geoh5py Documentation

Release 0.1.5

MiraGeoscience

Mar 14, 2022

CONTENTS

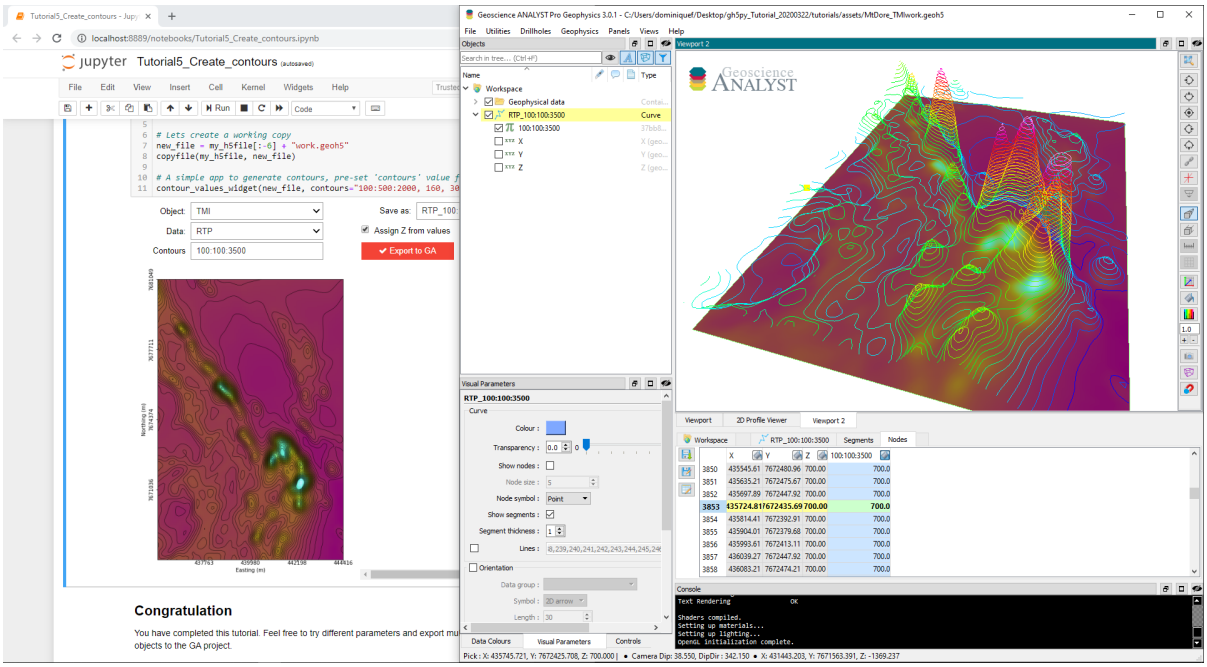
Welcome to the documentation page for **geoh5py**!

CHAPTER

ONE

IN SHORT

The **geoh5py** library has been created for the manipulation and storage of a wide range of geoscientific data (points, curve, surface, 2D and 3D grids) in geoh5 file format. Users will be able to directly leverage the powerful visualization capabilities of Geoscience ANALYST along with open-source code from the Python ecosystem.



CHAPTER

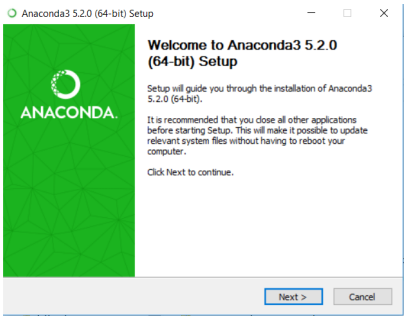
TWO

CONTENTS:

2.1 Installation

geoh5py is currently written for Python 3.7 or higher, and depends on NumPy and h5py.

Note: Users will likely want to take advantage of other packages available in the Python ecosystem. We therefore recommend using Anaconda to manage the installation.



Install **geoh5py** from PyPI:

```
$ pip install geoh5py
```

To install the latest development version of **geoh5py**, you can use `pip` with the latest GitHub `development` branch:

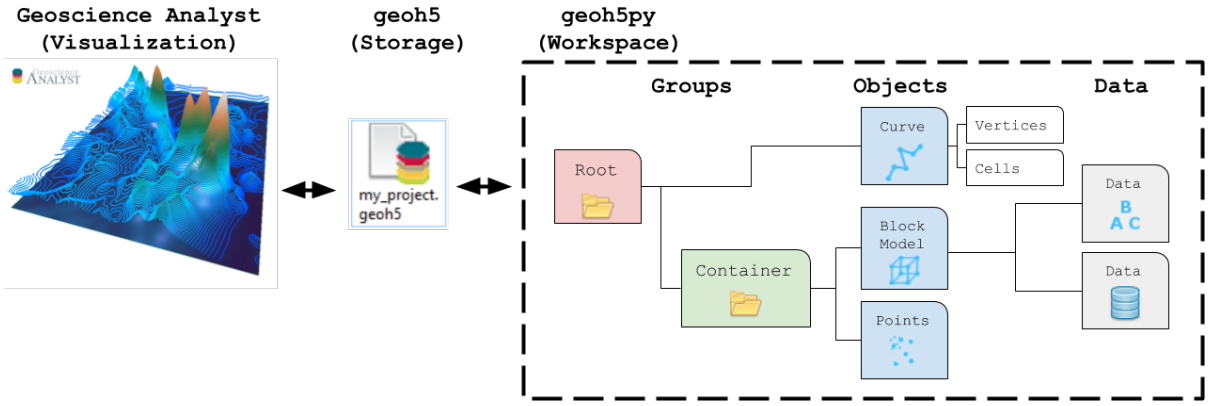
```
$ pip install git+https://github.com/MiraGeoscience/geoh5py.git
```

To work with **geoh5py** source code in development, install from GitHub:

```
$ git clone --recursive https://github.com/MiraGeoscience/geoh5py.git
$ cd geoh5py
$ python setup.py install
```

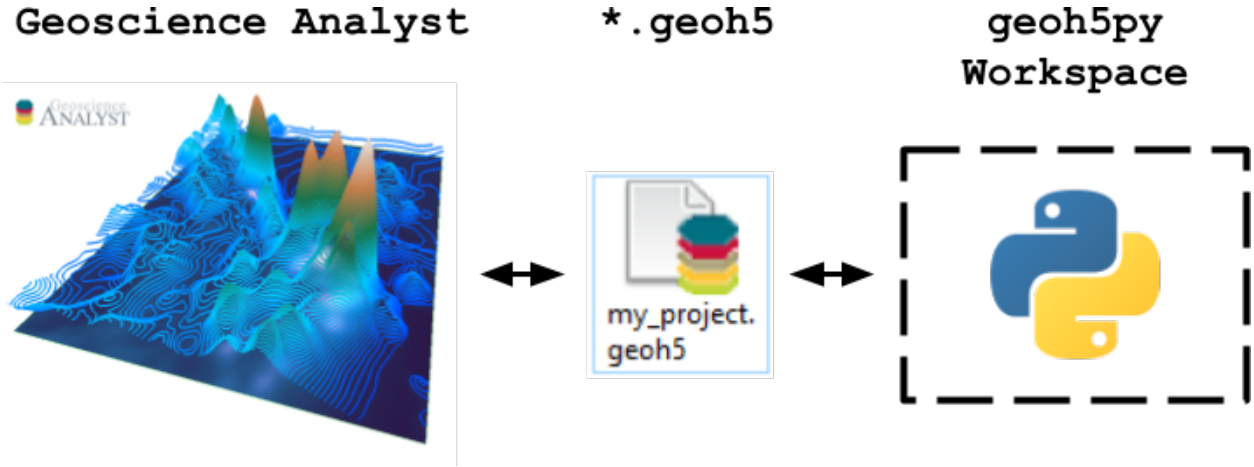
2.2 User Guide

This section provides information on how to use the **geoh5py** package, from the creation of a *Workspace* to the creation and manipulation of *Entities*



2.2.1 Workspace

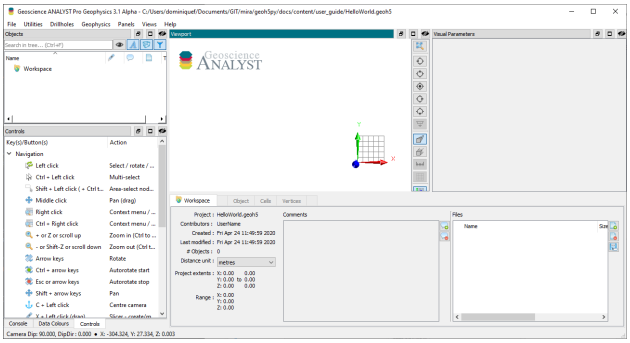
The core element of a project is the `Workspace`. A project Workspace holds core information about the author, version and all entities stored in the `geoh5` file. It also knows how to create the core structure needed by Geoscience ANALYST for visualization.



You can either open an existing project or create a new project by simply entering the desired file name.

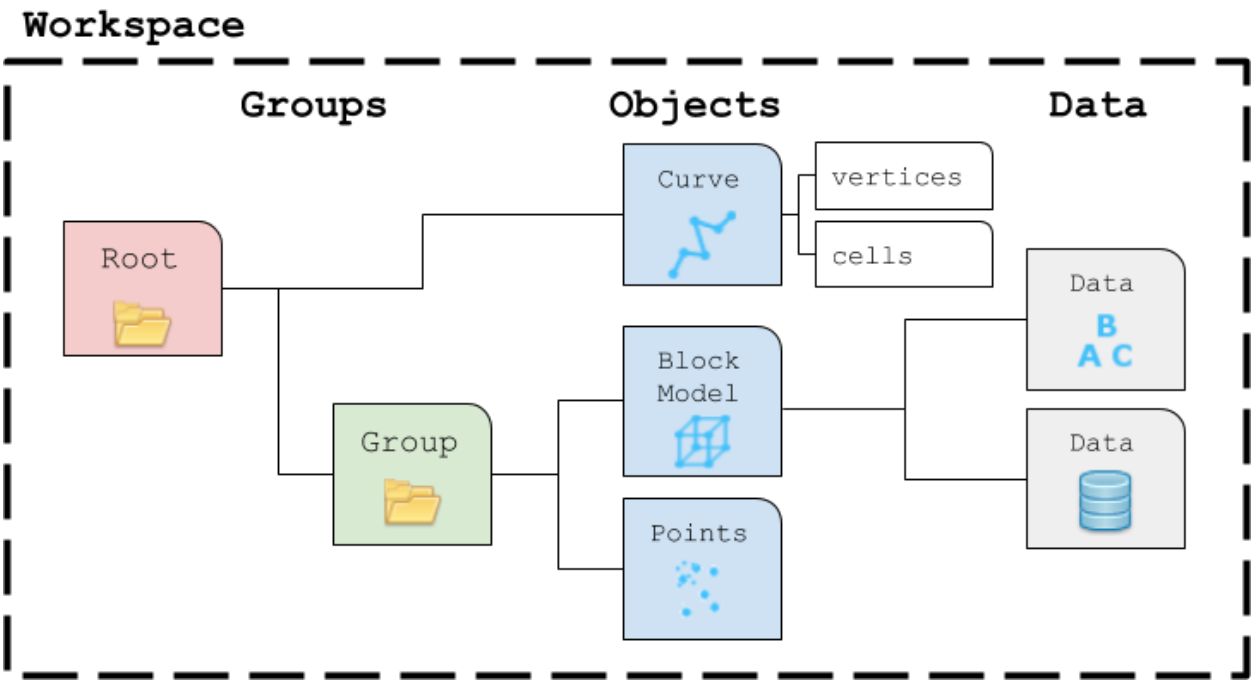
[1]:
```
from geoh5py.workspace import Workspace

# Create a new project
workspace = Workspace("my_project.geoh5")
```

Et voila!



2.2.2 Entities

This section introduces the different entities that can be created and stored in the `geoh5` file format.

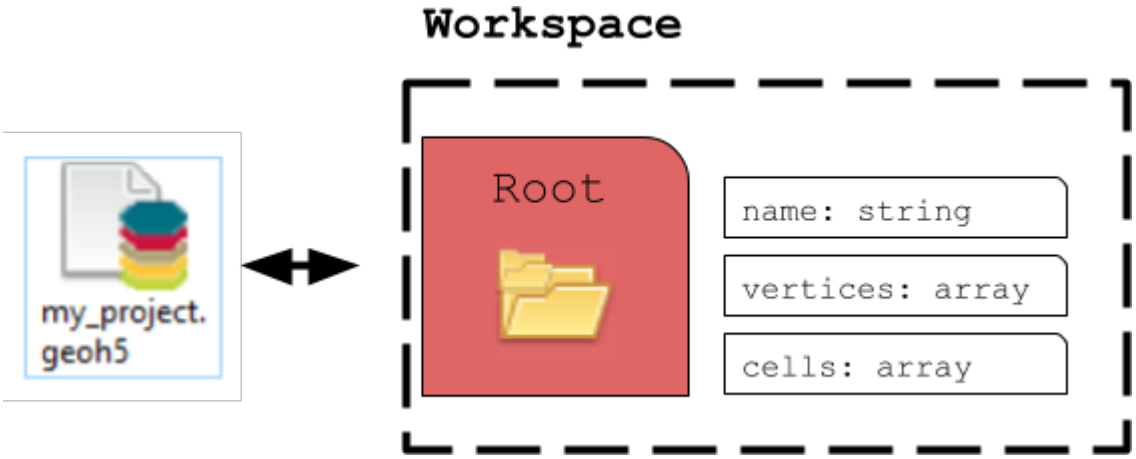


Groups

`Groups` are effectively containers for other entities, such as `Objects` (Points, Curve, Surface, etc.) and other `Groups`. Groups are used to establish `parent-child` relationships and to store information about a collection of entities.

RootGroup

By default, the parent of any new `Entity` is the workspace `RootGroup`. It is the only entity in the `Workspace` without a parent. Users rarely have to interect with the `Root` group as it is mainly used to maintain the overall project hierarchy.

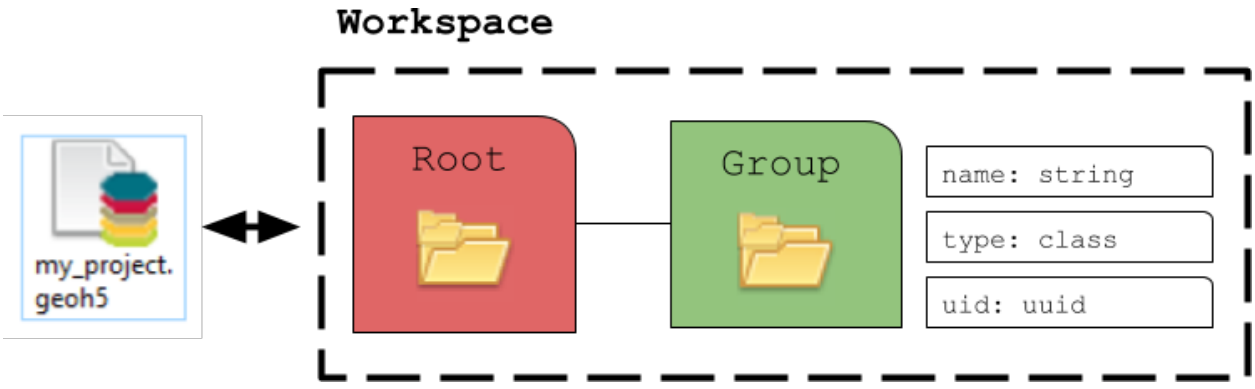


ContainerGroup

A `ContainerGroup` can easily be added to the workspace and can be assigned a `name` and `description`.

[1]:
```
from geoh5py.groups import ContainerGroup
from geoh5py.workspace import Workspace

# Create a blank project
workspace = Workspace("my_project.geoh5")

# Add a group
group = ContainerGroup.create(workspace, name='myGroup')
```

At creation, "`myGroup`" is written to the project `geoh5` file and visible in the Analyst project tree.



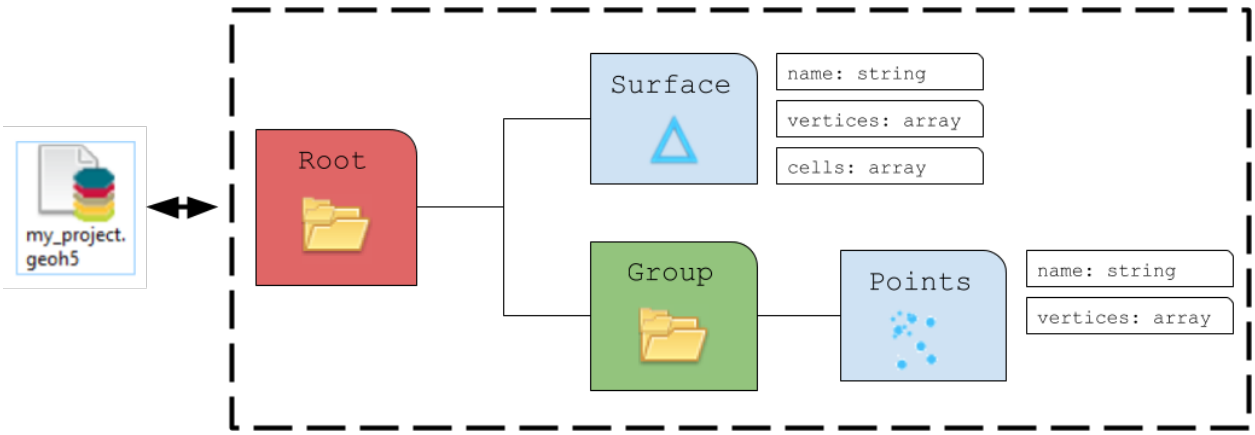
Any entity can be accessed by its `name` or `uid` (unique identifier):

[2]:
```
print(group.uid)
print(workspace.get_entity("myGroup")[0] == workspace.get_entity(group.uid)[0])
```

```
a1e952c9-f520-45a7-81f6-986466bef3b1
True
```

Objects

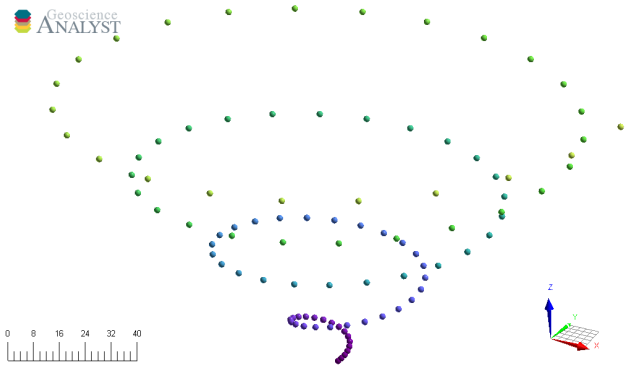
The `geoh5` format enables storing a wide variety of `Object` entities that can be displayed in 3D. This section describes the collection of `Objects` entities currently supported by `geoh5py`.



Points

The `Points` object consists of a list of `vertices` that define the location of actual data in 3D space. As for all other `Objects`, it can be created from an array of 3D coordinates and added to any group as follow:

[3]:
```
from geoh5py.workspace import Workspace
from geoh5py.objects import Points
import numpy as np

# Create a blank project
workspace = Workspace("my_project.geoh5")

# Generate a numpy array of xyz locations
n = 100
radius, theta = np.arange(n), np.linspace(0, np.pi*8, n)

x, y = radius * np.cos(theta), radius * np.sin(theta)
z = (x**2. + y**2.)**0.5
xyz = np.c_[x.ravel(), y.ravel(), z.ravel()] # Form a 2D array

# Create the Point object
points = Points.create(
    workspace,       # The target Workspace
    vertices=xyz     # Set vertices
)
```



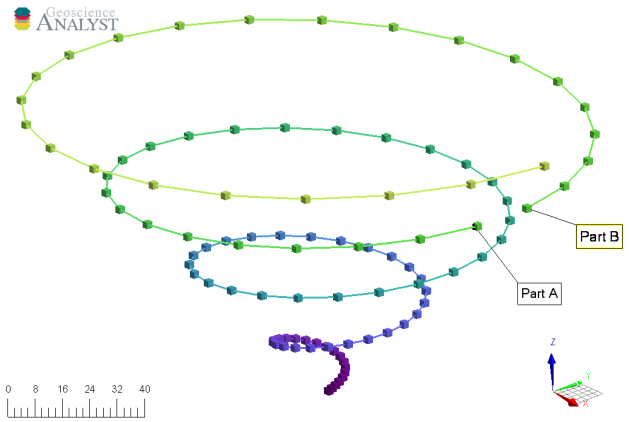
Curve

The `Curve` object, also known as a polyline, is often used to define contours, survey lines or geological contacts. It is a sub-class of the `Points` object with the added `cells` property, that defines the line segments connecting its `vertices`. By default, all vertices are connected sequentially following the order of the input `vertices`.

[4]:
```
from geoh5py.objects import Curve

# Create the Curve object
curve = Curve.create(
    workspace,       # The target Workspace
    vertices=xyz
)
```

Alternatively, the `cells` property can be modified, either directly or by assigning `parts` identification to each `vertices`:

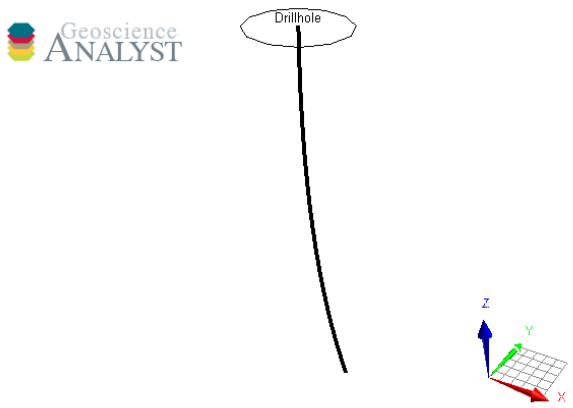
[5]:
```
# Split the curve into two parts
part_id = np.ones(n, dtype="int32")
part_id[:75] = 2

# Assign the part
curve.parts = part_id
workspace.finalize()
```



Drillhole

`Drillhole` objects are different from other objects as their 3D geometry is defined by the `collar` and `surveys` attributes. The `vertices` and `cells` properties are only instantiated when *interval or point log data* are added.

