GraphSpace Python Client Documentation

Release 1.0.0

Aditya Bharadwaj

Sep 22, 2017

Contents

1 Overview **3**
- 1.1 Why use graphspace_python? . . . 3
- 1.2 Who uses graphspace_python? . . . 3

2 Installing **5**
- 2.1 Installing with pip . . . 5
- 2.2 Installing from source . . . 5

3 Tutorial **7**
- 3.1 Connecting to GraphSpace . . . 7
- 3.2 Creating a graph . . . 7
- 3.3 Nodes . . . 7
- 3.4 Edges . . . 8
- 3.5 Graph Information . . . 9
- 3.6 Saving a graph on GraphSpace . . . 9
- 3.7 Fetching a graph from GraphSpace . . . 10
- 3.8 Updating a graph on GraphSpace . . . 10
- 3.9 Making a graph public on GraphSpace . . . 12
- 3.10 Making a graph private on GraphSpace . . . 13
- 3.11 Setting a default layout for a graph . . . 13
- 3.12 Unset default layout for a graph . . . 14
- 3.13 Deleting a graph on GraphSpace . . . 14
- 3.14 Creating a layout . . . 14
- 3.15 Node Positions . . . 15
- 3.16 Style . . . 15
- 3.17 Layout Information . . . 15
- 3.18 Saving a layout on GraphSpace . . . 15
- 3.19 Fetching a layout from GraphSpace . . . 16
- 3.20 Updating a layout on GraphSpace . . . 17
- 3.21 Deleting a layout on GraphSpace . . . 18
- 3.22 Creating a group . . . 18
- 3.23 Saving a group on GraphSpace . . . 18
- 3.24 Fetching a group from GraphSpace . . . 19
- 3.25 Updating a group on GraphSpace . . . 19
- 3.26 Fetching members of a group from GraphSpace . . . 20
- 3.27 Adding a member to a group on GraphSpace . . . 20
- 3.28 Deleting a member from a group on GraphSpace . . . 21

3.29 Fetching graphs shared with a group . . . 22
3.30 Sharing a graph with a group . . . 22
3.31 Unsharing a graph with a group . . . 23
3.32 Deleting a group on GraphSpace . . . 23
3.33 Responses . . . 24
3.34 Graphs endpoint responses . . . 24
3.35 Layouts endpoint responses . . . 24
3.36 Groups endpoint responses . . . 25
3.37 Groups member response . . . 25

4 Demos **27**

5 API Reference **29**
5.1 GraphSpace Client . . . 29
5.2 Graphs Endpoint Class . . . 30
5.3 Layouts Endpoint Class . . . 39
5.4 Groups Endpoint Class . . . 45
5.5 GSGraph Class . . . 56
5.6 GSLayout Class . . . 71
5.7 GSGroup Class . . . 78

6 Indices and tables **81**

Python Module Index **83**

Contents:

CHAPTER 1

Overview

graphspace_python is a Python client library for the GraphSpace REST API. It simplifies the process of authentication, request construction, and response parsing for Python developers using the GraphSpace API. This client library is built and tested on Python 2.7.

Why use graphspace_python?

graphspace_python allows a user to rapidly construct a network, add nodes and edges, modify their visual styles, and then upload the network, all within tens of lines of code. Moreover, the user need not know the details of the REST API to use this module. It is very easy to integrate this library into a user's software pipeline.

Who uses graphspace_python?

The potential audience for **graphspace_python** includes biologists, computer scientists and data scientists.

CHAPTER 2

Installing

Installing with pip

Install **graphspace_python** from PyPI using:

```
pip install graphspace_python
```

You can also get **graphspace_python** from the Python Package Index manually at http://pypi.python.org/pypi/graphspace_python. To use pip, you need to have setuptools installed.

Installing from source

You can install from source by downloading a source archive file (tar.gz or zip) or by checking out the source files from the Git source code repository.

graphspace_python is a pure Python package; you don't need a compiler to build or install it.

Source archive file

1. Download the source (tar.gz or zip file) from https://pypi.python.org/pypi/graphspace_python or get the latest development version from https://github.com/adbharadwaj/graphspace-python.
2. Unpack and change directory to the source directory (it should have the files requirements.txt and setup.py).
3. Run `python setup.py install` to build and install.

