools.types.**PCS_TYP = ['', 'GROUNDWATER_FLOW', 'LIQUID_FLOW', 'RICHARDS_FLOW', 'AIR_FLOW', 'MULTI_PHASE_FLOW', 'PS_GLOBAL', 'HEAT_TRANSPORT', 'DEFORMATION', 'MASS_TRANSPORT', 'OVERLAND_FLOW', 'FLUID_MOMENTUM', 'RANDOM_WALK', 'NO_PCS', 'TNEQ', 'TES', 'DEFORMATION_SINGLEFLOW_MONO', 'MULTI_COMPONENTIAL_FLOW']**

PCS types

Type
list

ogs5py.tools.types.**PCS_EXT = ['', '_GROUNDWATER_FLOW', '_LIQUID_FLOW', '_RICHARDS_FLOW', '_AIR_FLOW', '_MULTI_PHASE_FLOW', '_PS_GLOBAL', '_HEAT_TRANSPORT', '_DEFORMATION', '_MASS_TRANSPORT', '_OVERLAND_FLOW', '_FLUID_MOMENTUM', '_RANDOM_WALK', '_NO_PCS', '_TNEQ', '_TES', '_DEFORMATION_SINGLEFLOW_MONO', '_MULTI_COMPONENTIAL_FLOW']**

PCS file extensions with _

Type
list

ogs5py.tools.types.**PRIM_VAR = [[''], ['HEAD'], ['PRESSURE1'], ['PRESSURE1'], ['PRESSURE1', 'TEMPERATURE1'], ['PRESSURE1', 'PRESSURE2'], ['PRESSURE1, SATURATION2'], ['TEMPERATURE1'], ['DISPLACEMENT_X1', 'DISPLACEMENT_Y1', 'DISPLACEMENT_Z1'], [''], ['HEAD'], ['VELOCITY1_X', 'VELOCITY1_Y', 'VELOCITY1_Z'], [''], [''], [''], [''], [''], ['']]**

primary variables

Type
list

ogs5py.tools.types.**PRIM_VAR_BY_PCS = {'': [''], 'AIR_FLOW': ['PRESSURE1', 'TEMPERATURE1'], 'DEFORMATION': ['DISPLACEMENT_X1', 'DISPLACEMENT_Y1', 'DISPLACEMENT_Z1'], 'DEFORMATION_SINGLEFLOW_MONO': [''], 'FLUID_MOMENTUM': ['VELOCITY1_X', 'VELOCITY1_Y', 'VELOCITY1_Z'], 'GROUNDWATER_FLOW': ['HEAD'], 'HEAT_TRANSPORT': ['TEMPERATURE1'], 'LIQUID_FLOW': ['PRESSURE1'], 'MASS_TRANSPORT': [''], 'MULTI_COMPONENTIAL_FLOW': [''], 'MULTI_PHASE_FLOW': ['PRESSURE1', 'PRESSURE2'], 'NO_PCS': [''], 'OVERLAND_FLOW': ['HEAD'], 'PS_GLOBAL': ['PRESSURE1, SATURATION2'], 'RANDOM_WALK': [''], 'RICHARDS_FLOW': ['PRESSURE1'], 'TES': [''], 'TNEQ': ['']}**

primary variables per PCS

Type
dict

ogs5py.tools.types.**OGS_EXT = ['.msh', '.gli', '.ddc', '.pcs', '.rfd', '.cct', '.fct', '.bc', '.ic', '.st', '.mmp', '.msp', '.mfp', '.mcp', '.gem', '.krc', '.pqc', '.rei', '.pct', '.num', '.tim', '.out']**

all ogs file extensions

Type
list

ogs5py.tools.types.**MULTI_FILES = ['mpd', 'gli_ext', 'rfr', 'gem_init', 'asc']**

all ogs files that can occure multiple times

Type
list

3.3 Classes

OGS model Base Class

Class to setup an ogs model

OGS([task_root, task_id, output_dir])	Class for an OGS5 model.

ogs5py.OGS

class ogs5py.**OGS**(*task_root=None*, *task_id='model'*, *output_dir=None*)

Bases: `object`

Class for an OGS5 model.