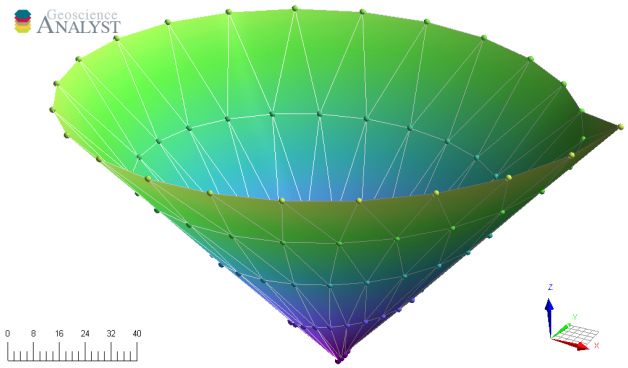
[6]:
```
from geoh5py.objects import Drillhole

# Create a simple well
total_depth = 100
dist = np.linspace(0, total_depth, 10)
azm = np.ones_like(dist) * 45.
dip = np.linspace(-89, -75, dist.shape[0])
collar = np.r_[0., 10., 10]

well = Drillhole.create(
    workspace, collar=collar, surveys=np.c_[dist, dip, azm]
)
```



Surface

The `Surface` object is also described by `vertices` and `cells` that form a net of triangles. If omitted on creation, the `cells` property is calculated using a 2D `scipy.spatial.Delaunay` triangulation.

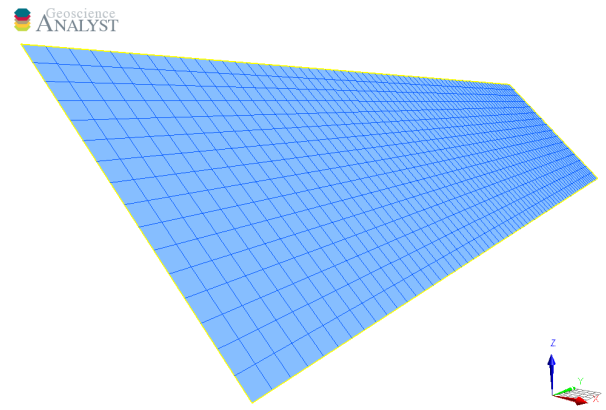
[7]:
```
from geoh5py.objects import Surface
from scipy.spatial import Delaunay

# Create a triangulated surface from points
surf_2D = Delaunay(xyz[:, :2])

# Create the Surface object
surface = Surface.create(
    workspace,
    vertices=points.vertices, # Add vertices
    cells=surf_2D.simplices
)
```



Grid2D

The `Grid2D` object defines a regular grid of `cells` often used to display model sections or to compute data derivatives. A `Grid2D` can be oriented in 3D space using the `origin`, `rotation` and `dip` parameters.

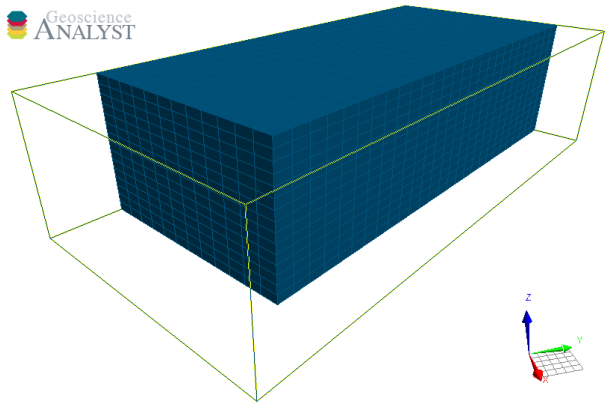
[8]:
```
from geoh5py.objects import Grid2D

# Create the Surface object
grid = Grid2D.create(
    workspace,
    origin = [25, -75, 50],
    u_cell_size = 2.5,
    v_cell_size = 2.5,
    u_count = 64,
    v_count = 16,
    rotation = 90.0,
    dip = 45.0,
)
```



BlockModel

The `BlockModel` object defines a rectilinear grid of cells, also known as a tensor mesh. The `cells` center position is determined by `cell_delimiters` (offsets) along perpendicular axes (`u`, `v`, `z`) and relative to the `origin`. `BlockModel` can be oriented horizontally by controlling the `rotation` parameter.

[9]:
```
from geoh5py.objects import BlockModel

# Create the Surface object
blockmodel = BlockModel.create(
    workspace,
    origin = [25, -100, 50],
    u_cell_delimiters=np.cumsum(np.ones(16) * 5), # Offsets along u
    v_cell_delimiters=np.cumsum(np.ones(32) * 5), # Offsets along v
    z_cell_delimiters=np.cumsum(np.ones(16) * -2.5),  # Offsets along z (down)
    rotation = 30.0
)
```

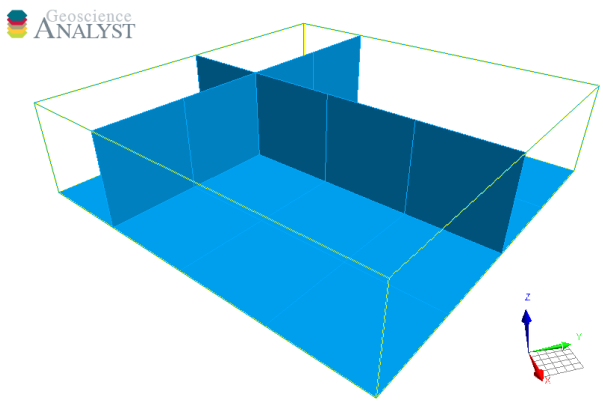


Octree

The `Octree` object is type of 3D grid that uses a tree structure to define `cells`. Each cell can be subdivided it into eight octants allowing for a more efficient local refinement of the mesh. The `Octree` object can also be oriented horizontally by controlling the `rotation` parameter.

[10]:
```
from geoh5py.objects import Octree

octree = Octree.create(
        workspace,
        origin=[25, -100, 50],
        u_count=16,       # Number of cells in power 2
        v_count=32,
        w_count=16,
        u_cell_size=5.0, # Base cell size (highest octree level)
        v_cell_size=5.0,
        w_cell_size=2.5, # Offsets along z (down)
        rotation=30,
)
```

By default, the octree mesh will be refined at the lowest level possible along each axes.



Data

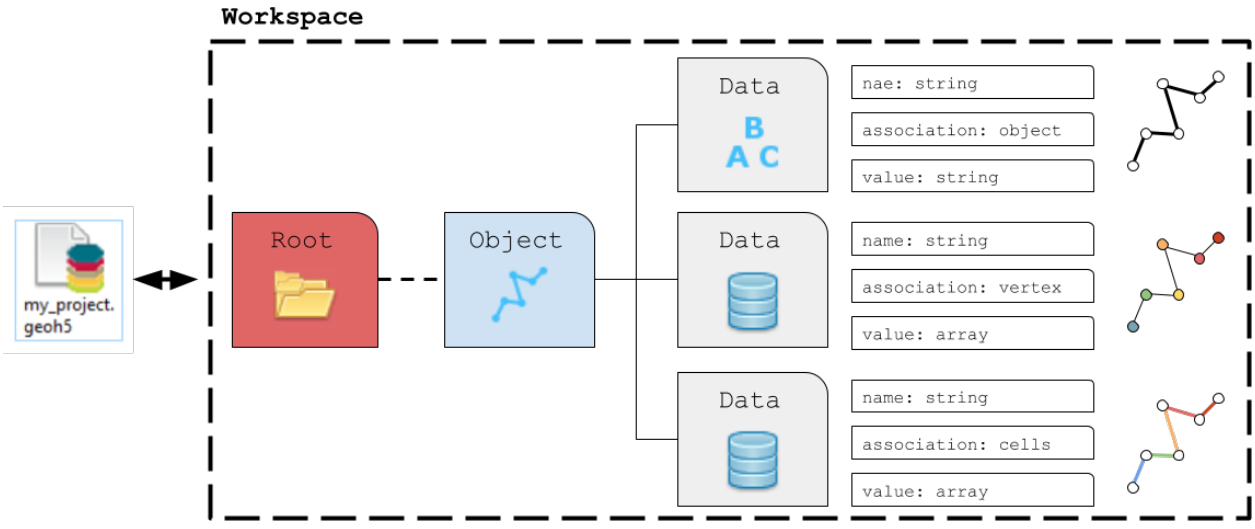
The `geoh5` format allows storing data (values) on different parts of an `Object`. The `data_association` can be one of:

- OBJECT: Single element characterizing the parent object
- VERTEX: Array of values associated with the parent object vertices
- CELL: Array of values associated with the parent object cells

Note: The length and order of the array provided must be consistent with the corresponding element of association.

The data types supported by `geoh5py` are:

- Arrays
- Integer
- Text
- Color_map



Add data

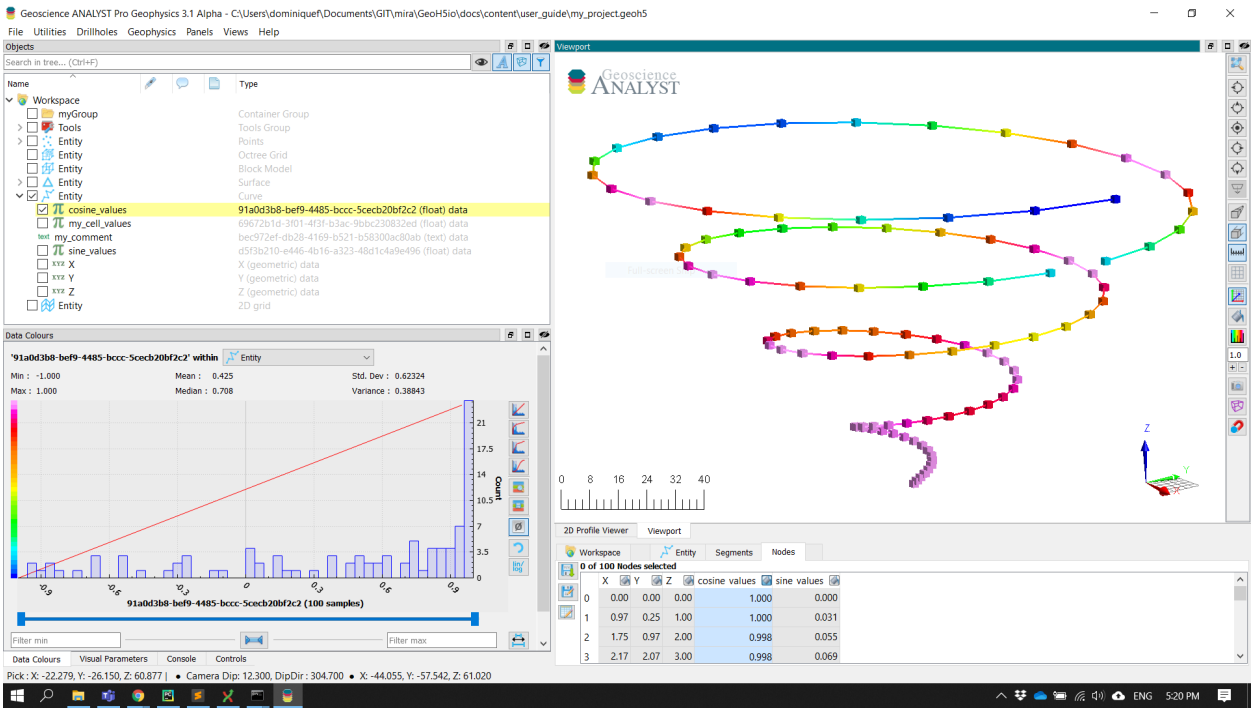
Data can be added to an `Object` entity using the `add_data` method.

[11]:
```
# Create a straight Curve object
curve = Curve.create(
    workspace,       # The target Workspace
    name='FlightLine3',
    vertices=np.c_[np.linspace(0, 100, 100), np.zeros(100), np.zeros(100)]
)

# Add a single string comment
curve.add_data({
    "my_comment": {
        "association":"OBJECT",
        "values": "hello_world"
    }
})

# Add a vector of floats
curve.add_data({
    "my_cell_values": {
        "association":"CELL",
        "values": np.random.randn(curve.n_cells)
    }
})

# Add multiple data vectors on a single call
data = {}
for ii in range(8):
    data[f"Period:{ii}"] = {
        "association":"VERTEX",
        "values": (ii+1) * np.cos(ii*curve.vertices[:, 0]*np.pi/curve.vertices[:, 0].
↪max()/4.)
    }

data_list = curve.add_data(data)
print([obj.name for obj in data_list])
```

```
['Period:0', 'Period:1', 'Period:2', 'Period:3', 'Period:4', 'Period:5', 'Period:6',
↪'Period:7']
```

If the `association` argument is omited, `geoh5py` will attempt to assign the data to the correct part based on the shape of the data values, either `object.n_values` or `object.n_cells`

The newly created data is directly added to the project's `geoh5` file and available for visualization:



Get data

Just like any `Entity`, data can be retrieved from the `Workspace` using the `get_entity` method. For convenience, `Objects` also have a `get_data_list` and `get_data` method that focusses only on their respective children `Data`.

[12]:
```
my_list = curve.get_data_list()
print(my_list, curve.get_data(my_list[0]))
```

```
['Period:0', 'Period:1', 'Period:2', 'Period:3', 'Period:4', 'Period:5', 'Period:6',
↪'Period:7', 'my_cell_values', 'my_comment'] [<geoh5py.data.float_data.FloatData object␣
↪at 0x7f47f2bd0c50>]
```

Well Data

In the case of *Drillhole* objects, data are added as either `interval log` or `point log` values.

Point Log Data

Log data are used to represent measurements recorded at discrete depths along the well path. A `depth` attribute is required on creation. If the `Drillhole` object already holds point log data, `geoh5py` will attempt to match collocated depths within tolerance. By default, depth markers within 1 centimeter are merged (`collocation_distance=1e-2`).

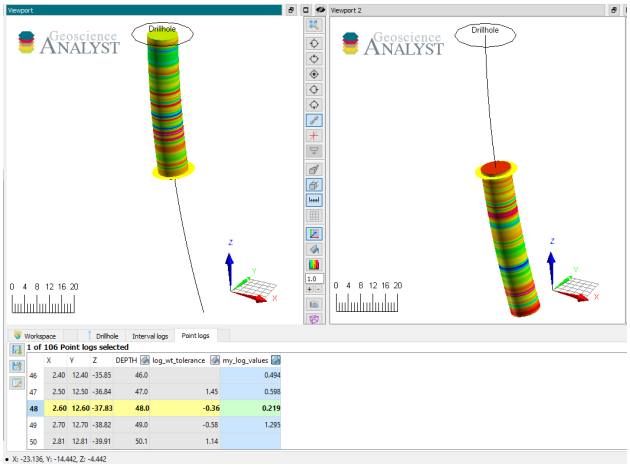
[13]:
```
depths_A = np.arange(0, 50.) # First list of depth

# Second list slightly offsetted on the first few depths
depths_B = np.arange(47.1, 100)
```

(continues on next page)

(continued from previous page)

```
# Add both set of log data with 0.5 m tolerance
well.add_data({
    "my_log_values": {
            "depth": depths_A,
            "values": np.random.randn(depths_A.shape[0]),
    },
    "log_wt_tolerance": {
            "depth": depths_B,
            "values": np.random.randn(depths_B.shape[0]),
            "collocation_distance": 0.5
    }
})
```

[13]:
```
[<geoh5py.data.float_data.FloatData at 0x7f482d66a590>,
 <geoh5py.data.float_data.FloatData at 0x7f47f2bd0c10>]
```



Interval Log Data

Interval log data are defined by constant values bounded by a start an end depth. A `from-to` attribute is expected on creation. Users can also control matching intervals by supplying a `tolerance` argument in meters (default `tolerance: 1e-3` meter).

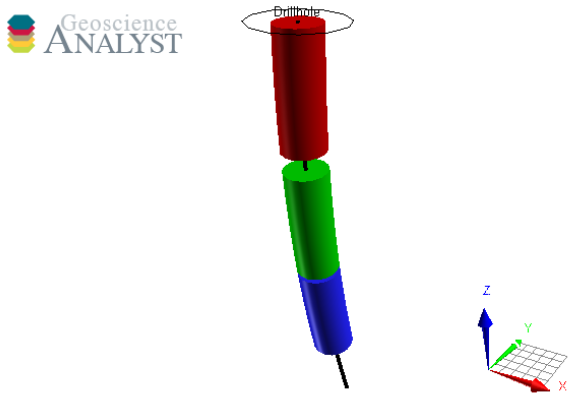
[14]:
```
# Add some geology as interval data
well.add_data({
    "interval_values": {
        "values": [1, 2, 3],
        "from-to": np.vstack([
            [0.25, 25.5],
            [30.1, 55.5],
            [56.5, 80.2]
        ]),
        "value_map": {
            1: "Unit_A",
            2: "Unit_B",
            3: "Unit_C"
        },
```

(continues on next page)

(continued from previous page)

```
        "type": "referenced",
    }
})
```

[14]:
```
<geoh5py.data.referenced_data.ReferencedData at 0x7f47f2b58b10>
```

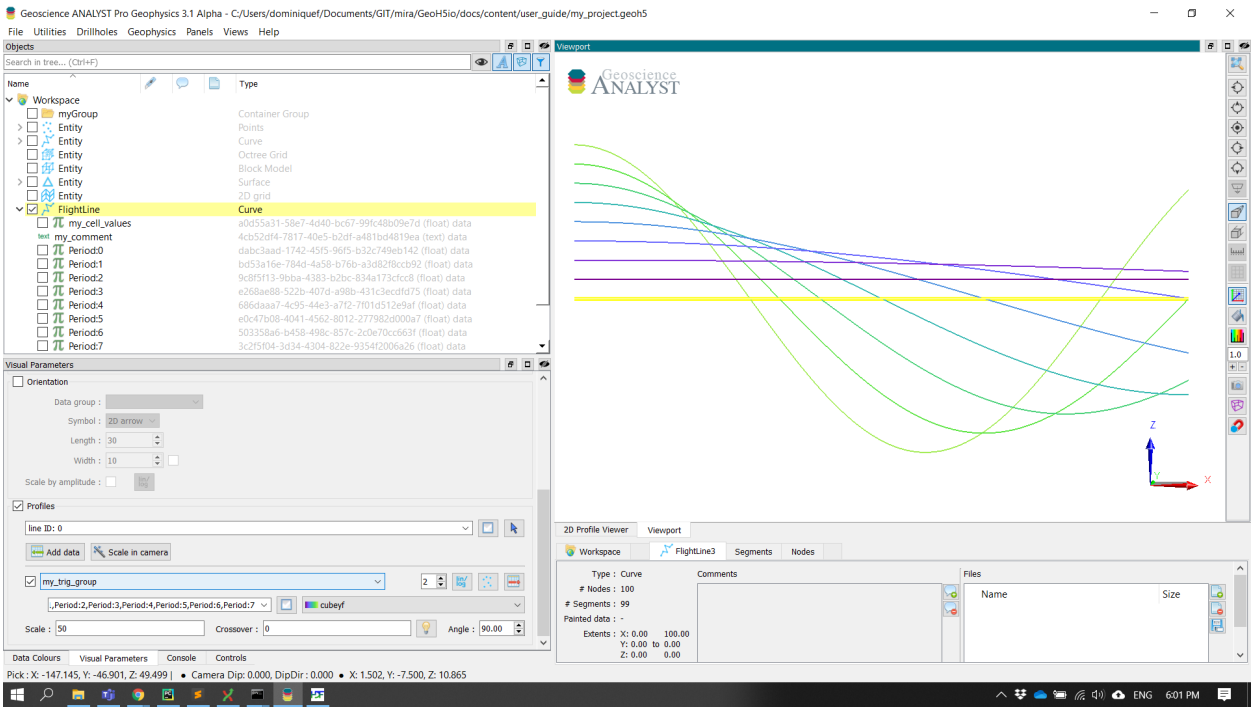


Property Groups

`Data` entities sharing the same parent `Object` and `association` can be linked within a `property_groups` and made available through profiling. This can be used to group data that would normally be stored as 2D array.

[15]:
```
# Add another VERTEX data and create a group with previous
curve.add_data_to_group([obj.name for obj in data_list], "my_trig_group")
```

[15]:
```
<geoh5py.groups.property_group.PropertyGroup at 0x7f47f2bc3710>
```

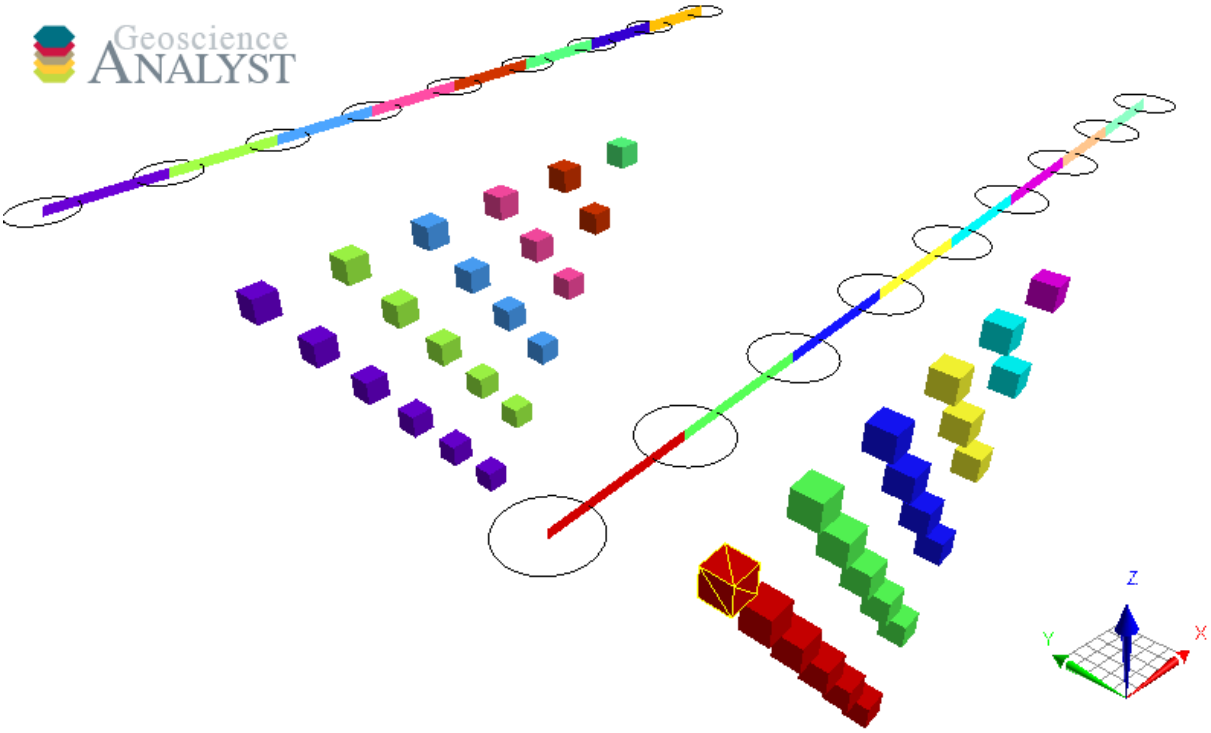


[16]:
```
# Update the geoh5 and re-write the Root
workspace.finalize()
```

2.2.3 Surveys

Direct Current and Induced Polarization (DC/IP)

This object is meant to handle direct-current resistivity surveys. The survey object is made up of two curve entities defining the transmitter (current) and receiver (potential) electrodes. The following example shows how to generate a survey from scratch.



