GEOS Python Packages

GEOS Contributors

May 07, 2024

CONTENTS

1 Python Tools Setup 1

2 Manual Installation 3

3 Development & Debugging 5

4 Packages 7

Python Module Index 39

Index 41

CHAPTER

ONE

PYTHON TOOLS SETUP

The preferred method to setup the GEOS python tools is to run the following command in the build directory:

```
make geosx_python_tools
```

The ats setup command also sets up the python tools:

```
make ats_environment
```

These will attempt to install the required packages into the python distribution indicated via the *Python3_EXECUTABLE* cmake variable (also used by pygeosx). If any package dependencies are missing, then the install script will attempt to fetch them from the internet using pip. After installation, these packages will be available for import within the associated python distribution, and a set of console scripts will be available within the GEOS build bin directory.

Note: To re-install or update an installed version of geosPythonTools, you can run the *make geosx_python_tools_clean* and *make geosx_python_tools* commands.

CHAPTER

TWO

MANUAL INSTALLATION

In some cases, you may need to manually install or update geosPythonPackages. To do this, you can clone a copy of the geosPythonPackages repository and install them using pip:

```
cd /path/to/store/python/tools
git clone https://github.com/GEOS-DEV/geosPythonPackages.git

# Install/upgrade geos_ats
cd geosPythonPackages/geos_ats_package
python -m pip install --upgrade .
```

Note: To upgrade an existing installation, the python executable in the above command should correspond to the version you indicated in your host config. If you have previously built the tools, this version will be linked in the build directory: *build_dir/bin/python*.

CHAPTER

THREE

DEVELOPMENT & DEBUGGING

Be default, the python environment setup commands target the "main" branch of geosPythonTools. To target another version of the tools, you can set the *GEOS_PYTHON_PACKAGES_BRANCH* cmake variable to the name of another valid branch (or git tag) in the host config file. In this case, the code will pull the most recent commit of the desired branch when building geosPythonTools.

Note: If you are working on significant updates to geosPythonTools, you should open a testing branch in the main GEOS repository that defines the *GEOS_PYTHON_PACKAGES_BRANCH* variable. This will ensure that your changes are tested as part of the GEOS CI.

If you need to debug one of the packages in geosPythonTools, we recommend using VSCode with the Python extension installed. Some of the packages contain specific entry point scripts that can be used to assist in this process.

CHAPTER

FOUR

PACKAGES

4.1 HDF5 Wrapper

The *hdf5_wrapper* python package adds a wrapper to *h5py* that greatly simplifies reading/writing to/from hdf5-format files.

4.1.1 Usage

Once loaded, the contents of a file can be navigated in the same way as a native python dictionary.

```
import hdf5_wrapper

data = hdf5_wrapper.hdf5_wrapper('data.hdf5')

test = data['test']
for k, v in data.items():
  print('key: %s, value: %s' % (k, str(v)))
```

If the user indicates that a file should be opened in write-mode (*w*) or read/write-mode (*a*), then the file can be created or modified. Note: for these changes to be written to the disk, the wrapper may need to be closed or deleted.

```
import hdf5_wrapper
import numpy as np

data = hdf5_wrapper.hdf5_wrapper('data.hdf5', mode='w')
data['string'] = 'string'
data['integer'] = 123
data['array'] = np.random.randn(3, 4, 5)
data['child'] = {'float': 1.234}
```

Existing dictionaries can be placed on the current level:

```
existing_dict = {'some': 'value'}
data.insert(existing_dict)
```

And external hdf5 format files can be linked together:

```
for k in ['child_a', 'child_b']:
  data.link(k, '%s.hdf5' % (k))
```

4.1.2 API

class hdf5_wrapper.wrapper.**hdf5_wrapper**(*fname: str = ''*, *target: h5py.File | None = None*, *mode: str = 'r'*)

A class for reading/writing hdf5 files, which behaves similar to a native dict

close() → None

Closes the database

copy() → Dict[str, Any]

Copy the entire database into memory

Returns
a dictionary holding the database contents

Return type
dict

get_copy() → Dict[str, Any]

Copy the entire database into memory

Returns
a dictionary holding the database contents

Return type
dict

insert(*x: Dict[str, Any] | hdf5_wrapper*) → None

Insert the contents of the target object to the current location

Parameters
x (*dict, hdf5_wrapper*) – the dictionary to insert

keys() → Iterable[str]

Get a list of groups and arrays located at the current level

Returns
a list of key names pointing to objects at the current level

Return type
list

link(*k: str*, *target: str*) → None

Link an external hdf5 file to this location in the database

Parameters

- **k** (*str*) – the name of the new link in the database
- **target** (*str*) – the path to the external database

values() → Iterable[*hdf5_wrapper* | Any]

Get a list of values located on the current level

4.2 GEOS ATS

The *geos_ats* python package includes tools for managing integrated tests for GEOS. It is built using the Automated Test System (ATS) package. The available console scripts for this package and its API are described below.

4.2.1 run_geos_ats

Primary entry point for running integrated tests.

Runs GEOS integrated tests

```
usage: run_geos_ats [-h] [-w WORKINGDIR] [-b BASELINEDIR] [-y YAML]
                    [--baselineArchiveName BASELINEARCHIVENAME]
                    [--baselineCacheDirectory BASELINECACHEDIRECTORY] [-d]
                    [-u] [-a ACTION] [-c CHECK] [-v VERBOSE]
                    [-r RESTARTCHECKOVERRIDES [RESTARTCHECKOVERRIDES ...]]
                    [--salloc SALLOC] [--sallocoptions SALLOCOPTIONS]
                    [--ats ATS [ATS ...]] [--machine MACHINE]
                    [--machine-dir MACHINE_DIR] [-l LOGS] [-f]
                    [--failIfTestsFail] [-n NUMNODES]
                    geos_bin_dir ats_target
```

Positional Arguments

geos_bin_dir GEOS binary directory.

ats_target ats file

Named Arguments

-w, --workingDir Root working directory

-b, --baselineDir Root baseline directory

-y, --yaml Path to YAML config file

Default: “”

--baselineArchiveName Baseline archive name

Default: “”

--baselineCacheDirectory Baseline cache directory

Default: “”

-d, --delete-old-baselines Automatically delete old baselines

Default: False

-u, --update-baselines Force baseline file update

Default: False

-a, --action Test actions options (run,rerun,continue,list,commands,reset,clean,veryclean,check,rebaseline,rebaselinefai

Default: “run”

-c, --check	Test check options (all,none,stopcheck,curvecheck,restartcheck) Default: "all"
-v, --verbose	Log verbosity options (debug,info,warnings,errors) Default: "info"
-r, --restartCheckOverrides	Restart check parameter override (name value) Default: []
--salloc	Used by the chaosM machine to first allocate nodes with salloc, before running the tests Default: True
--sallocoptions	Used to override all command-line options for salloc. No other options with be used or added. Default: ""
--ats	pass arguments to ats Default: []
--machine	name of the machine
--machine-dir	Search path for machine definitions
-l, --logs	
-f, --allow-failed-tests	Default: False
--failIfTestsFail	geos_ats normally exits with 0. This will cause it to exit with an error code if there was a failed test. Default: False
-n, -N, --numNodes	Default: 2

Note: Arguments can be passed to the underlying ATS system with the *–ats* argument.

Note: Other machine-specific options for ATS can be viewed by running *run_geos_ats –ats help*

4.2.2 Debugging

If for any reason you need to debug the geos_ats package, we recommend that you create a local copy of this entry point in the *build/integratedTests* directory: *geosPythonPackages/geos_ats_package/geos_ats/debug_geos_ats.py*. This script is designed as a debugger entry point, and will read the autogenerated run script that was built during setup. To use it, you must either have geos_ats installed in your target python environment, or a copy of geosPythonPackages in the expected location (*/usr/workspace/[username]/geosPythonPackages*).

We recommend that you use VSCode with the Python extension to debug geos_ats. To begin the debugging session, you simply need to load the entry script, set any initial breakpoints you desire, then select the Debug run option. Note that this approach can only be used to debug the python code associated with tests, and not the underlying GEOS tests.

4.2.3 API

Restart Check

class geos_ats.helpers.restart_check.**FileComparison**(*file_path*, *baseline_path*, *rtol*, *atol*, *regex_expressions*, *output*, *warnings_are_errors*, *skip_missing*, *diff_file=None*)

Class that compares two hdf5 files.

compareAttributes(*path*, *attrs*, *base_attrs*)

Compare two sets of attributes.

PATH [in]: The path at which the comparison takes place. ATTRS [in]: The hdf5 AttributeManager to compare. BASE_ATTRS [in]: The hdf5 AttributeManager to compare against.

compareData(*path*, *arr*, *base_arr*)

Compare the numerical portion of two datasets.