In this class everything for an OGS5 model can be specified.

Parameters

- **task_root** (`str`, optional) – Path to the destiny model folder. Default: cwd+"ogs5model"
- **task_id** (`str`, optional) – Name for the ogs task. Default: "model"
- **output_dir** (`str` or `None`, optional) – Path to the output directory. Default: `None`

Notes

The following Classes are present as attributes

bc
[Boundary Condition] Information of the Boundary Conditions for the model.

cct
[Communication Table] Information of the Communication Table for the model.

fct
[Function] Information of the Function definitions for the model.

gem
[geochemical thermodynamic modeling coupling] Information of the geochemical thermodynamic modeling coupling for the model.

gli
[Geometry] Information of the Geometry for the model.

ic
[Initial Condition] Information of the Initial Conditions for the model.

krc
[Kinetric Reaction] Information of the Kinetric Reaction for the model.

mcp
[reactive components for modelling chemical processes] Information of the reactive components for modelling chemical processes for the model.

mfp
[Fluid Properties] Information of the Fluid Properties for the model.

mmp
[Medium Properties] Information of the Medium Properties for the model.

msh
[Mesh] Information of the Mesh for the model.

msp
[Solid Properties] Information of the Solid Properties for the model.

num
[Settings for the numerical solver] Information of the numerical solver for the model.

out
[Output Settings] Information of the Output Settings for the model.

pcs
[Process settings] Information of the Process settings for the model.

pct
[Particle Definition for Random walk] Information of the Particles defined for Randomwalk setting.

pqc
[Phreqqc coupling (not supported yet)] Information of the Boundary Conditions for the model.

pqcdat
[Phreqqc coupling (the phreeqc.dat file)] phreeqc.dat file for the model. (just a line-wise file with no comfort)

rei
[Reaction Interface] Information of the Reaction Interface for the model.

rfd
[definition of time-curves for variing BCs or STs] Information of the time curves for the model.

st
[Source Term] Information of the Source Term for the model.

tim
[Time settings] Information of the Time settings for the model.

Additional

mpd
[Distributed Properties (list of files)] Information of the Distributed Properties for the model.

gli_ext
[list for external Geometry definition] External definition of surfaces (TIN) or polylines (POINT_VECTOR)

rfr
[list of restart files] RESTART files as defined in the INITIAL_CONDITION

gem_init
[list of GEMS3K input files (lst file)] given as GEMinit classes

asc
[list of ogs ASC files] This file type comes either from .tim .pcs or .gem

copy_files
[list of path-strings] Files that should be copied to the destiny folder.

Attributes

bot_com
Get and set the bottom comment for the ogs files.

has_output_dir
`bool`: State if the model has a output directory.

output_dir
`str`: output directory path of the ogs model.

task_id
str: task_id (name) of the ogs model.

task_root
Get and set the task_root path of the ogs model.

top_com
Get and set the top comment for the ogs files.

Methods

add_asc(asc_file)	Method to add a ASC file.
add_copy_file(path)	Method to add an arbitrary file that should be copied.
add_gem_init(gem_init_file)	Method to add a GEMS3K input file.
add_gli_ext(gli_ext_file)	Method to add an external Geometry definition file to the model.
add_mpd(mpd_file)	Method to add an ogs MEDIUM_PROPERTIES_DISTRIBUTED file to the model.
add_rfr(rfr_file)	Method to add an ogs RESTART file to the model.
del_asc([index])	Method to delete a ASC file.
del_copy_file([index])	Method to delete a copy-file.
del_gem_init([index])	Method to delete GEMS3K input file.
del_gli_ext([index])	Method to delete external Geometry file.
del_mpd([index])	Method to delete MEDIUM_PROPERTIES_DISTRIBUTED file.
del_rfr([index])	Method to delete RESTART file.
gen_script([script_dir, script_name, ...])	Generate a python script for the given model.
load_model(task_root[, task_id, ...])	Load an existing OGS5 model.
output_files([pcs, typ, element, output_dir])	Get a list of output file paths.
readpvd([pcs, output_dir])	Read the paraview pvd files of this OGS5 model.
readtec_point([pcs, output_dir])	Collect TECPLOT point output from this OGS5 model.
readtec_polyline([pcs, trim, output_dir])	Collect TECPLOT polyline output from this OGS5 model.
readvtk([pcs, output_dir])	Reader for vtk outputfiles of this OGS5 model.
reset()	Delete every content.
run_model([ogs_exe, ogs_name, print_log, ...])	Run the defined OGS5 model.
write_input()	Method to call all write_file() methods that are initialized.

