omf Documentation

Release 1.0.1

Global Mining Guidelines Group

Mar 22, 2019

Contents

Version: 1.0.1

API library for Open Mining Format, a new standard for mining data backed by the Global Mining Guidelines Group.

Warning: Pre-Release Notice

This is a Beta release of the Open Mining Format (OMF) and the associated Python API. The storage format and libraries might be changed in backward-incompatible ways and are not subject to any SLA or deprecation policy.

CHAPTER 1

Why?

An open-source serialization format and API library to support data interchange across the entire mining community.

CHAPTER 2

Scope

This library provides an abstracted object-based interface to the underlying OMF serialization format, which enables rapid development of the interface while allowing for future changes under the hood.

CHAPTER 3

Goals

- The goal of Open Mining Format is to standardize data formats across the mining community and promote collaboration
- The goal of the API library is to provide a well-documented, object-based interface for serializing OMF files

CHAPTER 4

Alternatives

OMF is intended to supplement the many alternative closed-source file formats used in the mining community.

CHAPTER 5

Connections

This library makes use of the properties open-source project, which is designed and publicly supported by Seequent.

CHAPTER 6

Installation

To install the repository, ensure that you have pip installed and run:

```
pip install omf
```

Or from github:

```
git clone https://github.com/gmggroup/omf.git
cd omf
pip install -e .
```

CHAPTER 7

3D Visualization

To easily visualize OMF project files and data objects in a pure Python environment, check out `omfvtk` (OMF-VTK) which provides tools for creating interactive renderings of OMF datasets using `vtki` (the vtkInterface).

Contents:

7.1 OMF API Index

The OMF API contains tools for creating *Project* and adding *PointSet*, *LineSet*, *Surface*, and *Volume*. These different elements may have *Data* or image *Texture*.

7.1.1 Project

Projects contain a list of *PointSet*, *LineSet*, *Surface*, and *Volume*. Projects can be serialized to file using `OMFWriter`:

```
proj = omf.Project()
...
proj.elements = [...]
...
OMFWriter(proj, 'outfile.omf')
```

For more details on how to build a project, see the *OMF API Example*.

class omf.base.**Project**(***kwargs*)
OMF Project for serializing to .omf file

Required Properties:

- **author** (`String`): Author, a unicode string
- **description** (`String`): Description, a unicode string
- **elements** (a list of *`ProjectElement`*): Project Elements, a list (each item is an instance of ProjectElement)

- **name** (String): Title, a unicode string
- **origin** (Vector3): Origin point for all elements in the project, a 3D Vector of <type 'float'> with shape (3), Default: [0.0, 0.0, 0.0]
- **revision** (String): Revision, a unicode string
- **units** (String): Spatial units of project, a unicode string

Optional Properties:

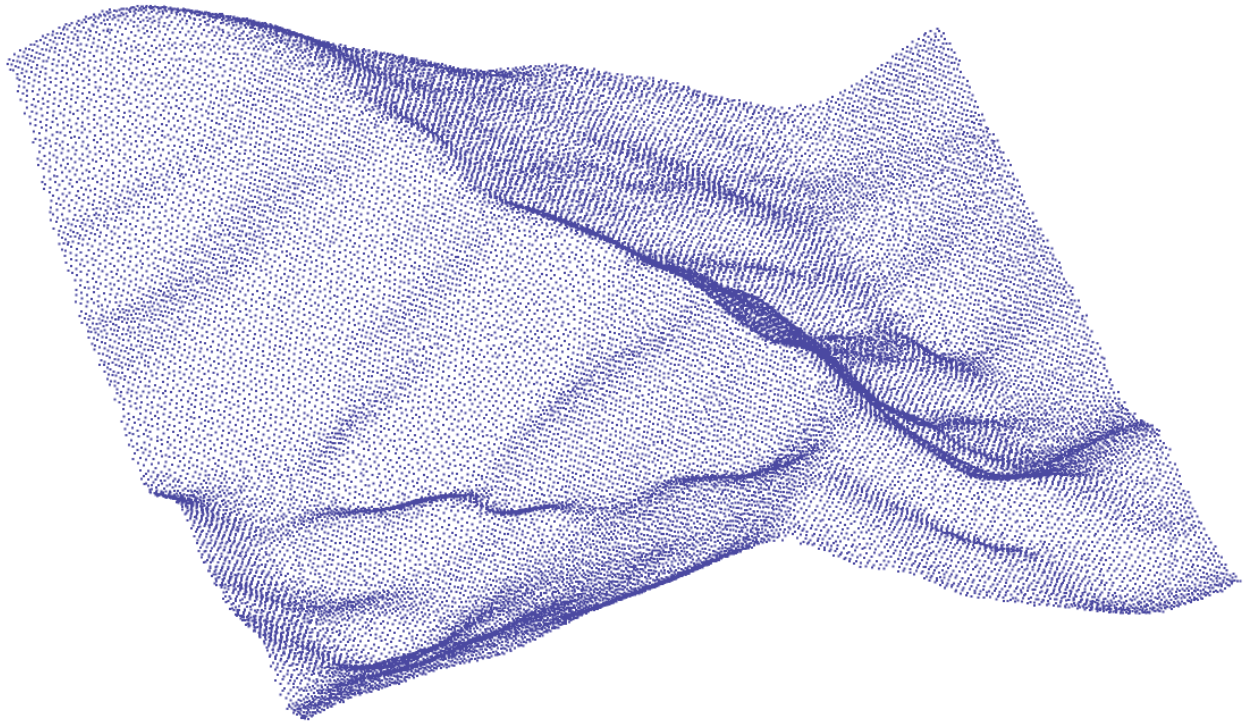
- **date** (DateTime): Date associated with the project data, a datetime object

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

7.1.2 PointSet

Transfering LIDAR point-cloud data from surveying software into 3D modelling software packages.



Element

class omf.pointset.**PointSetElement**(***kwargs*)

Contains mesh, data, textures, and options of a point set

Required Properties:

- **color** (Color): Solid color, a color, Default: random
- **description** (String): Description, a unicode string

- **geometry** (*PointSetGeometry*): Structure of the point set element, an instance of PointSetGeometry
- **name** (String): Title, a unicode string
- **subtype** (StringChoice): Category of PointSet, any of "point", "collar", "blasthole", Default: point

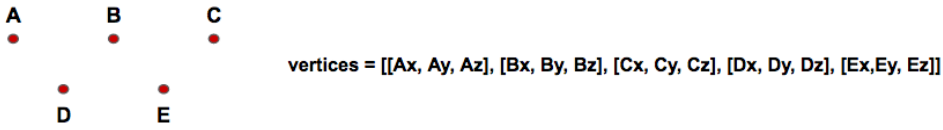
Optional Properties:

- **data** (a list of *ProjectElementData*): Data defined on the element, a list (each item is an instance of ProjectElementData)
- **textures** (a list of *ImageTexture*): Images mapped on the element, a list (each item is an instance of ImageTexture)

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

Geometry



class omf.pointset.**PointSetGeometry**(***kwargs*)

Contains spatial information of a point set

Required Properties:

- **origin** (Vector3): Origin of the Mesh relative to origin of the Project, a 3D Vector of <type 'float'> with shape (3), Default: [0.0, 0.0, 0.0]
- **vertices** (*Vector3Array*): Spatial coordinates of points relative to point set origin, an instance of Vector3Array

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

Data

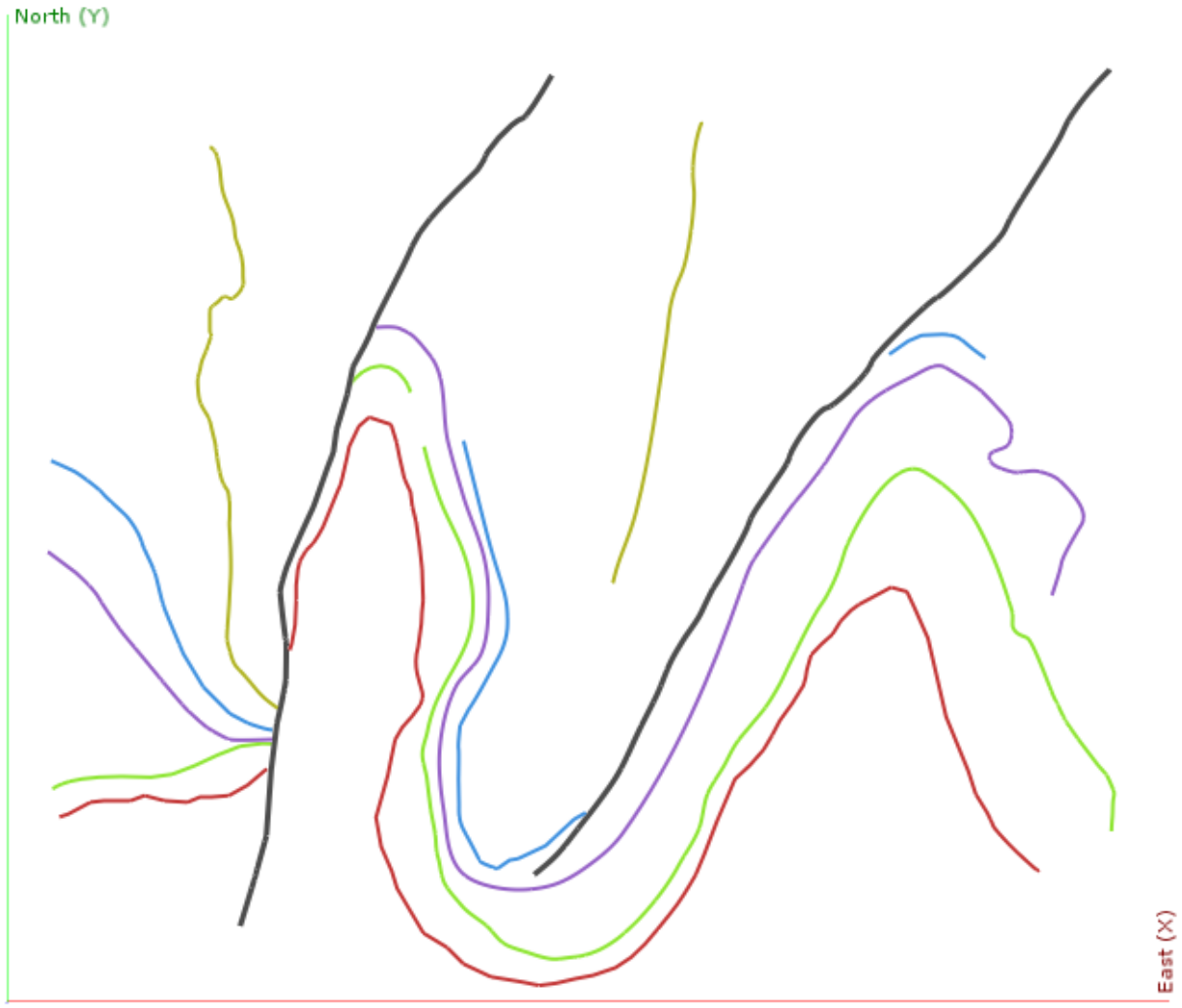
Data is a list of *data*. For PointSets, only `location='vertices'` is valid.

Textures

Textures are *ImageTexture* mapped to the PointSets.

7.1.3 LineSet

Transfer mapped geological contacts from a GIS software package into a 3D modelling software package to help construct a 3D model.



Element

class omf.lineset.**LineSetElement**(***kwargs*)

Contains mesh, data, and options of a line set

Required Properties:

- **color** (Color): Solid color, a color, Default: random
- **description** (String): Description, a unicode string
- **geometry** (*LineSetGeometry*): Structure of the line element, an instance of LineSetGeometry
- **name** (String): Title, a unicode string
- **subtype** (StringChoice): Category of LineSet, either "line" or "borehole", Default: line

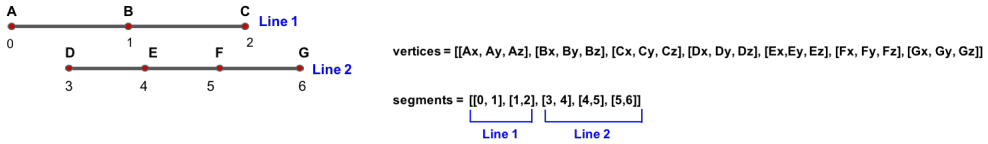
Optional Properties:

- **data** (a list of *ProjectElementData*): Data defined on the element, a list (each item is an instance of ProjectElementData)

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

Geometry



class omf.lineset.**LineSetGeometry**(***kwargs*)

Contains spatial information of a line set

Required Properties:

- **origin** (Vector3): Origin of the Mesh relative to origin of the Project, a 3D Vector of <type 'float'> with shape (3), Default: [0.0, 0.0, 0.0]
- **segments** (*Int2Array*): Endpoint vertex indices of line segments, an instance of Int2Array
- **vertices** (*Vector3Array*): Spatial coordinates of line vertices relative to line set origin, an instance of Vector3Array

Other Properties:

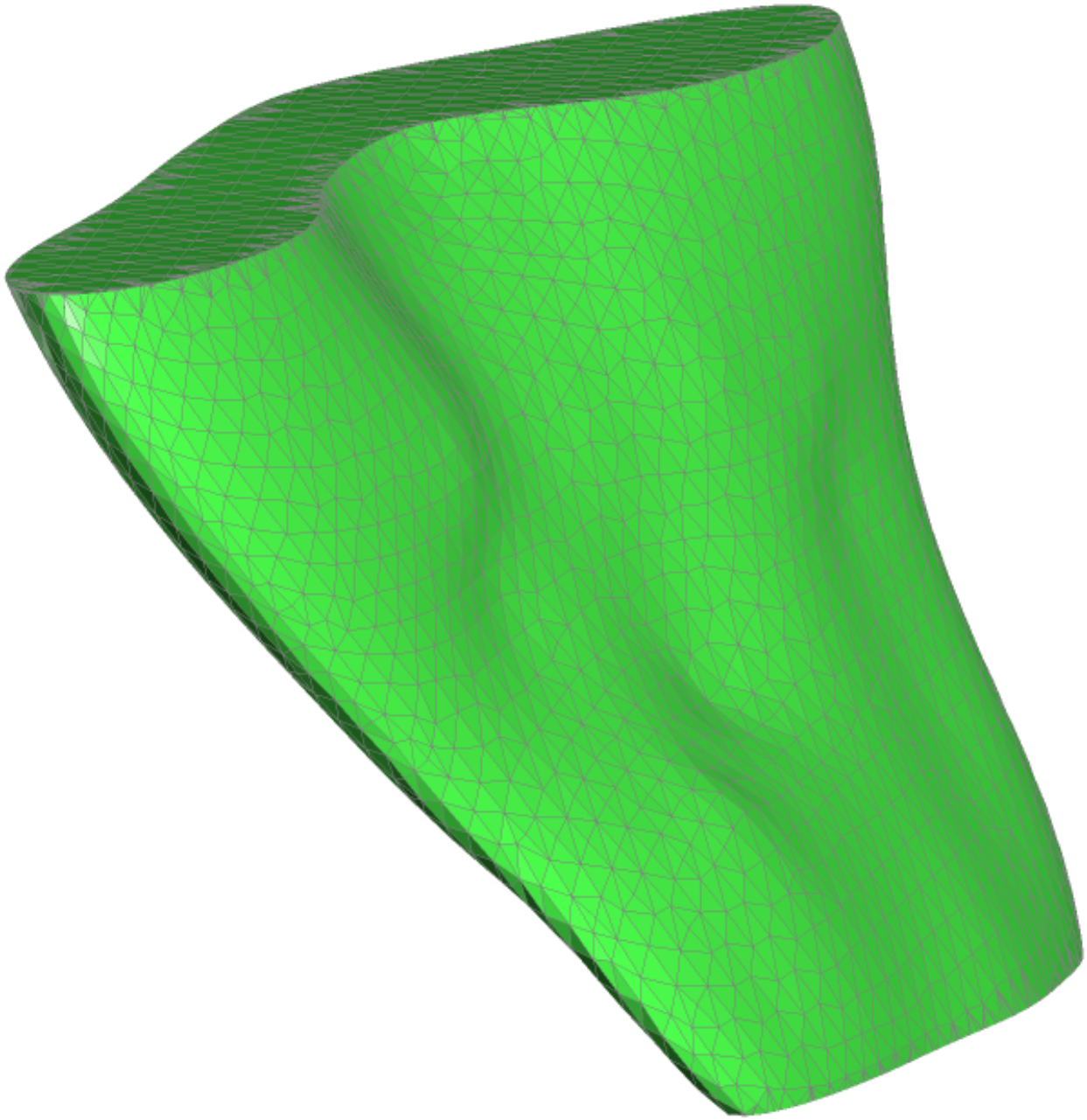
- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

Data

Data is a list of *data*. For Lines, `location='vertices'` and `location='segments'` are valid.

7.1.4 Surface

Transfer geological domains from 3D modelling software to Resource Estimation software.



Element

class omf.surface.**SurfaceElement**(***kwargs*)

Contains mesh, data, textures, and options of a surface

Required Properties:

- **color** (Color): Solid color, a color, Default: random
- **description** (String): Description, a unicode string
- **geometry** (*SurfaceGeometry*, *SurfaceGridGeometry*): Structure of the surface element, an instance of SurfaceGeometry or an instance of SurfaceGridGeometry

- **name** (String): Title, a unicode string
- **subtype** (StringChoice): Category of Surface, any of "surface", Default: surface

Optional Properties:

- **data** (a list of *ProjectElementData*): Data defined on the element, a list (each item is an instance of ProjectElementData)
- **textures** (a list of *ImageTexture*): Images mapped on the surface element, a list (each item is an instance of ImageTexture)

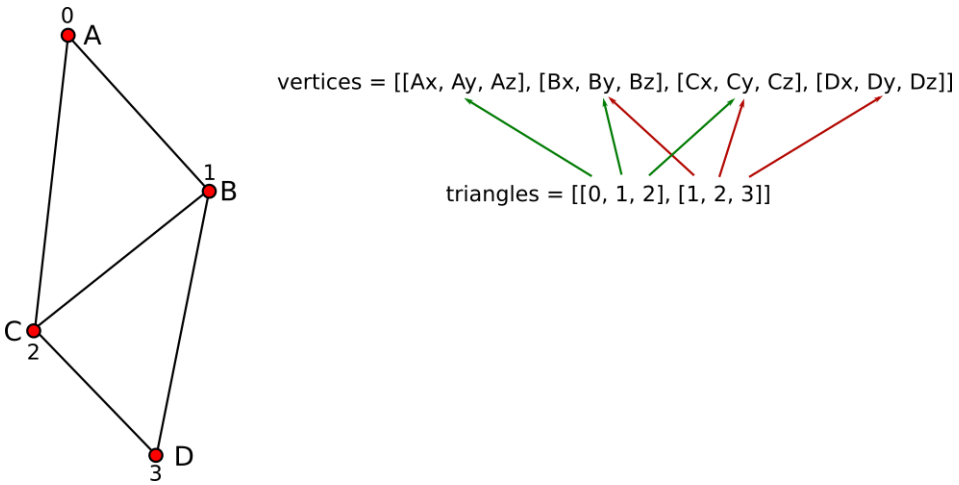
Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

Geometry

Surfaces have two available geometries: SurfaceGeometry, an unstructured triangular mesh, and SurfaceGridGeometry, a gridded mesh.

SurfaceGeometry



class omf.surface.**SurfaceGeometry**(***kwargs*)

Contains spatial information about a triangulated surface

Required Properties:

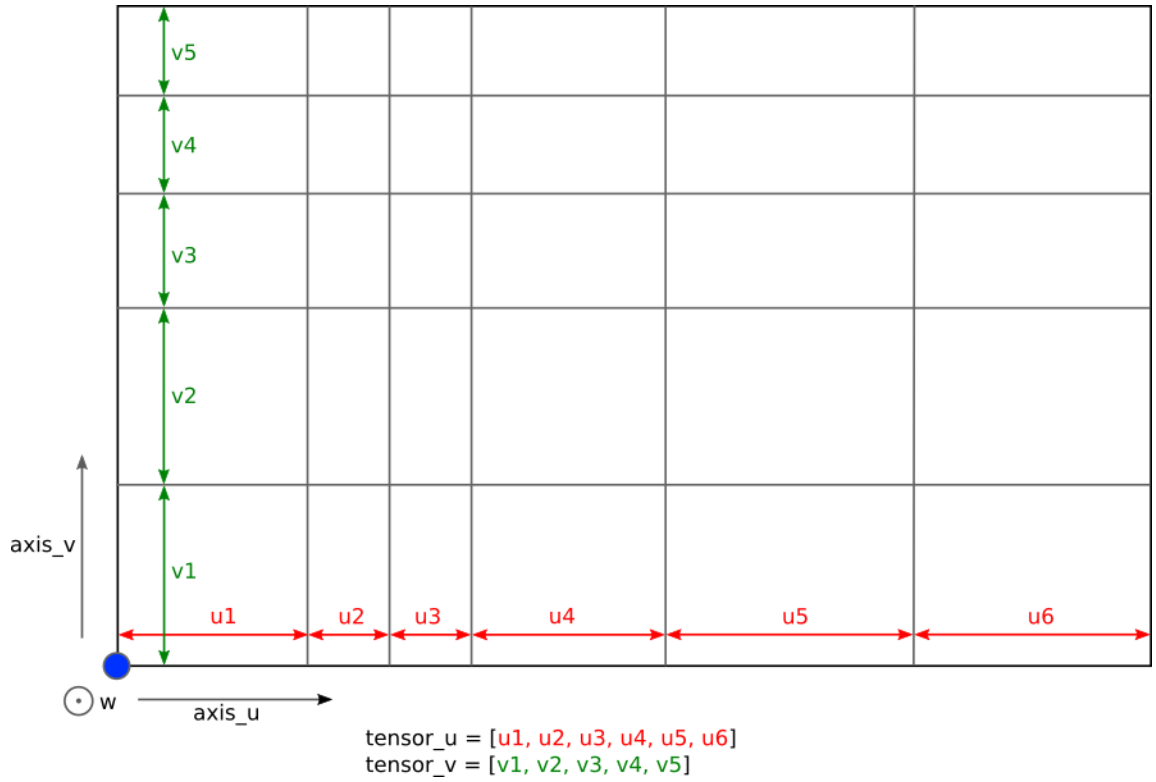
- **origin** (Vector3): Origin of the Mesh relative to origin of the Project, a 3D Vector of <type 'float'> with shape (3), Default: [0.0, 0.0, 0.0]
- **triangles** (*Int3Array*): Vertex indices of surface triangles, an instance of Int3Array
- **vertices** (*Vector3Array*): Spatial coordinates of vertices relative to surface origin, an instance of Vector3Array

Other Properties:

- **date_created** (GettableProperty): Date project was created

- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

SurfaceGridGeometry



class omf.surface.**SurfaceGridGeometry**(***kwargs*)

Contains spatial information of a 2D grid

Required Properties:

- **axis_u** (Vector3): Vector orientation of u-direction, a 3D Vector of <type 'float'> with shape (3), Default: X
- **axis_v** (Vector3): Vector orientation of v-direction, a 3D Vector of <type 'float'> with shape (3), Default: Y
- **origin** (Vector3): Origin of the Mesh relative to origin of the Project, a 3D Vector of <type 'float'> with shape (3), Default: [0.0, 0.0, 0.0]
- **tensor_u** (Array): Grid cell widths, u-direction, a list or numpy array of <type 'float'> with shape (*)
- **tensor_v** (Array): Grid cell widths, v-direction, a list or numpy array of <type 'float'> with shape (*)

Optional Properties:

- **offset_w** (*ScalarArray*): Node offset, an instance of ScalarArray

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified

- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

Data

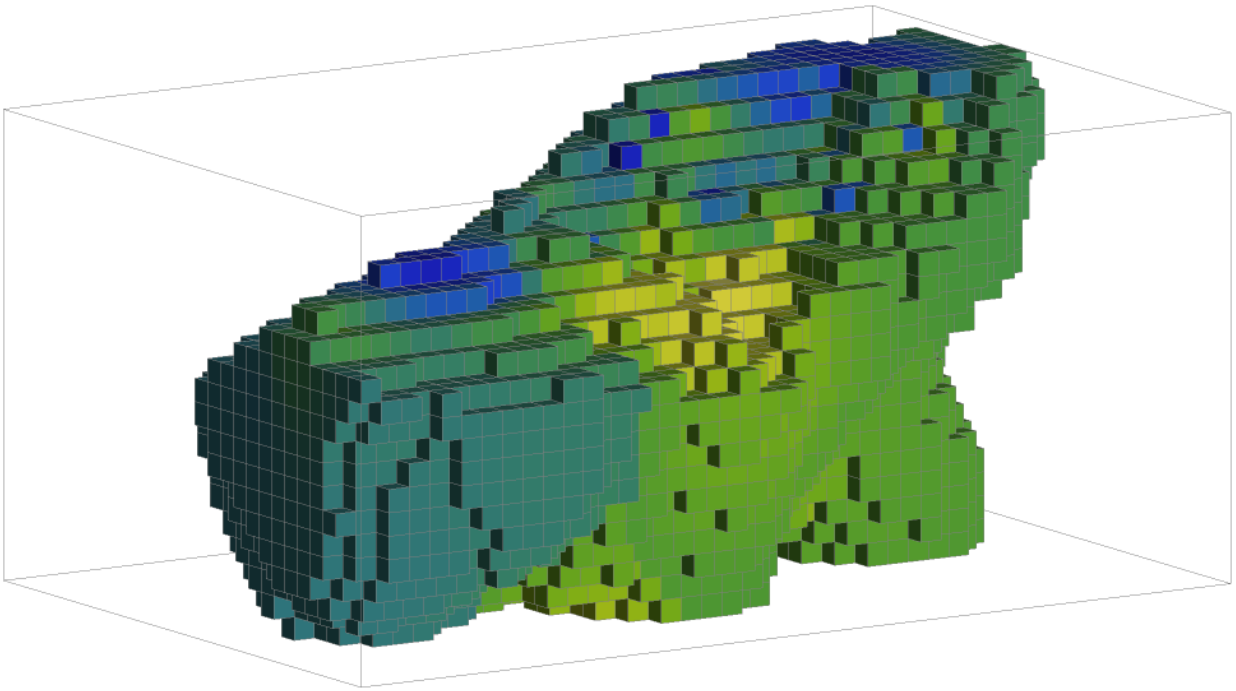
Data is a list of *data*. For Surfaces, `location='vertices'` and `location='faces'` are valid.

Textures

Textures are *ImageTexture* mapped to the Surface.

7.1.5 Volume

Transferring a block model from Resource Estimation software into Mine planning software.