PATH [in]: The path at which the comparison takes place. ARR [in]: The hdf5 Dataset to compare. BASE_ARR [in]: The hdf5 Dataset to compare against.

compareDatasets(*dset*, *base_dset*)

Compare two datasets.

DSET [in]: The Dataset to compare. BASE_DSET [in]: The Dataset to compare against.

compareFloatArrays(*path*, *arr*, *base_arr*)

Compares two arrays ARR and BASEARR of floating point values. Entries x1 and x2 are considered equal iff:

abs(x1 - x2) <= ATOL * (1 + max(abs(x2))) or abs(x1 - x2) <= RTOL * abs(x2).

To measure the degree of difference a scaling factor q is introduced. The goal is now to minimize q such that:

abs(x1 - x2) <= ATOL * (1 + max(abs(x2))) * q or abs(x1 - x2) <= RTOL * abs(x2) * q.

If RTOL * abs(x2) > ATOL * (1 + max(abs(x2))) q = abs(x1 - x2) / (RTOL * abs(x2)) else q = abs(x1 - x2) / (ATOL * (1 + max(abs(x2)))).

If the maximum value of q over all the entries is greater than 1.0 then the arrays are considered different and an error message is produced.

PATH [in]: The path at which the comparison takes place. ARR [in]: The hdf5 Dataset to compare. BASE_ARR [in]: The hdf5 Dataset to compare against.

compareFloatScalars(*path*, *val*, *base_val*)

Compare floating point scalars.

PATH [in]: The path at which the comparison occurs. VAL [in]: The value to compare. BASE_VAL [in]: The baseline value to compare against.

compareGroups(*group*, *base_group*)

Compare hdf5 groups. GROUP [in]: The Group to compare. BASE_GROUP [in]: The Group to compare against.

compareIntArrays(*path*, *arr*, *base_arr*)

Compare two integer datasets. Exact equality is used as the acceptance criteria.

PATH [in]: The path at which the comparison takes place. ARR [in]: The hdf5 Dataset to compare. BASE_ARR [in]: The hdf5 Dataset to compare against.

compareIntScalars(*path*, *val*, *base_val*)

Compare integer scalars.

PATH [in]: The path at which the comparison occurs. VAL [in]: The value to compare. BASE_VAL [in]: The baseline value to compare against.

compareStringArrays(*path*, *arr*, *base_arr*)

Compare two string datasets. Exact equality is used as the acceptance criteria.

PATH [in]: The path at which the comparison takes place. ARR [in]: The hdf5 Dataset to compare. BASE_ARR [in]: The hdf5 Dataset to compare against.

compareStringScalars(*path*, *val*, *base_val*)

Compare string scalars.

PATH [in]: The path at which the comparison occurs. VAL [in]: The value to compare. BASE_VAL [in]: The baseline value to compare against.

errorMsg(*path*, *message*, *add_to_diff=False*)

Issue an error which occurred at PATH in the files with the contents of MESSAGE. Sets self.different to True and rites the error to both stdout and OUTPUT.

PATH [in]: The path in the files at which the error occurred. MESSAGE [in]: The error message.

isExcluded(*path*)

Return True iff path matches any of the regex expressions in self.regex_expressions.

PATH [in]: The path to match.

warningMsg(*path*, *message*)

Issue a warning which occurred at PATH in the files with the contents of MESSAGE. Writes the warning to both stdout and OUTPUT. If WARNINGS_ARE_ERRORS then this is a wrapper around errorMsg.

PATH [in]: The path in the files at which the warning occurred. MESSAGE [in]: The warning message.

geos_ats.helpers.restart_check.**findMaxMatchingFile**(*file_path*)

Given a path FILE_PATH where the base name of FILE_PATH is treated as a regular expression find and return the path of the greatest matching file/folder or None if no match is found.

FILE_PATH [in]: The pattern to match.

Examples: '.*' will return the file/folder with the greatest name in the current directory. 'test/plot_*.hdf5' will return the file with the greatest name in the ./test directory that begins with 'plot' and ends with '.hdf5'.

geos_ats.helpers.restart_check.**main**()

Parses the command line arguments and executes the proper comparison. Writes output to both stdout and a '%s.restartcheck' file where the first part is the path of the file to compare.

Example

The file to compare is ./a/b/c.hdf5 the output will be a ./a/b/c.restartcheck file.

geos_ats.helpers.restart_check.**write**(*output*, *msg*)

Write MSG to both stdout and OUTPUT. OUTPUT [in/out]: File stream to write to. MSG [in]: Message to write.

geos_ats.helpers.restart_check.**writeHeader**(*file_pattern*, *file_path*, *baseline_pattern*, *baseline_path*, *args*)

Write the header.

FILE_PATTERN [in]: The pattern used to find the file to compare. FILE_PATH [in]: The path to the file to compare. BASELINE_PATTERN [in]: The pattern used to find the file to compare against. BASELINE_PATH [in]: THE path to the file to compare against. ARGS [in]: A dictionary of arguments to FileComparison.

Curve Check

geos_ats.helpers.curve_check.**check_diff**(*parameter_name*, *set_name*, *target*, *baseline*, *tolerance*, *errors*, *modifier='baseline'*)

Compute the L2-norm of the diff and compare to the set tolerance

Parameters

- **parameter_name** (*str*) – Parameter name
- **set_name** (*str*) – Set name
- **target** (*np.ndarray*) – Target value array
- **baseline** (*np.ndarray*) – Baseline value array
- **tolerance** (*float*) – Required tolerance of diff
- **errors** (*list*) – List to add any errors

Returns
Interpolated value array

Return type
np.ndarray

geos_ats.helpers.curve_check.**compare_time_history_curves**(*fname*, *baseline*, *curve*, *tolerance*, *output*, *output_n_column*, *units_time*, *script_instructions*)

Compute time history curves

Parameters

- **fname** (*str*) – Target curve file name
- **baseline** (*str*) – Baseline curve file name
- **curve** (*list*) – list containing pairs of value and set names to test
- **tolerance** (*list*) – Tolerance for curve diffs
- **output** (*str*) – Path to place output figures
- **ncol** (*int*) – Number of columns to use in the figure
- **units_time** (*str*) – Time units for the figure
- **script_instructions** (*list*) – List of (script, function, parameter, setname) values

Returns
warnings, errors

Return type
tuple

geos_ats.helpers.curve_check.**curve_check_figure**(*parameter_name*, *location_str*, *set_name*, *data*, *data_sizes*, *output_root*, *ncol*, *units_time*)

Generate figures associated with the curve check

Parameters

- **parameter_name** (*str*) – Parameter name
- **set_name** (*str*) – Set name
- **data** (*dict*) – Dictionary of curve data
- **data_sizes** (*dict*) – Dictionary of curve data sizes
- **output_root** (*str*) – Path of the folder to place the figures
- **ncol** (*int*) – Number of columns to use in the figure
- **units_time** (*str*) – Time units for the figure

geos_ats.helpers.curve_check.**curve_check_parser**()

Build the curve check parser

Returns
The curve check parser

Return type
argparse.parser

geos_ats.helpers.curve_check.**evaluate_external_script**(*script*, *fn*, *data*)

Evaluate an external script to produce the curve

Parameters

- **script** (*str*) – Path to a python script
- **fn** (*str*) – Name of the function to call

Returns
Curve values

Return type
np.ndarray

geos_ats.helpers.curve_check.**interpolate_values_time**(*ta*, *xa*, *tb*)

Interpolate array values in time

Parameters

- **ta** (*np.ndarray*) – Target time array
- **xa** (*np.ndarray*) – Target value array
- **tb** (*np.ndarray*) – Baseline time array

Returns
Interpolated value array

Return type
np.ndarray

geos_ats.helpers.curve_check.**main**()

Entry point for the curve check script

4.3 GEOS Mesh Tools

The *geosx_mesh_tools* python package includes tools for converting meshes from common formats (abaqus, etc.) to those that can be read by GEOS (gmsh, vtk). See *Python Tools Setup* for details on setup instructions, and External Mesh Guidelines for a detailed description of how to use external meshes in GEOS. The available console scripts for this package and its API are described below.

