omf Documentation

Release 1.0.1

Global Mining Guidelines Group

Jun 15, 2021

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

Version 2 of the Open Mining Format (OMF) and the associated Python API is under active development, and subject to backwards-incompatible changes at any time. The latest stable release of Version 1 is available on PyPI.

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 omfvista which provides a module for loading OMF datasets into PyVista mesh objects for 3D visualization and analysis.

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 *Projected 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

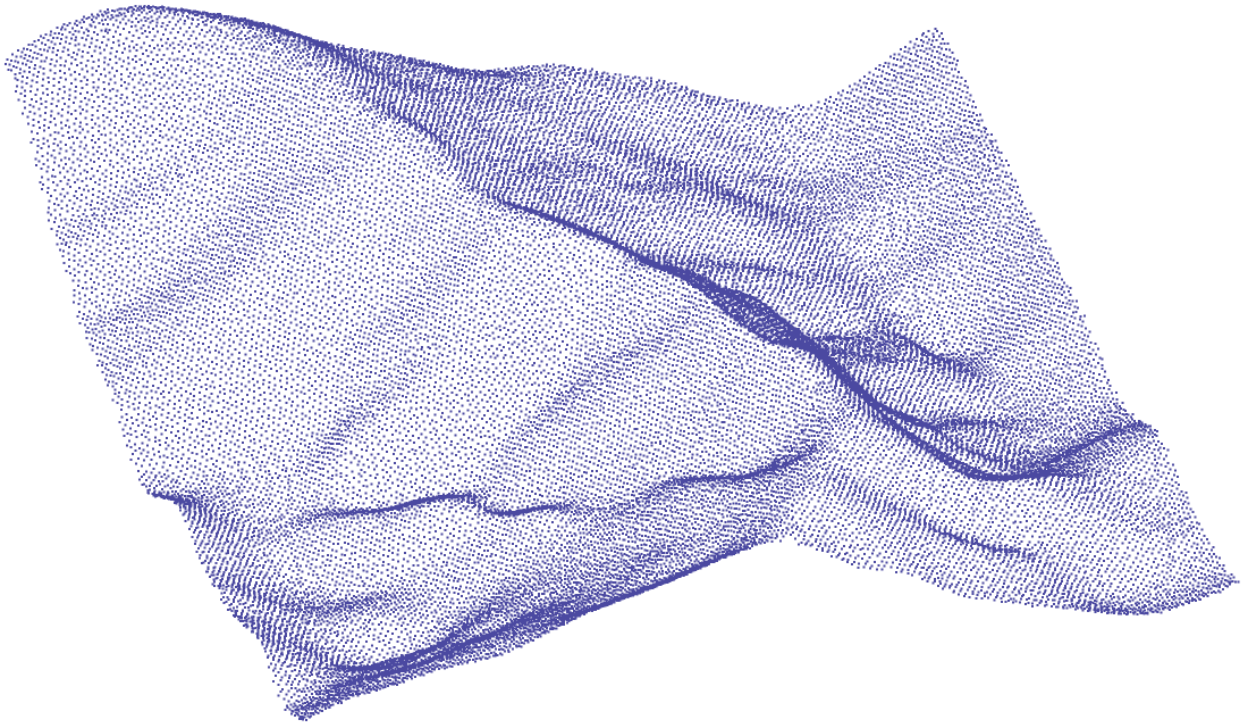
Required Properties:

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

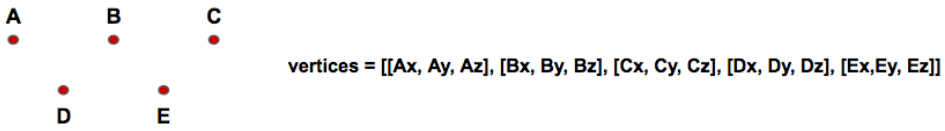
- **metadata** (*ArbitraryMetadataDict*): Project metadata, an arbitrary JSON-serializable dictionary, with certain keys validated against *ProjectMetadata*
- **name** (String): Title of the object, a unicode string
- **uid** (String): Unique identifier, a unicode string, Default: new instance of unicode

7.1.2 PointSet

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



Element



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

Contains point set spatial information and attributes

Required Properties:

- **description** (String): Description of the object, a unicode string
- **metadata** (*ArbitraryMetadataDict*): Element metadata, an arbitrary JSON-serializable dictionary, with certain keys validated against *ElementMetadata*
- **name** (String): Title of the object, a unicode string
- **subtype** (StringChoice): Category of PointSet, any of “point”, “collar”, “blasthole”, Default: point

- **uid** (`String`): Unique identifier, a unicode string, Default: new instance of unicode
- **vertices** (*`Array`*): Spatial coordinates of points relative to point set origin, an instance of Array

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 *`ProjectedTexture`*, a list of *`UVMappedTexture`*): Images mapped on the element, a list (each item is an instance of ProjectedTexture or an instance of UVMappedTexture)