Element

class omf.volume.**VolumeElement**(***kwargs*)

Contains mesh, data, and options of a volume

Required Properties:

- **color** (Color): Solid color, a color, Default: random
- **description** (String): Description, a unicode string
- **geometry** (*VolumeGridGeometry*): Structure of the volume element, an instance of VolumeGridGeometry
- **name** (String): Title, a unicode string
- **subtype** (StringChoice): Category of Volume, any of "volume", Default: volume

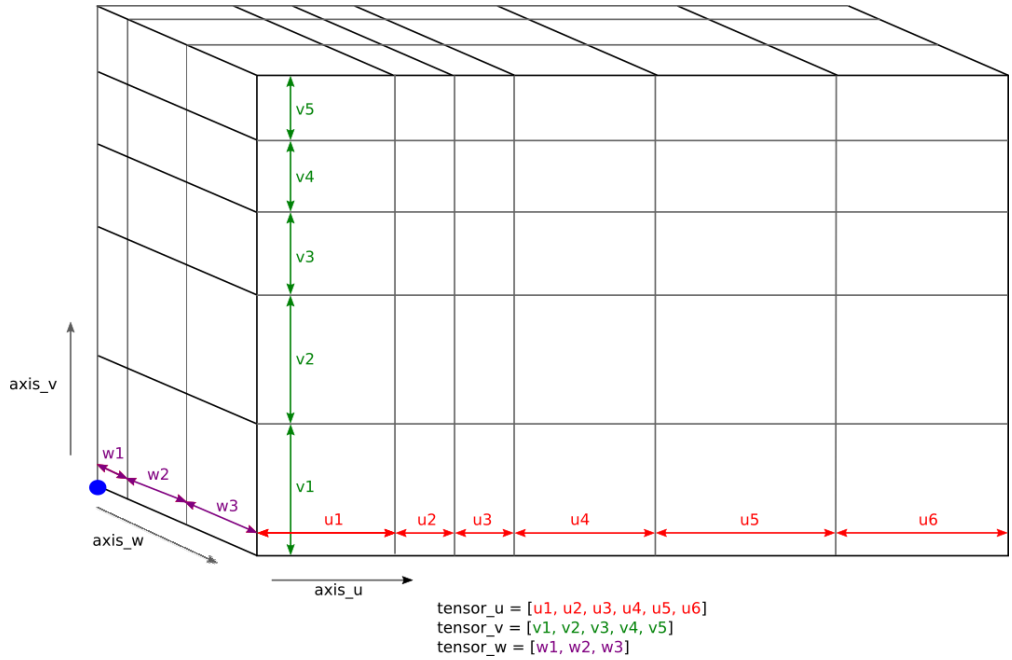
Optional Properties:

- **data** (a list of *ProjectElementData*): Data defined on the element, a list (each item is an instance of ProjectElementData)

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

Geometry



class omf.volume.**VolumeGridGeometry**(***kwargs*)

Contains spatial information of a 3D grid volume.

Required Properties:

- **axis_u** (Vector3): Vector orientation of u-direction, a 3D Vector of <type 'float'> with shape (3), Default: X
- **axis_v** (Vector3): Vector orientation of v-direction, a 3D Vector of <type 'float'> with shape (3), Default: Y
- **axis_w** (Vector3): Vector orientation of w-direction, a 3D Vector of <type 'float'> with shape (3), Default: Z
- **origin** (Vector3): Origin of the Mesh relative to origin of the Project, a 3D Vector of <type 'float'> with shape (3), Default: [0.0, 0.0, 0.0]
- **tensor_u** (Array): Tensor cell widths, u-direction, a list or numpy array of <type 'float'> with shape (*)
- **tensor_v** (Array): Tensor cell widths, v-direction, a list or numpy array of <type 'float'> with shape (*)
- **tensor_w** (Array): Tensor cell widths, w-direction, a list or numpy array of <type 'float'> with shape (*)

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

Data

Data is a list of *data*. For Volumes, `location='vertices'` and `location='cells'` are valid.

7.1.6 Data

ProjectElements include a list of ProjectElementData. These specify mesh location ('vertices', 'faces', etc.) as well as the array, name, and description. See class descriptions below for specific types of Data.

Mapping array values to a mesh is staightforward for unstructured meshes (those defined by vertices, segments, triangles, etc); the order of the data array simply corresponds to the order of the associated mesh parameter. For grid meshes, however, mapping 1D data array to the 2D or 3D grid requires correctly ordered unwrapping. The default is C-style, row-major ordering, `order='c'`. To align data this way, you may start with a numpy array that is size (x, y) for 2D data or size (x, y, z) for 3D data then use numpy's `flatten()` function with default order 'C'. Alternatively, if your data uses Fortran- or Matlab-style, column-major ordering, you may specify data `order='f'`.

Here is a code snippet to show data binding in action; this assumes the surface contains a mesh with 9 vertices and 4 faces (ie a 2x2 square grid).

```
>> ...
>> my_surface = omf.Surface(...)
>> ...
>> my_node_data = omf.ScalarData(
       name='Nine Numbers',
       array=[0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0],
       location='vertices',
       order='c'  # Default
   )
>> my_face_data = omf.ScalarData(
       name='Four Numbers',
       array=[0.0, 1.0, 2.0, 3.0],
       location='faces'
   )
>> my_surface.data = [
       my_face_data,
       my_node_data
   ]
```

ScalarData

class omf.data.**ScalarData**(***kwargs*)

Data array with scalar values

Required Properties:

- **array** (*ScalarArray*): scalar values at locations on a mesh (see location parameter), an instance of ScalarArray
- **description** (String): Description, a unicode string
- **location** (StringChoice): Location of the data on mesh, any of "vertices", "segments", "faces", "cells"

- **name** (String): Title, a unicode string

Optional Properties:

- **colormap** (*ScalarColormap*): colormap associated with the data, an instance of ScalarColormap

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

Vector3Data

class omf.data.**Vector3Data**(***kwargs*)

Data array with 3D vectors

Required Properties:

- **array** (*Vector3Array*): 3D vectors at locations on a mesh (see location parameter), an instance of Vector3Array
- **description** (String): Description, a unicode string
- **location** (StringChoice): Location of the data on mesh, any of "vertices", "segments", "faces", "cells"
- **name** (String): Title, a unicode string

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

Vector2Data

class omf.data.**Vector2Data**(***kwargs*)

Data array with 2D vectors

Required Properties:

- **array** (*Vector2Array*): 2D vectors at locations on a mesh (see location parameter), an instance of Vector2Array
- **description** (String): Description, a unicode string
- **location** (StringChoice): Location of the data on mesh, any of "vertices", "segments", "faces", "cells"
- **name** (String): Title, a unicode string

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

ColorData

class omf.data.**ColorData**(***kwargs*)

Data array of RGB colors specified as three integers 0-255 or color

If n x 3 integers is provided, these will simply be clipped to values between 0 and 255 inclusive; invalid colors will not error. This allows fast array validation rather than slow element-by-element list validation.

Other color formats may be used (ie String or Hex colors). However, for large arrays, validation of these types will be slow.