4.3.1 convert_abaqus

Compile an xml file with advanced features into a single file that can be read by GEOS.

```
usage: convert_abaqus [-h] [-v] input output
```

Positional Arguments

input	Input abaqus mesh file name
output	Output gmsh/vtu mesh file name

Named Arguments

-v, --verbose	Increase verbosity level Default: False

Note: For vtk format meshes, the user also needs to determine the region ID numbers and names of nodesets to import into GEOS. The following shows how these could look in an input XML file for a mesh with three regions (*REGIONA*, *REGIONB*, and *REGIONC*) and six nodesets (*xneg*, *xpos*, *yneg*, *ypos*, *zneg*, and *zpos*):

```
<Problem>
  <Mesh>
    <VTKMesh
      name="external_mesh"
      file="mesh.vtu"
      regionAttribute="REGIONA-REGIONB-REGIONC"
      nodesetNames="{ xneg, xpos, yneg, ypos, zneg, zpos }"/>
  </Mesh>

  <ElementRegions>
    <CellElementRegion
      name="ALL"
      cellBlocks="{ 0_tetrahedra, 1_tetrahedra, 2_tetrahedra }"
      materialList="{ water, porousRock }"
      meshBody="external_mesh"/>
  </ElementRegions>
</Problem>
```

4.3.2 API

geosx_mesh_tools.abaqus_converter.**convert_abaqus_to_gmsh**(*input_mesh: str*, *output_mesh: str*, *logger: Logger | None = None*) → int

Convert an abaqus mesh to gmsh 2 format, preserving nodeset information.

If the code encounters any issues with region/element indices, the conversion will attempt to continue, with errors indicated by -1 values in the output file.

Parameters

- **input_mesh** (*str*) – path of the input abaqus file
- **output_mesh** (*str*) – path of the output gmsh file
- **logger** (*logging.Logger*) – an instance of logging.Logger

Returns
Number of potential warnings encountered during conversion

Return type
int

geosx_mesh_tools.abaqus_converter.**convert_abaqus_to_vtu**(*input_mesh: str*, *output_mesh: str*, *logger: Logger | None = None*) → int

Convert an abaqus mesh to vtu format, preserving nodeset information.

If the code encounters any issues with region/element indices, the conversion will attempt to continue, with errors indicated by -1 values in the output file.

Parameters

- **input_mesh** (*str*) – path of the input abaqus file
- **output_mesh** (*str*) – path of the output vtu file
- **logger** (*logging.Logger*) – a logger instance

Returns
Number of potential warnings encountered during conversion

Return type
int

4.4 GEOS XML Tools

The *geosx_xml_tools* python package adds a set of advanced features to the GEOS xml format: units, parameters, and symbolic expressions. See *Python Tools Setup* for details on setup instructions, and Advanced XML Features for a detailed description of the input format. The available console scripts for this package and its API are described below.

4.4.1 convert_abaqus

Convert an abaqus format mesh file to gmsh or vtk format.

```
usage: preprocess_xml [-h] [-i INPUT] [-c COMPILED_NAME] [-s SCHEMA]
                      [-v VERBOSE] [-p PARAMETERS [PARAMETERS ...]]
```

Named Arguments

-i, --input	Input file name (multiple allowed)
-c, --compiled-name	Compiled xml file name (otherwise, it is randomly genrated)
	Default: “”
-s, --schema	GEOSX schema to use for validation
	Default: “”
-v, --verbose	Verbosity of outputs
	Default: 0
-p, --parameters	Parameter overrides (name value, multiple allowed)
	Default: []

4.4.2 format_xml

Formats an xml file.

```
usage: format_xml [-h] [-i INDENT] [-s STYLE] [-d DEPTH] [-a ALPHEBITIZE]
                  [-c CLOSE] [-n NAMESPACE]
                  input
```

Positional Arguments

input	Input file name

Named Arguments

-i, --indent	Indent size
	Default: 2
-s, --style	Indent style
	Default: 0
-d, --depth	Block separation depth
	Default: 2
-a, --alphebitize	Alphebetize attributes
	Default: 0

-c, --close	Close tag style
	Default: 0
-n, --namespace	Include namespace
	Default: 0

4.4.3 check_xml_attribute_coverage

Checks xml attribute coverage for files in the GEOS repository.

```
usage: check_xml_attribute_coverage [-h] [-r ROOT] [-o OUTPUT]
```

Named Arguments

-r, --root	GEOSX root
	Default: ""
-o, --output	Output file name
	Default: "attribute_test.xml"

4.4.4 check_xml_redundancy

Checks for redundant attribute definitions in an xml file, such as those that duplicate the default value.

```
usage: check_xml_redundancy [-h] [-r ROOT]
```

Named Arguments

-r, --root	GEOSX root
	Default: ""

4.4.5 API

Command line tools for geosx_xml_tools

geosx_xml_tools.main.**check_mpi_rank**() → int

Check the MPI rank

Returns
MPI rank

Return type
int

geosx_xml_tools.main.**format_geosx_arguments**(*compiled_name: str*, *unknown_args: Iterable[str]*) → Iterable[str]

Format GEOSX arguments

Parameters

- **compiled_name** (*str*) – Name of the compiled xml file
- **unknown_args** (*list*) – List of unprocessed arguments

Returns
List of arguments to pass to GEOSX

Return type
list

geosx_xml_tools.main.**preprocess_parallel**() → Iterable[str]

MPI aware xml preprocesing

geosx_xml_tools.main.**preprocess_serial**() → None

Entry point for the geosx_xml_tools console script

geosx_xml_tools.main.**wait_for_file_write_rank_0**(*target_file_argument: int | str = 0*, *max_wait_time: float = 100*, *max_startup_delay: float = 1*) → Callable[[Callable[[...], Any]], Callable[[...], Any]]

Constructor for a function decorator that waits for a target file to be written on rank 0

Parameters

- **target_file_argument** (*int, str*) – Index or keyword of the filename argument in the decorated function
- **max_wait_time** (*float*) – Maximum amount of time to wait (seconds)
- **max_startup_delay** (*float*) – Maximum delay allowed for thread startup (seconds)

Returns
Wrapped function

Tools for processing xml files in GEOSX

geosx_xml_tools.xml_processor.**apply_regex_to_node**(*node: lxml.etree.Element*) → None

Apply regexes that handle parameters, units, and symbolic math to each xml attribute in the structure.

Parameters
node (*lxml.etree.Element*) – The target node in the xml structure.

geosx_xml_tools.xml_processor.**generate_random_name**(*prefix: str = ''*, *suffix: str = '.xml'*) → str

If the target name is not specified, generate a random name for the compiled xml

Parameters

- **prefix** (*str*) – The file prefix (default = ‘’).
- **suffix** (*str*) – The file suffix (default = ‘.xml’)

Returns
Random file name

Return type
str

geosx_xml_tools.xml_processor.**merge_included_xml_files**(*root: lxml.etree.Element*, *fname: str*, *includeCount: int*, *maxInclude: int = 100*) → None

Recursively merge included files into the current structure.

Parameters

- **root** (*lxml.etree.Element*) – The root node of the base xml structure.

- **fname** (*str*) – The name of the target xml file to merge.
- **includeCount** (*int*) – The current recursion depth.
- **maxInclude** (*int*) – The maximum number of xml files to include (default = 100)

geosx_xml_tools.xml_processor.**merge_xml_nodes**(*existingNode: lxml.etree.Element*, *targetNode: lxml.etree.Element*, *level: int*) → None

Merge nodes in an included file into the current structure level by level.

Parameters

- **existingNode** (*lxml.etree.Element*) – The current node in the base xml structure.
- **targetNode** (*lxml.etree.Element*) – The node to insert.
- **level** (*int*) – The xml file depth.

geosx_xml_tools.xml_processor.**process**(*inputFiles: Iterable[str]*, *outputFile: str = ''*, *schema: str = ''*, *verbose: int = 0*, *parameter_override: List[Tuple[str, str]] = []*, *keep_parameters: bool = True*, *keep_includes: bool = True*) → str

Process an xml file

Parameters

- **inputFiles** (*list*) – Input file names.
- **outputFile** (*str*) – Output file name (if not specified, then generate randomly).
- **schema** (*str*) – Schema file name to validate the final xml (if not specified, then do not validate).
- **verbose** (*int*) – Verbosity level.
- **parameter_override** (*list*) – Parameter value overrides
- **keep_parameters** (*bool*) – If True, then keep parameters in the compiled file (default = True)
- **keep_includes** (*bool*) – If True, then keep includes in the compiled file (default = True)

Returns
Output file name

Return type
str

geosx_xml_tools.xml_processor.**validate_xml**(*fname: str*, *schema: str*, *verbose: int*) → None

Validate an xml file, and parse the warnings.