add_asc(*asc_file*)

Method to add a ASC file.

See ogs5py.ASC for further information

add_copy_file(*path*)

Method to add an arbitrary file that should be copied.

The base-name of the file will be keept and it will be copied to the task-root when the "write" routine is called.

add_gem_init(*gem_init_file*)

Method to add a GEMS3K input file.

This is usually generated by GEM-SELEKTOR.

See ogs5py.GEM and ogs5py.GEMinit for further information

add_gli_ext(*gli_ext_file*)

Method to add an external Geometry definition file to the model.

This is used for TIN definition in SURFACE or POINT_VECTOR definition in POLYLINE in the GLI file.

See ogs5py.GLI for further information

add_mpd(*mpd_file*)

Method to add an ogs MEDIUM_PROPERTIES_DISTRIBUTED file to the model.

This is used for disributed information in the MMP file.

See ogs5py.MPD for further information

add_rfr(*rfr_file*)

Method to add an ogs RESTART file to the model.

This is used for disributed information in the IC file.

See ogs5py.IC for further information

del_asc(*index=None*)

Method to delete a ASC file.

Parameters
index (*int or None, optional*) – The index of the ASC file that should be deleted. If None, all ASC files are deleted. Default: None

del_copy_file(*index=None*)

Method to delete a copy-file.

Parameters
index (*int or None, optional*) – The index of the copy-file that should be deleted. If None, all copy-files are deleted. Default: None

del_gem_init(*index=None*)

Method to delete GEMS3K input file.

Parameters
index (*int or None, optional*) – The index of the GEMS3K file that should be deleted. If None, all GEMS3K files are deleted. Default: None

del_gli_ext(*index=None*)

Method to delete external Geometry file.

Parameters
index (*int or None, optional*) – The index of the external gli file that should be deleted. If None, all external gli files are deleted. Default: None

del_mpd(*index=None*)

Method to delete MEDIUM_PROPERTIES_DISTRIBUTED file.

Parameters
index (*int or None, optional*) – The index of the mpd-file that should be deleted. If None, all mpd-files are deleted. Default: None

del_rfr(*index=None*)

Method to delete RESTART file.

Parameters
index (*int or None, optional*) – The index of the RESTART file that should be deleted. If None, all RESTART files are deleted. Default: None

`gen_script`(*script_dir='/home/docs/checkouts/readthedocs.org/user_builds/ogs5py/checkouts/stable/docs/source/ogs_scrip*
script_name='model.py', *ogs_cls_name='model'*, *task_root=None*, *task_id=None*,
output_dir=None, *separate_files=None*)

Generate a python script for the given model.

Parameters

- **script_dir** (*str*) – target directory for the script
- **script_name** (*str*) – name for the script file (including .py ending)
- **ogs_cls_name** (*str*) – name of the model in the script
- **task_root** (*str*) – used task_root in the script
- **task_id** (*str*) – used task_id in the script
- **output_dir** (*str*) – used output_dir in the script
- **separate_files** (*list of str or None*) – list of files, that should be written to separate files and then loaded from the script

Notes

This will only create BlockFiles from the script. GLI and MSH files as well as every other file are stored separately.