Data

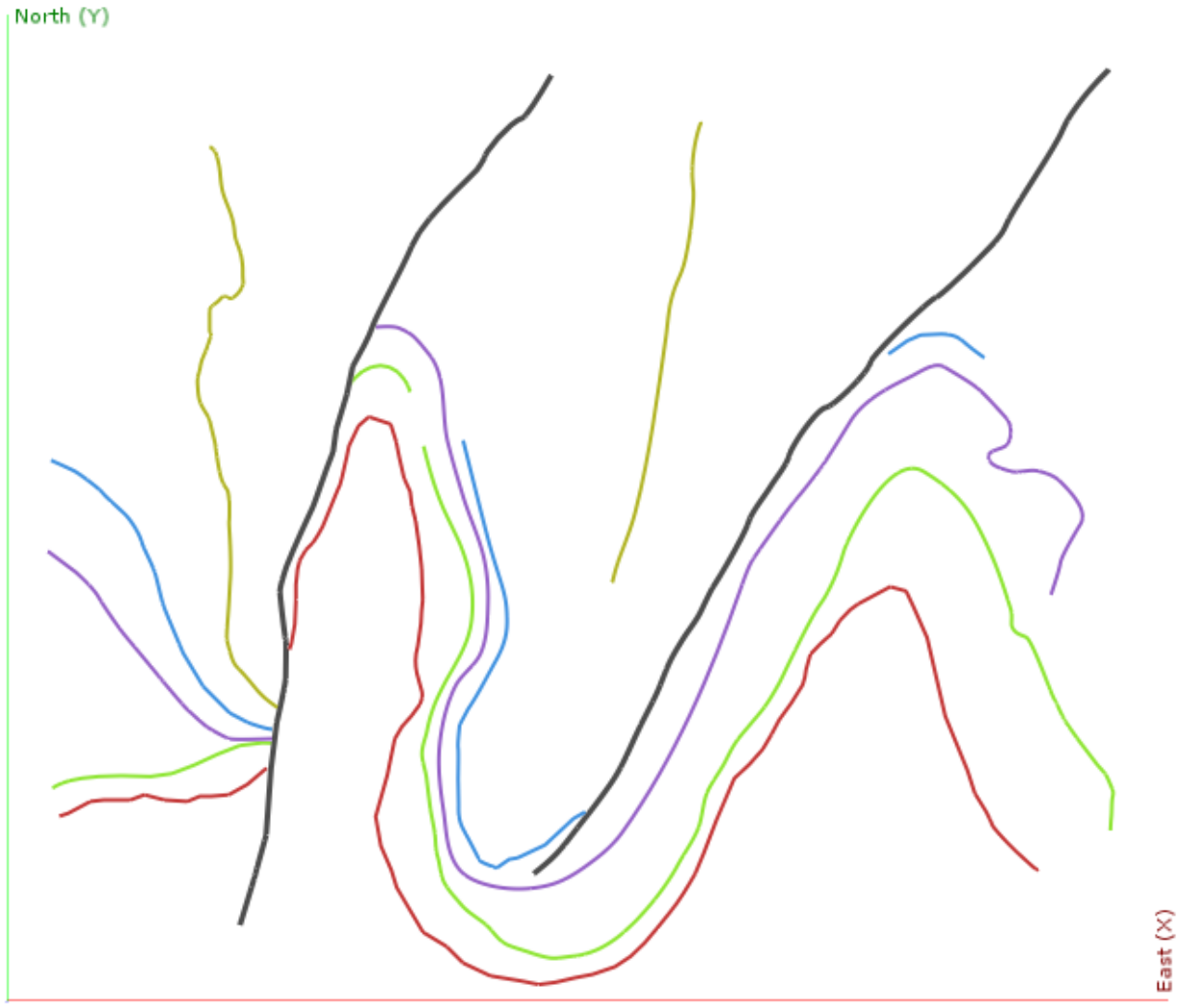
Data is a list of *data*. For PointSets, only `location='vertices'` is valid.

Textures

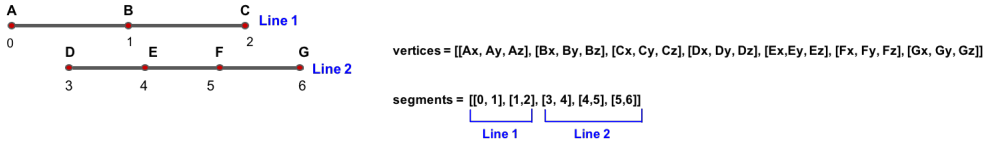
Textures are *ImageTexture* mapped to the PointSets.

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 line set spatial information and attributes

Required Properties:

- **description** (String): Description of the object, a unicode string
- **metadata** (*ArbitraryMetadataDict*): Element metadata, an arbitrary JSON-serializable dictionary, with certain keys validated against *ElementMetadata*
- **name** (String): Title of the object, a unicode string
- **subtype** (StringChoice): Category of LineSet, either “line” or “borehole”, Default: line

- **uid** (*String*): Unique identifier, a unicode string, Default: new instance of unicode
- **vertices** (*Array*): Spatial coordinates of line vertices relative to line set origin, an instance of Array

Optional Properties:

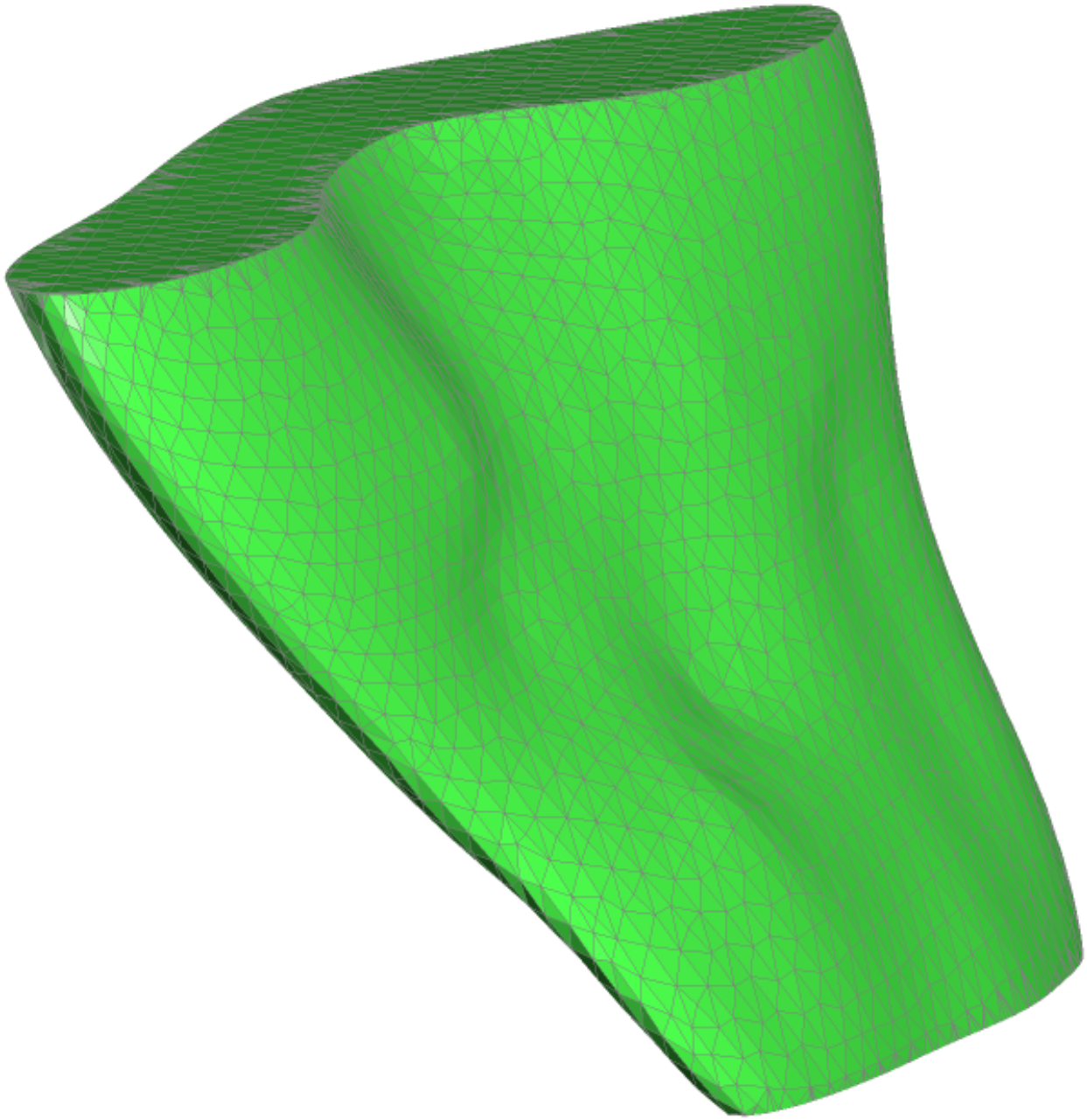
- **data** (a list of *ProjectElementData*): Data defined on the element, a list (each item is an instance of ProjectElementData)
- **segments** (*Array*): Endpoint vertex indices of line segments; if segments is not specified, the vertices are connected in order, equivalent to segments=[[0, 1], [1, 2], [2, 3], ...], an instance of Array