GitHub

1. Clone the **graphspace_python** repository (see https://github.com/adbharadwaj/graphspace-python for options).

```
git clone https://github.com/adbharadwaj/graphspace-python.git
```

2. Change directory to `graphspace-python`.
3. Run `python setup.py install` to build and install.

CHAPTER 3

Tutorial

graphspace-python allows user to do the following operations:

Start here to begin working with *graphspace-python*.

Connecting to GraphSpace

You can connect to GraphSpace using your username and password.

```
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
```

You can also set the api host using the *set_api_host()* method if you are using a different server.

```
>>> graphspace.set_api_host('localhost:8000')
```

Creating a graph

Create an empty graph with no nodes and no edges.

```
>>> from graphspace_python.graphs.classes.gsgraph import GSGraph
>>> G = GSGraph()
```

Nodes

You can add one node at a time using the *add_node()* method.

```
>>> # Adding a node 'a' with a given popup and label
>>> G.add_node('a', popup='sample node popup text', label='A')
>>> G.nodes(data=True)
[('a', {'id': 'a', 'popup': 'sample node popup text', 'name': 'a',
'label': 'A'})]
>>> # Adding style information for node 'a'
>>> G.add_node_style('a', shape='ellipse', color='red', width=90, height=90)
>>> G.get_style_json()
{'style': [{'style': {'border-color': '#000000', 'border-width': 1, 'height': 90,
'width': 90, 'shape': 'ellipse', 'border-style': 'solid', 'text-wrap': 'wrap',
'text-halign': 'center', 'text-valign': 'center', 'background-color': 'red'},
'selector': 'node[name="a"]'}]}
```

```
>>> # Adding a node 'b' with a given popup and label
>>> G.add_node('b', popup='sample node popup text', label='B')
>>> G.nodes(data=True)
[('a', {'id': 'a', 'popup': 'sample node popup text', 'name': 'a',
'label': 'A'}), ('b', {'id': 'b', 'popup': 'sample node popup text',
'name': 'b', 'label': 'B'})]
>>> # Adding style information for node 'b'
>>> G.add_node_style('b', shape='ellipse', color='blue', width=40, height=40)
>>> G.get_style_json()
{'style': [{'style': {'border-color': '#000000', 'border-width': 1, 'height': 90,
'width': 90, 'shape': 'ellipse', 'border-style': 'solid', 'text-wrap': 'wrap',
'text-halign': 'center', 'text-valign': 'center', 'background-color': 'red'},
'selector': 'node[name="a"]'}, {'style': {'border-color': '#000000', 'border-width'
: 1, 'height': 40, 'width': 40, 'shape': 'ellipse', 'border-style': 'solid',
'text-wrap': 'wrap', 'text-halign': 'center', 'text-valign': 'center', 'background-
color': 'blue'}, 'selector': 'node[name="b"]'}]}
```

Edges

You can also add one edge at a time using the *add_edge()* method.

```
>>> G.add_edge('a', 'b', directed=True, popup='sample edge popup')
>>> # Accessing edges
>>> G.edges(data=True)
[('a', 'b', {'source': 'a', 'popup': 'sample edge popup', 'is_directed':
True, 'target': 'b'})]
>>> # Direct access using subscript notation
>>> G['a']
{'b': {'source': 'a', 'popup': 'sample edge popup', 'is_directed': True,
'target': 'b'}}
>>> G['a']['b']
{'source': 'a', 'popup': 'sample edge popup', 'is_directed': True,
'target': 'b'}
>>> # Adding style information for edge
>>> G.add_edge_style('a', 'b', directed=True, edge_style='dotted')
>>> G.get_style_json()
{'style': [{'style': {'border-color': '#000000', 'border-width': 1, 'height': 90,
'width': 90, 'shape': 'ellipse', 'border-style': 'solid', 'text-wrap': 'wrap',
'text-halign': 'center', 'text-valign': 'center', 'background-color': 'red'},
'selector': 'node[name="a"]'}, {'style': {'border-color': '#000000', 'border-width'
: 1, 'height': 40, 'width': 40, 'shape': 'ellipse', 'border-style': 'solid',
'text-wrap': 'wrap', 'text-halign': 'center', 'text-valign': 'center', 'background-
```

```
color': 'blue'}, 'selector': 'node[name="b"]'}, {'style': {'width': 1.0, 'line-
color': '#000000', 'target-arrow-shape': 'triangle', 'line-style': 'dotted',
'target-arrow-fill': 'filled', 'target-arrow-color': '#000000'}, 'selector':
'edge[source="a"][target="b"]'}]}
```

Graph Information

You can add more meaningful information about the graph like name, description and tags.