`load_model`(*task_root*, *task_id=None*, *use_task_root=False*, *use_task_id=False*, *skip_files=None*,
skip_ext=None, *encoding=None*, *verbose=False*, *search_ext=None*)

Load an existing OGS5 model.

Parameters

- **task_root** (*str*) – Path to the destiny folder.
- **task_id** (*str or None, optional*) – Task ID of the model to load. If None is given, it will be determind by the found files. If multiple possible task_ids were found, the first one in alphabetic order will be used. Default: None
- **use_task_root** (*Bool, optional*) – State if the given task_root should be used for this model. Default: False
- **use_task_id** (*Bool, optional*) – State if the given task_id should be used for this model. Default: False
- **skip_files** (*list or None, optional*) – List of file-names, that should not be read. Default: None
- **skip_ext** (*list or None, optional*) – List of file-extensions, that should not be read. Default: None
- **encoding** (*str or None, optional*) – encoding of the given files. If `None` is given, the system standard is used. Default: `None`
- **verbose** (*bool, optional*) – Print information of the reading process. Default: False
- **search_ext** (*str*) – OGS extension that should be searched for. Default: ".pcs"

Notes

This method will search for all known OGS5 file-extensions in the given path (task_root). Additional files from:

- GLI (POINT_VECTOR + TIN)
- MMP (distributed media properties)

- IC (RESTART)
- GEM (GEM3SK init file)

will be read automatically.

If you get an `UnicodeDecodeError` try loading with:

```
encoding="ISO-8859-15"
```

---

output_files(*pcs=None*, *typ='VTK'*, *element=None*, *output_dir=None*)

Get a list of output file paths.

Parameters

- **pcs** (*string or None, optional*) – specify the PCS type that should be collected Possible values are:
  - None/"" (no PCS_TYPE specified in *.out)
  - "NO_PCS"
  - "GROUNDWATER_FLOW"
  - "LIQUID_FLOW"
  - "RICHARDS_FLOW"
  - "AIR_FLOW"
  - "MULTI_PHASE_FLOW"
  - "PS_GLOBAL"
  - "HEAT_TRANSPORT"
  - "DEFORMATION"
  - "MASS_TRANSPORT"
  - "OVERLAND_FLOW"
  - "FLUID_MOMENTUM"
  - "RANDOM_WALK"

  Default : None
- **typ** (*string, optional*) – Type of the output ("VTK", "PVD", "TEC_POINT" or "TEC_POLYLINE"). Default : "VTK"
- **element** (*string or None, optional*) – For tecplot output you can specify the name of the output element. (Point-name of Line-name from GLI file) Default: None

readpvd(*pcs='ALL'*, *output_dir=None*)

Read the paraview pvd files of this OGS5 model.

All concerned files are converted to a dictionary containing their data

Parameters

- **pcs** (*string or None, optional*) – specify the PCS type that should be collected Possible values are:
  - None/"" (no PCS_TYPE specified in *.out)
  - "NO_PCS"
  - "GROUNDWATER_FLOW"
  - "LIQUID_FLOW"
  - "RICHARDS_FLOW"

- "AIR_FLOW"
- "MULTI_PHASE_FLOW"
- "PS_GLOBAL"
- "HEAT_TRANSPORT"
- "DEFORMATION"
- "MASS_TRANSPORT"
- "OVERLAND_FLOW"
- "FLUID_MOMENTUM"
- "RANDOM_WALK"

You can get a list with all known PCS-types by setting PCS="ALL" Default : "ALL"

- **output_dir** (*:any:'None' or :class:'str', optional*) – Sometimes OGS5 doesn't put the output in the right directory. You can specify a separate output directory here in this case. Default: :any:'None'

Returns

result – keys are the point names and the items are the data from the corresponding files if pcs="ALL", the output is a dictionary with the PCS-types as keys

Return type

dict

`readtec_point`(*pcs='ALL'*, *output_dir=None*)

Collect TECPLOT point output from this OGS5 model.