Parameters

- **fname** (*str*) – Target xml file name.
- **schema** (*str*) – Schema file name.
- **verbose** (*int*) – Verbosity level.

geosx_xml_tools.xml_formatter.**format_attribute**(*attribute_indent: str*, *ka: str*, *attribute_value: str*) → str

Format xml attribute strings

Parameters

- **attribute_indent** (*str*) – Attribute indent string

- **ka** (*str*) – Attribute name
- **attribute_value** (*str*) – Attribute value

Returns
Formatted attribute value

Return type
str

geosx_xml_tools.xml_formatter.**format_file**(*input_fname: str*, *indent_size: int = 2*, *indent_style: bool = False*, *block_separation_max_depth: int = 2*, *alphebitize_attributes: bool = False*, *close_style: bool = False*, *namespace: bool = False*) → None

Script to format xml files

Parameters

- **input_fname** (*str*) – Input file name
- **indent_size** (*int*) – Indent size
- **indent_style** (*bool*) – Style of indentation (0=fixed, 1=hanging)
- **block_separation_max_depth** (*int*) – Max depth to separate xml blocks
- **alphebitize_attributes** (*bool*) – Alphebitize attributes
- **close_style** (*bool*) – Style of close tag (0=same line, 1=new line)
- **namespace** (*bool*) – Insert this namespace in the xml description

geosx_xml_tools.xml_formatter.**format_xml_level**(*output: TextIO*, *node: lxml.etree.Element*, *level: int*, *indent: str = ' '*, *block_separation_max_depth: int = 2*, *modify_attribute_indent: bool = False*, *sort_attributes: bool = False*, *close_tag_newline: bool = False*, *include_namespace: bool = False*) → None

Iteratively format the xml file

Parameters

- **output** (*file*) – the output text file handle
- **node** (*lxml.etree.Element*) – the current xml element
- **level** (*int*) – the xml depth
- **indent** (*str*) – the xml indent style
- **block_separation_max_depth** (*int*) – the maximum depth to separate adjacent elements
- **modify_attribute_indent** (*bool*) – option to have flexible attribute indentation
- **sort_attributes** (*bool*) – option to sort attributes alphabetically
- **close_tag_newline** (*bool*) – option to place close tag on a separate line
- **include_namespace** (*bool*) – option to include the xml namespace in the output

geosx_xml_tools.xml_formatter.**main**() → None

Script to format xml files

Parameters

- **input** (*str*) – Input file name
- **-i/--indent** (*int*) – Indent size

- **-s/--style** (*int*) – Indent style
- **-d/--depth** (*int*) – Block separation depth
- **-a/--alphebitize** (*int*) – Alphebitize attributes
- **-c/--close** (*int*) – Close tag style
- **-n/--namespace** (*int*) – Include namespace

Tools for managing units in GEOSX

class geosx_xml_tools.unit_manager.**UnitManager**

This class is used to manage unit definitions.

buildUnits() → None

Build the unit definitions.

regexHandler(*match: Match*) → str

Split the matched string into a scale and unit definition.

Parameters
match (*re.match*) – The matching string from the regex.

Returns
The string with evaluated unit definitions

Return type
str

Tools for managing regular expressions in geosx_xml_tools

class geosx_xml_tools.regex_tools.**DictRegexHandler**

This class is used to substitute matched values with those stored in a dict.

geosx_xml_tools.regex_tools.**SymbolicMathRegexHandler**(*match: Match*) → str

Evaluate symbolic expressions that are identified using the regex_tools.patterns[‘symbolic’].

Parameters
match (*re.match*) – A matching string identified by the regex.

geosx_xml_tools.xml_redundancy_check.**check_redundancy_level**(*local_schema: Dict[str, Any]*, *node: lxml.etree.Element*, *whitelist: Iterable[str] = ['component']*) → int

Check xml redundancy at the current level

Parameters

- **local_schema** (*dict*) – Schema definitions
- **node** (*lxml.etree.Element*) – current xml node
- **whitelist** (*list*) – always match nodes containing these attributes

Returns
Number of required attributes in the node and its children

Return type
int

geosx_xml_tools.xml_redundancy_check.**check_xml_redundancy**(*schema: Dict[str, Any]*, *fname: str*) → None

Check redundancy in an xml file

Parameters

- **schema** (*dict*) – Schema definitions
- **fname** (*str*) – Name of the target file

geosx_xml_tools.xml_redundancy_check.**main**() → None

Entry point for the xml attribute usage test script

Parameters

-r/--root (*str*) – GEOSX root directory

geosx_xml_tools.xml_redundancy_check.**process_xml_files**(*geosx_root: str*) → None

Test for xml redundancy

Parameters

geosx_root (*str*) – GEOSX root directory

geosx_xml_tools.attribute_coverage.**collect_xml_attributes**(*xml_types: Dict[str, Dict[str, Any]]*, *fname: str*, *folder: str*) → None

Collect xml attribute usage in a file

Parameters

- **xml_types** (*dict*) – dictionary containing attribute usage
- **fname** (*str*) – name of the target file
- **folder** (*str*) – the source folder for the current file

geosx_xml_tools.attribute_coverage.**collect_xml_attributes_level**(*local_types: Dict[str, Dict[str, Any]]*, *node: lxml.etree.Element*, *folder: str*) → None

Collect xml attribute usage at the current level

Parameters

- **local_types** (*dict*) – dictionary containing attribute usage
- **node** (*lxml.etree.Element*) – current xml node
- **folder** (*str*) – the source folder for the current file

geosx_xml_tools.attribute_coverage.**main**() → None

Entry point for the xml attribute usage test script

Parameters

- **-r/--root** (*str*) – GEOSX root directory
- **-o/--output** (*str*) – output file name

geosx_xml_tools.attribute_coverage.**parse_schema**(*fname: str*) → Dict[str, Dict[str, Any]]

Parse the schema file into the xml attribute usage dict

Parameters

fname (*str*) – schema name

Returns

Dictionary of attributes and children for the entire schema

Return type

dict

geosx_xml_tools.attribute_coverage.**parse_schema_element**(*root: lxml.etree.Element*, *node: lxml.etree.Element*, *xsd: str = '{http://www.w3.org/2001/XMLSchema}'*, *recursive_types: Iterable[str] = ['PeriodicEvent', 'SoloEvent', 'HaltEvent']*, *folders: Iterable[str] = ['src', 'examples']*) → Dict[str, Dict[str, Any]]

Parse the xml schema at the current level

Parameters

- **root** (*lxml.etree.Element*) – the root schema node
- **node** (*lxml.etree.Element*) – current schema node
- **xsd** (*str*) – the file namespace
- **recursive_types** (*list*) – node tags that allow recursive nesting
- **folders** (*list*) – folders to sort xml attribute usage into

Returns
Dictionary of attributes and children for the current node

Return type
dict

geosx_xml_tools.attribute_coverage.**process_xml_files**(*geosx_root: str*, *output_name: str*) → None

Test for xml attribute usage

Parameters

- **geosx_root** (*str*) – GEOSX root directory
- **output_name** (*str*) – output file name

geosx_xml_tools.attribute_coverage.**write_attribute_usage_xml**(*xml_types: Dict[str, Dict[str, Any]]*, *fname: str*) → None

Write xml attribute usage file

Parameters

- **xml_types** (*dict*) – dictionary containing attribute usage by xml type
- **fname** (*str*) – output file name

geosx_xml_tools.attribute_coverage.**write_attribute_usage_xml_level**(*local_types: Dict[str, Dict[str, Any]]*, *node: lxml.etree.Element*, *folders: Iterable[str] = ['src', 'examples']*) → None

Write xml attribute usage file at a given level

Parameters

- **local_types** (*dict*) – dict containing attribute usage at the current level
- **node** (*lxml.etree.Element*) – current xml node

Tools for reading/writing GEOSX ascii tables

geosx_xml_tools.table_generator.**read_GEOS_table**(*axes_files: Iterable[str]*, *property_files: Iterable[str]*) → Tuple[Iterable[ndarray], Dict[str, ndarray]]

Read an GEOS-compatible ascii table.

Parameters

- **axes_files** (*list*) – List of the axes file names in order.
- **property_files** (*list*) – List of property file names

Returns
List of axis definitions, dict of property values

Return type
tuple

geosx_xml_tools.table_generator.**write_GEOS_table**(*axes_values: Iterable[ndarray]*, *properties: Dict[str, ndarray]*, *axes_names: Iterable[str] = ['x', 'y', 'z', 't']*, *string_format: str = '%1.5e'*) → None

Write an GEOS-compatible ascii table.

Parameters

- **axes_values** (*list*) – List of arrays containing the coordinates for each axis of the table.
- **properties** (*dict*) – Dict of arrays with dimensionality/size defined by the axes_values
- **axes_names** (*list*) – Names for each axis (default = [‘x’, ‘y’, ‘z’, ‘t’])
- **string_format** (*str*) – Format for output values (default = %1.5e)

geosx_xml_tools.table_generator.**write_read_GEOS_table_example**() → None

Table read / write example.