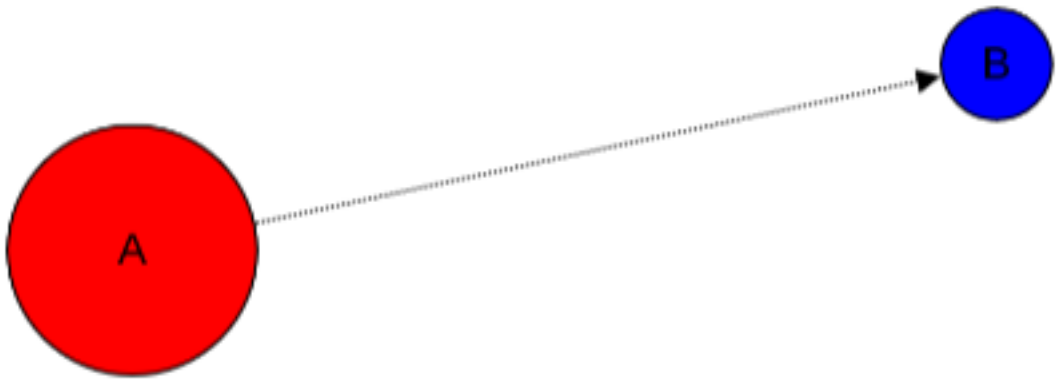
```
>>> G.set_name('My Sample Graph')
>>> G.get_name()
'My Sample Graph'
>>> G.set_tags(['sample'])
>>> G.get_tags()
['sample']
>>> G.set_data(data={
...     'description': 'my sample graph'
... })
>>> G.get_data()
{'description': 'my sample graph', 'name': 'My Sample Graph', 'tags': ['sample']}
```

Saving a graph on GraphSpace

You can save your graph online using the *post_graph()* method.

```
>>> graph = graphspace.post_graph(G)
>>> graph.get_name()
u'My Sample Graph'
>>> graph.id
29824
```

The saved graph will look like this on GraphSpace:



Fetching a graph from GraphSpace

You can retrieve your saved graph anytime from GraphSpace using the *get_graph()* method.

```
>>> graph = graphspace.get_graph(graph_name='My Sample Graph')
>>> graph.get_name()
u'My Sample Graph'
>>> graph.id
29824
>>> graph.get_is_public()
0
>>> graph.get_graph_json()
{u'elements': {u'nodes': [{u'data': {u'popup': u'sample node popup text', u'name':
u'a', u'id': u'a', u'label': u'A'}}, {u'data': {u'popup': u'sample node popup
text', u'name': u'b', u'id': u'b', u'label': u'B'}}], u'edges': [{u'is_directed':
0, u'data': {u'source': u'a', u'popup': u'sample edge popup', u'is_directed': True,
u'target': u'b', u'name': u'a-b'}}]}, u'data': {u'tags': [u'sample'], u'description'
: u'my sample graph', u'name': u'My Sample Graph'}}
>>> graph.get_style_json()
{u'style': [{u'style': {u'border-color': u'#000000', u'border-style': u'solid',
u'border-width': 1, u'width': 90, u'shape': u'ellipse', u'text-wrap': u'wrap',
u'text-halign': u'center', u'height': 90, u'text-valign': u'center', u'background-
color': u'red'}, u'selector': u'node[name="a"]'}, {u'style': {u'border-color':
u'#000000', u'border-style': u'solid', u'border-width': 1, u'width': 40, u'shape':
u'ellipse', u'text-wrap': u'wrap', u'text-halign': u'center', u'height': 40,
u'text-valign': u'center', u'background-color': u'blue'}, u'selector': u'node
[name="b"]'}, {u'style': {u'line-color': u'#000000', u'target-arrow-shape':
u'triangle', u'target-arrow-fill': u'filled', u'width': 1.0, u'line-style':
u'dotted', u'target-arrow-color': u'#000000'}, u'selector': u'edge[source="a"]
[target="b"]'}]}
```