Data

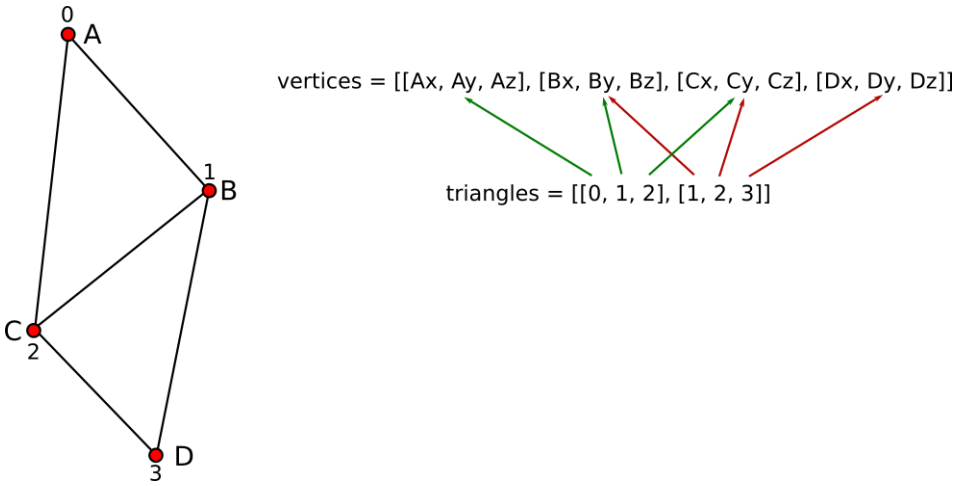
Data is a list of *data*. For Lines, `location='vertices'` and `location='segments'` are valid.

7.1.4 Surface

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



Elements



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

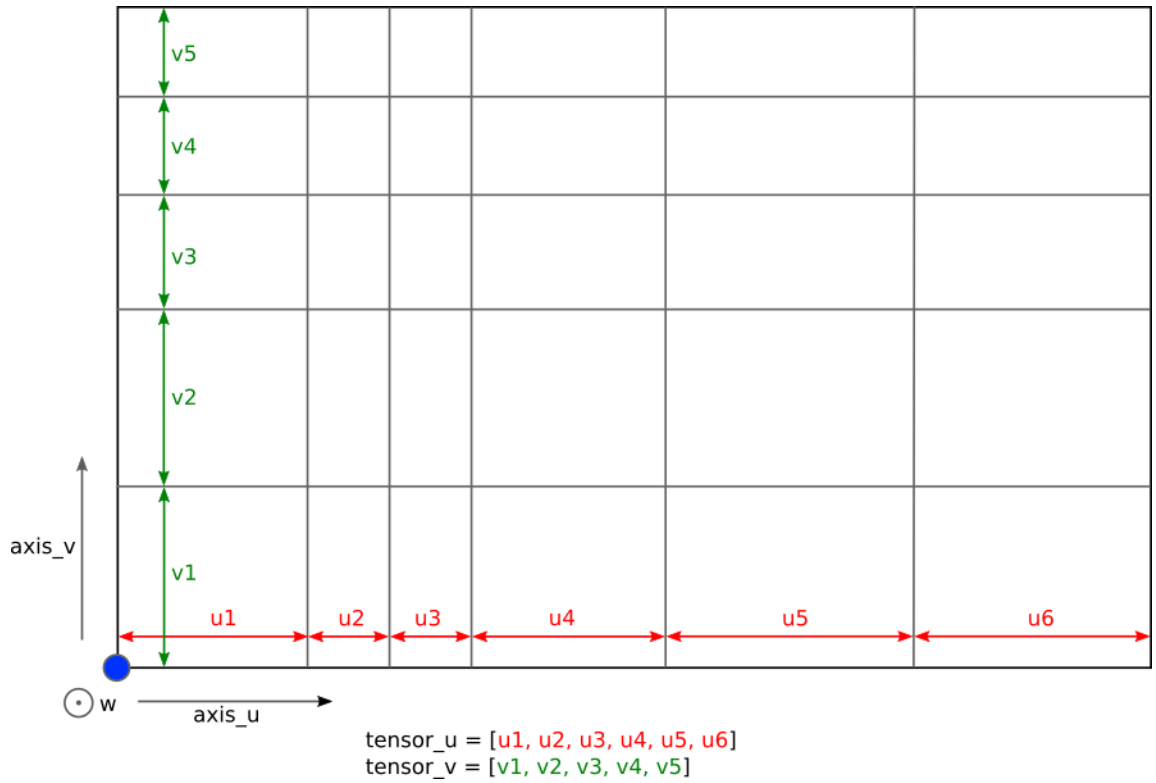
Contains triangulated surface spatial information and attributes