4.5 PyGEOSX Tools

The *pygeosx_tools* python package adds a variety of tools for working with pygeosx objects. These include common operations such as setting the value of geosx wrappers with python functions, parallel communication, and file IO. Examples using these tools can be found here: PYGEOSX Examples .

4.5.1 API

pygeosx_tools.wrapper.**allgather_wrapper**(*problem*, *key*, *ghost_key=''*)

Get a global copy of a wrapper as a numpy ndarray on all ranks

Parameters

- **problem** (*pygeosx.Group*) – GEOSX problem handle
- **target_key** (*str*) – Key for the target wrapper

Returns
The wrapper as a numpy ndarray

Return type
np.ndarray

pygeosx_tools.wrapper.**gather_wrapper**(*problem*, *key*, *ghost_key=''*)

Get a global copy of a wrapper as a numpy ndarray on rank 0

Parameters

- **problem** (*pygeosx.Group*) – GEOSX problem handle
- **target_key** (*str*) – Key for the target wrapper

Returns
The wrapper as a numpy ndarray

Return type
np.ndarray

pygeosx_tools.wrapper.**get_global_value_range**(*problem*, *key*)

Get the range of a target value across all processes

Parameters

- **problem** (*pygeosx.Group*) – GEOSX problem handle
- **target_key** (*str*) – Key for the target wrapper

Returns
The global min/max of the target

Return type
tuple

pygeosx_tools.wrapper.**get_matching_wrapper_path**(*problem*, *filters*)

Recursively search the group and its children for wrappers that match the filters A successful match is identified if the wrapper path contains all of the strings in the filter. For example, if filters=[‘a’, ‘b’, ‘c’], the following could match any of the following: ‘a/b/c’, ‘c/b/a’, ‘d/e/c/f/b/a/a’

Parameters

- **problem** (*pygeosx.Group*) – GEOSX problem handle
- **filters** (*list*) – a list of strings

Returns
Key of the matching wrapper

Return type
str

pygeosx_tools.wrapper.**get_wrapper**(*problem*, *target_key*, *write_flag=False*)

Get a local copy of a wrapper as a numpy ndarray

Parameters

- **filename** (*str*) – Catalog file name
- **problem** (*pygeosx.Group*) – GEOSX problem handle
- **target_key** (*str*) – Key for the target wrapper
- **write_flag** (*bool*) – Sets write mode (default=False)

Returns
The wrapper as a numpy ndarray

Return type
np.ndarray

pygeosx_tools.wrapper.**get_wrapper_par**(*problem*, *target_key*, *allgather=False*, *ghost_key=''*)

Get a global copy of a wrapper as a numpy ndarray. Note: if ghost_key is set, it will try to remove any ghost elements

Parameters

- **problem** (*pygeosx.Group*) – GEOSX problem handle
- **target_key** (*str*) – Key for the target wrapper
- **allgather** (*bool*) – Flag to trigger allgather across ranks (False)
- **ghost_key** (*str*) – Key for the corresponding ghost wrapper (default='')

Returns
The wrapper as a numpy ndarray

Return type
np.ndarray

pygeosx_tools.wrapper.**plot_history**(*records*, *output_root='.'*, *save_figures=True*, *show_figures=True*)

Plot the time-histories for the records structure. Note: If figures are shown, the GEOSX process will be blocked until they are closed

Parameters

- **records** (*dict*) – A dict of dicts containing the queries
- **output_root** (*str*) – Path to save figures (default = './')
- **save_figures** (*bool*) – Flag to indicate whether figures should be saved (default = True)
- **show_figures** (*bool*) – Flag to indicate whether figures should be drawn (default = False)

pygeosx_tools.wrapper.**print_global_value_range**(*problem*, *key*, *header*, *scale=1.0*, *precision='%1.4f'*)

Print the range of a target value across all processes

Parameters

- **problem** (*pygeosx.Group*) – GEOSX problem handle
- **target_key** (*str*) – Key for the target wrapper
- **header** (*str*) – Header to print with the range
- **scale** (*float*) – Multiply the range with this value before printing (default = 1.0)
- **precision** (*str*) – Format for printing the range (default = %1.4f)

Returns
The global min/max of the target

Return type
tuple

pygeosx_tools.wrapper.**run_queries**(*problem*, *records*)

Query the current GEOSX datastructure Note: The expected record request format is as follows. For now, the only supported query is to find the min/max values of the target record = {'path/of/wrapper': {'label': 'aperture (m)', # A label to include with plots 'scale': 1.0, # Value to scale results by 'history: [], # A list to store values over time 'fhandle': plt.figure() # A figure handle }}

Parameters

- **problem** (*pygeosx.Group*) – GEOSX problem handle
- **records** (*dict*) – A dict of dicts that specifies the queries to run

pygeosx_tools.wrapper.**search_datastructure_wrappers_recursive**(*group*, *filters*, *matching_paths*, *level=0*, *group_path=[]*)

Recursively search the group and its children for wrappers that match the filters

Parameters

- **problem** (*pygeosx.Group*) – GEOSX problem handle
- **filters** (*list*) – a list of strings
- **matching_paths** (*list*) – a list of matching values

pygeosx_tools.wrapper.**set_wrapper_to_value**(*problem*, *key*, *value*)

Set the value of a wrapper

Parameters

- **problem** (*pygeosx.Group*) – GEOSX problem handle
- **target_key** (*str*) – Key for the target wrapper
- **value** (*float*) – Value to set the wrapper

pygeosx_tools.wrapper.**set_wrapper_with_function**(*problem*, *target_key*, *input_keys*, *fn*, *target_index=-1*)

Set the value of a wrapper using a function

Parameters

- **problem** (*pygeosx.Group*) – GEOSX problem handle
- **target_key** (*str*) – Key for the target wrapper
- **input_keys** (*str, list*) – The input key(s)
- **fn** (*function*) – Vectorized function used to calculate target values
- **target_index** (*int*) – Target index to write the output (default = all)

pygeosx_tools.file_io.**load_tables**(*axes_names: Iterable[str]*, *property_names: Iterable[str]*, *table_root: str = './tables'*, *extension: str = 'csv'*) → Tuple[Iterable[ndarray], Dict[str, ndarray]]

Load a set of tables in GEOSX format

Parameters

- **axes_names** (*list*) – Axis file names in the target directory (with no extension)
- **property_names** (*list*) – Property file names in the target directory (with not extension)
- **table_root** (*str*) – Root path for the table directory
- **extension** (*str*) – Table file extension (default = 'csv')

Returns
List of axes values, and dictionary of table values

Return type
tuple

pygeosx_tools.file_io.**save_tables**(*axes: Iterable[ndarray]*, *properties: Dict[str, ndarray]*, *table_root: str = './tables'*, *axes_names: List[str] = []*) → None

Saves a set of tables in GEOSX format

The shape of these arrays should match the length of each axis in the specified order. The output directory will be created if it does not exist yet. If axes_names are not supplied, then they will be selected based on the dimensionality of the grid: 1D=[t]; 3D=[x, y, z]; 4D=[x, y, z, t].

Parameters

- **axes** (*list*) – A list of numpy ndarrays defining the table axes
- **properties** (*dict*) – A dict of numpy ndarrays defning the table values
- **table_root** (*str*) – The root path for the output directory
- **axes_names** (*list*) – A list of names for each potential axis (optional)

pygeosx_tools.mesh_interpolation.**apply_to_bins**(*fn: Callable[[float | ndarray], float]*, *position: ndarray*, *value: ndarray*, *bins: ndarray*, *collapse_edges: bool = True*)

Apply a function to values that are located within a series of bins Note: if a bin is empty, this function will fill a nan value

Parameters

- **fn** (*function*) – Function that takes a single scalar or array input
- **position** (*np.ndarray*) – A 1D list/array describing the location of each sample
- **value** (*np.ndarray*) – A 1D list/array of values at each location
- **bins** (*np.ndarray*) – The bin edges for the position data
- **collapse_edges** (*bool*) – Controls the behavior of edge-data (default=True)

Returns
an array of function results for each bin

Return type
np.ndarray

pygeosx_tools.mesh_interpolation.**extrapolate_nan_values**(*x*, *y*, *slope_scale=0.0*)

Fill in any nan values in two 1D arrays by extrapolating

Parameters

- **x** (*np.ndarray*) – 1D list/array of positions
- **y** (*np.ndarray*) – 1D list/array of values
- **slope_scale** (*float*) – value to scale the extrapolation slope (default=0.0)