Current Electrode (sources)

The `CurrentElectrode` entity defines the A-B dipole pairs used to inject current into the ground. It is a sub-class of the `Curve` object, defined by vertices (poles) and cells (dipole segments). Here we generate four (4) parallel EW lines with eight dipoles per line.

[1]:
```
import numpy as np
import uuid
from geoh5py.workspace import Workspace
from geoh5py.objects import CurrentElectrode, PotentialElectrode

# Create a new project
workspace = Workspace("my_project.geoh5")
```

(continues on next page)

(continued from previous page)

```
# Define the pole locations
n_poles = 9
n_lines = 2
x_loc, y_loc = np.meshgrid(np.linspace(0, 60, n_poles), np.linspace(-20, 20., n_lines))
vertices = np.c_[x_loc.ravel(), y_loc.ravel(), np.zeros_like(x_loc).ravel()]

# Assign a line ID to the poles (vertices)
parts = np.kron(np.arange(n_lines), np.ones(n_poles)).astype('int')

# Create the CurrentElectrode object
currents = CurrentElectrode.create(workspace, vertices=vertices, parts=parts)
```

At this stage the `CurrentElectrode` object has segments (`cells`) connecting all poles in series along line.

AB Cell ID

A key element of the DCIP survey objects is the `ab_cell_id` property. This `ReferenceData` contains the map referencing each cell of the `CurrentElectrode` object to a unique A-B source identifier with name.

currents.ab_cell_id.value_map.map = {
0: “Unknown”, 1: “AB_100”, 2: “AB_200”, … }

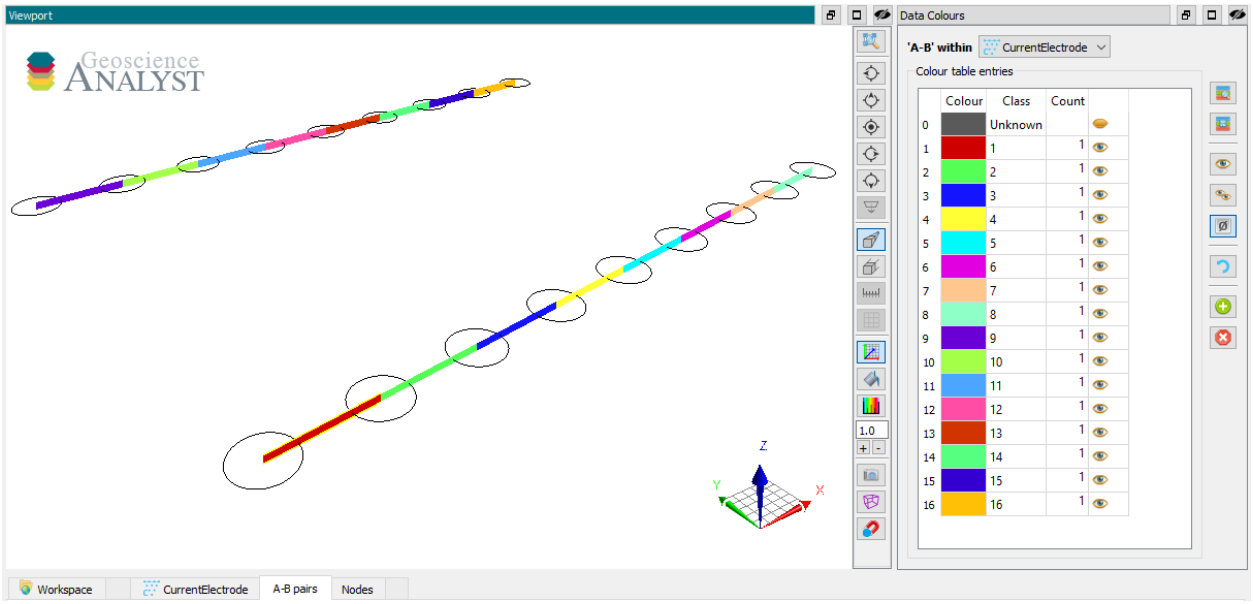
The utility function `add_default_ab_cell_id` can help generate this map with a simple name string incrementor.

[2]:
```
currents.add_default_ab_cell_id()
print(currents.ab_cell_id.values)
print(currents.ab_cell_id.value_map.map)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16]
{0: 'Unknown', 1: '1', 2: '2', 3: '3', 4: '4', 5: '5', 6: '6', 7: '7', 8: '8', 9: '9',␣
↪10: '10', 11: '11', 12: '12', 13: '13', 14: '14', 15: '15', 16: '16'}
```

Users may wish to alter the `value_map` names to carry forward specific survey information.

Note: The first entry `{0:Unknown}` is a reserved field used by Geoscience ANALYST to flag unknown data entries.



In this specific case, every `cell` on the curve corresponds to a unique dipole source current. For more complex survey configurations, users can edit the `cell` property in order to define different combinations of connections between poles.

Potential Electrode (receivers)

The `PotentialElectrode` object defines the M-N dipole pairs used to measure the electric potential (receivers). Just like the `CurrentElectrode`, it is a sub-class of the `Curve` object defined by vertices (poles) and cells (dipoles).

Although poles could be set independently on the `CurrentElectrode` and `PotentialElectrode` objects, here we re-uses the same locations for simplicity:

[3]:
```
potentials = PotentialElectrode.create(workspace, vertices=vertices)
```

The link between the sources `CurrentElectrode` and the receivers `PotentialElectrode` is established on creation with the `current_electrodes` argument.

The same can also be done after instantiation as

[4]:
```
potentials.current_electrodes = currents
```

or equivalently

[5]:
```
currents.potential_electrodes = potentials
```

In all the above cases, the link between the two objects gets encoded in their respective `metadata`.

[6]:
```
print(potentials.metadata == currents.metadata)
print(currents.metadata)
```

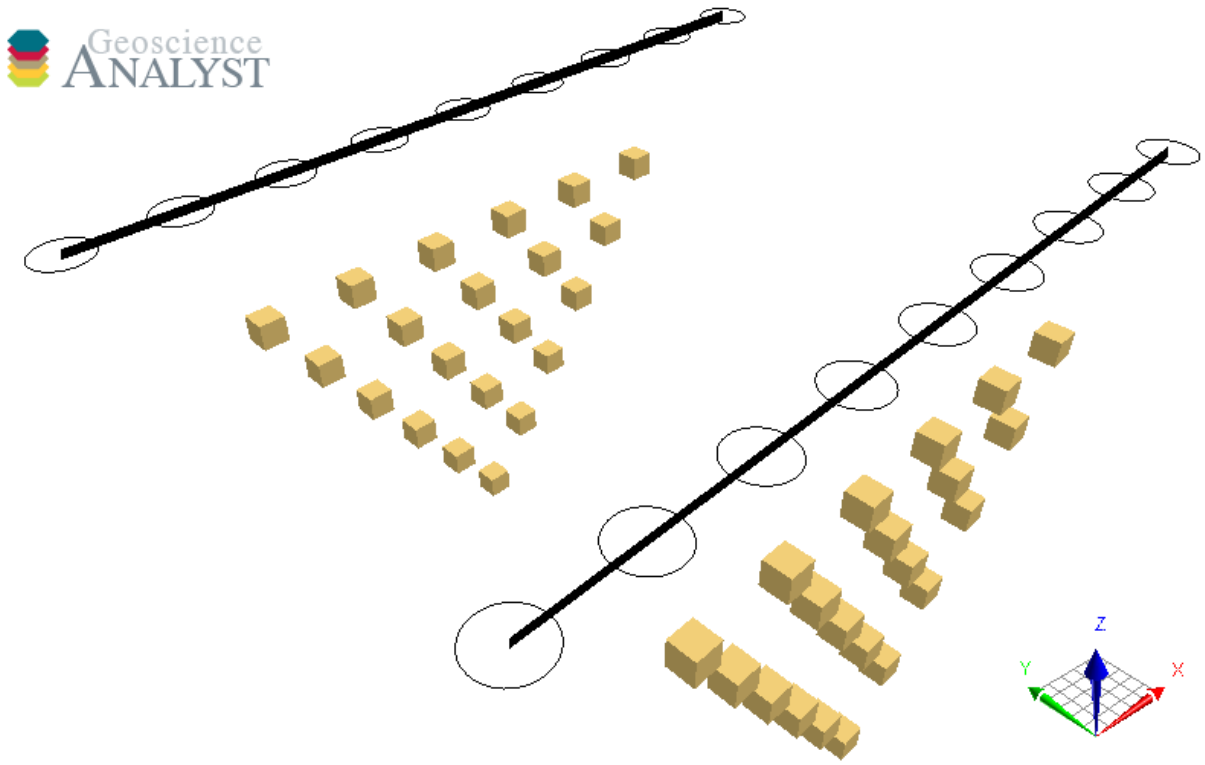
```
True
{'Current Electrodes': UUID('35fde588-45c1-47f3-9b92-b319e1f0c586'), 'Potential␣
↪Electrodes': UUID('28e6d74d-5952-4379-bb9e-90a261eced00')}
```

Next, we must define the receiver dipoles. The following routine generates a maximum of six (6) receivers dipoles per injection currents along line.

[7]:
```
N = 6
dipoles = []
current_id = []

for val in currents.ab_cell_id.values: # For each source dipole
    cell_id = int(currents.ab_map[val]) - 1 # Python 0 indexing
    line = currents.parts[currents.cells[cell_id, 0]]
    for m_n in range(N):
        dipole_ids = (currents.cells[cell_id, :] + 2 + m_n).astype("uint32") # Skip two␣
↪poles

        # Shorten the array as we get to the end of the line
        if (
            any(dipole_ids > (potentials.n_vertices - 1))
            or any(currents.parts[dipole_ids] != line)
        ):
            continue

        dipoles += [dipole_ids] # Save the receiver id
        current_id += [val] # Save the source id

potentials.cells = np.vstack(dipoles)
```

Finally, users need to create an association between each receiver dipole (M-N) to a dipole current (A-B). The mapping is done through the `ab_cell_id` property of the `PotentialElectrode`. An integer (ID) value must be assigned to each cell, corresponding to the *AB Cell ID* pairs stored on the associated `CurrentElectrode` object.

[8]:
```
potentials.ab_cell_id = np.asarray(current_id, dtype="int32")
```



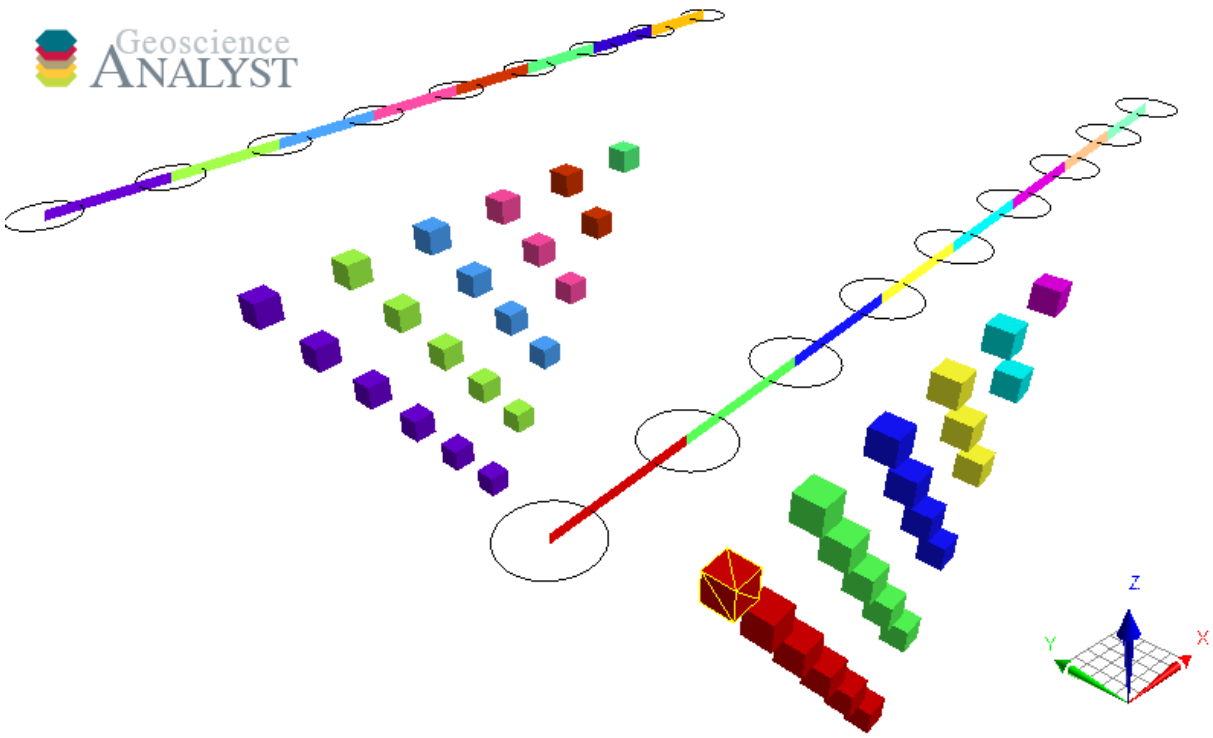
Note: The `ab_cell_id` property of the `CurrentElectrode` and `PotentialElectrode` are two different `ReferenceData` entities:

[9]:
```
print(potentials.ab_cell_id == currents.ab_cell_id)
```
```
False
```

but share the same `DataType` that holds the map of unique source dipoles.

[10]:
```
print(potentials.ab_cell_id.entity_type == currents.ab_cell_id.entity_type)
```
```
True
```

This link between `DataType` allows users to query the data by dipole sources and display the values as pseudo-section in Geoscience ANALYST.



2.3 geoh5py package

2.3.1 Subpackages

geoh5py.data package

Submodules

geoh5py.data.blob_data module

class geoh5py.data.blob_data.**BlobData**(*data_type:* geoh5py.data.data_type.DataType, ***kwargs*)

Bases: *geoh5py.data.data.Data*

classmethod primitive_type() → *geoh5py.data.primitive_type_enum.PrimitiveTypeEnum*

geoh5py.data.color_map module

class geoh5py.data.color_map.**ColorMap**(***kwargs*)

Bases: `object`

Records colors assigned to value ranges (where Value is the start of the range).

property name: str

`str`: Name of the colormap

property values: numpy.ndarray

`numpy.array`: Colormap defined by values and corresponding RGBA:

```
values = [
    [V_1, R_1, G_1, B_1, A_1],
    ..., [V_i, R_i, G_i, B_i, A_i]
]
```

where V (Values) are sorted floats defining the position of each RGBA. R (Red), G (Green), B (Blue) and A (Alpha) are integer values between [0, 255].

geoh5py.data.data module

class geoh5py.data.data.**Data**(*data_type:* geoh5py.data.data_type.DataType, ***kwargs*)

Bases: *geoh5py.shared.entity.Entity*

Base class for Data entities.

property association: *DataAssociationEnum* | None

DataAssociationEnum: Relationship made between the *values()* and elements of the *parent* object. Association can be set from a `str` chosen from the list of available *DataAssociationEnum* options.

property entity_type: *geoh5py.data.data_type.DataType*

DataType

classmethod find_or_create_type(*workspace:* workspace.Workspace, ***kwargs*) → *DataType*

Find or create a type for a given object class

Parameters **workspace** (*Current*) – Workspace

Returns A new or existing object type

property n_values: int | None

`int`: Number of expected data values based on *association*

abstract classmethod primitive_type() → *geoh5py.data.primitive_type_enum.PrimitiveTypeEnum*

property values

Data values

geoh5py.data.data_association_enum module

class geoh5py.data.data_association_enum.**DataAssociationEnum**(*value*)

Bases: `enum.Enum`

Known data association between *values* and the *parent* object. Available options:

CELL = 2

FACE = 4

GROUP = 5

OBJECT = 1

UNKNOWN = 0

VERTEX = 3

geoh5py.data.data_type module

class geoh5py.data.data_type.**DataType**(*workspace:* workspace.Workspace, ***kwargs*)
Bases: *geoh5py.shared.entity_type.EntityType*

DataType class

property color_map: *ColorMap* | None
ColorMap: Colormap used for plotting

The colormap can be set from a `dict` of sorted values with corresponding RGBA color.

```
color_map = {
    val_1: [r_1, g_1, b_1, a_1],
    ...,
    val_i: [r_i, g_i, b_i, a_i]
}
```

classmethod create(*workspace:* workspace.Workspace, *data_class: type[*data.Data*]*) → *DataType*
Creates a new instance of *DataType* with corresponding *PrimitiveTypeEnum*.

Parameters **data_class** – A *Data* implementation class.

Returns A new instance of *DataType*.

classmethod find_or_create(*workspace:* workspace.Workspace, ***kwargs*) → *DataType*
Find or creates an EntityType with given UUID that matches the given Group implementation class.

Parameters **workspace** – An active Workspace class

Returns A new instance of DataType.

classmethod for_x_data(*workspace:* workspace.Workspace) → *DataType*

classmethod for_y_data(*workspace:* workspace.Workspace) → *DataType*

classmethod for_z_data(*workspace:* workspace.Workspace) → *DataType*

property hidden: bool
`bool`: Hidden data [False]

property mapping: str
`str`: Color stretching type chosen from: 'linear', ['equal_area'], 'logarithmic', 'cdf', 'missing'

property number_of_bins: int | None
`int`: Number of bins used by the histogram [50]

property primitive_type: *PrimitiveTypeEnum* | None
PrimitiveTypeEnum

property transparent_no_data: bool
`bool`: Use transparent for no-data-value [True]

property units: str | None
`str`: Data units

property value_map: *ReferenceValueMap* | None
ReferenceValueMap: Reference value map for `ReferenceData`

The value_map can be set from a `dict` of sorted values with corresponding `str` description.

```
value_map = {
    val_1: str_1,
    ...,
    val_i: str_i
}
```

geoh5py.data.data_unit module

class geoh5py.data.data_unit.**DataUnit**(*unit_name: Optional[str] = None*)

Bases: `object`

Data unit

property name: str | None

geoh5py.data.datetime_data module

class geoh5py.data.datetime_data.**DatetimeData**(*data_type:* geoh5py.data.data_type.DataType, ***kwargs*)

Bases: *geoh5py.data.data.Data*

classmethod primitive_type() → *geoh5py.data.primitive_type_enum.PrimitiveTypeEnum*

geoh5py.data.filename_data module

class geoh5py.data.filename_data.**FilenameData**(*data_type:* geoh5py.data.data_type.DataType, ***kwargs*)

Bases: *geoh5py.data.data.Data*

classmethod primitive_type() → *geoh5py.data.primitive_type_enum.PrimitiveTypeEnum*

geoh5py.data.float_data module

class geoh5py.data.float_data.**FloatData**(*data_type:* geoh5py.data.data_type.DataType, ***kwargs*)

Bases: *geoh5py.data.numeric_data.NumericData*

Data container for floats values

classmethod ndv() → float

No-Data-Value

classmethod primitive_type() → *geoh5py.data.primitive_type_enum.PrimitiveTypeEnum*

geoh5py.data.geometric_data_constants module

class geoh5py.data.geometric_data_constants.**GeometricDataConstants**
Bases: object

classmethod primitive_type() → *geoh5py.data.primitive_type_enum.PrimitiveTypeEnum*

classmethod x_datatype_uid() → uuid.UUID

classmethod y_datatype_uid() → uuid.UUID

classmethod z_datatype_uid() → uuid.UUID

geoh5py.data.integer_data module

class geoh5py.data.integer_data.**IntegerData**(*data_type:* geoh5py.data.data_type.DataType, ***kwargs*)
Bases: *geoh5py.data.numeric_data.NumericData*

classmethod ndv() → int
No-Data-Value

classmethod primitive_type() → *geoh5py.data.primitive_type_enum.PrimitiveTypeEnum*

geoh5py.data.numeric_data module

class geoh5py.data.numeric_data.**NumericData**(*data_type:* geoh5py.data.data_type.DataType, ***kwargs*)
Bases: *geoh5py.data.data.Data*, abc.ABC

Data container for floats values

check_vector_length(*values*) → numpy.ndarray
Check for possible mismatch between the length of values stored and the expected number of cells or vertices.

classmethod primitive_type() → *geoh5py.data.primitive_type_enum.PrimitiveTypeEnum*

property values: numpy.ndarray

Returns values: An array of float values

geoh5py.data.primitive_type_enum module

class geoh5py.data.primitive_type_enum.**PrimitiveTypeEnum**(*value*)
Bases: enum.Enum

Known data type.