Required Properties:

- **description** (String): Description of the object, a unicode string
- **metadata** (*ArbitraryMetadataDict*): Element metadata, an arbitrary JSON-serializable dictionary, with certain keys validated against *ElementMetadata*
- **name** (String): Title of the object, a unicode string
- **subtype** (StringChoice): Category of Surface, any of "surface", Default: surface
- **triangles** (*Array*): Vertex indices of surface triangles, an instance of Array
- **uid** (String): Unique identifier, a unicode string, Default: new instance of unicode
- **vertices** (*Array*): Spatial coordinates of vertices relative to surface origin, an instance of Array

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 *ProjectedTexture*, a list of *UVMappedTexture*): Images mapped on the element, a list (each item is an instance of ProjectedTexture or an instance of UVMappedTexture)



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

Contains 2D grid spatial information and attributes

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
- **description** (String): Description of the object, a unicode string
- **metadata** (*ArbitraryMetadataDict*): Element metadata, an arbitrary JSON-serializable dictionary, with certain keys validated against *ElementMetadata*
- **name** (String): Title of the object, a unicode string
- **origin** (Vector3): Origin of the Mesh relative to Project coordinate reference system, a 3D Vector of <type 'float'> with shape (3), Default: [0.0, 0.0, 0.0]
- **subtype** (StringChoice): Category of Surface, any of "surface", Default: surface
- **tensor_u** (a list of Float): Grid cell widths, u-direction, a list (each item is a float in range [0.0, inf])
- **tensor_v** (a list of Float): Grid cell widths, v-direction, a list (each item is a float in range [0.0, inf])
- **uid** (String): Unique identifier, a unicode string, Default: new instance of unicode

Optional Properties:

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

- **textures** (a list of *ProjectedTexture*, a list of *UVMappedTexture*): Images mapped on the element, a list (each item is an instance of ProjectedTexture or an instance of UVMappedTexture)

Data

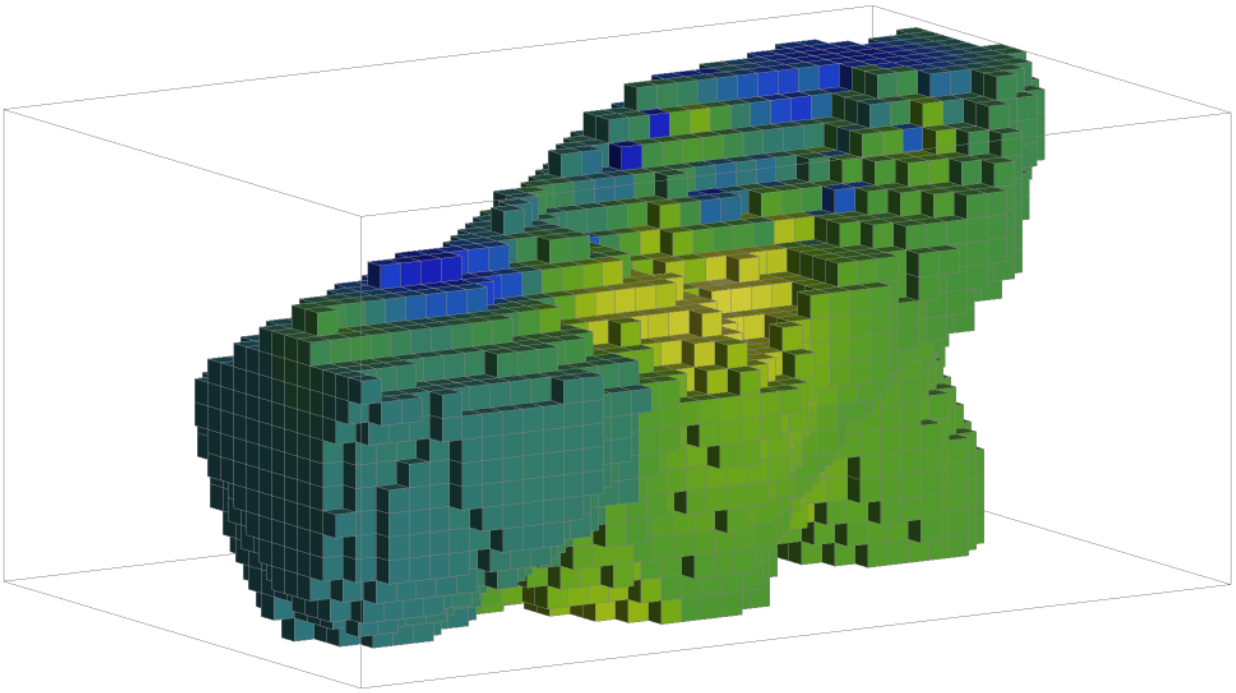
Data is a list of *data*. For Surfaces, `location='vertices'` and `location='faces'` are valid.