Returns
The input array with nan values replaced by extrapolated data

Return type
np.ndarray

pygeosx_tools.mesh_interpolation.**get_random_realization**(*x*, *bins*, *value*, *rand_fill=0*, *rand_scale=0*, *slope_scale=0*)

Get a random realization for a noisy signal with a set of bins

Parameters

- **x** (*np.ndarray*) – 1D list/array of positions
- **bins** (*np.ndarray*) – 1D list/array of bin edges
- **value** (*np.ndarray*) – 1D list/array of values
- **rand_fill** (*float*) – The standard deviation to use where data is not defined (default=0)
- **rand_scale** (*float*) – Value to scale the standard deviation for the realization (default=0)

- **slope_scale** (*float*) – Value to scale the extrapolation slope (default=0.0)

Returns
An array containing the random realization

Return type
np.ndarray

pygeosx_tools.mesh_interpolation.**get_realizations**(*x*, *bins*, *targets*)

Get random realizations for noisy signals on target bins

Parameters

- **x** (*np.ndarray*) – 1D list/array of positions
- **bins** (*np.ndarray*) – 1D list/array of bin edges
- **targets** (*dict*) – Dict of geosx target keys, inputs to get_random_realization

Returns
Dictionary of random realizations

Return type
dict

pygeosx_tools.well_log.**convert_E_nu_to_K_G**(*E*, *nu*)

Convert young's modulus and poisson's ratio to bulk and shear modulus

Parameters

- **E** (*float, np.ndarray*) – Young's modulus
- **nu** (*float, np.ndarray*) – Poisson's ratio

Returns
bulk modulus, shear modulus with same size as inputs

Return type
tuple

pygeosx_tools.well_log.**estimate_shmin**(*z*, *rho*, *nu*)

Estimate the minimum horizontal stress using the poisson's ratio

Parameters

- **z** (*float, np.ndarray*) – Depth
- **rho** (*float, np.ndarray*) – Density
- **nu** (*float, np.ndarray*) – Poisson's ratio

Returns
minimum horizontal stress

Return type
float

pygeosx_tools.well_log.**parse_las**(*fname*, *variable_start='~C'*, *body_start='~A'*)

Parse an las format log file

Parameters

- **fname** (*str*) – Path to the log file
- **variable_start** (*str*) – A string that indicates the start of variable header information (default = '~CURVE INFORMATION')

- **body_start** (*str*) – a string that indicates the start of the log body (default = '~A')

Returns
a dict containing the values and unit definitions for each variable in the log

Return type
np.ndarray

4.6 Time History Tools

timehistory.plot_time_history.**getHistorySeries**(*database*, *variable*, *setname*, *indices=None*, *components=None*)

Retrieve a series of time history structures suitable for plotting in addition to the specific set index and component for the time series

Parameters

- **database** (*hdf5_wrapper.hdf5_wrapper*) – database to retrieve time history data from
- **variable** (*str*) – the name of the time history variable for which to retrieve time-series data
- **setname** (*str*) – the name of the index set as specified in the geosx input xml for which to query time-series data
- **indices** (*int, list*) – the indices in the named set to query for, if None, defaults to all
- **components** (*int, list*) – the components in the flattened data types to retrieve, defaults to all

Returns
list of (time, data, idx, comp) timeseries tuples for each time history data component

Return type
list

4.7 Mesh Doctor

`mesh_doctor` is a `python` executable that can be used through the command line to perform various checks, validations, and tiny fixes to the `vtk` mesh that are meant to be used in `geos`. `mesh_doctor` is organized as a collection of modules with their dedicated sets of options. The current page will introduce those modules, but the details and all the arguments can be retrieved by using the `--help` option for each module.

4.7.1 Modules

To list all the modules available through `mesh_doctor`, you can simply use the `--help` option, which will list all available modules as well as a quick summary.

```
$ python mesh_doctor.py --help
usage: mesh_doctor.py [-h] [-v] [-q] -i VTK_MESH_FILE
                      {collocated_nodes,element_volumes,fix_elements_orderings,generate_
↪cube,generate_fractures,generate_global_ids,non_conformal,self_intersecting_elements,
↪supported_elements}
                      ...
```

(continues on next page)

(continued from previous page)

```
Inspects meshes for GEOSX.

positional arguments:
  {collocated_nodes,element_volumes,fix_elements_orderings,generate_cube,generate_
↪fractures,generate_global_ids,non_conformal,self_intersecting_elements,supported_
↪elements}
        Modules
    collocated_nodes
        Checks if nodes are collocated.
    element_volumes
        Checks if the volumes of the elements are greater than "min".
    fix_elements_orderings
        Reorders the support nodes for the given cell types.
    generate_cube
        Generate a cube and its fields.
    generate_fractures
        Splits the mesh to generate the faults and fractures. [EXPERIMENTAL]
    generate_global_ids
        Adds globals ids for points and cells.
    non_conformal
        Detects non conformal elements. [EXPERIMENTAL]
    self_intersecting_elements
        Checks if the faces of the elements are self intersecting.
    supported_elements
        Check that all the elements of the mesh are supported by GEOSX.

options:
  -h, --help
        show this help message and exit
  -v    Use -v 'INFO', -vv for 'DEBUG'. Defaults to 'WARNING'.
  -q    Use -q to reduce the verbosity of the output.
  -i VTK_MESH_FILE, --vtk-input-file VTK_MESH_FILE

Note that checks are dynamically loaded.
An option may be missing because of an unloaded module.
Increase verbosity (-v, -vv) to get full information.
```

Then, if you are interested in a specific module, you can ask for its documentation using the `mesh_doctor module_name --help` pattern. For example

```
$ python mesh_doctor.py collocated_nodes --help
usage: mesh_doctor.py collocated_nodes [-h] --tolerance TOLERANCE

options:
  -h, --help            show this help message and exit
  --tolerance TOLERANCE
                        [float]: The absolute distance between two nodes for
                        them to be considered collocated.
```

`mesh_doctor` loads its module dynamically. If a module can't be loaded, `mesh_doctor` will proceed and try to load other modules. If you see a message like

```
[1970-04-14 03:07:15,625][WARNING] Could not load module "collocated_nodes": No module␣
↪named 'vtkmodules'
```

then most likely `mesh_doctor` could not load the `collocated_nodes` module, because the `vtk` python package was not found. Thereafter, the documentation for module `collocated_nodes` will not be displayed. You can solve this issue by installing the dependencies of `mesh_doctor` defined in its `requirements.txt` file (`python -m pip install -r requirements.txt`).

Here is a list and brief description of all the modules available.

collocated_nodes

Displays the neighboring nodes that are closer to each other than a prescribed threshold. It is not uncommon to define multiple nodes for the exact same position, which will typically be an issue for `geos` and should be fixed.

```
$ python mesh_doctor.py collocated_nodes --help
usage: mesh_doctor.py collocated_nodes [-h] --tolerance TOLERANCE

options:
  -h, --help            show this help message and exit
  --tolerance TOLERANCE
                        [float]: The absolute distance between two nodes for
                        them to be considered collocated.
```

element_volumes

Computes the volumes of all the cells and displays the ones that are below a prescribed threshold. Cells with negative volumes will typically be an issue for `geos` and should be fixed.

```
$ python mesh_doctor.py element_volumes --help
usage: mesh_doctor.py element_volumes [-h] --min 0.0

options:
  -h, --help  show this help message and exit
  --min 0.0   [float]: The minimum acceptable volume. Defaults to 0.0.
```

fix_elements_orderings

It sometimes happens that an exported mesh does not abide by the `vtk` orderings. The `fix_elements_orderings` module can rearrange the nodes of given types of elements. This can be convenient if you cannot regenerate the mesh.

```
$ python mesh_doctor.py fix_elements_orderings --help
usage: mesh_doctor.py fix_elements_orderings [-h]
                                             [--Hexahedron 1,6,5,4,7,0,2,3]
                                             [--Prism5 8,2,0,7,6,9,5,1,4,3]
                                             [--Prism6 11,2,8,10,5,0,9,7,6,1,4,3]
                                             [--Pyramid 3,4,0,2,1]
                                             [--Tetrahedron 2,0,3,1]
                                             [--Voxel 1,6,5,4,7,0,2,3]
                                             [--Wedge 3,5,4,0,2,1] --output
                                             OUTPUT
```

(continues on next page)