Available options:

BLOB = 6

DATETIME = 8

FILENAME = 5

FLOAT = 2

GEOMETRIC = 9

INTEGER = 1

INVALID = 0

REFERENCED = 4

TEXT = 3

VECTOR = 7

geoh5py.data.reference_value_map module

class geoh5py.data.reference_value_map.**ReferenceValueMap**(*color_map: dict[int, str] = None*)
Bases: abc.ABC

Maps from reference index to reference value of ReferencedData.

property map
dict: A reference dictionary mapping values to strings

geoh5py.data.referenced_data module

class geoh5py.data.referenced_data.**ReferencedData**(*data_type:* geoh5py.data.data_type.DataType, ***kwargs*)
Bases: *geoh5py.data.integer_data.IntegerData*

Reference data described by indices and associated strings.

classmethod primitive_type() → *geoh5py.data.primitive_type_enum.PrimitiveTypeEnum*

property value_map
Pointer to the data.data_type.DataType.value_map

geoh5py.data.text_data module

class geoh5py.data.text_data.**CommentsData**(*data_type:* geoh5py.data.data_type.DataType, ***kwargs*)
Bases: *geoh5py.data.data.Data*

Comments added to an Object or Group. Stored as a list of dictionaries with the following keys:

```
comments = [
    {
        "Author": "username",
        "Date": "2020-05-21T10:12:15",
        "Text": "A text comment."
    },
]
```

classmethod primitive_type() → *geoh5py.data.primitive_type_enum.PrimitiveTypeEnum*

property values: list[dict] | None
list List of comments

class geoh5py.data.text_data.**TextData**(*data_type:* geoh5py.data.data_type.DataType, ***kwargs*)
Bases: *geoh5py.data.data.Data*

classmethod primitive_type() → *geoh5py.data.primitive_type_enum.PrimitiveTypeEnum*

property values: str | None
str Text value.

geoh5py.data.unknown_data module

class geoh5py.data.unknown_data.**UnknownData**(*data_type:* geoh5py.data.data_type.DataType, *association:* geoh5py.data.data_association_enum.DataAssociationEnum, *name: str*, *uid: Optional[uuid.UUID] = None*)

Bases: *geoh5py.data.data.Data*

classmethod primitive_type() → *geoh5py.data.primitive_type_enum.PrimitiveTypeEnum*

Module contents

geoh5py.groups package

Submodules

geoh5py.groups.container_group module

class geoh5py.groups.container_group.**ContainerGroup**(*group_type:* geoh5py.groups.group_type.GroupType, ***kwargs*)

Bases: *geoh5py.groups.group.Group*

The type for the basic Container group.

classmethod default_type_uid() → uuid.UUID

geoh5py.groups.custom_group module

class geoh5py.groups.custom_group.**CustomGroup**(*group_type:* geoh5py.groups.group_type.GroupType, ***kwargs*)

Bases: *geoh5py.groups.group.Group*

A custom group, for an unlisted Group type.

classmethod default_type_uid() → uuid.UUID | None

geoh5py.groups.drillhole_group module

class geoh5py.groups.drillhole_group.**DrillholeGroup**(*group_type:* geoh5py.groups.group_type.GroupType, ***kwargs*)

Bases: *geoh5py.groups.group.Group*

The type for the group containing drillholes.

classmethod default_type_uid() → uuid.UUID

geoh5py.groups.giftools_group module

class geoh5py.groups.giftools_group.**GiftoolsGroup**(*group_type:* geoh5py.groups.group_type.GroupType, ***kwargs*)

Bases: *geoh5py.groups.group.Group*

The type for a GIFtools group.

classmethod default_type_uid() → uuid.UUID

geoh5py.groups.group module

class geoh5py.groups.group.**Group**(*group_type:* geoh5py.groups.group_type.GroupType, ***kwargs*)

Bases: *geoh5py.shared.entity.Entity*

Base Group class

add_comment(*comment: str*, *author: Optional[str] = None*)

Add text comment to an object.

Parameters

- **comment** – Text to be added as comment.
- **author** – Author's name or `contributors`.

property comments

Fetch a *CommentsData* entity from children.

abstract classmethod default_type_uid() → uuid.UUID | None

property entity_type: *geoh5py.groups.group_type.GroupType*

classmethod find_or_create_type(*workspace:* workspace.Workspace, ***kwargs*) → *GroupType*

geoh5py.groups.group_type module

class geoh5py.groups.group_type.**GroupType**(*workspace:* workspace.Workspace, ***kwargs*)

Bases: *geoh5py.shared.entity_type.EntityType*

property allow_delete_content: bool

`bool`: [True] Allow to delete the group *children*.

property allow_move_content: bool

`bool`: [True] Allow to move the group *children*.

static create_custom(*workspace:* workspace.Workspace, ***kwargs*) → *GroupType*

Creates a new instance of GroupType for an unlisted custom Group type with a new auto-generated UUID.

classmethod find_or_create(*workspace:* workspace.Workspace, *entity_class*, ***kwargs*) → *GroupType*

Find or creates an EntityType with given UUID that matches the given Group implementation class.

Parameters

- **workspace** – An active Workspace class
- **entity_class** – An Group implementation class.

Returns A new instance of GroupType.

geoh5py.groups.notype_group module

class geoh5py.groups.notype_group.**NoTypeGroup**(*group_type:* geoh5py.groups.group_type.GroupType, ***kwargs*)

Bases: *geoh5py.groups.group.Group*

A group with no type.

classmethod default_type_uid() → uuid.UUID

geoh5py.groups.property_group module

class geoh5py.groups.property_group.**PropertyGroup**(***kwargs*)

Bases: abc.ABC

Property group listing data children of an object. This group is not registered to the workspace and only visible to the parent object.

property association: *geoh5py.data.data_association_enum.DataAssociationEnum*
DataAssociationEnum Data association

property attribute_map: dict
dict Attribute names mapping between geoh5 and geoh5py

property name: str
str Name of the group

property parent: *geoh5py.shared.entity.Entity*
The parent *ObjectBase*

property properties: list[uuid.UUID]
List of unique identifiers for the *Data* contained in the property group.

property property_group_type: str

property uid: uuid.UUID
uuid.UUID Unique identifier

geoh5py.groups.root_group module

class geoh5py.groups.root_group.**RootGroup**(*group_type:* geoh5py.groups.group_type.GroupType, ***kwargs*)

Bases: *geoh5py.groups.notype_group.NoTypeGroup*

The Root group of a workspace.

property parent
Parental entity of root is always None

Module contents

geoh5py.io package

Submodules

geoh5py.io.h5_reader module

class geoh5py.io.h5_reader.**H5Reader**
Bases: object

Class to read information from a geoh5 file.

static bool_value(*value: numpy.int8*) → bool

classmethod fetch_attributes(*file: str | h5py.File*, *uid: uuid.UUID*, *entity_type: str*) → tuple[dict, dict, dict]
Get attributes of an *Entity*.

Parameters

- **file** – h5py.File or name of the target geoh5 file
- **uid** – Unique identifier
- **entity_type** – Type of entity from 'group', 'data', 'object', 'group_type', 'data_type', 'object_type'

Returns

attributes: [obj:*dict* of attributes for the *Entity*]

type_attributes: [obj:*dict* of attributes for the EntityType]

property_groups: [obj:*dict* of data uuid.UUID]

classmethod fetch_cells(*file: str | h5py.File*, *uid: uuid.UUID*) → np.ndarray | None
Get an object's *cells*.

Parameters

- **file** – h5py.File or name of the target geoh5 file
- **uid** – Unique identifier of the target object.

Return cells numpy.ndarray of int.

classmethod fetch_children(*file: str | h5py.File*, *uid: uuid.UUID*, *entity_type: str*) → dict
Get *children* of an *Entity*.

Parameters

- **file** – h5py.File or name of the target geoh5 file
- **uid** – Unique identifier
- **entity_type** – Type of entity from 'group', 'data', 'object', 'group_type', 'data_type', 'object_type'

Return children [{uuid: type}, . . .] List of dictionaries for the children uid and type

classmethod fetch_coordinates(*file: str | h5py.File*, *uid: uuid.UUID*, *name: str*) → np.ndarray | None
Get an object coordinates data.

Parameters

- **file** – h5py.File or name of the target geoh5 file
- **uid** – Unique identifier of the target object
- **name** – Type of coordinates 'vertices', 'trace' or 'surveys'

Return surveys numpy.ndarray of [x, y, z] coordinates

classmethod fetch_delimiters(*file: str | h5py.File*, *uid: uuid.UUID*) → tuple[np.ndarray, np.ndarray, np.ndarray]

Get the delimiters of a *BlockModel*.

Parameters

- **file** – h5py.File or name of the target geoh5 file
- **uid** – Unique identifier of the target entity.

Returns

u_delimiters: [obj:*numpy.ndarray* of u_delimiters]

v_delimiters: [obj:*numpy.ndarray* of v_delimiters]

z_delimiters: [obj:*numpy.ndarray* of z_delimiters]

classmethod fetch_metadata(*file: str | h5py.File*, *uid: uuid.UUID*) → str | dict | None

Fetch the metadata of an entity.

classmethod fetch_octree_cells(*file: str | h5py.File*, *uid: uuid.UUID*) → np.ndarray

Get *Octree cells*.

Parameters

- **file** – h5py.File or name of the target geoh5 file
- **uid** – Unique identifier of the target entity.

Return octree_cells numpy.ndarray of int.

classmethod fetch_project_attributes(*file: str | h5py.File*) → dict[Any, Any]

Get attributes of an *Entity*.

Parameters **file** – h5py.File or name of the target geoh5 file

Return attributes dict of attributes.

classmethod fetch_property_groups(*file: str | h5py.File*, *uid: uuid.UUID*) → dict[str, dict[str, str]]

Get the property groups.

Parameters

- **file** – h5py.File or name of the target geoh5 file
- **uid** – Unique identifier of the target entity

Return property_group_attributes dict of property groups and respective attributes.

```
property_group = {
    "group_1": {"attribute": value, ...},
    ...,
    "group_N": {"attribute": value, ...},
}
```

classmethod fetch_trace_depth(*file: str | h5py.File*, *uid: uuid.UUID*) → np.ndarray

Get an object *trace_depth* data

Parameters

- **file** – h5py.File or name of the target geoh5 file
- **uid** – Unique identifier of the target object

Return surveys numpy.ndarray of [x, y, z] coordinates

classmethod fetch_uuids(*file: str | h5py.File*, *entity_type: str*) → list
Fetch all uuids of a given type from geoh5

Parameters

- **file** – h5py.File or name of the target geoh5 file
- **entity_type** – Type of entity from 'group', 'data', 'object', 'group_type', 'data_type', 'object_type'

Return uuids [uuid1, uuid2, . . .] List of uuids

classmethod fetch_value_map(*file: str | h5py.File*, *uid: uuid.UUID*) → dict
Get data value_map

Parameters

- **file** – h5py.File or name of the target geoh5 file
- **uid** – Unique identifier of the target entity

Return value_map dict of {int: str}

classmethod fetch_values(*file: str | h5py.File*, *uid: uuid.UUID*) → float | None
Get data *values*

Parameters

- **file** – h5py.File or name of the target geoh5 file
- **uid** – Unique identifier of the target entity

Return values numpy.array of float

static format_type_string(*string*)

key_map = {'cells': 'Cells', 'color_map': 'Color map', 'octree_cells': 'Octree Cells', 'property_groups': 'PropertyGroups', 'surveys': 'Surveys', 'trace': 'Trace', 'trace_depth': 'TraceDepth', 'values': 'Data', 'vertices': 'Vertices'}

static str_from_utf8_bytes(*value: bytes | str*) → str

static uuid_str(*value: uuid.UUID*) → str

static uuid_value(*value: str*) → uuid.UUID

geoh5py.io.h5_writer module

class geoh5py.io.h5_writer.**H5Writer**
Bases: object

Writing class to a geoh5 file.

static bool_value(*value: numpy.int8*) → bool
Convert integer to bool.

classmethod create_dataset(*entity_handle*, *dataset: numpy.ndarray*, *label: str*)
Create a dataset on geoh5.

Parameters

- **entity_handle** – Pointer to a hdf5 group
- **dataset** – Array of values to be written
- **label** – Name of the dataset on file

classmethod create_geoh5(*file: str | h5py.File*, *workspace:* workspace.Workspace)
Add the geoh5 core structure.

Parameters

- **file** – Name or handle to a geoh5 file.
- **workspace** – *Workspace* object defining the project structure.

Return h5file Pointer to a geoh5 file.

classmethod fetch_handle(*file: str | h5py.File*, *entity*, *return_parent: bool = False*)
Get a pointer to an *Entity* in geoh5.

Parameters

- **file** – Name or handle to a geoh5 file
- **entity** – Target *Entity*
- **return_parent** – Option to return the handle to the parent entity.

Return entity_handle HDF5 pointer to an existing entity, parent or None if not found.

classmethod finalize(*file: str | h5py.File*, *workspace:* workspace.Workspace)
Add/replace the *RootGroup* in geoh5.

Parameters

- **file** – Name or handle to a geoh5 file
- **workspace** – Workspace object defining the project structure.

key_map = {'cells': 'Cells', 'color_map': 'Color map', 'metadata': 'Metadata', 'octree_cells': 'Octree Cells', 'property_groups': 'PropertyGroups', 'surveys': 'Surveys', 'trace': 'Trace', 'trace_depth': 'TraceDepth', 'values': 'Data', 'vertices': 'Vertices'}

static remove_child(*file: str | h5py.File*, *uid: uuid.UUID*, *ref_type: str*, *parent:* Entity)
Remove a child from a parent.

Parameters

- **file** – Name or handle to a geoh5 file
- **uid** – uuid of the target *Entity*
- **ref_type** – Input type from: 'Types', 'Groups', 'Objects' or 'Data
- **parent** – Remove entity from parent.

static remove_entity(*file: str | h5py.File*, *uid: uuid.UUID*, *ref_type: str*, *parent:* Entity *= None*)
Remove an entity and its type from the target geoh5 file.

Parameters

- **file** – Name or handle to a geoh5 file
- **uid** – uuid of the target *Entity*
- **ref_type** – Input type from: 'Types', 'Groups', 'Objects' or 'Data

- **parent** – Remove entity from parent.

classmethod save_entity(*file: str | h5py.File*, *entity*, *add_children: bool = True*)
Write an *Entity* to geoh5 with its *children*.

Parameters

- **file** – Name or handle to a geoh5 file.
- **entity** – Target *Entity*.
- **add_children** – Add *children*.

str_type = dtype('O')

classmethod update_attributes(*file: str | h5py.File*, *entity*)
Update the attributes of an *Entity*.

Parameters

- **file** – Name or handle to a geoh5 file.
- **entity** – Target *Entity*.

static uuid_str(*value: uuid.UUID*) → str
Convert `uuid.UUID` to string used in geoh5.

static uuid_value(*value: str*) → uuid.UUID
Convert string to `uuid.UUID`.

classmethod write_attributes(*file: str | h5py.File*, *entity*)
Write attributes of an *Entity*.

Parameters

- **file** – Name or handle to a geoh5 file.
- **entity** – Entity with attributes to be added to the geoh5 file.

classmethod write_cell_delimiters(*file: str | h5py.File*, *entity*)
Add cell delimiters (u, v, z) to a *BlockModel*.

Parameters

- **file** – Name or handle to a geoh5 file
- **entity** – Target entity

classmethod write_cells(*file: str | h5py.File*, *entity*)
Add *cells*.

Parameters

- **file** – Name or handle to a geoh5 file
- **entity** – Target entity

classmethod write_color_map(*file: str | h5py.File*, *entity_type: shared.EntityType*)
Add *ColorMap* to a *DataType*.

Parameters

- **file** – Name or handle to a geoh5 file
- **entity_type** – Target entity_type with color_map

classmethod write_coordinates(*file: str | h5py.File*, *entity*, *attribute*)
Add `surveys` of an object.

Parameters

- **file** – Name or handle to a geoh5 file.
- **entity** – Target entity.
- **attribute** – Name of the attribute to be written to geoh5

classmethod write_data_values(*file: str | h5py.File*, *entity*, *attribute*)
Add data *values*.

Parameters

- **file** – Name or handle to a geoh5 file.
- **entity** – Target entity.
- **attribute** – Name of the attribute to be written to geoh5

classmethod write_entity(*file: str | h5py.File*, *entity*)
Add an *Entity* and its attributes to geoh5. The function returns a pointer to the entity if already present on file.

Parameters

- **file** – Name or handle to a geoh5 file.
- **entity** – Target *Entity*.

Return entity Pointer to the written entity. Active link if "close_file" is False.

classmethod write_entity_type(*file: str | h5py.File*, *entity_type: shared.EntityType*)
Add an *EntityType* to geoh5.

Parameters

- **file** – Name or handle to a geoh5 file.
- **entity_type** – Entity with type to be added.

Return type Pointer to *EntityType* in geoh5.

classmethod write_octree_cells(*file: str | h5py.File*, *entity*)
Add `cells` of an `Octree` object to geoh5.

Parameters

- **file** – Name or handle to a geoh5 file.
- **entity** – Target entity_type with color_map.

classmethod write_properties(*file: str | h5py.File*, *entity:* Entity)
Add properties of an *Entity*.

Parameters

- **file** – Name or handle to a geoh5 file.
- **entity** – Target *Entity*.

classmethod write_property_groups(*file: str | h5py.File*, *entity*)
Write *PropertyGroup* associated with an *Entity*.

Parameters

- **file** – Name or handle to a geoh5 file.
- **entity** – Target *Entity*.

classmethod write_to_parent(*file: str | h5py.File*, *entity:* Entity, *recursively=False*)
Add/create an *Entity* and add it to its parent.

Parameters

- **file** – Name or handle to a geoh5 file.
- **entity** – Entity to be added or linked to a parent in geoh5.
- **recursively** – Add parents recursively until reaching the *RootGroup*.

classmethod write_value_map(*file: str | h5py.File*, *entity_type: shared.EntityType*)
Add *ReferenceValueMap* to a *DataType*.

Parameters

- **file** – Name or handle to a geoh5 file
- **entity_type** – Target entity_type with value_map

classmethod write_visible(*file: str | h5py.File*, *entity*)
Needs revision once Visualization is implemented

Parameters

- **file** – Name or handle to a geoh5 file
- **entity** – Target entity

Module contents

geoh5py.objects package

Subpackages

geoh5py.objects.surveys package

Submodules

geoh5py.objects.surveys.direct_current module

class geoh5py.objects.surveys.direct_current.**CurrentElectrode**(*object_type:* geoh5py.objects.object_type.ObjectType, ***kwargs*)

Bases: *geoh5py.objects.surveys.direct_current.PotentialElectrode*

Ground direct current electrode (transmitter).

add_default_ab_cell_id()
Utility function to set ab_cell_id's based on curve cells.

copy(*parent=None*, *copy_children: bool = True*)
Function to copy a survey to a different parent entity.

Parameters

- **parent** – Target parent to copy the entity under. Copied to current *parent* if None.
- **copy_children** – Create copies of all children entities along with it.

Return entity Registered Entity to the workspace.

property current_electrodes
The associated current electrode object (sources).

classmethod default_type_uid() → uuid.UUID

Returns Default unique identifier

property potential_electrodes: *PotentialElectrode* | None
The associated potential_electrodes (receivers)

class geoh5py.objects.surveys.direct_current.**PotentialElectrode**(*object_type:* geoh5py.objects.object_type.ObjectType, ***kwargs*)

Bases: *geoh5py.objects.curve.Curve*

Ground potential electrode (receiver).

property ab_cell_id: *ReferencedData* | None
Reference data entity mapping cells to a unique current dipole.

property ab_map: dict | None
Get the ReferenceData.value_map of the ab_value_id

copy(*parent=None*, *copy_children: bool = True*)
Function to copy a survey to a different parent entity.

Parameters

- **parent** – Target parent to copy the entity under. Copied to current *parent* if None.
- **copy_children** – Create copies of all children entities along with it.

Return entity Registered Entity to the workspace.

property current_electrodes
The associated current electrode object (sources).

classmethod default_type_uid() → uuid.UUID

Returns Default unique identifier

property metadata: dict | None
Metadata attached to the entity.

property potential_electrodes
The associated potential_electrodes (receivers)

geoh5py.objects.surveys.magnetics module

class geoh5py.objects.surveys.magnetics.**AirborneMagnetics**(*object_type:* geoh5py.objects.object_type.ObjectType, ***kwargs*)

Bases: *geoh5py.objects.curve.Curve*

An airborne magnetic survey object.

> **Warning:** Partially implemented.

classmethod default_type_uid() → uuid.UUID

Returns Default unique identifier

Module contents

Submodules

geoh5py.objects.block_model module

class geoh5py.objects.block_model.**BlockModel**(*object_type:* geoh5py.objects.object_type.ObjectType, ***kwargs*)

Bases: *geoh5py.objects.object_base.ObjectBase*

Rectilinear 3D tensor mesh defined by three perpendicular axes. Each axis is divided into discrete intervals that define the cell dimensions. Nodal coordinates are determined relative to the origin and the sign of cell delimiters. Negative and positive cell delimiters are accepted to denote relative offsets from the origin.

property centroids
numpy.array, shape (*n_cells*, 3): Cell center locations in world coordinates.

```
centroids = [
    [x_1, y_1, z_1],
    ...,
    [x_N, y_N, z_N]
]
```

classmethod default_type_uid() → uuid.UUID

Returns Default unique identifier

property n_cells: int | None
int: Total number of cells

property origin: numpy.ndarray
numpy.array of float, shape (3,): Coordinates of the origin.

property rotation: float
float: Clockwise rotation angle (degree) about the vertical axis.

property shape: tuple | None
list of int, len (3,): Number of cells along the u, v and z-axis

property u_cell_delimiters: np.ndarray | None
numpy.array of float: Nodal offsets along the u-axis relative to the origin.

property u_cells: np.ndarray | None
numpy.array of float, shape (*shape* [0],): Cell size along the u-axis.

property v_cell_delimiters: np.ndarray | None
numpy.array of float: Nodal offsets along the v-axis relative to the origin.

property v_cells: np.ndarray | None
numpy.array of float, shape (*shape* [1],): Cell size along the v-axis.

property z_cell_delimiters: np.ndarray | None
numpy.array of float: Nodal offsets along the z-axis relative to the origin (positive up).

property z_cells: np.ndarray | None
numpy.array of float, shape (*shape* [2],): Cell size along the z-axis

geoh5py.objects.curve module

class geoh5py.objects.curve.**Curve**(*object_type:* geoh5py.objects.object_type.ObjectType, ***kwargs*)

Bases: *geoh5py.objects.points.Points*

Curve object defined by a series of line segments (*cells*) connecting *vertices*.

property cells: np.ndarray | None

numpy.ndarray of int, shape (*, 2): Array of indices defining segments connecting vertices. Defined based on *parts* if set by the user.

property current_line_id

classmethod default_type_uid() → uuid.UUID

Returns Default unique identifier

property parts

numpy.array of int, shape (*n_vertices*, 2): Group identifiers for vertices connected by line segments as defined by the *cells* property. The definition of the *cells* property get modified by the setting of parts.

property unique_parts

list of int: Unique *parts* identifiers.

geoh5py.objects.drillhole module

class geoh5py.objects.drillhole.**Drillhole**(*object_type:* geoh5py.objects.object_type.ObjectType, ***kwargs*)

Bases: *geoh5py.objects.points.Points*

Drillhole object class defined by

> **Warning:** Not yet implemented.

add_data(*data: dict*, *property_group: str = None*) → *Data* | list[*Data*]

Create *Data* specific to the drillhole object from dictionary of name and arguments. A keyword 'depth' or 'from-to' with corresponding depth values is expected in order to locate the data along the well path.