Textures

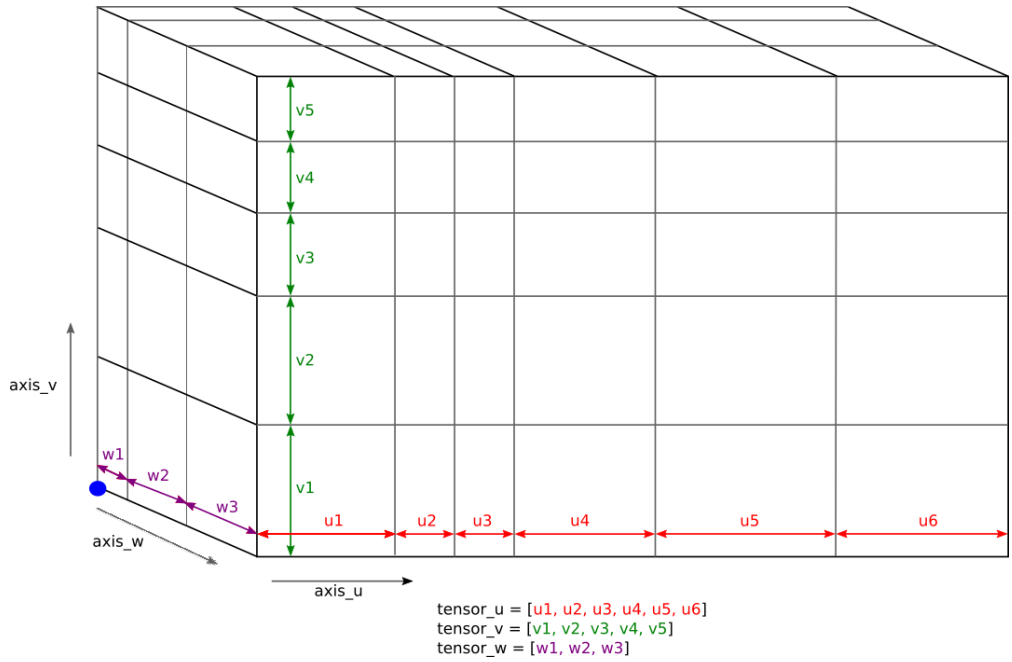
Textures are *ImageTexture* mapped to the Surface.

7.1.5 Volume

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



Element



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

Contains 3D grid volume spatial information and attributes

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
- **description** (String): Description of the object, a unicode string
- **metadata** (*ArbitraryMetadataDict*): Element metadata, an arbitrary JSON-serializable dictionary, with certain keys validated against *ElementMetadata*
- **name** (String): Title of the object, a unicode string
- **origin** (Vector3): Origin of the Mesh relative to Project coordinate reference system, a 3D Vector of <type 'float'> with shape (3), Default: [0.0, 0.0, 0.0]
- **subtype** (StringChoice): Category of Volume, any of "volume", Default: volume
- **tensor_u** (a list of Float): Tensor cell widths, u-direction, a list (each item is a float in range [0.0, inf])
- **tensor_v** (a list of Float): Tensor cell widths, v-direction, a list (each item is a float in range [0.0, inf])
- **tensor_w** (a list of Float): Tensor cell widths, w-direction, a list (each item is a float in range [0.0, inf])
- **uid** (String): Unique identifier, a unicode string, Default: new instance of unicode

Optional Properties:

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

class omf.blockmodel.**RegularBlockModel**(***kwargs*)

Block model with constant spacing in each dimension

Required Properties:

- **description** (String): Description of the object, a unicode string
- **metadata** (*ArbitraryMetadataDict*): Element metadata, an arbitrary JSON-serializable dictionary, with certain keys validated against *ElementMetadata*
- **name** (String): Title of the object, a unicode string
- **num_blocks** (a list of Integer): Number of blocks along u, v, and w axes, a list (each item is an integer in range [1, inf]) with length of 3
- **size_blocks** (a list of Float): Size of blocks in the u, v, and w dimensions, a list (each item is a float in range [0, inf]) with length of 3
- **uid** (String): Unique identifier, a unicode string, Default: new instance of unicode

Optional Properties:

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

Other Properties:

- **cbc** (dynamic Array): Compressed block count - for regular block models this must have length equal to the product of num_blocks and all values must be 1 (if attributes exist on the block) or 0; the default is an array of 1s, a list or numpy array of <type 'int'>, <type 'bool'> with shape (*) created dynamically
- **cbi** (dynamic Array): Compressed block index - used for indexing attributes into the block model; must have length equal to the product of num_blocks plus 1 and monotonically increasing, a list or numpy array of <type 'int'> with shape (*) created dynamically

Data

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

7.1.6 Composite Element

Composite Elements are used to compose multiple other elements into a single, more complex, grouped object.

Element

class omf.composite.**CompositeElement**(***kwargs*)

Element constructed from other primitive elements

Required Properties:

- **description** (String): Description of the object, a unicode string
- **elements** (a list of *RegularBlockModel*, a list of *LineSetElement*, a list of *PointSetElement*, a list of *SurfaceElement*, a list of *SurfaceGridElement*, a list of *VolumeGridElement*): Elements grouped into one composite element, a list (each item is an instance of RegularBlockModel or an instance of LineSetElement or an instance of PointSetElement or an instance of SurfaceElement or an instance of SurfaceGridElement or an instance of VolumeGridElement)
- **metadata** (*ArbitraryMetadataDict*): Element metadata, an arbitrary JSON-serializable dictionary, with certain keys validated against *ElementMetadata*

- **name** (`String`): Title of the object, a unicode string
- **uid** (`String`): Unique identifier, a unicode string, Default: new instance of unicode

Optional Properties:

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

Data

Data is a list of *data*. For Composite Elements, only `location='elements'` is valid. However, Data may also be defined on the child `elements`