(continued from previous page)

```
                                          [--data-mode binary, ascii]

options:
  -h, --help            show this help message and exit
  --Hexahedron 1,6,5,4,7,0,2,3
                        [list of integers]: node permutation for "Hexahedron".
  --Prism5 8,2,0,7,6,9,5,1,4,3
                        [list of integers]: node permutation for "Prism5".
  --Prism6 11,2,8,10,5,0,9,7,6,1,4,3
                        [list of integers]: node permutation for "Prism6".
  --Pyramid 3,4,0,2,1   [list of integers]: node permutation for "Pyramid".
  --Tetrahedron 2,0,3,1
                        [list of integers]: node permutation for
                        "Tetrahedron".
  --Voxel 1,6,5,4,7,0,2,3
                        [list of integers]: node permutation for "Voxel".
  --Wedge 3,5,4,0,2,1   [list of integers]: node permutation for "Wedge".
  --output OUTPUT       [string]: The vtk output file destination.
  --data-mode binary, ascii
                        [string]: For ".vtu" output format, the data mode can
                        be binary or ascii. Defaults to binary.
```

generate_cube

This module conveniently generates cubic meshes in `vtk`. It can also generate fields with simple values. This tool can also be useful to generate a trial mesh that will later be refined or customized.

```
$ python mesh_doctor.py generate_cube --help
usage: mesh_doctor.py generate_cube [-h] [--x 0:1.5:3] [--y 0:5:10] [--z 0:1]
                                    [--nx 2:2] [--ny 1:1] [--nz 4]
                                    [--fields name:support:dim [name:support:dim ...]]
                                    [--cells] [--no-cells] [--points]
                                    [--no-points] --output OUTPUT
                                    [--data-mode binary, ascii]

options:
  -h, --help            show this help message and exit
  --x 0:1.5:3           [list of floats]: X coordinates of the points.
  --y 0:5:10            [list of floats]: Y coordinates of the points.
  --z 0:1               [list of floats]: Z coordinates of the points.
  --nx 2:2              [list of integers]: Number of elements in the X
                        direction.
  --ny 1:1              [list of integers]: Number of elements in the Y
                        direction.
  --nz 4                [list of integers]: Number of elements in the Z
                        direction.
  --fields name:support:dim [name:support:dim ...]
                        Create fields on CELLS or POINTS, with given dimension
                        (typically 1 or 3).
  --cells               [bool]: Generate global ids for cells. Defaults to
                        true.
```

(continues on next page)

(continued from previous page)

```
  --no-cells            [bool]: Don't generate global ids for cells.
  --points              [bool]: Generate global ids for points. Defaults to
                        true.
  --no-points           [bool]: Don't generate global ids for points.
  --output OUTPUT       [string]: The vtk output file destination.
  --data-mode binary, ascii
                        [string]: For ".vtu" output format, the data mode can
                        be binary or ascii. Defaults to binary.
```

generate_fractures

For a conformal fracture to be defined in a mesh, `geos` requires the mesh to be split at the faces where the fracture gets across the mesh. The `generate_fractures` module will split the mesh and generate the multi-block `vtk` files.

```
$ python mesh_doctor.py generate_fractures --help
usage: mesh_doctor.py generate_fractures [-h] --policy field,
                                         internal_surfaces [--name NAME]
                                         [--values VALUES] --output OUTPUT
                                         [--data-mode binary, ascii]
                                         --fracture-output FRACTURE_OUTPUT
                                         [--fracture-data-mode binary, ascii]

options:
  -h, --help            show this help message and exit
  --policy field, internal_surfaces
                        [string]: The criterion to define the surfaces that
                        will be changed into fracture zones. Possible values
                        are "field, internal_surfaces"
  --name NAME           [string]: If the "field" policy is selected, defines
                        which field will be considered to define the
                        fractures. If the "internal_surfaces" policy is
                        selected, defines the name of the attribute will be
                        considered to identify the fractures.
  --values VALUES       [list of comma separated integers]: If the "field"
                        policy is selected, which changes of the field will be
                        considered as a fracture. If the "internal_surfaces"
                        policy is selected, list of the fracture attributes.
  --output OUTPUT       [string]: The vtk output file destination.
  --data-mode binary, ascii
                        [string]: For ".vtu" output format, the data mode can
                        be binary or ascii. Defaults to binary.
  --fracture-output FRACTURE_OUTPUT
                        [string]: The vtk output file destination.
  --fracture-data-mode binary, ascii
                        [string]: For ".vtu" output format, the data mode can
                        be binary or ascii. Defaults to binary.
```

generate_global_ids

When running `geos` in parallel, *global ids* can be used to refer to data across multiple ranks. The `generate_global_ids` can generate *global ids* for the imported `vtk` mesh.

```
$ python mesh_doctor.py generate_global_ids --help
usage: mesh_doctor.py generate_global_ids [-h] [--cells] [--no-cells]
                                          [--points] [--no-points] --output
                                          OUTPUT [--data-mode binary, ascii]

options:
  -h, --help            show this help message and exit
  --cells               [bool]: Generate global ids for cells. Defaults to
                        true.
  --no-cells            [bool]: Don't generate global ids for cells.
  --points              [bool]: Generate global ids for points. Defaults to
                        true.
  --no-points           [bool]: Don't generate global ids for points.
  --output OUTPUT       [string]: The vtk output file destination.
  --data-mode binary, ascii
                        [string]: For ".vtu" output format, the data mode can
                        be binary or ascii. Defaults to binary.
```

non_conformal

This module will detect elements which are close enough (there's a user defined threshold) but which are not in front of each other (another threshold can be defined). *Close enough* can be defined in terms or proximity of the nodes and faces of the elements. The angle between two faces can also be precribed. This module can be a bit time consuming.

```
$ python mesh_doctor.py non_conformal --help
usage: mesh_doctor.py non_conformal [-h] [--angle_tolerance 10.0]
                                    [--point_tolerance POINT_TOLERANCE]
                                    [--face_tolerance FACE_TOLERANCE]

options:
  -h, --help            show this help message and exit
  --angle_tolerance 10.0
                        [float]: angle tolerance in degrees. Defaults to 10.0
  --point_tolerance POINT_TOLERANCE
                        [float]: tolerance for two points to be considered
                        collocated.
  --face_tolerance FACE_TOLERANCE
                        [float]: tolerance for two faces to be considered
                        "touching".
```

self_intersecting_elements

Some meshes can have cells that auto-intersect. This module will display the elements that have faces intersecting.

```
$ python mesh_doctor.py self_intersecting_elements --help
usage: mesh_doctor.py self_intersecting_elements [-h]
                                                 [--min 2.220446049250313e-16]

options:
  -h, --help            show this help message and exit
  --min 2.220446049250313e-16
                        [float]: The tolerance in the computation. Defaults to
                        your machine precision 2.220446049250313e-16.
```

supported_elements

geos supports a specific set of elements. Let's cite the standard elements like *tetrahedra*, *wedges*, *pyramids* or *hexahedra*. But also prismes up to 11 faces. geos also supports the generic VTK_POLYHEDRON/42 elements, which are converted on the fly into one of the elements just described.

The supported_elements check will validate that no unsupported element is included in the input mesh. It will also verify that the VTK_POLYHEDRON cells can effectively get converted into a supported type of element.

```
$ python mesh_doctor.py supported_elements --help
usage: mesh_doctor.py supported_elements [-h] [--chunck_size 1] [--nproc 2]

options:
  -h, --help       show this help message and exit
  --chunck_size 1  [int]: Defaults chunk size for parallel processing to 1
  --nproc 2        [int]: Number of threads used for parallel processing.
                   Defaults to your CPU count 2.
```

PYTHON MODULE INDEX

g

geos_ats.helpers.curve_check, 13
geos_ats.helpers.restart_check, 11
geosx_mesh_tools.abaqus_converter, 16
geosx_xml_tools.attribute_coverage, 23
geosx_xml_tools.main, 18
geosx_xml_tools.regex_tools, 22
geosx_xml_tools.table_generator, 24
geosx_xml_tools.unit_manager, 22
geosx_xml_tools.xml_formatter, 20
geosx_xml_tools.xml_processor, 19
geosx_xml_tools.xml_redundancy_check, 22

h

hdf5_wrapper.wrapper, 8

p

pygeosx_tools.file_io, 28
pygeosx_tools.mesh_interpolation, 29
pygeosx_tools.well_log, 30
pygeosx_tools.wrapper, 25

t

timehistory.plot_time_history, 31

INDEX

A

allgather_wrapper() (*in module pygeosx_tools.wrapper*), 25
apply_regex_to_node() (*in module geosx_xml_tools.xml_processor*), 19
apply_to_bins() (*in module pygeosx_tools.mesh_interpolation*), 29