Parameters

- **pcs** (*string or None, optional*) – specify the PCS type that should be collected Possible values are:
  - None/"" (no PCS_TYPE specified in *.out)
  - "NO_PCS"
  - "GROUNDWATER_FLOW"
  - "LIQUID_FLOW"
  - "RICHARDS_FLOW"
  - "AIR_FLOW"
  - "MULTI_PHASE_FLOW"
  - "PS_GLOBAL"
  - "HEAT_TRANSPORT"
  - "DEFORMATION"
  - "MASS_TRANSPORT"
  - "OVERLAND_FLOW"
  - "FLUID_MOMENTUM"
  - "RANDOM_WALK"

  You can get a list with all known PCS-types by setting PCS="ALL" Default : "ALL"

- **output_dir** (*:any:'None' or :class:'str', optional*) – Sometimes OGS5 doesn't put the output in the right directory. You can specify a separate output directory here in this case. Default: :any:'None'

Returns
result – Keys are the point names and the items are the data from the corresponding files. If pcs="ALL", the output is a dictionary with the PCS-types as keys.

Return type
dict

readtec_polyline(*pcs='ALL'*, *trim=True*, *output_dir=None*)

Collect TECPLOT polyline output from this OGS5 model.

Parameters

- **pcs** (*string or None, optional*) – specify the PCS type that should be collected Possible values are:
  - None/"" (no PCS_TYPE specified in *.out)
  - "NO_PCS"
  - "GROUNDWATER_FLOW"
  - "LIQUID_FLOW"
  - "RICHARDS_FLOW"
  - "AIR_FLOW"
  - "MULTI_PHASE_FLOW"
  - "PS_GLOBAL"
  - "HEAT_TRANSPORT"
  - "DEFORMATION"
  - "MASS_TRANSPORT"
  - "OVERLAND_FLOW"
  - "FLUID_MOMENTUM"
  - "RANDOM_WALK"

  You can get a list with all known PCS-types by setting pcs="ALL" Default : "ALL"
- **output_dir** (*:any:'None' or :class:'str', optional*) – Sometimes OGS5 doesn't put the output in the right directory. You can specify a separate output directory here in this case. Default: :any:'None'
- **trim** (*Bool, optional*) – if the ply_ids are not continuous, there will be "None" values in the output list. If trim is "True" these values will be eliminated. If there is just one output for a polyline, the list will be eliminated and the output will be the single dict. Default : True

Returns
result – keys are the Polyline names and the items are lists sorted by the ply_id (it is assumed, that the ply_ids are continuous, if not, the corresponding list entries are "None") if pcs="ALL", the output is a dictionary with the PCS-types as keys

Return type
dict

readvtk(*pcs='ALL'*, *output_dir=None*)

Reader for vtk outputfiles of this OGS5 model.

Parameters

- **pcs** (*string or None, optional*) – specify the PCS type that should be collected Possible values are:
  - None/"" (no PCS_TYPE specified in *.out)
  - "NO_PCS"
  - "GROUNDWATER_FLOW"
  - "LIQUID_FLOW"
  - "RICHARDS_FLOW"
  - "AIR_FLOW"
  - "MULTI_PHASE_FLOW"
  - "PS_GLOBAL"
  - "HEAT_TRANSPORT"
  - "DEFORMATION"
  - "MASS_TRANSPORT"
  - "OVERLAND_FLOW"
  - "FLUID_MOMENTUM"
  - "RANDOM_WALK"

  You can get a list with all known PCS-types by setting PCS="ALL" Default : None
- **output_dir** (*:any:'None' or :class:'str', optional*) – Sometimes OGS5 doesn't put the output in the right directory. You can specify a separate output directory here in this case. Default: :any:'None'

Returns

result – keys are the point names and the items are the data from the corresponding files if pcs="ALL", the output is a dictionary with the PCS-types as keys

Return type

dict

`reset()`

Delete every content.

`run_model(ogs_exe=None, ogs_name='ogs', print_log=True, save_log=True, log_path=None, log_name=None, timeout=None)`

Run the defined OGS5 model.