Parameters **data** – Dictionary of data to be added to the object, e.g.

```
data_dict = {
    "data_A": {
        'values', [v_1, v_2, ...],
        "from-to": numpy.ndarray,
        },
    "data_B": {
        'values', [v_1, v_2, ...],
        "depth": numpy.ndarray,
        },
}
```

Returns List of new Data objects.

add_vertices(*xyz*)
Function to add vertices to the drillhole

property cells: np.ndarray | None
numpy.ndarray of int, shape (*, 2): Array of indices defining segments connecting vertices.

property collar
numpy.array of float, shape (3,): Coordinates of the collar

property cost
float: Cost estimate of the drillhole

property default_collocation_distance
Minimum collocation distance for matching depth on merge

classmethod default_type_uid() → uuid.UUID

desurvey(*depths*)
Function to return x, y, z coordinates from depth.

property deviation_x
numpy.ndarray: Store the change in x-coordinates along the well path.

property deviation_y
numpy.ndarray: Store the change in y-coordinates along the well path.

property deviation_z
numpy.ndarray: Store the change in z-coordinates along the well path.

property locations
numpy.ndarray: Lookup array of the well path x,y,z coordinates.

property planning
str: Status of the hole: ["Default", "Ongoing", "Planned", "Completed"]

sort_depths()
Read the 'DEPTH' data and sort all Data.values if needed

property surveys
numpy.array of float, shape (3,): Coordinates of the surveys

property trace: np.ndarray | None
numpy.array: Drillhole trace defining the path in 3D

property trace_depth: np.ndarray | None
numpy.array: Drillhole trace depth from top to bottom

validate_interval_data(*from_to*, *input_values*, *collocation_distance=0.0001*)
Compare new and current depth values, append new vertices if necessary and return an augmented values vector that matches the vertices indexing.

validate_log_data(*depth*, *input_values*, *collocation_distance=0.0001*)
Compare new and current depth values, append new vertices if necessary and return an augmented values vector that matches the vertices indexing.

geoh5py.objects.geo_image module

class geoh5py.objects.geo_image.**GeoImage**(*object_type:* geoh5py.objects.object_type.ObjectType, ***kwargs*)

Bases: *geoh5py.objects.object_base.ObjectBase*

Image object class.

> **Warning:** Not yet implemented.

classmethod default_type_uid() → uuid.UUID

geoh5py.objects.grid2d module

class geoh5py.objects.grid2d.**Grid2D**(*object_type:* geoh5py.objects.object_type.ObjectType, ***kwargs*)

Bases: *geoh5py.objects.object_base.ObjectBase*

Rectilinear 2D grid of uniform cell size. The grid can be oriented in 3D space through horizontal *rotation* and *dip* parameters. Nodal coordinates are determined relative to the origin and the sign of cell delimiters.

property cell_center_u: np.ndarray | None
numpy.array of float, shape(*u_count*,): Cell center local coordinate along the u-axis.

property cell_center_v: np.ndarray | None
numpy.array of float shape(*u_count*,): The cell center local coordinate along the v-axis.

property centroids: np.ndarray | None
numpy.array of float, shape (*n_cells*, 3): Cell center locations in world coordinates.

```
centroids = [
    [x_1, y_1, z_1],
    ...,
    [x_N, y_N, z_N]
]
```

classmethod default_type_uid() → uuid.UUID

Returns Default unique identifier

property dip: float
float: Dip angle from horizontal (positive down) in degrees.

property n_cells: int | None
int: Total number of cells.

property origin: numpy.ndarray
numpy.array of float, shape (3,): Coordinates of the origin.

property rotation: float
float: Clockwise rotation angle (degree) about the vertical axis.

property shape: tuple | None
list of int, len (2,): Number of cells along the u and v-axis.

property u_cell_size: float | None
float: Cell size along the u-axis.

property u_count: int | None
int: Number of cells along u-axis

property v_cell_size: float | None
float: Cell size along the v-axis

property v_count: int | None
int: Number of cells along v-axis

property vertical: bool | None
bool: Set the grid to be vertical.

geoh5py.objects.label module

class geoh5py.objects.label.**Label**(*object_type:* geoh5py.objects.object_type.ObjectType, ***kwargs*)
Bases: *geoh5py.objects.object_base.ObjectBase*

Label object for annotation in viewport.

> **Warning:** Not yet implemented.

classmethod default_type_uid() → uuid.UUID

geoh5py.objects.notype_object module

class geoh5py.objects.notype_object.**NoTypeObject**(*object_type:* geoh5py.objects.object_type.ObjectType, ***kwargs*)
Bases: *geoh5py.objects.object_base.ObjectBase*

Generic Data object without a registered type

classmethod default_type_uid() → uuid.UUID

geoh5py.objects.object_base module

class geoh5py.objects.object_base.**ObjectBase**(*object_type:* geoh5py.objects.object_type.ObjectType, ***kwargs*)
Bases: *geoh5py.shared.entity.Entity*

Object base class.

add_comment(*comment: str*, *author: Optional[str] = None*)
Add text comment to an object.

Parameters

- **comment** – Text to be added as comment.
- **author** – Name of author or defaults to *contributors*.

add_data(*data: dict*, *property_group: str = None*) → *Data* | list[*Data*]
Create *Data* from dictionary of name and arguments. The provided arguments can be any property of the target Data class.

Parameters **data** – Dictionary of data to be added to the object, e.g.

```
data = {
    "data_A": {
        'values', [v_1, v_2, ...],
        'association': 'VERTEX'
        },
    "data_B": {
        'values', [v_1, v_2, ...],
        'association': 'CELLS'
        },
}
```

Returns List of new Data objects.

add_data_to_group(*data: list | Data | uuid.UUID | str*, *name: str*) → *PropertyGroup*
Append data children to a *PropertyGroup* All given data must be children of the parent object.

Parameters

- **data** – *Data* object, *uid* or *name* of data.
- **name** – Name of a *PropertyGroup*. A new group is created if none exist with the given name.

Returns The target property group.

property cells
numpy.array of int: Array of indices defining the connection between *vertices*.

property comments
Fetch a *CommentsData* entity from children.

abstract classmethod default_type_uid() → uuid.UUID

property entity_type: *geoh5py.objects.object_type.ObjectType*
EntityType: Object type.

property faces

find_or_create_property_group(***kwargs*) → *geoh5py.groups.property_group.PropertyGroup*
Find or create *PropertyGroup* from given name and properties.

Parameters **kwargs** – Any arguments taken by the *PropertyGroup* class.

Returns A new or existing *PropertyGroup*

classmethod find_or_create_type(*workspace:* workspace.Workspace, ***kwargs*) → *ObjectType*
Find or create a type instance for a given object class.

Parameters **workspace** – Target *Workspace*.

Returns The ObjectType instance for the given object class.

get_data(*name: str*) → list[*Data*]
Get a child *Data* by name.

Parameters **name** – Name of the target child data

Returns A list of children Data objects

get_data_list() → list[str]
Get a list of names of all children *Data*.

Returns List of names of data associated with the object.

property last_focus: str
bool: Object visible in camera on start.

property n_cells: int | None
int: Number of cells.

property n_vertices: int | None
int: Number of vertices.

property property_groups: list[*PropertyGroup*]
list of *PropertyGroup*.

remove_data_from_group(*data: list | Data | uuid.UUID | str*, *name: str = None*)
Remove data children to a *PropertyGroup* All given data must be children of the parent object.

Parameters

- **data** – *Data* object, *uid* or *name* of data.
- **name** – Name of a *PropertyGroup*. A new group is created if none exist with the given name.

validate_data_association(*attribute_dict*)
Get a dictionary of attributes and validate the data 'association' keyword.

static validate_data_type(*attribute_dict*)
Get a dictionary of attributes and validate the type of data.

property vertices
numpy.array of float, shape (*, 3): Array of x, y, z coordinates defining the position of points in 3D space.

geoh5py.objects.object_type module

class geoh5py.objects.object_type.**ObjectType**(*workspace:* workspace.Workspace, ***kwargs*)
Bases: *geoh5py.shared.entity_type.EntityType*

Object type class

static create_custom(*workspace:* workspace.Workspace) → *ObjectType*
Creates a new instance of ObjectType for an unlisted custom Object type with a new auto-generated UUID.

Parameters **workspace** – An active Workspace class

classmethod find_or_create(*workspace:* workspace.Workspace, *entity_class*, ***kwargs*) → *ObjectType*
Find or creates an EntityType with given uuid.UUID that matches the given Group implementation class.

It is expected to have a single instance of EntityType in the Workspace for each concrete Entity class.

Parameters

- **workspace** – An active Workspace class
- **entity_class** – An Group implementation class.

Returns A new instance of GroupType.

geoh5py.objects.octree module

class geoh5py.objects.octree.**Octree**(*object_type:* geoh5py.objects.object_type.ObjectType, ***kwargs*)

Bases: *geoh5py.objects.object_base.ObjectBase*

Octree mesh class that uses a tree structure such that cells can be subdivided it into eight octants.

base_refine()

Refine the mesh to its base octree level resulting in a single cell along the shortest dimension.

property centroids

numpy.array of float, shape (*n_cells*, 3): Cell center locations in world coordinates.

```
centroids = [
    [x_1, y_1, z_1],
    ...,
    [x_N, y_N, z_N]
]
```

classmethod default_type_uid() → uuid.UUID

property n_cells: int | None

int: Total number of cells in the mesh

property octree_cells: np.ndarray | None

numpy.ndarray of int, shape (*n_cells*, 4): Array defining the i, j, k position and size of each cell. The size defines the width of a cell in number of base cells.

```
cells = [
    [i_1, j_1, k_1, size_1],
    ...,
    [i_N, j_N, k_N, size_N]
]
```

property origin

numpy.array of float, shape (3,): Coordinates of the origin

property rotation: float

float: Clockwise rotation angle (degree) about the vertical axis.

property shape: tuple | None

list of int, len (3,): Number of cells along the u, v and w-axis.

property u_cell_size: float | None

float: Base cell size along the u-axis.

property u_count: int | None

int: Number of cells along u-axis.

property v_cell_size: float | None

float: Base cell size along the v-axis.

property v_count: int | None

int: Number of cells along v-axis.

property w_cell_size: float | None

float: Base cell size along the w-axis.

property w_count: int | None

int: Number of cells along w-axis.

geoh5py.objects.points module

class geoh5py.objects.points.**Points**(*object_type:* geoh5py.objects.object_type.ObjectType, ***kwargs*)
Bases: *geoh5py.objects.object_base.ObjectBase*

Points object made up of vertices.

classmethod default_type_uid() → uuid.UUID

property vertices: np.ndarray | None
vertices

geoh5py.objects.surface module

class geoh5py.objects.surface.**Surface**(*object_type:* geoh5py.objects.object_type.ObjectType, ***kwargs*)
Bases: *geoh5py.objects.points.Points*

Surface object defined by vertices and cells

property cells: np.ndarray | None
Array of vertices index forming triangles :return cells: `numpy.array` of `int`, shape (“*”, 3)

classmethod default_type_uid() → uuid.UUID

Module contents

geoh5py.shared package

Submodules

geoh5py.shared.coord3d module

class geoh5py.shared.coord3d.**Coord3D**(*xyz: numpy.ndarray = array([[0.0, 4.64625126e-310, 0.0]])*)
Bases: `object`

Coordinate of vertices.

> **Warning:** Replaced by `numpy.array`

property locations: numpy.ndarray

property x: float

property y: float

property z: float

geoh5py.shared.date_time module

class geoh5py.shared.date_time.**DateTime**
Bases: object

Time stamp

> **Warning:** Not implemented

geoh5py.shared.distance_unit module

class geoh5py.shared.distance_unit.**DistanceUnit**
Bases: object

Units

> **Warning:** Not implemented

geoh5py.shared.entity module

class geoh5py.shared.entity.**Entity**(***kwargs*)
Bases: abc.ABC

Base Entity class

add_children(*children: list[shared.Entity]*)

Parameters **children** – Add a list of entities as *children*

property allow_delete: bool
bool Entity can be deleted from the workspace.

property allow_move: bool
bool Entity can change *parent*

property allow_rename: bool
bool Entity can change name

property attribute_map: dict
dict Correspondence map between property names used in geoh5py and geoh5.

property children
list Children entities in the workspace tree

copy(*parent=None*, *copy_children: bool = True*)
Function to copy an entity to a different parent entity.

Parameters

- **parent** – Target parent to copy the entity under. Copied to current *parent* if None.
- **copy_children** – Create copies of all children entities along with it.

Return entity Registered Entity to the workspace.

classmethod create(*workspace*, ***kwargs*)
Function to create an entity.

Parameters

- **workspace** – Workspace to be added to.
- **kwargs** – List of keyword arguments defining the properties of a class.

Return entity Registered Entity to the workspace.

abstract property entity_type: shared.EntityType

property existing_h5_entity: bool
bool Entity already present in *h5file*.

classmethod fix_up_name(*name: str*) → str
If the given name is not a valid one, transforms it to make it valid :return: a valid name built from the given name. It simply returns the given name if it was already valid.

property metadata: str | dict | None
Metadata attached to the entity.

property modified_attributes
list[str] List of attributes to be updated in associated workspace *h5file*.

property name: str
str Name of the entity

property parent

property public: bool

bool Entity is accessible in the workspace tree and other parts of the the user interface in ANALYST.

reference_to_uid(*value:* Entity | *str* | *uuid.UUID*) → list[uuid.UUID]
General entity reference translation.

Parameters **value** – Either an *Entity*, string or uuid

Returns List of unique identifier associated with the input reference.

remove_children(*children: list[shared.Entity]*)
Remove children from the list of children entities.

Parameters **children** – List of entities

> **Warning:** Removing a child entity without re-assigning it to a different parent may cause it to become inactive. Inactive entities are removed from the workspace by *remove_none_referents()*.

property uid
uuid.UUID The unique identifier of an entity, either as stored in geoh5 or generated in uuid4() format.

property visible: bool
bool Entity visible in camera (checked in ANALYST object tree).

property workspace
Workspace to which the Entity belongs to.

geoh5py.shared.entity_type module

class geoh5py.shared.entity_type.**EntityType**(*workspace: ws.Workspace*, ***kwargs*)
Bases: abc.ABC

property attribute_map
dict Correspondence map between property names used in geoh5py and geoh5.

property description
str Entity type description.

property existing_h5_entity: bool
bool Entity already present in *h5file*.

classmethod find(*workspace: ws.Workspace*, *type_uid: uuid.UUID*) → TEntityType | None
Finds in the given Workspace the EntityType with the given UUID for this specific EntityType implementation class.

Returns EntityType of None

property modified_attributes
list[str] List of attributes to be updated in associated workspace *h5file*.

property name: str | None

property uid: uuid.UUID
uuid.UUID The unique identifier of an entity, either as stored in geoh5 or generated in uuid4() format.

property workspace: ws.Workspace
Workspace registering this type.

geoh5py.shared.file_name module

class geoh5py.shared.file_name.**FileName**
Bases: object

File name

Warning: Not implemented

geoh5py.shared.utils module

geoh5py.shared.utils.**compare_entities**(*object_a*, *object_b*, *ignore: list | None = None*, *decimal: int = 6*)

geoh5py.shared.utils.**fetch_h5_handle**(*file: str | h5py.File*) → h5py.File
Open in read+ mode a geoh5 file from string. If receiving a file instead of a string, merely return the given file.

Parameters **file** – Name or handle to a geoh5 file.

Return h5py.File Handle to an opened h5py file.

geoh5py.shared.utils.**match_values**(*vec_a*, *vec_b*, *collocation_distance=0.0001*)
Find indices of matching values between two arrays, within collocation_distance.

Param vec_a, list or numpy.ndarray Input sorted values

Param vec_b, list or numpy.ndarray Query values

Returns indices, numpy.ndarray Pairs of indices for matching values between the two arrays such that vec_a[ind[:, 0]] == vec_b[ind[:, 1]].

geoh5py.shared.utils.**merge_arrays**(*head*, *tail*, *replace='A->B'*, *mapping=None*, *collocation_distance=0.0001*, *return_mapping=False*)

Given two numpy.arrays of different length, find the matching values and append both arrays.

Param head, numpy.array of float First vector of shape(M,) to be appended.

Param tail, numpy.array of float Second vector of shape(N,) to be appended

Param mapping=None, numpy.ndarray of int Optional array where values from the head are replaced by the tail.

Param collocation_distance=1e-4, float Tolerance between matching values.

Returns numpy.array shape(O,) Unique values from head to tail without repeats, within collocation_distance.

geoh5py.shared.version_number module

class geoh5py.shared.version_number.**VersionNumber**(*number: float*)

Bases: object

Version

> **Warning:** Not implemented

property value: float

geoh5py.shared.version_string module

class geoh5py.shared.version_string.**VersionString**(*value: str*)

Bases: object

Version of Geoscience ANALYST

> **Warning:** Not implemented

property value: str

geoh5py.shared.vertex_index module

class geoh5py.shared.vertex_index.**VertexIndex**

Bases: object

Indices

> **Warning:** Not implemented

geoh5py.shared.weakref_utils module

geoh5py.shared.weakref_utils.**get_clean_ref**(*some_dict: dict[K, ReferenceType[T]]*, *key: K*) → T | None
Gets the referent value for the given `key` in a `some_dict` of `weakref` values. In case `key` points to a reference to a deleted value, remove that key from `some_dict` on the fly, and returns None.

Parameters

- **some_dict** – The dictionary of `weakref` values.
- **key** – The key

Returns the referent value for `key` if found in the the dictionary, else None.

geoh5py.shared.weakref_utils.**insert_once**(*some_dict: dict[K, ReferenceType]*, *key: K*, *value*)
Check if the reference to an Entity with uuid is already in use.

Parameters

- **some_dict** – Dictionary of UUID keys and weakref values.
- **key** – UUID key to be checked.
- **value** – Entity to be checked

Returns Dictionary with clean weakref

geoh5py.shared.weakref_utils.**remove_none_referents**(*some_dict: dict[K, ReferenceType]*)
Removes any key from the given `some_dict` where the value is a reference to a deleted value (that is where referent of the `weakref` value is None).

Parameters **some_dict** – The dictionary to be cleaned up.

Module contents

geoh5py.workspace package

Submodules

geoh5py.workspace.workspace module

class geoh5py.workspace.workspace.**Workspace**(*h5file: str = 'Analyst.geoh5'*, ***kwargs*)
Bases: `object`

The Workspace class manages all Entities created or imported from the *geoh5* structure.

The basic requirements needed to create a Workspace are:

Parameters **h5file** – File name of the target *geoh5* file. A new project is created if the target file cannot by found on disk.

activate()
Makes this workspace the active one.

In case the workspace gets deleted, Workspace.active() safely returns None.

static active() → *geoh5py.workspace.workspace.Workspace*
Get the active workspace.

property attribute_map: dict
Mapping between names used in the geoh5 database.

property contributors: numpy.ndarray
numpy.array of str List of contributors name.

copy_to_parent(*entity*, *parent*, *copy_children: bool = True*, *omit_list: tuple = ()*)
Copy an entity to a different parent with copies of children.

Parameters

- **entity** – Entity to be copied.
- **parent** – Target parent to copy the entity under.
- **copy_children** – Copy all children of the entity.
- **omit_list** – List of property names to omit on copy

Returns The Entity registered to the workspace.

classmethod create(*entity:* geoh5py.shared.entity.Entity, ***kwargs*) → *geoh5py.shared.entity.Entity*
Create and register a new Entity.

Parameters

- **entity** – Entity to be created
- **kwargs** – List of attributes to set on new entity

Return entity The new entity

create_data(*entity_class*, *entity_kwargs: dict*, *entity_type_kwargs: dict* | DataType) → *Entity* | None
Create a new Data entity with attributes.

Parameters

- **entity_class** – *Data* class.
- **entity_kwargs** – Properties of the entity.
- **entity_type_kwargs** – Properties of the entity_type.

Returns The newly created entity.

create_entity(*entity_class*, *save_on_creation: bool = True*, *file: str | h5py.File | None = None*, ***kwargs*)
→ *Entity* | None
Function to create and register a new entity and its entity_type.

Parameters

- **entity_class** – Type of entity to be created
- **save_on_creation** – Save the entity to h5file immediately
- **file** – h5py.File or name of the target geoh5 file

Return entity Newly created entity registered to the workspace

create_object_or_group(*entity_class*, *entity_kwargs: dict*, *entity_type_kwargs: dict*) → *Entity* | None
Create an object or a group with attributes.

Parameters

- **entity_class** – *ObjectBase* or *Group* class.
- **entity_kwargs** – Attributes of the entity.
- **entity_type_kwargs** – Attributes of the entity_type.

Returns A new Object or Group.

property data: list[*data.Data*]
Get all active Data entities registered in the workspace.

deactivate()
Deactivate this workspace if it was the active one, else does nothing.

property distance_unit: str
`str` Distance unit used in the project.

fetch_cells(*uid: uuid.UUID*, *file: str | h5py.File | None = None*) → np.ndarray
Fetch the cells of an object from the source h5file.

Parameters

- **uid** – Unique identifier of target entity.
- **file** – `h5py.File` or name of the target geoh5 file

Returns Cell object with vertices index.

fetch_children(*entity:* Entity, *recursively: bool = False*, *file: str | h5py.File | None = None*)
Recover and register children entities from the h5file

Parameters

- **entity** – Parental entity
- **recursively** – Recover all children down the project tree
- **file** – `h5py.File` or name of the target geoh5 file

fetch_coordinates(*uid: uuid.UUID*, *name: str*, *file: str | h5py.File | None = None*) → np.ndarray
Fetch the survey values of drillhole objects

Parameters

- **uid** – Unique identifier of target entity
- **file** – `h5py.File` or name of the target geoh5 file

Return values Array of [Depth, Dip, Azimuth] defining the drillhole path.

fetch_delimiters(*uid: uuid.UUID*, *file: str | h5py.File | None = None*) → tuple[np.ndarray, np.ndarray, np.ndarray]
Fetch the delimiter attributes from the source h5file.

