gmshparser

Release 0.1.0

Jun 04, 2020

Contents

Package author: Jukka Aho (@ahojukka5)

Gmshparser is a small Python package which aims to do only one thing: parse Gmsh mesh file format. Package does not have any external dependencies to other packages and it aims to be a simple stand-alone solution for a common problem: how to import mesh to your favourite research FEM code?

Project is hosted on GitHub: https://github.com/ahojukka5/gmshparser. Project is licensed under MIT license. Please see the *project license* for further details.

CHAPTER 1

Mesh formats

Mesh formats can be found from: https://gmsh.info/doc/texinfo/gmsh.html#MSH-file-format

Format 4.1:

```
$MeshFormat
4.1 0 8          MSH4.1, ASCII
$EndMeshFormat
$Nodes
1 6 1 6          1 entity bloc, 6 nodes total, min/max node tags: 1 and 6
2 1 0 6          2D entity (surface) 1, no parametric coordinates, 6 nodes
1                  node tag #1
2                  node tag #2
3                  etc.
4
5
6
0. 0. 0.           node #1 coordinates (0., 0., 0.)
1. 0. 0.           node #2 coordinates (1., 0., 0.)
1. 1. 0.           etc.
0. 1. 0.
2. 0. 0.
2. 1. 0.
$EndNodes
$Elements
1 2 1 2          1 entity bloc, 2 elements total, min/max element tags: 1 and 2
2 1 3 2          2D entity (surface) 1, element type 3 (4-node quad), 2 elements
1 1 2 3 4          quad tag #1, nodes 1 2 3 4
2 2 5 6 3          quad tag #2, nodes 2 5 6 3
$EndElements
$NodeData
1                1 string tag:
"A scalar view"    the name of the view ("A scalar view")
1                1 real tag:
0.0                the time value (0.0)
3                3 integer tags:
```

(continues on next page)

(continued from previous page)

```
0                  the time step (0; time steps always start at 0)
1                  1-component (scalar) field
6                  6 associated nodal values
1 0.0            value associated with node #1 (0.0)
2 0.1            value associated with node #2 (0.1)
3 0.2            etc.
4 0.0
5 0.2
6 0.4
$EndNodeData
```

CHAPTER 2

API Documentation

2.1 Externals

External classes and functions are the public API of the package.

The main command used to parse mesh is *gmshparser.parse*.

parse(*filename: str*) → gmshparser.mesh.Mesh
Parse Gmsh .msh file and return *Mesh* object.

Package contains data structures to describe nodes, node entities, elements and element entities.

class Node
Node.

get_coordinates() → Tuple[float, float, float]
Get the coordinates of the node.

get_tag() → int
Get node tag (node id).

set_coordinates(*coordinates: Tuple[float, float, float]*)
Set the coordinates of the node.

set_tag(*tag: int*)
Set node tag (node id).

class NodeEntity
NodeEntity class holds nodes for one block.

add_node(*node: gmshparser.node.Node*)
Add new node to entity.

get_dimension() → int
Get the dimension of the entity.

get_node(*tag: int*) → gmshparser.node.Node
Get node from entity by its tag.

get_nodes() → List[gmshparser.node.Node]
Get all nodes in this entity.

get_number_of_nodes() → int
Get the number of nodes of the entity.

get_number_of_parametric_coordinates() → int
Get the number of parametric coordinates of the entity.

get_tag() → int
Get the tag of the entity.

set_dimension(*dimension: int*)
Set the dimension of the entity to *dimension*.

set_number_of_nodes(*number_of_nodes: int*)
Set the number of nodes of the entity.

set_number_of_parametric_coordinates(*npar: int*)
Set the number of parametric coordinates of the entity.

set_tag(*tag: int*)
Set the tag of the entity.

class Element
Element.

get_connectivity() → List[int]
Get element connectivity.

get_tag()
Get element tag.

set_connectivity(*connectivity: List[int]*)
Set element connectivity.

set_tag(*tag: int*)
Set element tag.