Parameters

- **ogs_exe** (*str or None, optional*) – path to the ogs executable. If `None` is given, the default sys path will be searched with `which`. Can be a folder containing the exe with basename: ogs_name. It will first look in the *OGS5PY_CONFIG* folder. Default: None
- **ogs_name** (*str or None, optional*) – Name of to the ogs executable to search for. Just used if ,ogs_exe is `None`. Default: `"ogs"`
- **print_log** (*bool, optional*) – state if the ogs output should be displayed in the terminal. Default: True
- **save_log** (*bool, optional*) – state if the ogs output should be saved to a file. Default: True
- **log_path** (*str or None, optional*) – Path, where the log file should be saved. Default: None (the defined output directory or the task_root directory)

- **log_name** (*str or None, optional*) – Name of the log file. Default: None (task_id+time+"_log.txt")
- **timeout** (*int or None, optional*) – Time to wait for OGS5 to finish in seconds. Default: None

Returns
success – State if OGS5 terminated 'normally'. (Allways true on Windows.)

Return type
bool

`write_input()`
Method to call all write_file() methods that are initialized.

`property bot_com`
Get and set the bottom comment for the ogs files.

`property has_output_dir`
State if the model has a output directory.

Type
bool

`property output_dir`
output directory path of the ogs model.

Type
str

`property task_id`
task_id (name) of the ogs model.

Type
str

`property task_root`
Get and set the task_root path of the ogs model.

`property top_com`
Get and set the top comment for the ogs files.

File Classes

Classes for all OGS5 Files. See: *ogs5py.fileclasses*

ASC(**OGS_Config)	Class for the ogs ASC file.
BC(**OGS_Config)	Class for the ogs BOUNDARY CONDITION file.
CCT(**OGS_Config)	Class for the ogs COMMUNICATION TABLE file.
DDC(**OGS_Config)	Class for the ogs MPI DOMAIN DECOMPOSITION file.
FCT(**OGS_Config)	Class for the ogs FUNCTION file.
GEM(**OGS_Config)	Class for the ogs GEOCHEMICAL THERMODYNAMIC MODELING COUPLING file.
GEMinit([lst_name, dch, ipm, dbr, ...])	Class for GEMS3K input file.
GLI([gli_dict])	Class for the ogs GEOMETRY file.
GLIext([typ, data, name, file_ext, ...])	Class for an external definition for the ogs GEOMETRY file.
IC(**OGS_Config)	Class for the ogs INITIAL_CONDITION file.
RFR([variables, data, units, headers, name, ...])	Class for the ogs RESTART file, if the DIS_TYPE in IC is set to RESTART.
KRC(**OGS_Config)	Class for the ogs KINETRIC REACTION file.
MCP(**OGS_Config)	Class for the ogs COMPONENT_PROPERTIES file.
MFP(**OGS_Config)	Class for the ogs FLUID PROPERTY file.
MMP(**OGS_Config)	Class for the ogs MEDIUM_PROPERTIES file.
MPD([name, file_ext])	Class for the ogs MEDIUM_PROPERTIES_DISTRIBUTED file.
MSH([mesh_list])	Class for a multi layer mesh file that contains multiple '#FEM_MSH' Blocks.
MSP(**OGS_Config)	Class for the ogs SOLID_PROPERTIES file.
NUM(**OGS_Config)	Class for the ogs NUMERICS file.
OUT(**OGS_Config)	Class for the ogs OUTPUT file.
PCS(**OGS_Config)	Class for the ogs PROCESS file.
PCT([data, s_flag, task_root, task_id])	Class for the ogs Particle file, if the PCS TYPE is RANDOM_WALK.
PQC(**OGS_Config)	Class for the ogs PHREEQC interface file.
PQCdat(**OGS_Config)	Class for the ogs PHREEQC dat file.
REI(**OGS_Config)	Class for the ogs REACTION_INTERFACE file.
RFD(**OGS_Config)	Class for the ogs USER DEFINED TIME CURVES file.
ST(**OGS_Config)	Class for the ogs SOURCE_TERM file.
TIM(**OGS_Config)	Class for the ogs TIME_STEPPING file.