Parameters

- **uid** – Unique identifier of target data object.
- **file** – `h5py.File` or name of the target geoh5 file

Returns Arrays of delimiters along the u, v, and w axis (u_delimiters, v_delimiters, z_delimiters).

fetch_metadata(*uid: uuid.UUID*, *file: str | h5py.File | None = None*)
Fetch the metadata of an entity from the source h5file.

Returns

fetch_octree_cells(*uid: uuid.UUID*, *file: str | h5py.File | None = None*) → np.ndarray
Fetch the octree cells ordering from the source h5file

Parameters

- **uid** – Unique identifier of target entity
- **file** – `h5py.File` or name of the target geoh5 file

Return values Array of [i, j, k, dimension] defining the octree mesh

fetch_or_create_root(*h5file: h5py._hl.files.File*)

fetch_property_groups(*entity:* Entity, *file: str | h5py.File | None = None*) → list[*PropertyGroup*]
Fetch all property_groups on an object from the source h5file

Parameters

- **entity** – Target object
- **file** – `h5py.File` or name of the target geoh5 file

fetch_trace_depth(*uid: uuid.UUID*, *file: str | h5py.File | None = None*) → np.ndarray
Fetch the trace depth information of a drillhole objects

Parameters

- **uid** – Unique identifier of target entity
- **file** – `h5py.File` or name of the target geoh5 file

Returns Array of trace depth values.

fetch_values(*uid: uuid.UUID*, *file: str | h5py.File | None = None*) → float | None
Fetch the data values from the source h5file.

Parameters

- **uid** – Unique identifier of target data object.
- **file** – `h5py.File` or name of the target geoh5 file

Returns Array of values.

finalize(*file: str | h5py.File | None = None*)
Finalize the h5file by checking for updated entities and re-building the Root

Parameters **file** – `h5py.File` or name of the target geoh5 file

find_data(*data_uid: uuid.UUID*) → *Entity* | None
Find an existing and active Data entity.

find_entity(*entity_uid: uuid.UUID*) → *Entity* | None
Get all active entities registered in the workspace.

find_group(*group_uid: uuid.UUID*) → *group.Group* | None
Find an existing and active Group object.

find_object(*object_uid: uuid.UUID*) → *object_base.ObjectBase* | None
Find an existing and active Object.

find_type(*type_uid: uuid.UUID*, *type_class: type[*EntityType*]*) → *EntityType* | None
Find an existing and active EntityType

Parameters **type_uid** – Unique identifier of target type

property ga_version: str
`str` Version of Geoscience Analyst software.

get_entity(*name: str | uuid.UUID*) → list[*Entity* | None]
Retrieve an entity from one of its identifier, either by name or `uuid.UUID`.

Parameters **name** – Object identifier, either name or uuid.

Returns List of entities with the same given name.

property groups: list[groups.Group]
Get all active Group entities registered in the workspace.

property h5file: str

Str Target *geoh5* file name with path.

property list_data_name: dict[uuid.UUID, str]
`dict` of `uuid.UUID` keys and name values for all registered Data.

property list_entities_name: dict[uuid.UUID, str]

Returns `dict` of `uuid.UUID` keys and name values for all registered Entities.

property list_groups_name: dict[uuid.UUID, str]
`dict` of `uuid.UUID` keys and name values for all registered Groups.

property list_objects_name: dict[uuid.UUID, str]
`dict` of `uuid.UUID` keys and name values for all registered Objects.

load_entity(*uid: uuid.UUID*, *entity_type: str*, *parent:* Entity *= None*, *file: str | h5py.File | None = None*) → *Entity* | None
Recover an entity from geoh5.

Parameters

- **uid** – Unique identifier of entity
- **entity_type** – One of entity type 'group', 'object', 'data' or 'root'
- **file** – `h5py.File` or name of the target geoh5 file

Return entity Entity loaded from geoh5

property name: str
`str` Name of the project.

property objects: list[objects.ObjectBase]
Get all active Object entities registered in the workspace.

remove_children(*parent*, *children: list*, *file: str | h5py.File | None = None*)
Remove a list of entities from a parent.

remove_entity(*entity:* Entity, *file: str | h5py.File | None = None*)
Function to remove an entity and its children from the workspace

remove_none_referents(*referents: dict[uuid.UUID, ReferenceType]*, *rtype: str*, *file: str | h5py.File | None = None*)
Search and remove deleted entities

remove_recursively(*entity:* Entity, *file: str | h5py.File | None = None*)
Delete an entity and its children from the workspace and geoh5 recursively

property root: *Entity* | None
RootGroup entity.

save_entity(*entity:* Entity, *add_children: bool = True*, *file: str | h5py.File | None = None*)
Save or update an entity to geoh5.

Parameters

- **entity** – Entity to be written to geoh5.
- **add_children** – Add children entities to geoh5.

- **file** – h5py.File or name of the target geoh5

property types: list[*EntityType*]
Get all active entity types registered in the workspace.

validate_file(*file*) → h5py._hl.files.File
Validate the h5file name

property version: float
float Version of the geoh5 file format.

property workspace
This workspace instance itself.

geoh5py.workspace.workspace.**active_workspace**(*workspace:* geoh5py.workspace.workspace.Workspace)

Module contents

2.3.2 Module contents

2.4 GEOH5 Format

2.4.1 Abou

The GEOH5 format aims to provide a robust means of handling large quantities of diverse data required in geoscience. The file

structure builds on the generic qualities of the Geoscience ANALYST data model, and attempts to maintain a certain level of simplicity and consistency throughout. It is based entirely on free and open HDF5 technology. Given that this specification is public, the file format could, with further investment and involvement, become a useful exchange format for the broader geoscientific community.

Why GEOH5?

- **Leverages propertie**
  Fast I/O, compression, cross-platform
- **Content readable ar**
  We recommend using HDFView, along with Geoscience ANALYST, when learning the format.
- **Easily extensible to**
  It is intended

for Geoscience ANALYST to preserve data it does not understand (and generally be very tolerant with regards to missing information) when loading and saving geoh5 files. This will allow third parties to write to this format fairly easily, as well as include additional information not included in this spec for their own purposes. In the current implementation, Geoscience ANALYST automatically removes unnecessary information on save.

2.4.2 Defin

The following sections define the structure

and components making up the GEOH5 file format.

Workspace

The bulk of the data is accessible both directly by `UUID` through the `flat` HDF5 groups or through a **hierarchy of hard links** structured as follow:

Data
Flat con-

tainer for all data entities

Groups
Flat container for all group entities

Objects
Flat container for all object entities

Root
Optional hard link to `workspace` group, top of group hierarchy.

Types

- *Data Types*: Flat container

for all data types

- *Group Types*: Flat container for all group types
- *Object Types*: Flat container for all object types

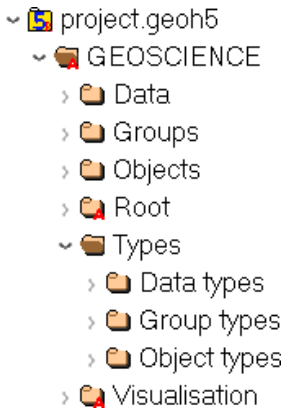
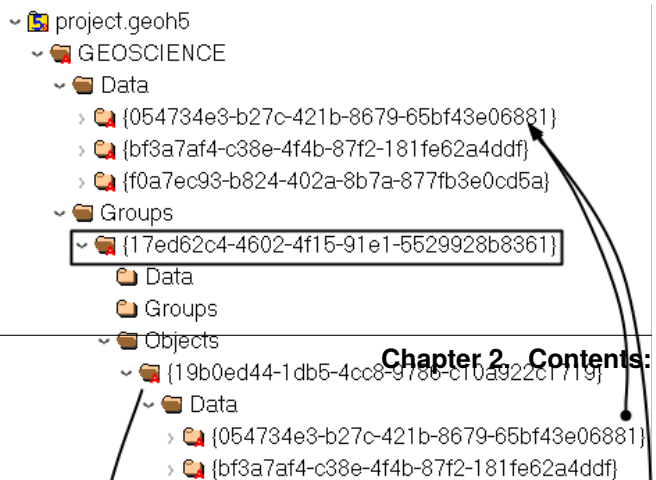


Fig. 2.1: As seen in HDFView

While all groups, objects and data entities are written into their respective base folder, they also hold links to their children entities to allow for traversals. There is no data duplication, merely multiple references (pointers) to the data storage on file. Types are shared (and thus generally written to file first). All groups, objects and data must include a hard link to their type.



Attributes

Version `double` Version of specification used by this file

Distance unit `str` *feet* or (default) *metres* Distance unit of all data enclosed

Contributors 1D array of `str` (Optional) List of users who contributed to this workspace

Groups

Groups are simple container for other groups and objects. They are often used to assign special meanings to a collection of entities or to create specialized software functionality. See the *Group Types* section for the list of supported group types.

Note: Though this file format technically allows objects/groups to appear within multiple groups simultaneously (overlapping lists), this is not currently supported by Geoscience ANALYST.

Attributes

Name `str` Name of the object displayed in the project tree.

ID `str`, *UUID* Unique identifier of the group.

Visible `int`, 0 or (default) 1 Set visible in the 3D camera (checked in the object tree).

Public `int`, 0 or (default) 1 Set accessible in the object tree and other parts of the the user interface.

Clipping IDs 1D array of *UUID* (Optional) List of unique identifiers of clipping plane objects.

Allow delete `int`, 0 or (default) 1 (Optional) User interface allows deletion.

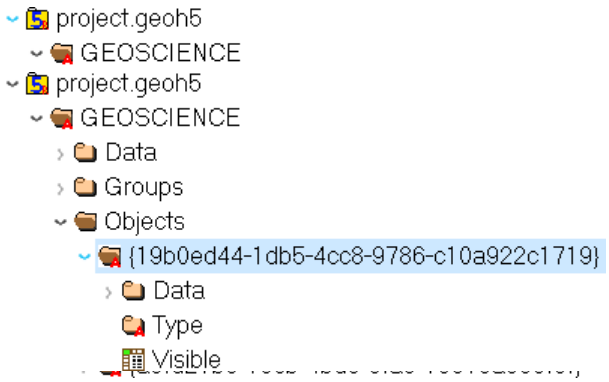
Allow move `int`, 0 or (default) 1 (Optional) User interface allows moving to another parent group.

Allow rename `int`, 0 or (default) 1 (Optional) User interface allows renaming.

Metadata (`int`, optional) (Optional) Any additional text attached to the group.

Objects

Objects are containers for `Data` with spatial information. Most (not all) object geometry is described in terms of vertices (3D locations) and cells (groupings of vertices such as triangles or segments). The exact requirements and interpretation depends on the type. Additional information may also be stored for some specific types. See the *Object Types* section for the list of supported objects.



Attributes

Name `str` Name of the object displayed in the project tree.

ID `str` Unique identifier (*UUID*) of the group.

Visible `int`, 0 or (default) 1 Set visible in the 3D camera (checked in the object tree).

Public `int`, 0 or (default) 1 Set accessible in the object tree and other parts of the the user interface.

Clipping IDs 1D array of `UUID` (Optional) List of unique identifiers of clipping plane objects.

Allow delete `int`, 0 or (default) 1 (Optional) User interface allows deletion.

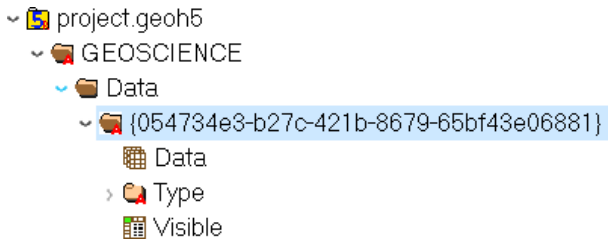
Allow move `int`, 0 or (default) 1 (Optional) User interface allows moving to another parent group.

Allow rename `int`, 0 or (default) 1 (Optional) User interface allows renaming.

Metadata (`int`, optional) (Optional) Any additional text attached to the group.

Data

Containers for data values of various types. Data are currently **always stored as a 1D array**, even in the case of single-value data with the `Object` association (in which case it is a 1D array of length 1). See the *Data Types* section for the list of supported data types.



Attributes

Association `str` Describes which part the property is tied to. Must be one of: *Unknown*, *Object*, *Cell*, *Vertex*, *Face* or *Group*

Name `str` Name of the data displayed in the project tree.

ID `str` Unique identifier (*UUID*) of the group.

Visible `int`, 0 or 1 (Optional) Set visible in the 3D camera (checked in the object tree).

Clipping IDs 1D array of `UUID` (Optional) List of unique identifiers of clipping plane objects.

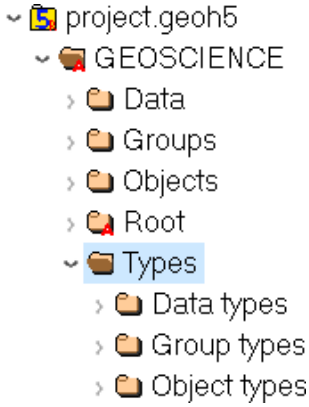
Allow delete `int`, 0 or (default) 1 (Optional) User interface allows deletion.

Allow rename `int`, 0 or (default) 1 (Optional) User interface allows renaming.

Public `int`, 0 or (default) 1 (Optional) Set accessible in the object tree and other parts of the the user interface.

Types

While they are structured similarly, **each group, object or set of data has a type that defines how its HDF5 datasets should be interpreted**. This type is shared among any number of entities (groups/objects/data sets).



Group Types

The following section describes the supported group types.

Group Types

To be further documented

Container

UUID : {61FBB4E8-A480-11E3-8D5A-2776BDF4F982}

Simple container with no special meaning. Default in Geoscience ANALYST.

Drillholes

UUID : {825424FB-C2C6-4FEA-9F2B-6CD00023D393}

Container restricted to containing drillhole objects, and which may provide convenience functionality for the drillholes within.

No Type (Root)

UUID : {dd99b610-be92-48c0-873c-5b5946ea2840}

The `Root` group defines the tree structure used in Geoscience ANALYST describing the parent-child relationships of entities. If absent, any Groups/Objects/Data will be brought into Geoscience ANALYST under the workspace group, still respecting any defined hierarchy links.

Tools

UUID : {a2befc38-3207-46aa-95a2-16b40117a5d8}

Group for slicer and label objects.

Not yet geoh5py implemented

To be further documented

Maxwell

UUID : {1c4122b2-8e7a-4ec3-8d6e-c818495adac7}

Group for Maxwell plate modeling application.

Not yet geoh5py implemented

To be further documented

GIFTools

GIFtools group containers

GIFtools Project

UUID : {585b3218-c24b-41fe-ad1f-24d5e6e8348a}

Not yet geoh5py implemented

To be documented

GIF Executables

UUID : {afae95ef-c2a7-4aec-9800-0d19bd2c2c07}

Not yet geoh5py implemented

To be documented

gzinv3d

UUID : {20eb4ff8-bdfe-43f3-8745-f418dcc9e14a}

Not yet geoh5py implemented

To be documented

gzfor3d

UUID : {a4857df0-d175-4824-ac5d-cecfdcc2f20b}

Not yet geoh5py implemented

To be documented

magfor3d

UUID : {6b8189ac-a479-4fe7-b4fc-92279aee5a41}

Not yet geoh5py implemented

To be documented

maginv3d

UUID : {b99e8db8-e118-4042-864e-9e1128f2d1e6}

Not yet geoh5py implemented

To be documented

mvifwd

UUID : {14c41f47-bcee-4a63-8192-fa42a1741052}

Not yet geoh5py implemented

To be documented

ggfor3d

UUID : {c8a8424d-ab12-482e-82ee-b198fcfd5859}

Not yet geoh5py implemented

To be documented

gginv3d

UUID : {0f080369-b3a3-464c-83fa-9b3c1efa9895}

Not yet geoh5py implemented

To be documented

mviinv

UUID : {9472b5cb-a285-4257-a2e8-68a3d33aa1f2}

Not yet geoh5py implemented

To be documented

octgrvde

UUID : {4e043415-a0ea-4cef-bf89-2771e27b346c}

Not yet geoh5py implemented

To be documented

octmagde

UUID : {f8217512-296d-4cc0-afcb-6c07a20581fe}

Not yet geoh5py implemented

To be documented

dcinv3d

UUID : {ae416ab8-0e72-4f37-8873-5cc0909433bb}

Not yet geoh5py implemented

To be documented

ipinv3d

UUID : {9f9543a0-e857-4a56-ab66-9f21e2b002c6}

Not yet geoh5py implemented

To be documented

e3dmt

UUID : {8cf239e3-63a6-4813-adf8-9714293b602e}

Not yet geoh5py implemented

To be documented

dcoctree_inv

UUID : {54d296de-0588-472c-9a62-480098303394}

Not yet geoh5py implemented

To be documented

dcoctree_fwd

UUID : {A522D641-6CB7-421B-836B-A14C0D9C7801}

Not yet geoh5py implemented

To be documented

ipoctree_inv

UUID : {d9fd455e-ea94-40f5-9d86-e7c49c7b5005}

Not yet geoh5py implemented

To be documented

dcipf3d

UUID : {59b5338d-596c-4049-9aa4-6979700e00ff}

Not yet geoh5py implemented

To be documented

Geoscience INTEGRATOR Groups

Geoscience INTEGRATOR

UUID : {61449477-3833-11e4-a7fb-fcddabfddab1}

Not yet geoh5py implemented

To be documented

Project

UUID : {56f6f03e-3833-11e4-a7fb-fcddabfddab1}

Not yet geoh5py implemented

To be documented

Query Group

UUID : {85756113-592a-4088-b374-f32c8fac37a2}

Not yet geoh5py implemented

To be documented

Neighbourhoods

UUID : {2a5b7faa-41d1-4437-afac-934933eae6eb}

Not yet geoh5py implemented

To be documented

Map File Group

UUID : {1f684938-2baf-4a01-ac71-e50c30cc0685}

Not yet geoh5py implemented

To be documented

Maps Group

UUID : {4d65f8c3-a015-4c01-b411-412c0f4f0884}

Not yet geoh5py implemented

To be documented

Airborne

UUID : {812f3b2a-fdae-4752-8391-3b657953a983}

Not yet geoh5py implemented

To be documented

Ground

UUID : {a9d05630-7a80-4bda-89a2-feca0dc7a83e}

Not yet geoh5py implemented

To be documented

Borehole

UUID : {f6f011a9-2e52-4f99-b842-a524ad9fdf03}

Not yet geoh5py implemented

To be documented

Geoscience INTEGRATOR Themes

Data

UUID : {51fdf764-3833-11e4-a7fb-fcddabfddab1}

Not yet geoh5py implemented

To be documented

Interpretation

UUID : {05e96011-3833-11e4-a7fb-fcddabfddab1}

Not yet geoh5py implemented

To be documented

Microseismic

UUID : {2bddbaaf-3829-11e4-8654-fcddabfddab1}

Not yet geoh5py implemented

To be documented

Ground Deformation

UUID : {7ade974c-3829-11e4-9cce-fcddabfddab1}

Not yet geoh5py implemented

To be documented

Production Area

UUID : {55ccb6d9-016c-47cd-824f-077214dc44db}

Not yet geoh5py implemented

To be documented

Blasting

UUID : {e2040afa-3829-11e4-a70e-fcddabfddab1}

Not yet geoh5py implemented

To be documented

Stress

UUID : {460e31c8-3829-11e4-a70e-fcddabfddab1}

Not yet geoh5py implemented

To be documented

Drillholes and Wells

UUID : {5d9b6a8c-3829-11e4-93fc-fcddabfddab1}

Not yet geoh5py implemented

To be documented

Mobile Equipment

UUID : {e7f63d21-3833-11e4-a7fb-fcddabfddab1}

Not yet geoh5py implemented

To be documented

Fixed Plant

UUID : {fad14ac4-3833-11e4-a7fb-fcddabfddab1}

Not yet geoh5py implemented

To be documented

Earth Model Points

UUID : {fcd708da-3833-11e4-a7fb-fcddabfddab1}

Not yet geoh5py implemented

To be documented

Earth Model Regular 3D Grids

UUID : {79b41607-ffa2-4825-a270-44dd48807a03}

Not yet geoh5py implemented

To be documented

Observation Points

UUID : {f65e521c-a763-427b-97bf-d0b4e5689e0d}

Not yet geoh5py implemented

To be documented

Targets & Anomalies

UUID : {e41c2308-0f35-47dd-8562-d0fd354406f8}

Not yet geoh5py implemented

To be documented

Targets

UUID : {af0925ba-3dc5-4fe6-ab35-9e0ef568023f}

Not yet geoh5py implemented

To be documented

Anomalies

UUID : {51bcc3e9-1d66-4c83-847e-5c852fc9de58}

Not yet geoh5py implemented

To be documented

Fusion Model

UUID : {3d69be5b-3833-11e4-a7fb-fcddabfddab1}

Not yet geoh5py implemented

To be documented

Deformation

UUID : {5caf35fa-3d0e-11e4-939f-f5f83219c4e0}

Not yet geoh5py implemented

To be documented

Mine Production

UUID : {7508bc11-3829-11e4-9cce-fcddabfddab1}

Not yet geoh5py implemented

To be documented

Earth Models

UUID : {adee3b2a-3829-11e4-a70e-fcddabfddab1}

Not yet geoh5py implemented

To be documented

Mine Models

UUID : {e53a8b3e-3829-11e4-a70e-fcddabfddab1}

Not yet geoh5py implemented

To be documented

Samples

UUID : {1cde9996-cda7-40f0-8c20-faeb4e926748}

Not yet geoh5py implemented

To be documented

Geochemistry & Mineralogy

UUID : {ed00094f-3da1-485f-8c4e-b52f6f171ea4}

Not yet geoh5py implemented

To be documented

Rock Properties

UUID : {cbeb3920-a1a9-46f8-ab2b-7dfdf79c8a00}

Not yet geoh5py implemented

To be documented

Incidents

UUID : {136cb431-c7d2-4992-a5ab-46a6e16b6726}

Not yet geoh5py implemented

To be documented

Mine Infrastructure

UUID : {cff33bb0-ef43-4b06-8070-266940ab9d06}

Not yet geoh5py implemented

To be documented

3D Structural Surfaces

UUID : {a246f9e0-2b67-4efd-bd3d-742bfe06178b}

Not yet geoh5py implemented

To be documented

3D Domains

UUID : {f69979b0-5ba1-417a-93d4-778146049014}

Not yet geoh5py implemented

To be documented

3D Geological Contact Surfaces

UUID : {0bf96ee1-7fa4-41a2-bc8a-7cd76426ba18}

Not yet geoh5py implemented

To be documented

Remote Sensing and Air Pohotos

UUID : {386f2c57-1893-40bb-bd1c-95552b90e514}

Not yet geoh5py implemented

To be documented

Inversions

UUID : {7a7b14af-23d9-4897-9cdb-8d586fefa025}

Not yet geoh5py implemented

To be documented

Topography

UUID : {c162ddd2-a9de-4dac-b6a2-3cc6e011d7c3}

Not yet geoh5py implemented

To be documented

Culture

UUID : {dd51ca09-34d7-4c30-a0d0-ef9e61ea5e9d}

Not yet geoh5py implemented

To be documented

Claims, boundaries

UUID : {6e430b33-4ab8-45c1-896d-c47525185ce0}

Not yet geoh5py implemented

To be documented

Ventilation

UUID : {d049e5a0-aadb-4448-a0f1-fe560c6d26f9}

Not yet geoh5py implemented

To be documented

Gas Monitoring

UUID : {bc8540b0-d814-46ac-b897-b5a528d5d1d6}

Not yet geoh5py implemented

To be documented

Ventilation & Gas Monitoring

UUID : {8ebd9b52-801e-4461-b7e6-e1aa0a8742b3}

Not yet geoh5py implemented

To be documented

Other

UUID : {79b61598-7385-4b63-8513-636ecde9c150}

Not yet geoh5py implemented

To be documented

Airborne

UUID : {3d0e8578-7764-48cf-8db8-6c83d6411762}

Not yet geoh5py implemented

To be documented

Ground

UUID : {47d6f059-b56a-46c7-8fc7-a0ded87360c3}

Not yet geoh5py implemented

To be documented

Integrator Borehole

UUID : {9c69ef80-b45c-4f5c-ac55-996a99dc299f}

Not yet geoh5py implemented

To be documented

Geophysics

UUID : {151778d9-6cc0-4e72-ba08-2a80a4fb967f}

Not yet geoh5py implemented

To be documented

Geotechnical

UUID : {391a616b-3833-11e4-a7fb-fcddabfddab1}

Not yet geoh5py implemented

To be documented

Equipment

UUID : {8beac9ff-3829-11e4-8654-fcddabfddab1}

Not yet geoh5py implemented

To be documented

Attributes

Name `str` Name of the group displayed in the project tree.