7.1.7 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
   ]
```

NumericData

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

Data array with scalar values

Required Properties:

- **array** (*Array*): Numeric values at locations on a mesh (see location parameter); these values must be scalars, an instance of Array
- **description** (String): Description of the object, a unicode string
- **location** (StringChoice): Location of the data on mesh, any of "vertices", "segments", "faces", "cells", "elements"
- **metadata** (*ArbitraryMetadataDict*): Attribute metadata, an arbitrary JSON-serializable dictionary, with certain keys validated against *AttributeMetadata*
- **name** (String): Title of the object, a unicode string
- **uid** (String): Unique identifier, a unicode string, Default: new instance of unicode

Optional Properties:

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

VectorData

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

Data array with vector values

This data type cannot have a colormap, since you cannot map colormaps to vectors.

Required Properties:

- **array** (*Array*): Numeric vectors at locations on a mesh (see location parameter); these vectors may be 2D or 3D, an instance of Array
- **description** (String): Description of the object, a unicode string
- **location** (StringChoice): Location of the data on mesh, any of "vertices", "segments", "faces", "cells", "elements"
- **metadata** (*ArbitraryMetadataDict*): Attribute metadata, an arbitrary JSON-serializable dictionary, with certain keys validated against *AttributeMetadata*
- **name** (String): Title of the object, a unicode string
- **uid** (String): Unique identifier, a unicode string, Default: new instance of unicode

StringData

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

Data consisting of a list of strings or datetimes

Required Properties:

- **array** (*StringList*): String values at locations on a mesh (see location parameter); these values may be DateTimes or arbitrary strings, an instance of StringList
- **description** (String): Description of the object, a unicode string
- **location** (StringChoice): Location of the data on mesh, any of "vertices", "segments", "faces", "cells", "elements"
- **metadata** (*ArbitraryMetadataDict*): Attribute metadata, an arbitrary JSON-serializable dictionary, with certain keys validated against *AttributeMetadata*
- **name** (String): Title of the object, a unicode string
- **uid** (String): Unique identifier, a unicode string, Default: new instance of unicode

MappedData

class omf.data.**MappedData**(***kwargs*)
Data array of indices linked to legend values or -1 for no data

Required Properties:

- **array** (*Array*): indices into 1 or more legends for locations on a mesh, an instance of Array
- **description** (String): Description of the object, 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", "elements"
- **metadata** (*ArbitraryMetadataDict*): Attribute metadata, an arbitrary JSON-serializable dictionary, with certain keys validated against *AttributeMetadata*
- **name** (String): Title of the object, a unicode string
- **uid** (String): Unique identifier, a unicode string, Default: new instance of unicode

Legend

class omf.data.**Legend**(***kwargs*)
Legends to be used with MappedData indices

Required Properties:

- **description** (String): Description of the object, a unicode string
- **metadata** (*ArbitraryMetadataDict*): Basic object metadata, an arbitrary JSON-serializable dictionary, with certain keys validated against *BaseMetadata*
- **name** (String): Title of the object, a unicode string
- **uid** (String): Unique identifier, a unicode string, Default: new instance of unicode
- **values** (a list of Color, a list of String, a list of Integer, a list of Float): values for mapping indexed data, a list (each item is a color) or a list (each item is a unicode string) or a list (each item is an integer) or a list (each item is a float)

Colormap

class omf.data.**Colormap**(***kwargs*)
Length-128 color gradient with min/max values, used with ScalarData

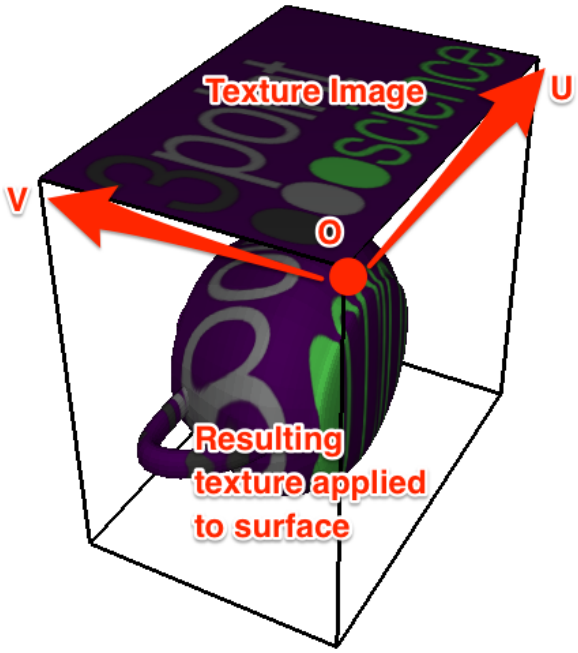
Required Properties:

- **description** (String): Description of the object, a unicode string
- **gradient** (*Array*): N x 3 Array of RGB values between 0 and 255 which defines the color gradient, an instance of Array
- **limits** (a list of Float): Data range associated with the gradient, a list (each item is a float) with length of 2
- **metadata** (*ArbitraryMetadataDict*): Basic object metadata, an arbitrary JSON-serializable dictionary, with certain keys validated against *BaseMetadata*
- **name** (String): Title of the object, a unicode string

- **uid** (`String`): Unique identifier, a unicode string, Default: new instance of unicode

7.1.8 Projected Texture

Projected 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.ProjectedTexture(
       origin=[0.0, 0.0, 0.0],
       axis_u=[1.0, 0.0, 0.0],
       axis_v=[0.0, 1.0, 0.0],
```

(continues on next page)

(continued from previous page)

```
        image='image1.png'
    )
>> my_tex_2 = omf.ProjectedTexture(
        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,
        my_tex_2
    ]
```

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

Contains an image that can be projected onto 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 of the object, a unicode string
- **image** (ImagePNG): PNG image file, a PNG image file, valid modes include (u'ab+', u'rb+', u'wb+', u'rb')
- **metadata** (*ArbitraryMetadataDict*): Basic object metadata, an arbitrary JSON-serializable dictionary, with certain keys validated against *BaseMetadata*
- **name** (String): Title of the object, 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]
- **uid** (String): Unique identifier, a unicode string, Default: new instance of unicode

7.1.9 UV Mapped Textures

Rather than being projected onto points or a surface, UV Mapped Textures are given normalized UV coordinates which correspond to element vertices. This allows arbitrary mapping of images to surfaces.

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

Contains an image that is UV mapped to a geometry

Required Properties:

- **description** (String): Description of the object, a unicode string
- **image** (ImagePNG): PNG image file, a PNG image file, valid modes include (u'ab+', u'rb+', u'wb+', u'rb')
- **metadata** (*ArbitraryMetadataDict*): Basic object metadata, an arbitrary JSON-serializable dictionary, with certain keys validated against *BaseMetadata*
- **name** (String): Title of the object, a unicode string
- **uid** (String): Unique identifier, a unicode string, Default: new instance of unicode

- **uv_coordinates** (*Array*): Normalized UV coordinates mapping the image to element vertices; for values outside 0-1 the texture repeats at every integer level, and NaN indicates no texture at a vertex, an instance of Array

7.1.10 Array Types

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

Array

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

Class with unique ID and data array

Required Properties:

- **array** (Array): Shared Scalar Array, a list or numpy array of <type 'int'>, <type 'float'>, <type 'bool'> with shape (*) or (*, *)
- **uid** (String): Unique identifier, a unicode string, Default: new instance of unicode

Other Properties:

- **datatype** (dynamic StringChoice): Array data type string, any of "Int32Array", "Int64Array", "Uint32Array", "BooleanArray", "Int8Array", "Int16Array", "Uint8Array", "Uint16Array", "Float64Array", "Uint64Array", "Float32Array" created dynamically
- **shape** (dynamic a list of Integer): Shape of the array, a list (each item is an integer) created dynamically
- **size** (dynamic Integer): Size of array in bits, an integer created dynamically

StringList

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

Array-like class with unique ID and string-list array

Required Properties:

- **array** (a list of DateTime, a list of String): List of datetimes or strings, a list (each item is a datetime object) or a list (each item is a unicode string)
- **uid** (String): Unique identifier, a unicode string, Default: new instance of unicode

Other Properties:

- **datatype** (dynamic StringChoice): List data type string, either "DateTimeArray" or "StringArray" created dynamically
- **shape** (dynamic a list of Integer): Shape of the string list, a list (each item is an integer) created dynamically
- **size** (dynamic Integer): Size of string list dumped to JSON in bits, an integer created dynamically

ArrayInstanceProperty

class omf.data.**ArrayInstanceProperty**(*doc*, ***kwargs*)

Instance property for OMF Array objects

This adds additional shape and dtype validation.

7.1.11 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 geometric definition. ProjectElements include PointSet, LineSet, Surface, and Volume

Required Properties:

- **description** (String): Description of the object, a unicode string
- **metadata** (*ArbitraryMetadataDict*): Element metadata, an arbitrary JSON-serializable dictionary, with certain keys validated against *ElementMetadata*
- **name** (String): Title of the object, a unicode string
- **uid** (String): Unique identifier, a unicode string, Default: new instance of unicode

Optional Properties:

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

ProjectElement Data

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

Data array with values at specific locations on the mesh

Required Properties:

- **description** (String): Description of the object, a unicode string
- **location** (StringChoice): Location of the data on mesh, any of "vertices", "segments", "faces", "cells", "elements"
- **metadata** (*ArbitraryMetadataDict*): Attribute metadata, an arbitrary JSON-serializable dictionary, with certain keys validated against *AttributeMetadata*
- **name** (String): Title of the object, a unicode string
- **uid** (String): Unique identifier, a unicode string, Default: new instance of unicode

Content Model

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

ContentModel is a UidModel with name, description, and metadata

Required Properties:

- **description** (String): Description of the object, a unicode string
- **metadata** (*ArbitraryMetadataDict*): Basic object metadata, an arbitrary JSON-serializable dictionary, with certain keys validated against *BaseMetadata*
- **name** (String): Title of the object, a unicode string
- **uid** (String): Unique identifier, a unicode string, Default: new instance of unicode

Uid Model

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

class omf.base.**UidModel**(***kwargs*)
UidModel is a HasProperties object with uid

Required Properties:

- **uid** (String): Unique identifier, a unicode string, Default: new instance of unicode

Metadata Classes

class omf.base.**ProjectMetadata**(***kwargs*)
Validated metadata properties for Projects

Optional Properties:

- **author** (String): Author of the project, a unicode string
- **coordinate_reference_system** (String): EPSG or Proj4 plus optional local transformation string, a unicode string
- **date** (*StringDateTime*): Date associated with the project data, a datetime object
- **date_created** (*StringDateTime*): Date object was created, a datetime object
- **date_modified** (*StringDateTime*): Date object was modified, a datetime object
- **revision** (String): Revision, a unicode string

class omf.base.**ElementMetadata**(***kwargs*)
Validated metadata properties for Elements

Optional Properties:

- **color** (Color): Solid element color, a color
- **coordinate_reference_system** (String): EPSG or Proj4 plus optional local transformation string, a unicode string
- **date_created** (*StringDateTime*): Date object was created, a datetime object
- **date_modified** (*StringDateTime*): Date object was modified, a datetime object
- **opacity** (Float): Element opacity, a float in range [0, 1]

class omf.base.**AttributeMetadata**(***kwargs*)
Validated metadata properties for Attributes

Optional Properties:

- **date_created** (*StringDateTime*): Date object was created, a datetime object
- **date_modified** (*StringDateTime*): Date object was modified, a datetime object
- **units** (String): Units of attribute values, a unicode string

class omf.base.**BaseMetadata**(***kwargs*)
Validated metadata properties for all objects

Optional Properties:

- **date_created** (*StringDateTime*): Date object was created, a datetime object
- **date_modified** (*StringDateTime*): Date object was modified, a datetime object

class omf.base.**ArbitraryMetadataDict**(*doc*, *metadata_class*, ***kwargs*)
Custom property class for metadata dictionaries

This property accepts JSON-compatible dictionary with any arbitrary fields. However, an additional `metadata_class` is specified to validate specific fields.

class omf.base.**StringDateTime**(*doc*, ***kwargs*)
DateTime property validated to be a string

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

(continues on next page)

(continued from previous page)

```
)

lin = omf.LineSetElement(
    name='Random Line',
    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',
    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.SurfaceGridElement(
    name='gridsurf',
    tensor_u=np.ones(10).astype(float),
    tensor_v=np.ones(15).astype(float),
    origin=[50., 50., 50.],
    axis_u=[1., 0, 0],
    axis_v=[0, 0, 1.],
    offset_w=np.random.rand(11*16),
    data=[
        omf.ScalarData(
            name='rand vert data',
            array=np.random.rand(11*16),
            location='vertices',
        ),
        omf.ScalarData(
            name='rand face data',
            array=np.random.rand(10*15),
```

(continues on next page)