3.4 Functions

Geometric

Geometric routines

hull_deform(x_in, y_in, z_in[, niv_top, ...])	Providing a transformation function to deform a given mesh.

Searching

Routine to search for a valid ogs id in a directory

search_task_id(task_root[, search_ext])	Search for OGS model names in the given path.

Formatting

Routines to format/generate data in the right way for the input

by_id(array[, ids])	Return a flattend array side-by-side with the array-element ids.
specialrange(val_min, val_max, steps[, typ])	Calculation of special point ranges.
generate_time(time_array[, time_start, ...])	Return a dictionary for the ".tim" file.

Downloading

Routine to download OGS5.

download_ogs([version, system, path, name, ...])	Download the OGS5 executable.
add_exe(ogs_exe[, dest_name])	Add an OGS5 exe to OGS5PY_CONFIG.
reset_download()	Reset all downloads in OGS5PY_CONFIG.
OGS5PY_CONFIG	Standard config path for ogs5py.

Plotting

Routine to download OGS5.

show_vtk(vtkfile[, log_scale])	Display a given mesh colored by its material ID.

Information

OGS_EXT	all ogs file extensions
PCS_TYP	PCS types
PRIM_VAR_BY_PCS	primary variables per PCS

CHAPTER 4

CHANGELOG

All notable changes to **ogs5py** will be documented in this file.

4.1 1.3.0 - 2023-04

See #18

Enhancements

- move to `src/` base package structure
- use hatchling as build backend
- drop py36 support
- added archive support
- simplify documentation

Bugfixes

- remove usage of deprecated `np.int`

4.2 1.2.2 - 2022-05-25

Bugfixes

- `MSH.load` now uses `engine="python"` as fallback in pandas to successfully read ogs meshes in some corner cases #16
- removed redundant `from io import open` which were there for py2 compatibility #16

4.3 1.2.1 - 2022-05-15

Enhancements

- `MSH.import_mesh` can handle `meshio.Mesh` as input now #13

Changes

- `pygmsh` support was removed. You can't use pygmsh Geometry objects to generate meshes anymore. Please generate beforehand and import the generated mesh. Other generators are using `gmsh` directly now. #13

4.4 1.2.0 - 2022-05-15

Enhancements

- move to a `pyproject.toml` based installation: d5ea756
- move from `develop`/`master` branches to a single `main` branch
- use GitHub Actions for CI: b6811ce
- use f-strings where possible #11
- simplified documentation #11
- added changelog to documentation #11
- added citation file and paper reference #11
- use Python 3 style classes #11

Changes

- `downlaod_ogs` only supports version "5.7", "5.7.1" and "5.8" since the CI for OGS5 was shut down: 8b1cc91

Bugfixes

- make `downlaod_ogs` work again 8b1cc91
- documentation fix in `GLI.add_polyline` #7
- require `pygmsh<7` for now #11

4.5 1.1.1 - 2020-04-02

Bugfixes

- check if `__version__` is present (only if installed)

4.6 1.1.0 - 2020-03-22

Bugfixes

- meshio 4 was not compatible
- fixed integer type in exporting meshes with element/material IDs
- better check for OGS5 success on Windows

Changes

- drop py2.7 support

4.7 1.0.5 - 2019-11-18

Bugfixes

- `MSH.set_material_id`: better handling of non-int single values: f34d2e5
- `MSH.show`: better handling of material IDs: 26b4610
- `GLI.add_polyline`: Adding polyline by point-names was not possible: 17dd199

Additions

- better integration of pygmsh: 570afd9
- new functions `specialrange` and `generate_time`: e5f3aba
- updated examples

4.8 1.0.4 - 2019-09-10

Bugfixes

- ogs5py was not usable offline: 0f98c32
- `add_exe` was not recognizing operation system: 89b07e5

Additions

- new sub-keywords for OUT (added to OGS5 in Aug 19) when using TECPLOT (TECPLOT_ELEMENT_OUTPUT_CELL_CENTERED, TECPLOT_ZONES_FOR_MG): ebcb22a