ID `str` Unique identifier (*UUID*) of the group type.

Description `str` (Optional) Description of the type.

Allow move contents `int`, 0 or (default) 1 (Optional) User interface allows deletion of the content.

Allow delete contents `int`, 0 or (default) 1 (Optional) User interface allows deletion of the content.

Object Types

Objects are containers for data values with spatial information. The following section describes the supported object types.

ANALYST Objects

To be further documented

Points

UUID : {202C5DB1-A56D-4004-9CAD-BAAFD8899406}

To be further documented

Object defined by vertices only - no cell data.

Curve

UUID : {6A057FDC-B355-11E3-95BE-FD84A7FFCB88}

To be further documented

Each cell contains two vertex indices, representing a segment.

Surface

UUID : {F26FEBA3-ADED-494B-B9E9-B2BBCBE298E1}

To be further documented

Each cell contains three vertex indices, representing a triangle.

Block model

UUID : {B020A277-90E2-4CD7-84D6-612EE3F25051}

To be further documented

Each cell represents a point of a 3D rectilinear grid. For a 3D cell index (i,j,k) along axes U,V and Z of length nU, nV and nZ respectively,

```
cell index = k + i*nZ + j*nU*nZ
```

Without rotation angles, U points eastwards, V points northwards, and Z points upwards. Since their geometry is defined entirely by the additional data described below, block models do not require a `Vertices` or `Cells` dataset.

Datasets

U cell delimiters 1D array of `double` Distances of cell edges from origin along the U axis (first value should be 0)

V cell delimiters 1D array of `double` Distances of cell edges from origin along the V axis (first value should be 0)

Z cell delimiters 1D array of `double` Distances of cell edges from origin upwards along the vertical axis (first value should be 0)

Attributes

Origin composite type

[*X* `double`, *Y* `double`, *Z* `double`]

Origin point of grid

Rotation `double` (default) 0 Counterclockwise angle (degrees) of rotation around the vertical axis in degrees.

2D Grid

UUID : {48f5054a-1c5c-4ca4-9048-80f36dc60a06}

To be further documented

Each cell represents a point in a regular 2D grid. For a 2D cell index (i,j) within axes U and V containing nU and nV cells respectively,

```
cell index = i + j*nU
```

Without rotation angles, U points eastwards and V points northwards. Since their geometry is defined entirely by the additional data described below, 2D grids do not require a Vertices or Cells dataset.

Attributes

Origin composite type

[*X* `double`, *Y* `double`, *Z* `double`]

Origin point of the grid.

U Size `double` Length of U axis

U Count `double` Number of cells along U axis

V Size `double` Length of V axis

V Count `double` Number of cells along V axis

Rotation `double` (Optional) Counterclockwise angle (degrees) of rotation around the vertical axis at the Origin.

Vertical `char`, 0(false, default) or 1(true)) (Optiona) If true, V axis is vertical (and rotation defined around the V axis)

Drillhole

UUID : {7CAEBF0E-D16E-11E3-BC69-E4632694AA37}

To be further documented

Vertices represent points along the drillhole path (support for data rather than the drillhole geometry itself) and must have a `Depth` property value. Cells contain two vertices and represent intervals along the drillhole path (and are a support for interval data as well). Cells may overlap with each other to accommodate the different sampling intervals of various data.

Attributes

Collar composite type

[*X* `double`, *Y* `double`, *Z* `double`]

Collar location

Datasets

Surveys 1D composite array

[*Depth* `double`, *Dip* `double`, *Azimuth* `double`]

Survey locations

Trace 1D composite array

[*X* `double`, *Y* `double`, *Z* `double`]

Points forming the drillhole path from collar to end of hole. Must contain at least two points.

Geoimage

UUID : {77AC043C-FE8D-4D14-8167-75E300FB835A}

Not yet geoh5py implemented

To be further documented

Vertices represent the four corners of the geolocated image. No cell data. An object-associated file-type data containing the image to display is expected to exist under this object.

Note: Should be arranged as a rectangle currently, since Geoscience ANALYST does not currently support skewed images.

Label

UUID : {E79F449D-74E3-4598-9C9C-351A28B8B69E}

Not yet geoh5py implemented

To be further documented

Has no vertices nor cell data

Attributes

Target position composite type

[*X* `double`, *Y* `double`, *Z* `double`]

The target location of the label

Label position composite type

[*X* `double`, *Y* `double`, *Z* `double`] (Optional - Defaults to same as target position) The location where the text of the label is displayed

Slicer

UUID : {238f961d-ae63-43de-ab64-e1a079271cf5}

Not yet geoh5py implemented

To be further documented

Target

UUID : {46991a5c-0d3f-4c71-8661-354558349282}

Not yet geoh5py implemented

To be further documented

ioGAS Points

UUID : {d133341e-a274-40e7-a8c1-8d32fb7f7eaf}

Not yet geoh5py implemented

To be further documented

Maxwell Plate

UUID : {878684e5-01bc-47f1-8c67-943b57d2e694}

Not yet geoh5py implemented

To be further documented

Octree

UUID : {4ea87376-3ece-438b-bf12-3479733ded46}

Not yet geoh5py implemented

To be further documented

Text Object

UUID : {c00905d1-bc3b-4d12-9f93-07fcf1450270}

Not yet geoh5py implemented

To be further documented

Potential Electrode

UUID : {275ecee9-9c24-4378-bf94-65f3c5fbe163}

Not yet geoh5py implemented

To be further documented

Current Electrode

UUID : {9b08bb5a-300c-48fe-9007-d206f971ea92}

Not yet geoh5py implemented

To be further documented

VP Model

UUID : {7d37f28f-f379-4006-984e-043db439ee95}

Not yet geoh5py implemented

To be further documented

Airborne EM

UUID : {fdf7d01e-97ab-43f7-8f2c-b99cc10d8411}

Not yet geoh5py implemented

To be further documented

Airborne TEM Rx

UUID : {19730589-fd28-4649-9de0-ad47249d9aba}

Not yet geoh5py implemented

To be further documented

Airborne TEM Tx

UUID : {58c4849f-41e2-4e09-b69b-01cf4286cded}

Not yet geoh5py implemented

To be further documented

Airborne FEM Rx

UUID : {b3a47539-0301-4b27-922e-1dde9d882c60}

Not yet geoh5py implemented

To be further documented

Airborne FEM Tx

UUID : {a006cf3e-e24a-4c02-b904-2e57b9b5916d}

Not yet geoh5py implemented

To be further documented

Airborne Gravity

UUID : {b54f6be6-0eb5-4a4e-887a-ba9d276f9a83}

Not yet geoh5py implemented

To be further documented

Airborne Magnetics

UUID : {4b99204c-d133-4579-a916-a9c8b98cfccb}

Not yet geoh5py implemented

To be further documented

Ground Gravity

UUID : {5ffa3816-358d-4cdd-9b7d-e1f7f5543e05}

Not yet geoh5py implemented

To be further documented

Ground Magnetics

UUID : {028e4905-cc97-4dab-b1bf-d76f58b501b5}

Not yet geoh5py implemented

To be further documented

Ground Gradient IP

UUID : {68b16515-f424-47cd-bb1a-a277bf7a0a4d}

Not yet geoh5py implemented

To be further documented

Ground EM

UUID : {09f1212f-2bdd-4dea-8bbd-f66b1030dfcd}

Not yet geoh5py implemented

To be further documented

Ground TEM Rx

UUID : {41018a45-01a0-4c61-a7cb-9f32d8159df4}

Not yet geoh5py implemented

To be further documented

Ground TEM Tx

UUID : {98a96d44-6144-4adb-afbe-0d5e757c9dfc}

Not yet geoh5py implemented

To be further documented

Ground TEM Rx (large-loop)

UUID : {deebe11a-b57b-4a03-99d6-8f27b25eb2a8}

Not yet geoh5py implemented

To be further documented

Ground TEM Tx (large-loop)

UUID : {17dbbfbb-3ee4-461c-9f1d-1755144aac90}

Not yet geoh5py implemented

To be further documented

Ground FEM Rx

UUID : {a81c6b0a-f290-4bc8-b72d-60e59964bfe8}

Not yet geoh5py implemented

To be further documented

Ground FEM Tx

UUID : {f59d5a1c-5e63-4297-b5bc-43898cb4f5f8}

Not yet geoh5py implemented

To be further documented

Magnetotellurics

UUID : {b99bd6e5-4fe1-45a5-bd2f-75fc31f91b38}

Not yet geoh5py implemented

To be further documented

ZTEM Rx

UUID : {0b639533-f35b-44d8-92a8-f70ecff3fd26}

Not yet geoh5py implemented

To be further documented

ZTEM Base Stations

UUID : {f495cd13-f09b-4a97-9212-2ea392aeb375}

Not yet geoh5py implemented

To be further documented

Geoscience INTEGRATOR Objects

List object types specific to INTEGRATOR.

Points

UUID : {6832acf3-78aa-44d3-8506-9574a3510c44}

Not yet geoh5py implemented

To be documented

Microseismic

UUID : {b1388138-5463-11e4-93e8-d3b5f5e17625}

Not yet geoh5py implemented

To be documented

Ground Deformation

UUID : {65a66246-59f7-11e4-aa15-123b93f75cba}

Not yet geoh5py implemented

To be documented

Production Area

UUID : {fc560104-9898-4dbf-9711-07519eb1fc84}

Not yet geoh5py implemented

To be documented

Fixed Plant

UUID : {2dda99b0-9980-4f25-820c-01eb7053b42d}

Not yet geoh5py implemented

To be documented

Blasting

UUID : {20cfa317-e98c-4612-9016-414fb1d9375d}

Not yet geoh5py implemented

To be documented

Mobile Equipment

UUID : {53108442-1664-41ed-99ea-ff4dd273e86c}

Not yet geoh5py implemented

To be documented

Stress

UUID : {60ce697d-59f7-42e0-bb58-88374f1d303a}

Not yet geoh5py implemented

To be documented

Earth Model

UUID : {c4268ef8-6b55-11e4-ab63-ca5fbc5c6e8b}

Not yet geoh5py implemented

To be documented

Mine Model

UUID : {ffd1ae8a-70bc-11e4-bf08-53db6953e95a}

Not yet geoh5py implemented

To be documented

Incidents

UUID : {aca8b138-634e-444c-8698-697b91f4cff9}

Not yet geoh5py implemented

To be documented

Mine infrastructures

UUID : {d15ef4a2-6fc4-40c9-ab3e-11647a81dbe1}

Not yet geoh5py implemented

To be documented

3D Structural Surfaces

UUID : {a69aca26-79d8-4074-bd58-dc2202674071}

Not yet geoh5py implemented

To be documented

3D Domains

UUID : {3ecb7f52-b32c-470d-b2f5-c8b0c2b6dff4}

Not yet geoh5py implemented

To be documented

3D Geological Contact Surfaces

UUID : {46d697f1-50a4-4905-a467-04d5c1e7634c}

Not yet geoh5py implemented

To be documented

Remote Sensing and Air Photos

UUID : {b952c7c5-b636-4f6d-9a59-0cbacd84a332}

Not yet geoh5py implemented

To be documented

Inversions

UUID : {b062ffb4-c57d-49a3-9e96-fa26e7b06e7e}

Not yet geoh5py implemented

To be documented

Topography

UUID : {849635b9-1362-40f1-9edd-f45039ff89ac}

Not yet geoh5py implemented

To be documented

Culture

UUID : {849635b9-1362-40f1-9edd-f45039ff89ac}

Not yet geoh5py implemented

To be documented

Claims, boundaries

UUID : {849635b9-1362-40f1-9edd-f45039ff89ac}

Not yet geoh5py implemented

To be documented

Geophysics

UUID : {80413650-58f0-4c99-94af-48f70affbb65}

Not yet geoh5py implemented

To be documented

Ventilation

UUID : {1cc34f3d-fc50-41d9-8210-d93a73b2c7b4}

Not yet geoh5py implemented

To be documented

Gas Monitoring

UUID : {27e44723-9787-48be-9b0e-67f14d60890b}

Not yet geoh5py implemented

To be documented

Other

UUID : {4ed901bb-0303-43cd-9618-a481f5688844}

Not yet geoh5py implemented

To be documented

Airborne

UUID : {c9f70e63-a30f-428b-bee2-02eed5dde43d}

Not yet geoh5py implemented

To be documented

Ground

UUID : {d9f91038-c7a1-4b72-b3f1-ac7760da16ac}

Not yet geoh5py implemented

To be documented

Borehole

UUID : {0bf977b4-bda8-45d7-9c89-9a41d50849bd}

Not yet geoh5py implemented

To be documented

Neighbourhood Surface

UUID : {88087fb8-76ae-445b-9cdf-68dbce530404}

Not yet geoh5py implemented

To be documented

Attributes

Name `str` Name of the object displayed in the project tree.

ID `str` Unique identifier (*UUID*) of the group type.

Description `str` (Optional) Description of the type.

Data Types

New data types can be created at will by software or users to describe object or group properties. Data of the same type can exist on any number of objects or groups of any type, and each instance can be associated with vertices, cells or the object/group itself. Some data type identifiers can also be reserved as a means of identifying a specific kind of data.

Attributes

Name `str` Name of the object displayed in the project tree.

ID `str` Unique identifier (*UUID*) of the data type.

Unlike `Groups` and `Objects`, `Data` entities do not generally have fixed identifier `Type`. Multiple data entities linked by a type will share common properties (color map, units, etc.). Exceptions to this rule are the fixed:

Geoscience INTEGRATOR Data Set

Microseismic

UUID : {9f9cfec8-3829-11e4-8654-fcddabfddab1}

Not yet geoh5py implemented

To be documented

Ground Deformation

UUID : {c455c010-3829-11e4-9cce-fcddabfddab1}

Not yet geoh5py implemented

To be documented

Production Area

UUID : {a8c95123-349f-4461-b9a4-74f76e659a56}

Not yet geoh5py implemented

To be documented

Blasting

UUID : {f55d8ae4-3829-11e4-a70e-fcddabfddab1}

Not yet geoh5py implemented

To be documented

Stress

UUID : {f8324cdc-3829-11e4-a70e-fcddabfddab1}

Not yet geoh5py implemented

To be documented

Fixed Plant

UUID : {31fc7ef1-3833-11e4-a7fb-fcddabfddab1}

Not yet geoh5py implemented

To be documented

Mobile Equipment

UUID : {75eafc96-3833-11e4-a7fb-fcddabfddab1}

Not yet geoh5py implemented

To be documented

Drillholes & Wells

UUID : {faf65f94-3829-11e4-93fc-fcddabfddab1}

Not yet geoh5py implemented

To be documented

Earth Model Points

UUID : {f38a481f-3833-11e4-a7fb-fcddabfddab1}

Not yet geoh5py implemented

To be documented

Earth Model Regular Grid

UUID : {f1a5e5a6-e651-40dc-b184-7827e792ffb3}

Not yet geoh5py implemented

To be documented

Observation Points

UUID : {d89f963d-16ba-4c6b-87d7-9b95410ab2fb}

Not yet geoh5py implemented

To be documented

Targets

UUID : {528e278e-529c-4d95-b8d1-731cf6ae6b5f}

Not yet geoh5py implemented

To be documented

Anomalies

UUID : {b83ac7a5-5217-4ff1-b0bc-2e8d514655ae}

Not yet geoh5py implemented

To be documented

Fusion Model

UUID : {bc99c8a9-3833-11e4-a7fb-fcddabfddab1}

Not yet geoh5py implemented

To be documented

Samples

UUID : {433ab253-1a73-4414-9159-031cdbf7a9e1}

Not yet geoh5py implemented

To be documented

Geochemistry & Mineralogy

UUID : {72f29283-a4f6-4fc0-a1a8-1417ce5fcbec}

Not yet geoh5py implemented

To be documented

Rock Properties

UUID : {4a067ffb-9d20-46b7-8bd8-a6dde20e8b89}

Not yet geoh5py implemented

To be documented

Incidents

UUID : {016dfd26-7d9b-49a6-97d8-cb31c37e404b}

Not yet geoh5py implemented

To be documented

Mine Infrastructure

UUID : {5e34fb33-86ec-49bb-a3d4-5b21fb158a14}

Not yet geoh5py implemented

To be documented

3D Structural Surfaces

UUID : {0a7fef75-26ba-4e80-9d38-89a76044f908}

Not yet geoh5py implemented

To be documented

3D Domains

UUID : {97909249-8584-40d5-9378-a1fb5b86a3ab}

Not yet geoh5py implemented

To be documented

3D Geological Contact Surfaces

UUID : {c63403e2-b635-4b23-998b-1748fe503f81}

Not yet geoh5py implemented

To be documented

Remote Sensing & Air Photos

UUID : {db53c93a-c57a-4911-92f2-9d0c811268b8}

Not yet geoh5py implemented

To be documented

Inversions

UUID : {57a84d8d-6a33-4dfa-a9f2-66b32e495c7f}

Not yet geoh5py implemented

To be documented

Topography

UUID : {5923bd49-6302-4f8b-963a-cba57ac757ae}

Not yet geoh5py implemented

To be documented

Culture

UUID : {bbccf928-d410-4d59-b737-4b4c1f8c84ca}

Not yet geoh5py implemented

To be documented

Claims, boundaries

UUID : {1bcba5c4-c33a-4682-ac47-88694ca67905}

Not yet geoh5py implemented

To be documented

Geophysics

UUID : {9b097cc1-66cb-4088-83dd-c447cba542df}

Not yet geoh5py implemented

To be documented

Ventilation

UUID : {b716d06a-8104-4086-a029-b10d1a545b49}

Not yet geoh5py implemented

To be documented

Gas Monitoring

UUID : {844354fa-41ae-416c-b33f-bf5bfbedc8f5}

Not yet geoh5py implemented

To be documented

Other

UUID : {7bebe936-2e04-4bd6-b050-b128ec5c078d}

Not yet geoh5py implemented

To be documented

Primitive type `str`

Specifies the kind of data values stored as `HDF5 dataset`. Must be one of:

Primitive Types

To be further documented

Float

- Stored as a 1D array of 32-bit float type
- No data value: 1.175494351e-38 (FLT_MIN, considering use of NaN)

Integer

- Stored as a 1D array of 32-bit integer type
- No data value: –2147483648 (INT_MIN, considering use of NaN)

Text

- Stored as a 1D array of UTF-8 encoded, variable-length string type
- No data value : empty string

Referenced

- Stored as a 1D array of 32-bit unsigned integer type (native)
- Value map : (1D composite type array dataset - Key (unsigned int), Value (variable-length utf8 string)) must exist under type
- No data value : 0 (key is tied to value “Unknown”)

DateTime

- Stored as a 1D array of variable-length strings formatted according to the ISO 8601 extended specification for representations of UTC dates and times (Qt implementation), taking the form YYYY-MM-DDTHH:mm:ss[Z|[+|-]HH:mm]
- No data value : empty string

Filename

- Stored as a 1D array of UTF-8 encoded, variable-length string type designating a file name
- For each file name within "Data", an opaque dataset named after the filename must be added under the Data instance, containing a complete binary dump of the file
- Different files (under the same object/group) must be saved under different names
- No data value : empty string

Blob

- Stored as a 1D array of 8-bit char type (native) (value '0' or '1')
- For each index set to 1, an opaque dataset named after the index (e.g. "1", "2", etc) must be added under the Data instance, containing the binary data tied to that index
- No data value : 0

Description `str` (Optional) Description of the type.