B

buildUnits() (*geosx_xml_tools.unit_manager.UnitManager method*), 22

C

check_diff() (*in module geos_ats.helpers.curve_check*), 13
check_mpi_rank() (*in module geosx_xml_tools.main*), 18
check_redundancy_level() (*in module geosx_xml_tools.xml_redundancy_check*), 22
check_xml_redundancy() (*in module geosx_xml_tools.xml_redundancy_check*), 22
close() (*hdf5_wrapper.wrapper.hdf5_wrapper method*), 8
collect_xml_attributes() (*in module geosx_xml_tools.attribute_coverage*), 23
collect_xml_attributes_level() (*in module geosx_xml_tools.attribute_coverage*), 23
compare_time_history_curves() (*in module geos_ats.helpers.curve_check*), 13
compareAttributes() (*geos_ats.helpers.restart_check.FileComparison method*), 11
compareData() (*geos_ats.helpers.restart_check.FileComparison method*), 11
compareDatasets() (*geos_ats.helpers.restart_check.FileComparison method*), 11
compareFloatArrays() (*geos_ats.helpers.restart_check.FileComparison method*), 11
compareFloatScalars() (*geos_ats.helpers.restart_check.FileComparison method*), 11
compareGroups() (*geos_ats.helpers.restart_check.FileComparison method*), 11
compareIntArrays() (*geos_ats.helpers.restart_check.FileComparison method*), 11
compareIntScalars() (*geos_ats.helpers.restart_check.FileComparison method*), 11
compareStringArrays() (*geos_ats.helpers.restart_check.FileComparison method*), 12
compareStringScalars() (*geos_ats.helpers.restart_check.FileComparison method*), 12
convert_abaqus_to_gmsh() (*in module geosx_mesh_tools.abaqus_converter*), 16
convert_abaqus_to_vtu() (*in module geosx_mesh_tools.abaqus_converter*), 16
convert_E_nu_to_K_G() (*in module pygeosx_tools.well_log*), 30
copy() (*hdf5_wrapper.wrapper.hdf5_wrapper method*), 8
curve_check_figure() (*in module geos_ats.helpers.curve_check*), 13
curve_check_parser() (*in module geos_ats.helpers.curve_check*), 14

D

DictRegexHandler (*class in geosx_xml_tools.regex_tools*), 22

E

errorMsg() (*geos_ats.helpers.restart_check.FileComparison method*), 12
estimate_shmin() (*in module pygeosx_tools.well_log*), 30
evaluate_external_script() (*in module geos_ats.helpers.curve_check*), 14
extrapolate_nan_values() (*in module pygeosx_tools.mesh_interpolation*), 29

F

FileComparison (*class in geos_ats.helpers.restart_check*), 11
findMaxMatchingFile() (*in module geos_ats.helpers.restart_check*), 12
format_attribute() (*in module geosx_xml_tools.xml_formatter*), 20
format_file() (*in module geosx_xml_tools.xml_formatter*), 21
format_geosx_arguments() (*in module geosx_xml_tools.main*), 18
format_xml_level() (*in module geosx_xml_tools.xml_formatter*), 21

G

gather_wrapper() (*in module pygeosx_tools.wrapper*), 25
generate_random_name() (*in module geosx_xml_tools.xml_processor*), 19
geos_ats.helpers.curve_check
    module, 13
geos_ats.helpers.restart_check
    module, 11
geosx_mesh_tools.abaqus_converter
    module, 16
geosx_xml_tools.attribute_coverage
    module, 23
geosx_xml_tools.main
    module, 18
geosx_xml_tools.regex_tools
    module, 22
geosx_xml_tools.table_generator
    module, 24
geosx_xml_tools.unit_manager
    module, 22
geosx_xml_tools.xml_formatter
    module, 20
geosx_xml_tools.xml_processor
    module, 19
geosx_xml_tools.xml_redundancy_check
    module, 22
get_copy() (*hdf5_wrapper.wrapper.hdf5_wrapper method*), 8
get_global_value_range() (*in module pygeosx_tools.wrapper*), 26
get_matching_wrapper_path() (*in module pygeosx_tools.wrapper*), 26
get_random_realization() (*in module pygeosx_tools.mesh_interpolation*), 29
get_realizations() (*in module pygeosx_tools.mesh_interpolation*), 30
get_wrapper() (*in module pygeosx_tools.wrapper*), 26
get_wrapper_par() (*in module pygeosx_tools.wrapper*), 26
getHistorySeries() (*in module timehistory.plot_time_history*), 31

H

hdf5_wrapper (*class in hdf5_wrapper.wrapper*), 8
hdf5_wrapper.wrapper
    module, 8

I

insert() (*hdf5_wrapper.wrapper.hdf5_wrapper method*), 8
interpolate_values_time() (*in module geos_ats.helpers.curve_check*), 14
isExcluded() (*geos_ats.helpers.restart_check.FileComparison method*), 12

K

keys() (*hdf5_wrapper.wrapper.hdf5_wrapper method*), 8

L

link() (*hdf5_wrapper.wrapper.hdf5_wrapper method*), 8
load_tables() (*in module pygeosx_tools.file_io*), 28

M

main() (*in module geos_ats.helpers.curve_check*), 14
main() (*in module geos_ats.helpers.restart_check*), 12
main() (*in module geosx_xml_tools.attribute_coverage*), 23
main() (*in module geosx_xml_tools.xml_formatter*), 21
main() (*in module geosx_xml_tools.xml_redundancy_check*), 23
merge_included_xml_files() (*in module geosx_xml_tools.xml_processor*), 19
merge_xml_nodes() (*in module geosx_xml_tools.xml_processor*), 20
module
    geos_ats.helpers.curve_check, 13
    geos_ats.helpers.restart_check, 11
    geosx_mesh_tools.abaqus_converter, 16
    geosx_xml_tools.attribute_coverage, 23
    geosx_xml_tools.main, 18
    geosx_xml_tools.regex_tools, 22
    geosx_xml_tools.table_generator, 24
    geosx_xml_tools.unit_manager, 22
    geosx_xml_tools.xml_formatter, 20
    geosx_xml_tools.xml_processor, 19
    geosx_xml_tools.xml_redundancy_check, 22
    hdf5_wrapper.wrapper, 8
    pygeosx_tools.file_io, 28
    pygeosx_tools.mesh_interpolation, 29
    pygeosx_tools.well_log, 30

pygeosx_tools.wrapper, 25
timehistory.plot_time_history, 31

P

parse_las() (*in module pygeosx_tools.well_log*), 30
parse_schema() (*in module geosx_xml_tools.attribute_coverage*), 23
parse_schema_element() (*in module geosx_xml_tools.attribute_coverage*), 23
plot_history() (*in module pygeosx_tools.wrapper*), 27
preprocess_parallel() (*in module geosx_xml_tools.main*), 19
preprocess_serial() (*in module geosx_xml_tools.main*), 19
print_global_value_range() (*in module pygeosx_tools.wrapper*), 27
process() (*in module geosx_xml_tools.xml_processor*), 20
process_xml_files() (*in module geosx_xml_tools.attribute_coverage*), 24
process_xml_files() (*in module geosx_xml_tools.xml_redundancy_check*), 23
pygeosx_tools.file_io
module, 28
pygeosx_tools.mesh_interpolation
module, 29
pygeosx_tools.well_log
module, 30
pygeosx_tools.wrapper
module, 25

R

read_GEOS_table() (*in module geosx_xml_tools.table_generator*), 24
regexHandler() (*geosx_xml_tools.unit_manager.UnitManager method*), 22
run_queries() (*in module pygeosx_tools.wrapper*), 27

S

save_tables() (*in module pygeosx_tools.file_io*), 28
search_datastructure_wrappers_recursive() (*in module pygeosx_tools.wrapper*), 27
set_wrapper_to_value() (*in module pygeosx_tools.wrapper*), 28
set_wrapper_with_function() (*in module pygeosx_tools.wrapper*), 28
SymbolicMathRegexHandler() (*in module geosx_xml_tools.regex_tools*), 22

T

timehistory.plot_time_history
module, 31

U

UnitManager (*class in geosx_xml_tools.unit_manager*), 22

V

validate_xml() (*in module geosx_xml_tools.xml_processor*), 20
values() (*hdf5_wrapper.wrapper.hdf5_wrapper method*), 8

W

wait_for_file_write_rank_0() (*in module geosx_xml_tools.main*), 19
warningMsg() (*geos_ats.helpers.restart_check.FileComparison method*), 12
write() (*in module geos_ats.helpers.restart_check*), 12
write_attribute_usage_xml() (*in module geosx_xml_tools.attribute_coverage*), 24
write_attribute_usage_xml_level() (*in module geosx_xml_tools.attribute_coverage*), 24
write_GEOS_table() (*in module geosx_xml_tools.table_generator*), 25
write_read_GEOS_table_example() (*in module geosx_xml_tools.table_generator*), 25
writeHeader() (*in module geos_ats.helpers.restart_check*), 12