class ElementEntity
ElementEntity class holds elements for one block.

add_element(*element: gmshparser.element.Element*)
Add a new element to the entity.

get_dimension() → int
Get the dimension of the element entity.

get_element(*tag: int*) → gmshparser.element.Element
Get an element from the entity.

get_element_type() → int
Get element type in element entity.

get_elements() → List[gmshparser.element.Element]
Return all the elements of this entity.

get_number_of_elements() → int
Get the number of elements in entity.

get_tag() → int
Get the tag of the element entity.

set_dimension(*dimension: int*)
Set the dimension of element entity.

set_element_type(*element_type: int*)
Set element type in element entity.

set_number_of_elements(*number_of_elements: int*)
Set the number of elements in entity.

set_tag(*tag: int*)
Set the tag of the element entity.

The main class is *Mesh*, which collects everything together.

class Mesh
Mesh is the main class of the package.

add_element_entity(*element_entity: gmshparser.element_entity.ElementEntity*)
Add element entity to mesh.

add_node_entity(*node_entity: gmshparser.node_entity.NodeEntity*)
Add node entity to mesh.

get_ascii() → bool
Get a boolean flag whether this mesh is ASCII of binary

get_element_entities() → List[gmshparser.element_entity.ElementEntity]
Get all element entities as dictionary.

get_element_entity(*dim: int*, *tag: int*) → gmshparser.element_entity.ElementEntity
Get element entity based on dimension *dim* and tag *tag*.

get_max_element_tag() → int
Get element maximum tag.

get_max_node_tag() → int
Get node maximum tag.

get_min_element_tag() → int
Get element minimum tag.

get_min_node_tag() → int
Get node minimum tag.

get_name() → str
Get the name of the mesh.

get_node_entities() → List[gmshparser.node_entity.NodeEntity]
Get all node entities of mesh.

get_node_entity(*dim: int*, *tag: int*)
Get node entity based on dimension and tag.

get_number_of_element_entities() → int
Get number of element entities.

get_number_of_elements() → int
Get number of elements.

get_number_of_node_entities() → int
Get number of node entities.

get_number_of_nodes() → int
Get number of nodes.

get_precision() → int
Get the precision of the mesh

get_version() → str
Get the version of the Mesh object

has_element_entity(*dim: int*, *tag: int*) → bool
Test does mesh have element entity with *(dim, tag)*.

has_node_entity(*dim: int*, *tag: int*) → bool
Test does mesh have node entity of dimension *dim* and tag *tag*.

set_ascii(*is_ascii: bool*)
Set a boolean flag whether this mesh is ASCII or binary

set_max_element_tag(*max_element_tag: int*)
Set element maximum tag.

set_max_node_tag(*max_node_tag: int*)
Set node maximum tag.

set_min_element_tag(*min_element_tag: int*)
Set element minimum tag.

set_min_node_tag(*min_node_tag: int*)
Set node minimum tag.

set_name(*name: str*)
Set the name of the mesh.

set_number_of_element_entities(*number_of_element_entities: int*)
Set number of element entities.

set_number_of_elements(*number_of_elements: int*)
Set number of elements.

set_number_of_node_entities(*number_of_node_entities: int*)
Set number of node entities.

set_number_of_nodes(*number_of_nodes: int*)
Set number of nodes.

set_precision(*precision: int*)
Set the precision of the mesh (8)

set_version(*version: str*)
Set the version of the Mesh object

2.2 Internals

Internal classes and functions are the private API of the package. They can change without any warning.

2.2.1 Functions

parse_ints(*io: TextIO*) → List[int]
Parse first line of io to list of integers.

:param io :: TextIO: Object supporting *readline()*

Returns A list of integers

Return type integers :: List[int]

Examples

```
>>> data = StringIO("1 2 3 4")
>>> parse_ints(data)
[1, 2, 3, 4]
```

parse_floats(*io: TextIO*) → List[float]
Parse first line of io to list of floats.