Units `str` (Optional) Data units

Color map 1D compound array

[*Value* `double`, *Red* `uint`, *Green* `uint`, *Blue* `uint`, *Alpha* `uint`]

(Optional) Records colors assigned to value ranges. The *Value* mark the start of the range)

Value map (1D compound array dataset)

[*Key* `uint`, *Value* `str`]

Required only for reference data types (classifications)

Transparent no data `int`, 0 or (default) 1 (Optional) Whether or not absence of data/filtered data should be hidden in the viewport.

Hidden `int`, 0 or (default) 1 (Optional) Whether or not the data type should appear in the data type list.

Scientific notation `int`, 0 or (default) 1 (Optional) Whether or not the data values of this type should be displayed in scientific notation.

Precision `int` (Optional) The number of decimals (or significant digits in case of scientific notation) used when displayed data values of this type.

Number of bins `int`, default=50 (Optional) Number of bins used when displaying histogram

Duplicate type on copy `int`, 0 or (default) 1 (Optional) When enabled, a separate copy of this data type will be created and used when data of this type is copied.

2.4.3 Standards

General notes on formatting.

- All text data and attributes are variable-length and use UTF-8 encoding
- All numeric data uses INTEL PC native types
- Boolean values are stored using char (0:false, 1:true)
- Anything found in a geoh5 v1.0 file which is not mentioned in this document is optional information

2.4.4 External Links

- HDFView.
- Precompiled binaries for multiple platforms
- **Libraries for accessing HDF5 data**
  - C, C, .NET
  - Python
  - Matlab

2.5 UI.JSON Format

2.5.1 About

The **ui.json** format provides a User Interface (UI) between geoh5py and Geoscience ANALYST Pro. The file structure is built on an array of JSON objects, each representing a parameter that is used in a python script. An object contains members that control the style and behaviour of the UI. In general only a **label** and **value** member is required in each object, however as outlined below, there are many types of input and dependencies that can be drawn on throughout the file. On output from Geoscience ANALYST, the value and whether the parameter is enabled will be updated or added to each JSON. Extra objects in the JSON are allowed and are ignored, but written out by Geoscience ANALYST. In general, objects will be put in order that they are set in the JSON. The exception is data parameters that depend on object parameters. Placing those parameters in the same group will ensure that they are close in the UI.

2.5.2 Input Objects

Within the **ui.json** file, each JSON object with **value** and **label** members will be considered a parameter to the UI. The following JSON objects could also be present:

run_command `str` Name of python script excluding the .py extension (i.e., "run_me" for run_me.py) required for Geoscience ANALYST Pro to run on save or auto-load.

conda_environment `str` Optional name of conda environment to activate when running the python script in *run_command*

title `str` Optional title of user interface window

2.5.3 Object Members

Each JSON object with the following members become a parameter in the user interface. Each object must have the members `label` and `value`. Each member will contribute to the appearence and behaviour within Geoscience ANALYST>. The possible members that can be given to all parameter objects are:

label `str` Required string describing parameter. A colon will automatically be added within Geoscience ANALYST, so this should be omitted.

value `str`, `int`, `bool` , **or** `float` This require member takes a different form, including empty, depending on the *parameter type*. The value is updated when written from Geoscience ANALYST.

main `bool` If set to true, the parameter is shown in the first tab and will throw an error if not given and not optional. Optional parameters may be set to main. When main is not given or is false, the parameter will be under the *Optional Parameters* tab.

tooltip `str` String describing the parameter in detail that appears when the mouse hovers over it.

optional `bool` *true* or *false* on whether the parameter is optional. On output, check if *enabled* is set to true.

enabled `bool` *true* or *false* if the parameter is enabled. The default is true. If a parameter is optional and not enabled, it will start as disabled (grey and inactive in the UI).

group `str` Name of the group to which the parameter belongs. Adds a box and name around the parameters with the same case-sensitive group name.

groupOptional `bool` If true, adds a checkbox in the top of the group box next to the name. The group parameters will be disabled if not checked. The initial statedepends on the **groupDependency** and **groupDependencyType** members and the **enabled** member of the group's parameters.

dependency `str` The name of the object of which this object is dependent upon. The dependency parameter should be optional or boolean parameter (i.e., has a checkbox).

dependencyType `str` What happens when the dependency member is checked. Options are `enabled` or `disabled`

groupDependency `str` The name of the object of which the group of the parameter is dependent upon. This member will also require the **groupOptional** member to be present and set to `true`. Be sure that the object is not within the group.

groupDependencyType `str` What happens when the group's dependency parameter is checked. Options are `enabled` or `disabled`.

2.5.4 Parameter Types

There are other JSON members that may be available or required based on the parameter type. The following sections define different parameter types that can be found in the **ui.json** format.

Boolean parameter

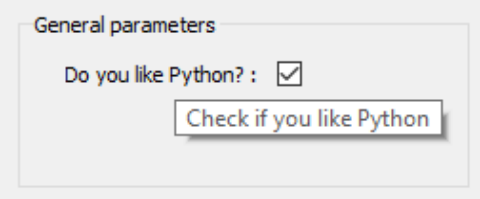
A parameter named "input" that has a `bool` value.

```
{
"input":{
"main": true,
"label": "Do you like Python?",
"value": true,
"tooltip": "Check if you like Python"
```

(continues on next page)

(continued from previous page)

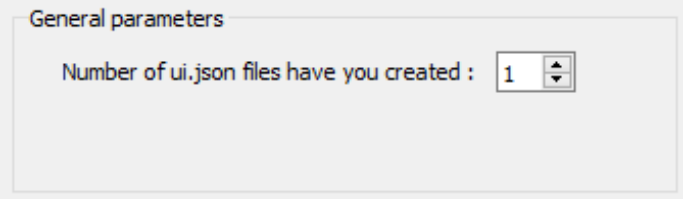
```
}
}
```



Integer parameter

A parameter that has an `int` value. The optional parameters `min` and `max` invoke a validator to insure the bound(s) are enforced.

```
{
"file_xp":{
"main": true,
"label": "Number of ui.json files have you created",
"value": 1,
"min": 0,
"max": 100
}
}
```



Float parameter

A parameter that has a `float` value. The optional parameters are:

min `float` Minimum value allowed for validator of the **value** member. The default is the minimum numeric limits of float.

max `float` Maximum value allowed for validator of the **value** member. The default is the maximum numeric limits of float.

lineEdit `bool` Boolean whether to use a line edit (**true**) or a spin box (**false**). The default is true.

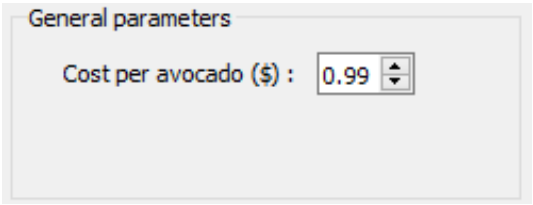
precision `int` Number of decimal places in the line edit or spin box

```
{
"avacado": {
"main": true,
"label": "Cost per avocado ($)",
```

(continues on next page)

(continued from previous page)

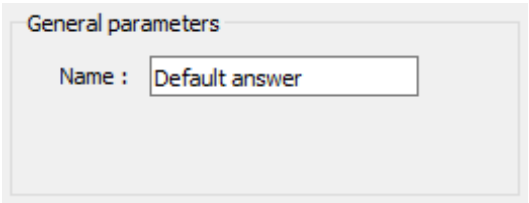
```
"value": 0.99,
"min": 0.29,
"precision": 2,
"lineEdit": false,
"max": 2.79
}
}
```



String parameter

For a simple string parameter, use an empty `str` value to have an empty string. Only a `label` and `value` is required.

```
{
"my_string": {
"main": true,
"label": "Name",
"value": "Default answer"
}
}
```



Multi-choice string parameter

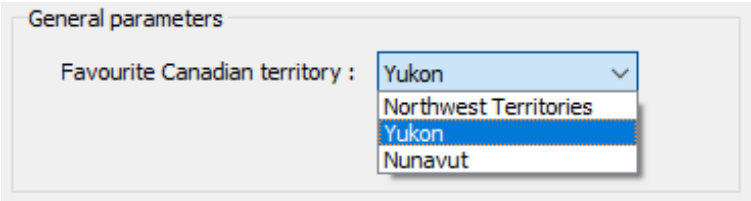
For a drop-down selection, add a `choiceList` member with an array of strings (`str`)

```
{
"favourites":
{
"choiceList": ["Northwest Territories",
"Yukon",
"Nunavut"],
"main": true,
"label": "Favourite Canadian territory",
"value": "Yukon"
```

(continues on next page)

(continued from previous page)

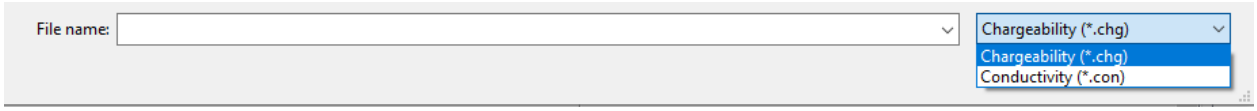
```
}
}
```



File parameter

A file parameter comes with an icon to choose the file, with a `str` value. Extra members of the file object parameter are **fileDescription** and **fileType**. Both of these are `str` types and can be arrays, but must be of the same length

```
{
"model_file": {
"fileDescription": ["Chargeability", "Conductivity"],
"fileType": ["chg", "con"],
"main": true,
"label": "DC/IP model file",
"value": ""
}
}
```



Geoscience ANALYST Object parameter

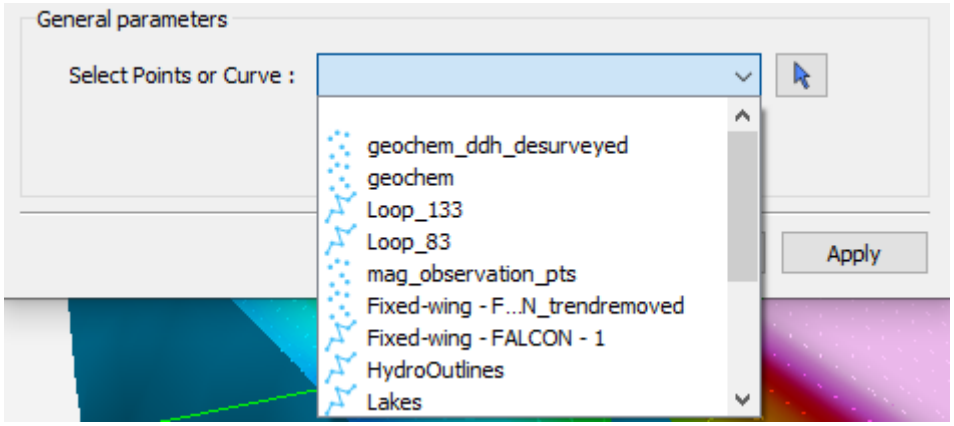
To choose an object from a dropdown menu, the universally unique identifier (UUID) of the *Object Type*: is required for the filtering of objects. This is given as a single or array of `str` in the member **meshType**. The icon to pick the object comes with this parameter. The value returned is the *UUID* of the Geoscience ANALYST object selected.

```
{
"interesting_object": {
"meshType": ["{202C5DB1-A56D-4004-9CAD-BAAFD8899406}" ,
   "{6A057FDC-B355-11E3-95BE-FD84A7FFCB88}"],
```

(continues on next page)

(continued from previous page)

```
"main": true,
"label": "Select Points or Curve",
"value": ""
}
}
```



Geoscience ANALYST Data parameter

Creating a parameter to choose a Geoscience ANALYST object's data requires extra members:

dataType `str` Describes the type of data to filter. One or more (as an array) of these key words: `Integer`, `Float`, `Text`, `Referenced`, `Vector`, `DataTime`, `Geometric`, `Boolean`, or `Text`.

association `str` Describes the geometry of the data. One or more of these key words: `Vertex`, `Cell`, or `Face`.

parent `str` Either a *UUID* of the parent or the name of the *Object parameter* JSON object to allow the user to choose the mesh.

isValue `bool` Describes whether to read the **value** (`float`) or **property** (`str`) member. If not given, the value member is an *UUID* and is considered a *drop-down data parameter*. If this member is given along with **property**, then an icon is added to the UI element, which switches between te **value** (line edit) and **property** (drop-down) choices. This value is updated on export depending on the style choice (`float` or `str`)

property `str`. Data *UUID* that is selected when **isValue** is present. Geoscience ANALYST Pro will update this value on export.

min `float` Optional minimum value allowed for validator of the **value** member. The default is the minimum numeric limits of float.

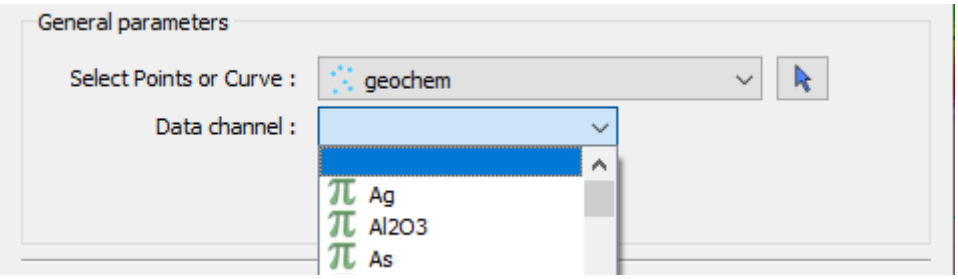
max `float` Optional maximum value allowed for validator of the **value** member. The default is the maximum numeric limits of float.

precision `int` Optional number of decimal places for the value.

Drop-down data parameter

In this example, the object parameter *data_mesh* is also given for reference.

```
{
"data_channel": {
"main": true,
"association": "Vertex",
"dataType": "Float",
"label": "Data channel",
"parent": "data_mesh",
"value": ""
},
"data_mesh": {
"main": true,
 "meshType": ["{202C5DB1-A56D-4004-9CAD-BAAFD8899406}" ,
   "{6A057FDC-B355-11E3-95BE-FD84A7FFCB88}"],
"main": true,
"label": "Select Points or Curve",
"value": ""
}
}
```



Data or value parameter

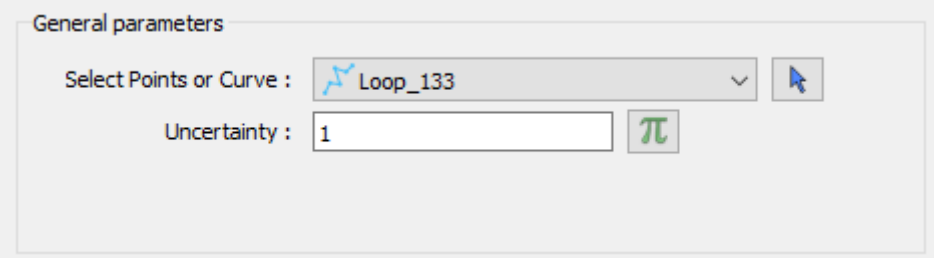
In some cases, a parameter may take its data from a Geoscience ANALYST object or simply a `float` value. The use of the member **isValue** and **property** together allows for the UI to switch between these two cases. In the top image, the **isValue** is true, so the **value** member of 1.0 will initially be active. When the icon is clicked, the type of input is switched to the **property** member (bottom image). The **uncertainty channel** object also depends on the **data_mesh** object. The drop-down selection will filter data from the chosen object that is located on the vertices and is float. The **isValue** is set to false upon export in this case.

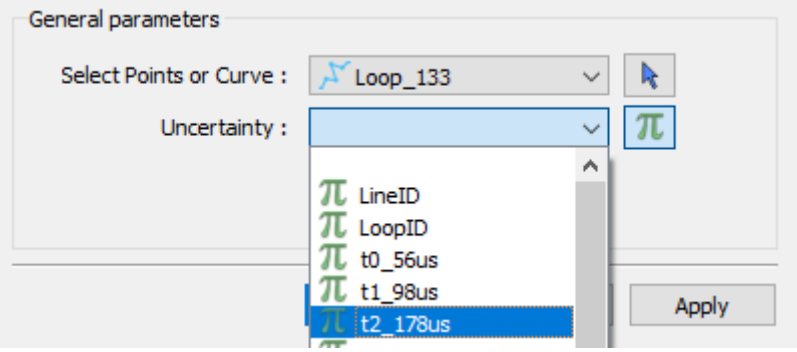
```
{
"uncertainty_channel": {
"main": true,
"association": "Vertex",
"dataType": "Float",
"isValue": true,
"property": "",
"min": 0.001,
"label": "Uncertainty",
```

(continues on next page)

(continued from previous page)

```
"parent": "data_mesh",
"value": 1.0
},
"data_mesh": {
"main": true,
 "meshType": ["{202C5DB1-A56D-4004-9CAD-BAAFD8899406}" ,
   "{6A057FDC-B355-11E3-95BE-FD84A7FFCB88}"],
"main": true,
"label": "Select Points or Curve",
"value": ""
}
}
```





Dependencies on other parameters

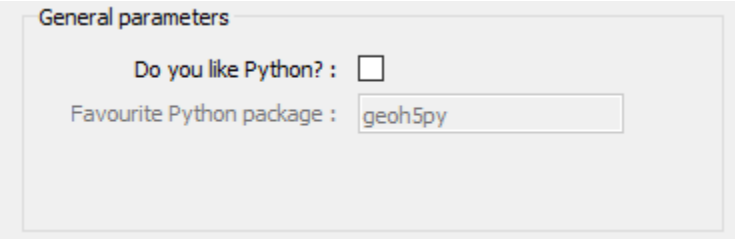
Use the **dependency** and **dependencyType** members to create dependencies. The parameter driving the dependency should set **optional** to true or be a *Boolean parameter'*. Below are a couple of examples. The first initializes the *favourite_package* parameter as disabled until the *python_interest* parameter is checked. The second shows the opposite when the **enabled** member is set to true.

```
{
"python_interest": {
"main": true,
"label": "Do you like Python?",
"value": false,
"tooltip": "Check if you like Python"
},
"favourite_package": {
"main": true,
```

(continues on next page)

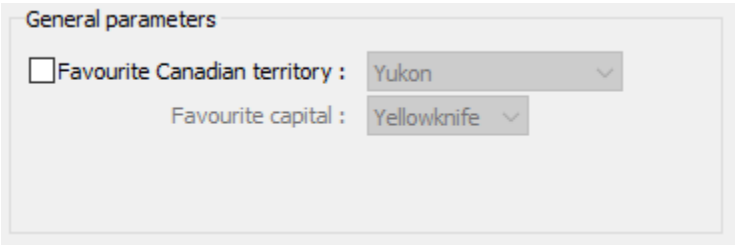
(continued from previous page)

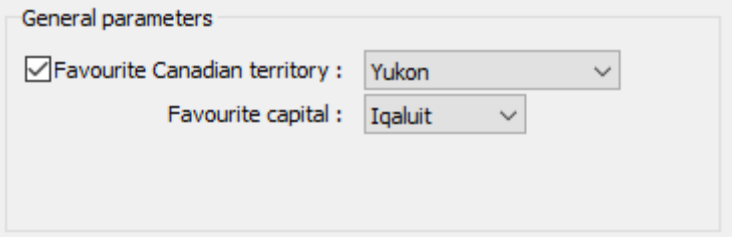
```
"label": "Favourite Python package",
"value": "geoh5py",
"dependency": "python_interest",
"dependencyType": "enabled"
}
}
```



The next example has a dependency on an optional parameter. The **enabled** member is set to false so that it is not automatically checked. The *city* and *territory* parameters will be enabled when the *territory* checkbox is checked.

```
{
"territory": {
"choiceList": ["Northwest Territories",
"Yukon",
"Nunavut"],
"main": true,
"label": "Favourite Canadian territory",
"value": "Yukon",
"optional": true,
"enabled": false
},
"city": {
"main": true,
"choiceList": ["Yellowknife",
"Whitehorse",
"Iqaluit"],
"label": "Favourite capital",
"value": "",
"dependency": "territory",
"dependencyType": "enabled"
}
}
```





2.5.5 Exporting from Geoscience ANALYST

When a **ui.json** is saved with Geoscience ANALYST Pro, the following object members are updated or added:

- The **value** member with the appropriate type
- The **enabled** member `bool` for whether the parameter is enabled
- The *Data parameter* will also have updated **isValue** and **property** members. The **isValue** `bool` member is *true* if the **value** member was selected and *false* if the **property** member was selected.

The following JSON objects will be written (and overwritten if given) upon export from Geoscience ANALYST Pro:

- monitoring_directory `str` the absolute path of a monitoring directory. Workspace files written to this folder will be automatically processed by Geoscience ANALYST.
- workspace_geoh5 `str` the absolute path to the current workspace (if previously saved) being used
- geoh5 `str` the absolute path to the geoh5 written containing all the objects of the workspace within the parameters of the **ui.json**. One only needs to use this workspace along with the JSON file to access the objects with geoh5py.

2.5.6 Tips on creating UIs

Here are a few tips on creating good looking UIs:

- Keep labels short and concise. Be consistent with capitalization and do not include the colons. Geoscience ANALYST will add colons and align them.
- Write detailed tooltips.
- Group related objects, but do not use a group if there are fewer than 3 objects.
- The **main** member is for general, required parameters. Do not include this member with every object, unless there are only a handful of objects. Objects that are in the required parameters without a valid value will invoke an error when exporting or running from Geoscience ANALYST. "Non-main" members are designated to a second page under *Optional parameters*.
- Utilize **optional** object members and dependencies. If a single workspace object input is optional, use the *Object parameter* rather than two parameters with a dependency.

2.5.7 External Links

- JSON Terminology
- Universally Unique IDentifier (UUID)
- C++ JSON Library

2.6 Release Notes

2.6.1 Release 0.1.6 - 2021/12/09

- Fix StatsCache on value changes.
- Fix crash if data values are None.
- Clean up for linters

2.6.2 Release 0.1.5 - 2021/11/05

- Fix for copying of direct-current survey.
- Fix documentation.

2.6.3 Release 0.1.4 - 2021/08/31

- Add direct_current survey type and related documentation.
- Fix for drillholes with single survey location anywhere along the borehole.
- Fix for entity.parent setter. Changes are applied directly to the target workspace.
- Improve Typing.

2.7 Feedback

Have comments or suggestions? Submit feedback. All the content can be found on our github repository.

2.7.1 Contributors

- Mira Geoscience

PYTHON MODULE INDEX

g

INDEX

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z