(continued from previous page)

```
            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.VolumeGridElement(
    name='vol',
    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)
        ),
    ],
)

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

assert proj.validate()

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

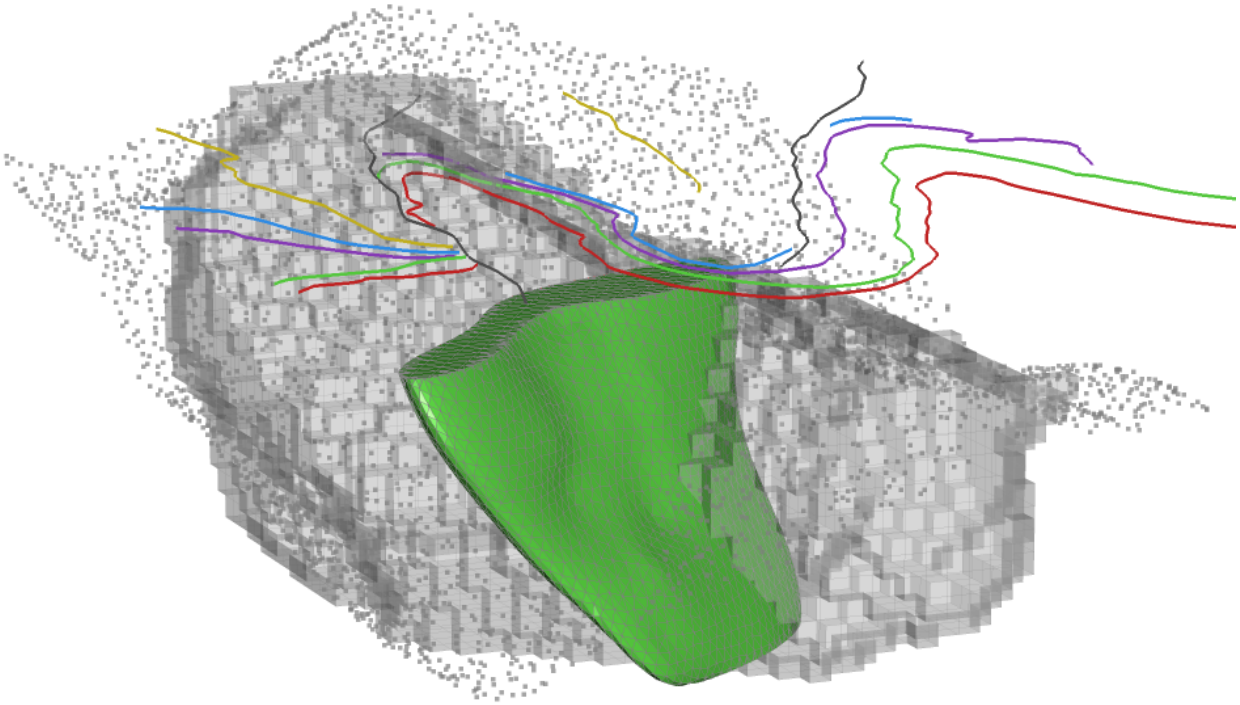
Piecewise building example:

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

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:

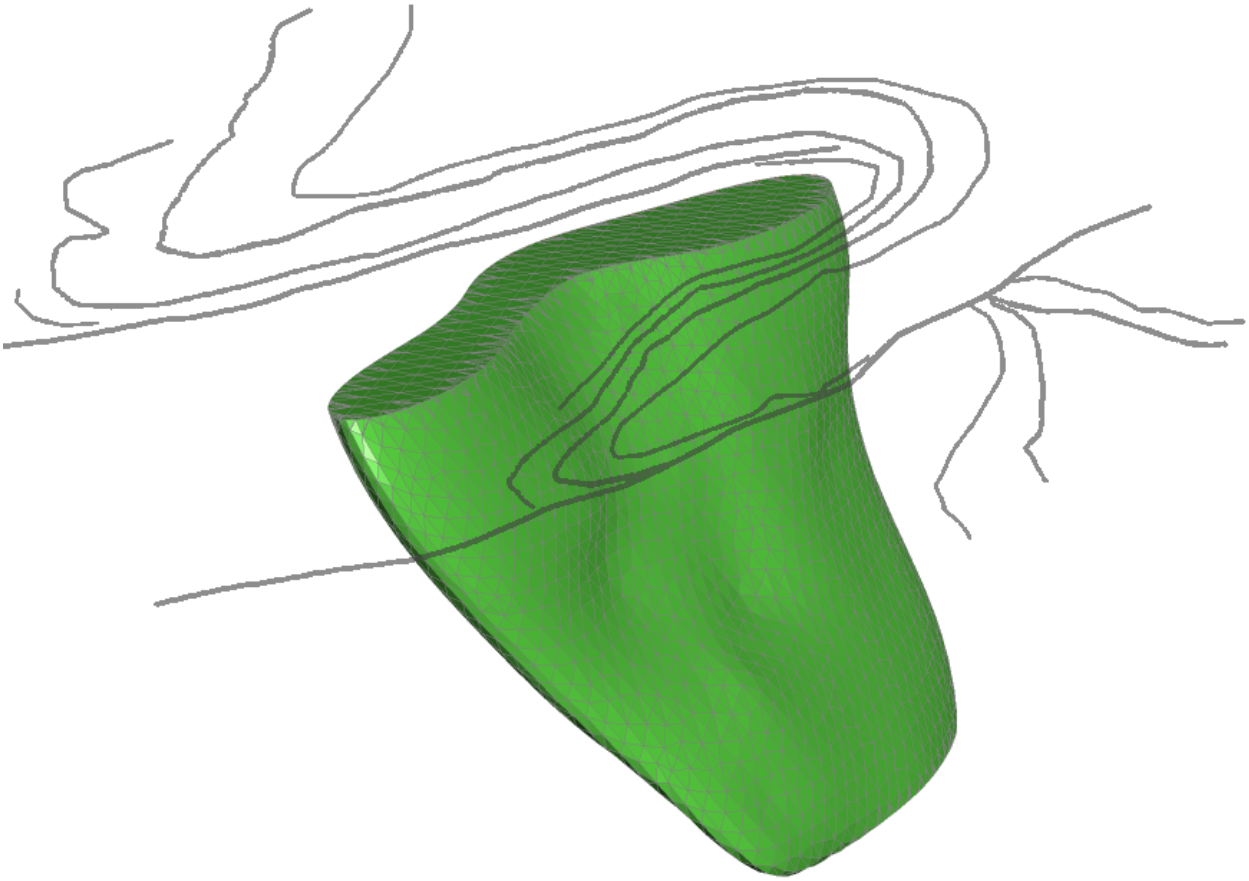
- 4 byte magic number: `b'\x81\x82\x83\x84'`
- 32 byte version string: `'OMF-v0.9.0'` (other bytes empty)
- 16 byte project uid (in little-endian bytes)
- 8 byte unsigned long long (little-endian): JSON start location in file

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