Changes

- RFR Class was refactored to allow multiple variables: 3c1c445

4.9 1.0.3 - 2019-08-23

Bugfixes

- `MSH.show` TempFile was not working on Windows: c0d0960

4.10 1.0.2 - 2019-08-22

Bugfixes

- Don't fix QT_API for MAYAVI and use vtk for export: 33398ad
- `PopenSpawn` has no close attribute on Windows: 12f05d6

4.11 1.0.1 - 2019-08-22

Bugfixes

- `download_ogs(version="latest" build="PETSC")` was not working: 552503b

4.12 1.0.0 - 2019-08-22

Bugfixes

- `GLI.add_polyline` now allows integer coordinates for points: bf5d684
- `MSH.centroids` are now calculated as center of mass instead of center of element nodes: b0708a6
- `MSH.show` was not working: 6a0489b
- `OGS.run_model` has now a better check for OGS success: 143d0ab
- `GMSH` interface was updated to new meshio-API: d3e0594
- `RFR` file was not written: 41e55f3
- `BC` new sub-key TIME_INTERVAL was missing: 94ec5c5

Additions

- `download_ogs` downloads a system dependent OGS5 executable: ede32e4
- `add_exe` add a self compiled OGS5 executable: ede32e4
- `MSH.import_mesh` now allows the import of material_id and element_id if given as cell_data in the external mesh: 00a77fa
- `MSH.export_mesh` now automatically exports material_id (already the case before) and element_id. Also you can now export additional `point_data` and `field_data`: 00a77fa
- New method `MSH.set_material_id` to set the material IDs for specific elements: 4b11c6a

- `MSH.show` now can show additional cell_data: ffd7604
- New routine `show_vtk` to show vtk output with mayavi: f640c19
- New method `OGS.output_files` to get a list of output files: 2f5f102
- New attribute `file_name` for files: 632c2e7
- BlockFile: new method `append_to_block`: efc9aac
- `OGS.gen_script` now allowes multiple subkeys: 2cd344b

Changes

- `MSH.export_mesh` argument `add_data_by_id` renamed to `cell_data_by_id`: 00a77fa
- `OGS.run_model` argument `ogs_root` renamed to `ogs_exe`: 6fcdb61
- Files that can occure multiple times (mpd, rfr, ...) are better handled now: 4a9c9d2
- `ogs5py` is now licensed under the MIT license: ae96c0e
- Extra named files now get their name by keyword `name`: 632c2e7

4.13 0.6.5 - 2019-07-05

Bugfixes

- gli.add_polyline: Adding polyline by given point IDs was not possible: 3ec23af

Additions

- New swap_axis routine in msh and gli: You can now easily swap axis of a mesh. If you have generated a 2D mesh in x-y you can get a x-z cross-section by swapping the y and z axis: 3ec23af

4.14 0.6.4 - 2019-05-16

Bugfixes

- generator bugfix: more decimals to combine meshes: 51211a6
- Adopt new meshio container style: b08260b 167fb2c
- Better checking for the ogs executable: bffa41a 6c3895f
- Suppressing VTK Errors: cfa4671
- DOC updates: 9131e0c 7a27d8f fc4314c

Additions

- New routine to zip data by IDs: 9accf6e
- reading routines in the OGS Class: 592bc50
- New `del_block` routine in Blockfiles: 8d15a90

4.15 0.6.3 - 2019-03-21

Bugfixes

- The used method os.makedirs has no keyword argument 'exist_ok' in python 2.7, so we prevent using it. See: 40fea36

4.16 0.6.2 - 2019-03-21

Bugfixes

- The vtk reading routine could not read multiple scalar cell data. See: 568e7be

4.17 0.6.1 - 2019-01-22

Bugfixes

- The BlockFile reading routine was cutting of the given keys. See: d82fd30

4.18 0.6.0 - 2019-01-22

First release of ogs5py.

PYTHON MODULE INDEX

O

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W