:param io :: TextIO: Object supporting *readline()*

Returns A list of floats

Return type floats :: List[float]

Examples

```
>>> data = StringIO("1.1 2.2 3.3 4.4")
>>> parse_floats(data)
[1.1, 2.2, 3.3, 4.4]
```

2.2.2 Classes

Parsers must be inherited from *AbstractParser* and they must implement function *parse*, which is responsible of parsing a section.

class AbstractParser
AbstractParser is a superclass of all other parsers.

All other parsers must inheric `AbstractParser` and implement their own static methods `parse` and `get_section_name`.

The first argument of the `parse` is a mutable `mesh` object, which parser modifies in-place. The second argument is `io`, where parser reads the text file line by line using *readline()*. Parser must stop reading the file to the section end mark, e.g. `$EndNodes` in the case of parser which is responsible to parse nodes, starting from a section start mark `$Nodes`.

Another must-to-implement static method is `get_section_name()`, which must return the name of the line where this parser should activate. For example, if the section name is `$Nodes`, then `get_section_name()` must return string `$Nodes`.

class MainParser(*parsers=[<class 'gmshparser.mesh_format_parser.MeshFormatParser'>, <class 'gmshparser.nodes_parser.NodesParser'>, <class 'gmshparser.elements_parser.ElementsParser'>]*)
The main parser class, using other parsers.

class MeshFormatParser

class NodesParser

class ElementsParser
ElementParser is responsible to parse data between tags $Elements and $EndElements.

CHAPTER 3

MIT License

CHAPTER 4

Installing package

Package can be installed using a standard package installing tool pip:

```
pip install gmshparser
```

Development version can be installed from GitHub repository, again, using pip:

```
pip install git+git://github.com/ahojukka5/gmshparser.git
```

CHAPTER 5

Usage

The usage of the package is quite straightforward. Import libary and read mesh using `gmshparser.parse`, given the filename of the mesh:

```
import gmshparser
mesh = gmshparser.parse("data/testmesh.msh")
print(mesh)
```

Output:

```
Mesh name: data/testmesh.msh
Mesh version: 4.1
Number of nodes: 6
Minimum node tag: 1
Maximum node tag: 6
Number of node entities: 1
Number of elements: 2
Minimum element tag: 1
Maximum element tag: 2
Number of element entities: 1
```

All nodes are stored in node entities and all elements are stored in element entities. To access nodes, one must first loop all node entities and after that all nodes in node entity:

```
for entity in mesh.get_node_entities():
    for node in entity.get_nodes():
        nid = node.get_tag()
        ncoords = node.get_coordinates()
        print("Node id = %s, node coordinates = %s" % (nid, ncoords))
```

Output:

```
Node id = 1, node coordinates = (0.0, 0.0, 0.0)
Node id = 2, node coordinates = (1.0, 0.0, 0.0)
Node id = 3, node coordinates = (1.0, 1.0, 0.0)
```

(continues on next page)

(continued from previous page)

```
Node id = 4, node coordinates = (0.0, 1.0, 0.0)
Node id = 5, node coordinates = (2.0, 0.0, 0.0)
Node id = 6, node coordinates = (2.0, 1.0, 0.0)
```

Accessing elements is done in a similar way, first entities and then elements. Element type is given in each entity. For example, here code 3 means linear quadrangle:

```
for entity in mesh.get_element_entities():
    eltype = entity.get_element_type()
    print("Element type: %s" % eltype)
    for element in entity.get_elements():
        elid = element.get_tag()
        elcon = element.get_connectivity()
        print("Element id = %s, connectivity = %s" % (elid, elcon))
```

Output:

```
Element type: 3
Element id = 1, connectivity = [1, 2, 3, 4]
Element id = 2, connectivity = [2, 5, 6, 3]
```

CHAPTER 6

Contributing to the project

Like in other open source projects, contributions are always welcome to this too! If you have some great ideas how to make this package better, feature requests etc., you can open an issue on gmshparser's issue tracker or contact me (ahojukka5 at gmail.com) directly.

Index