Required Properties:

- **array** (*Int3Array*, *ColorArray*): RGB color values at locations on a mesh (see location parameter), an instance of Int3Array or an instance of ColorArray
- **description** (String): Description, a unicode string
- **location** (StringChoice): Location of the data on mesh, any of "vertices", "segments", "faces", "cells"
- **name** (String): Title, a unicode string

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

StringData

class omf.data.**StringData**(***kwargs*)

Data array with text entries

Required Properties:

- **array** (*StringArray*): text at locations on a mesh (see location parameter), an instance of StringArray
- **description** (String): Description, a unicode string
- **location** (StringChoice): Location of the data on mesh, any of "vertices", "segments", "faces", "cells"
- **name** (String): Title, a unicode string

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

DateTimeData

class omf.data.**DateTimeData**(***kwargs*)

Data array with DateTime entries

Required Properties:

- **array** (*DateTimeArray*): datetimes at locations on a mesh (see location parameter), an instance of DateTimeArray

- **description** (String): Description, a unicode string
- **location** (StringChoice): Location of the data on mesh, any of "vertices", "segments", "faces", "cells"
- **name** (String): Title, a unicode string

Optional Properties:

- **colormap** (*DateTimeColormap*): colormap associated with the data, an instance of DateTimeColormap

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

MappedData

class omf.data.**MappedData**(***kwargs*)

Data array of indices linked to legend values or -1 for no data

Required Properties:

- **array** (*ScalarArray*): indices into 1 or more legends for locations on a mesh, an instance of ScalarArray
- **description** (String): Description, a unicode string
- **legends** (a list of *Legend*): legends into which the indices map, a list (each item is an instance of Legend)
- **location** (StringChoice): Location of the data on mesh, any of "vertices", "segments", "faces", "cells"
- **name** (String): Title, a unicode string

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

Legend

class omf.data.**Legend**(***kwargs*)

Legends to be used with DataMap indices

Required Properties:

- **description** (String): Description, a unicode string
- **name** (String): Title, a unicode string
- **values** (*ColorArray*, *DateTimeArray*, *StringArray*, *ScalarArray*): values for mapping indexed data, an instance of ColorArray or an instance of DateTimeArray or an instance of StringArray or an instance of ScalarArray

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified

- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

ScalarColormap

class omf.data.**ScalarColormap**(***kwargs*)

Length-128 color gradient with min/max values, used with ScalarData

Required Properties:

- **description** (String): Description, a unicode string
- **gradient** (*ColorArray*): length-128 ColorArray defining the gradient, an instance of ColorArray
- **limits** (a list of Float): Data range associated with the gradient, a list (each item is a float) with length of 2
- **name** (String): Title, a unicode string

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

DateTimeColormap

class omf.data.**DateTimeColormap**(***kwargs*)

Length-128 color gradient with min/max values, used with DateTimeData

Required Properties:

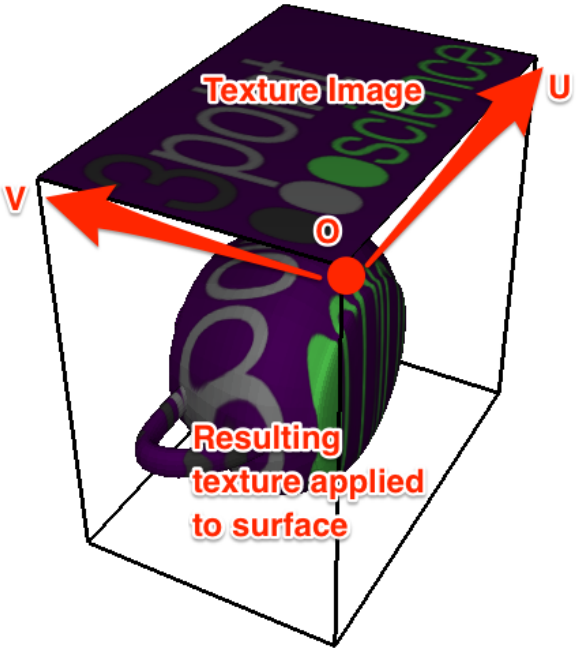
- **description** (String): Description, a unicode string
- **gradient** (*ColorArray*): length-128 ColorArray defining the gradient, an instance of ColorArray
- **limits** (a list of DateTime): Data range associated with the gradient, a list (each item is a datetime object) with length of 2
- **name** (String): Title, a unicode string

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

7.1.7 Texture

Textures are images that exist in space and are mapped to their corresponding elements. Unlike data, they do not need to correspond to mesh nodes or cell centers. This image shows how textures are mapped to a surface. Their position is defined by an origin and axis vectors then they are mapped laterally to the element position.



Like data, multiple textures can be applied to a element; simply provide a list of textures. Each of these textures provides an origin point and two extent vectors for the plane defining where images rests. The *axis_** properties define the extent of that image out from the origin. Given a rectangular PNG image, the *origin* is the bottom left, *origin* + *axis_u* is the bottom right, and *origin* + *axis_v* is the top left. This allows the image to be rotated and/or skewed. These values are independent of the corresponding Surface; in fact, there is nothing requiring the image to actually align with the Surface.

```
>> ...
>> my_surface = omf.SurfaceElement(...)
>> ...
>> my_tex_1 = omf.ImageTexture(
       origin=[0.0, 0.0, 0.0],
       axis_u=[1.0, 0.0, 0.0],
       axis_v=[0.0, 1.0, 0.0],
       image='image1.png'
   )
>> my_tex_2 = omf.ImageTexture(
       origin=[0.0, 0.0, 0.0],
       axis_u=[1.0, 0.0, 0.0],
       axis_v=[0.0, 0.0, 1.0],
       image='image2.png'
   )
>> my_surface.textures = [
       my_tex_1,
```

(continues on next page)

(continued from previous page)

```
        my_tex_2
    ]
```

class omf.texture.**ImageTexture**(***kwargs*)

Contains an image that can be mapped to a point set or surface

Required Properties:

- **axis_u** (Vector3): Vector corresponding to the image x-axis, a 3D Vector of <type 'float'> with shape (3), Default: X
- **axis_v** (Vector3): Vector corresponding to the image y-axis, a 3D Vector of <type 'float'> with shape (3), Default: Y
- **description** (String): Description, a unicode string
- **image** (ImagePNG): PNG image file, a PNG image file, valid modes include (u'ab+', u'rb+', u'wb+', u'rb')
- **name** (String): Title, a unicode string
- **origin** (Vector3): Origin point of the texture, a 3D Vector of <type 'float'> with shape (3), Default: [0.0, 0.0, 0.0]

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

7.1.8 Array Types

Array classes exist allow arrays to be shared across different objects.

ScalarArray

class omf.data.**ScalarArray**(*array=None*, ***kwargs*)

Class with unique ID and data array

Required Properties:

- **array** (Array): Shared Scalar Array, a list or numpy array of <type 'float'>, <type 'int'> with shape (*)

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

Vector3Array

class omf.data.**Vector3Array**(*array=None*, ***kwargs*)

Shared array of 3D vectors

Required Properties:

- **array** (Vector3Array): Shared Vector3 Array, a list of Vector3 of <type 'float'> with shape (*, 3)

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

Vector2Array

class omf.data.**Vector2Array**(*array=None*, ***kwargs*)

Shared array of 2D vectors

Required Properties:

- **array** (Vector2Array): Shared Vector2 Array, a list of Vector2 of <type 'float'> with shape (*, 2)

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

Int3Array

class omf.data.**Int3Array**(*array=None*, ***kwargs*)

Shared n x 3 array of integers

Required Properties:

- **array** (Array): Shared n x 3 Int Array, a list or numpy array of <type 'int'> with shape (*, 3)

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

Int2Array

class omf.data.**Int2Array**(*array=None*, ***kwargs*)

Shared n x 2 array of integers

Required Properties:

- **array** (Array): Shared n x 2 Int Array, a list or numpy array of <type 'int'> with shape (*, 2)

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

ColorArray

class omf.data.**ColorArray**(*array=None*, ***kwargs*)
Shared array of Colors

Required Properties:

- **array** (a list of Color): Shared array of Colors, a list (each item is a color)

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

StringArray

class omf.data.**StringArray**(*array=None*, ***kwargs*)
Shared array of text strings

Required Properties:

- **array** (a list of String): Shared array of text strings, a list (each item is a unicode string)

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

DateTimeArray

class omf.data.**DateTimeArray**(*array=None*, ***kwargs*)
Shared array of DateTimes

Required Properties:

- **array** (a list of DateTime): Shared array of DateTimes, a list (each item is a datetime object)

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

7.1.9 Other Classes

ProjectElement

Available elements are *PointSet*, *LineSet*, *Surface*, and *Volume*; *Project* are built with elements.

class omf.base.**ProjectElement**(***kwargs*)
Base ProjectElement class for OMF file

ProjectElement subclasses must define their mesh. ProjectElements include PointSet, LineSet, Surface, and Volume

Required Properties:

- **color** (Color): Solid color, a color, Default: random
- **description** (String): Description, a unicode string
- **name** (String): Title, a unicode string

Optional Properties:

- **data** (a list of *ProjectElementData*): Data defined on the element, a list (each item is an instance of ProjectElementData)

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

ProjectElement Data

class omf.base.**ProjectElementData**(***kwargs*)

Data array with values at specific locations on the mesh

Required Properties:

- **description** (String): Description, a unicode string
- **location** (StringChoice): Location of the data on mesh, any of "vertices", "segments", "faces", "cells"
- **name** (String): Title, a unicode string

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

Content Model

class omf.base.**ContentModel**(***kwargs*)

ContentModel is a UidModel with title and description

Required Properties:

- **description** (String): Description, a unicode string
- **name** (String): Title, a unicode string

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

Uid Model

UidModel gives all content a name, description, and unique uid.

class omf.base.**UidModel**(***kwargs*)

UidModel is a HasProperties object with uid

Other Properties:

- **date_created** (GettableProperty): Date project was created
- **date_modified** (GettableProperty): Date project was modified
- **uid** (Uuid): Unique identifier, a unique ID auto-generated with uuid.uuid4()

7.2 OMF API Example

This (very impractical) example shows usage of the OMF API.

Also, this example builds elements all at once. They can also be initialized with no arguments, and properties can be set one-by-one (see code snippet at bottom of page).

```
import numpy as np
import omf

proj = omf.Project(
    name='Test project',
    description='Just some assorted elements'
)

pts = omf.PointSetElement(
    name='Random Points',
    description='Just random points',
    geometry=omf.PointSetGeometry(
        vertices=np.random.rand(100, 3)
    ),
    data=[
        omf.ScalarData(
            name='rand data',
            array=np.random.rand(100),
            location='vertices'
        ),
        omf.ScalarData(
            name='More rand data',
            array=np.random.rand(100),
            location='vertices'
        )
    ],
    textures=[
        omf.ImageTexture(
            name='test image',
            image='test_image.png',
            origin=[0, 0, 0],
            axis_u=[1, 0, 0],
            axis_v=[0, 1, 0]
        ),
        omf.ImageTexture(
            name='test image',
```

(continues on next page)

(continued from previous page)

```
            image='test_image.png',
            origin=[0, 0, 0],
            axis_u=[1, 0, 0],
            axis_v=[0, 0, 1]
        )
    ],
    color='green'
)

lin = omf.LineSetElement(
    name='Random Line',
    geometry=omf.LineSetGeometry(
        vertices=np.random.rand(100, 3),
        segments=np.floor(np.random.rand(50, 2)*100).astype(int)
    ),
    data=[
        omf.ScalarData(
            name='rand vert data',
            array=np.random.rand(100),
            location='vertices'
        ),
        omf.ScalarData(
            name='rand segment data',
            array=np.random.rand(50),
            location='segments'
        )
    ],
    color='#0000FF'
)

surf = omf.SurfaceElement(
    name='trisurf',
    geometry=omf.SurfaceGeometry(
        vertices=np.random.rand(100, 3),
        triangles=np.floor(np.random.rand(50, 3)*100).astype(int)
    ),
    data=[
        omf.ScalarData(
            name='rand vert data',
            array=np.random.rand(100),
            location='vertices'
        ),
        omf.ScalarData(
            name='rand face data',
            array=np.random.rand(50),
            location='faces'
        )
    ],
    color=[100, 200, 200]
)

grid = omf.SurfaceElement(
    name='gridsurf',
    geometry=omf.SurfaceGridGeometry(
        tensor_u=np.ones(10).astype(float),
        tensor_v=np.ones(15).astype(float),
        origin=[50., 50., 50.],
```

(continues on next page)

(continued from previous page)

```
        axis_u=[1., 0, 0],
        axis_v=[0, 0, 1.],
        offset_w=np.random.rand(11, 16).flatten()
    ),
    data=[
        omf.ScalarData(
            name='rand vert data',
            array=np.random.rand(11, 16).flatten(),
            location='vertices'
        ),
        omf.ScalarData(
            name='rand face data',
            array=np.random.rand(10, 15).flatten(order='f'),
            location='faces'
        )
    ],
    textures=[
        omf.ImageTexture(
            name='test image',
            image='test_image.png',
            origin=[2., 2., 2.],
            axis_u=[5., 0, 0],
            axis_v=[0, 2., 5.]
        )
    ]
)

vol = omf.VolumeElement(
    name='vol',
    geometry=omf.VolumeGridGeometry(
        tensor_u=np.ones(10).astype(float),
        tensor_v=np.ones(15).astype(float),
        tensor_w=np.ones(20).astype(float),
        origin=[10., 10., -10]
    ),
    data=[
        omf.ScalarData(
            name='Random Data',
            location='cells',
            array=np.random.rand(10, 15, 20).flatten()
        )
    ]
)

proj.elements = [pts, lin, surf, grid, vol]

assert proj.validate()

omf.OMFWriter(proj, 'omfproj.omf')
```

Piecewise building example:

```
...
pts = omf.PointSetElement()
pts.name = 'Random Points',
pts.mesh = omf.PointSetGeometry()
pts.mesh.vertices = np.random.rand(100, 3)
```

(continues on next page)

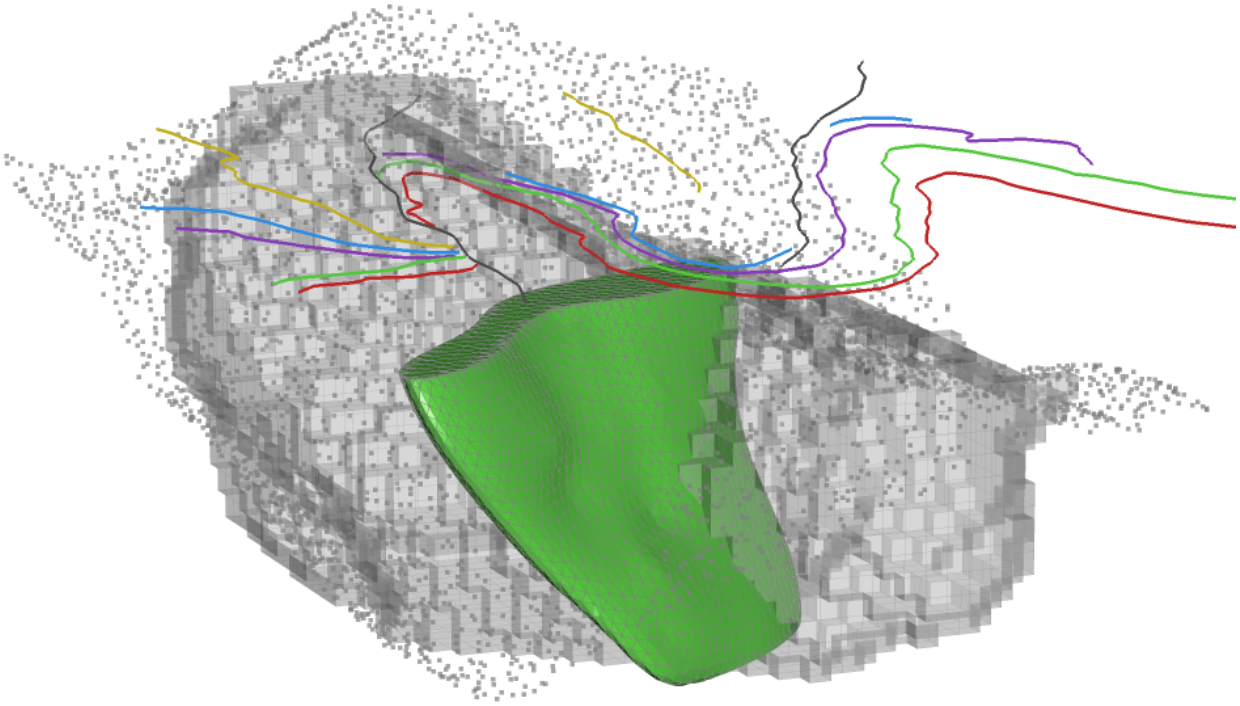
(continued from previous page)

```
...
```

7.3 OMF IO API

7.3.1 OMF Writer

Batch export multiple different object types from a geological modeling software package.



class omf.fileio.**OMFWriter**(*project*, *fname*)

OMFWriter serializes a OMF project to a file

```
proj = omf.project()
...
omf.OMFWriter(proj, 'outfile.omf')
```

The output file starts with a 60 byte header:

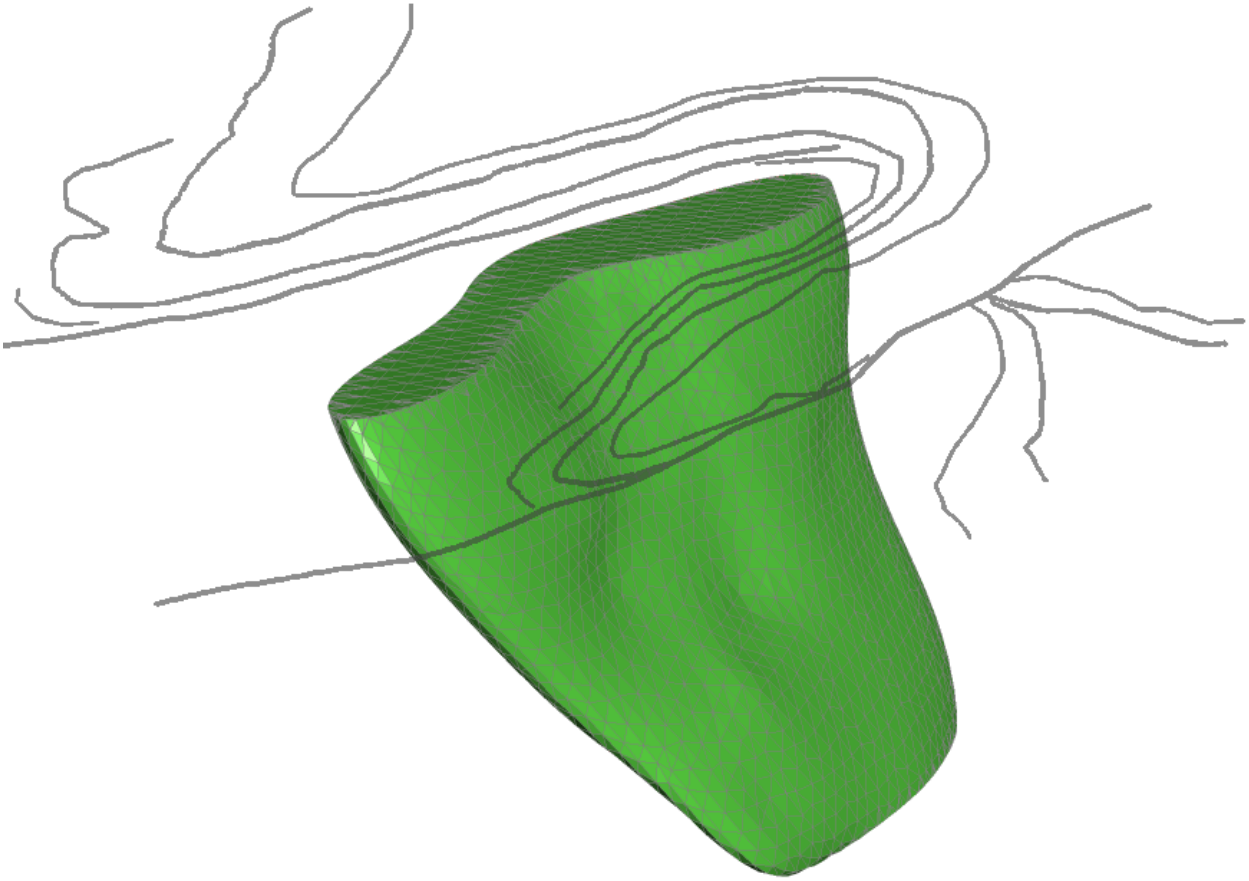
- 4 byte magic number: `b'\x81\x82\x83\x84'`
- 32 byte version string: `'OMF-v0.9.0'` (other bytes empty)
- 16 byte project uid (in little-endian bytes)
- 8 byte unsigned long long (little-endian): JSON start location in file

Following the header is a binary data blob.

Following the binary is a UTF-8 encoded JSON dictionary containing all elements of the project keyed by UID string. Objects can reference each other by UID, and arrays and images contain pointers to their data in the binary blob.

7.3.2 OMF Reader

Select which objects from the file are to be imported into a 3D visualization software.



class omf.fileio.**OMFReader**(*fopen*)

OMFReader deserializes an OMF file.

```
# Read all elements
reader = omf.OMFReader('infile.omf')
project = reader.get_project()

# Read all PointSets:
reader = omf.OMFReader('infile.omf')
project = reader.get_project_overview()
uids_to_import = [element.uid for element in project.elements
                  if isinstance(element, omf.PointSetElement)]
filtered_project = reader.get_project(uids_to_import)
```

CHAPTER 8

Index

- genindex

Index