You can retrieve a graph by id as well.

```
>>> graph = graphspace.get_graph(graph_id=29824)
```

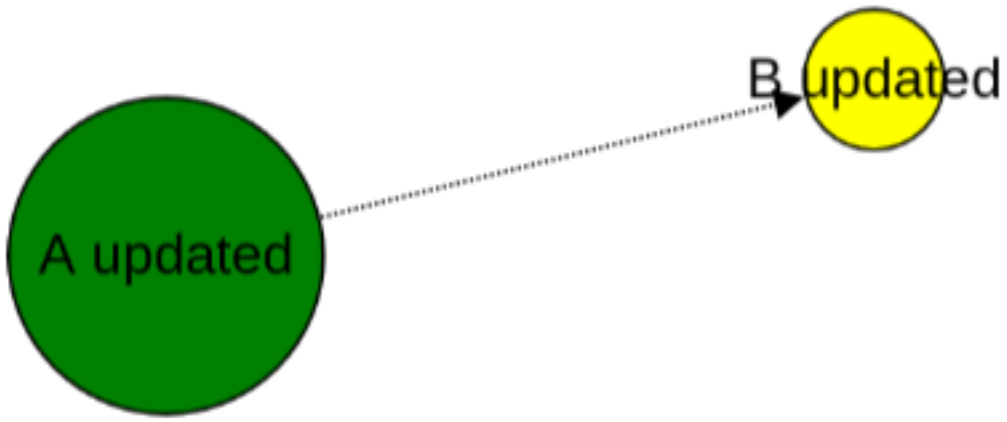
Updating a graph on GraphSpace

You can also update your graph anytime using the *update_graph()* method.

```
>>> G = GSGraph()
>>> G.add_node('a', popup='sample node popup text', label='A updated')
>>> G.add_node_style('a', shape='ellipse', color='green', width=90, height=90)
>>> G.add_node('b', popup='sample node popup text', label='B updated')
>>> G.add_node_style('b', shape='ellipse', color='yellow', width=40, height=40)
>>> G.add_edge('a', 'b', directed=True, popup='sample edge popup')
>>> G.add_edge_style('a', 'b', directed=True, edge_style='dotted')
>>> G.set_name('My Sample Graph')
>>> G.set_data(data={
...     'description': 'my sample graph'
... })
>>> G.set_is_public(1)
>>> graph = graphspace.update_graph(G)
>>> graph.get_name()
u'My Sample Graph'
>>> graph.get_is_public()
```

```
1
>>> graph.get_data()
{u'description': u'my sample graph', 'name': u'My Sample Graph', 'tags': [u'sample']}
>>> graph.get_graph_json()
{'elements': {'nodes': [{u'data': {u'popup': u'sample node popup text', u'name':
u'a', u'id': u'a', u'label': u'A updated'}}, {u'data': {u'popup': u'sample node
popup text', u'name': u'b', u'id': u'b', u'label': u'B updated'}}], 'edges': [{
u'data': {u'source': u'a', u'popup': u'sample edge popup', u'is_directed': True,
u'target': u'b', u'name': u'a-b'}, u'is_directed': 0}]}, 'data': {u'description':
u'my sample graph', 'name': u'My Sample Graph', 'tags': [u'sample']}}
>>> graph.get_style_json()
{u'style': [{u'style': {u'border-color': u'#000000', u'border-style': u'solid',
u'border-width': 1, u'width': 90, u'shape': u'ellipse', u'text-wrap': u'wrap',
u'text-halign': u'center', u'height': 90, u'text-valign': u'center', u'background-
color': u'green'}, u'selector': u'node[name="a"]'}, {u'style': {u'border-color':
u'#000000', u'border-style': u'solid', u'border-width': 1, u'width': 40, u'shape':
u'ellipse', u'text-wrap': u'wrap', u'text-halign': u'center', u'height': 40,
u'text-valign': u'center', u'background-color': u'yellow'}, u'selector': u'node
[name="b"]'}, {u'style': {u'line-color': u'#000000', u'target-arrow-shape':
u'triangle', u'target-arrow-fill': u'filled', u'width': 1.0, u'line-style':
u'dotted', u'target-arrow-color': u'#000000'}, u'selector': u'edge[source="a"]
[target="b"]'}]}
```

The updated graph will look like this on GraphSpace:

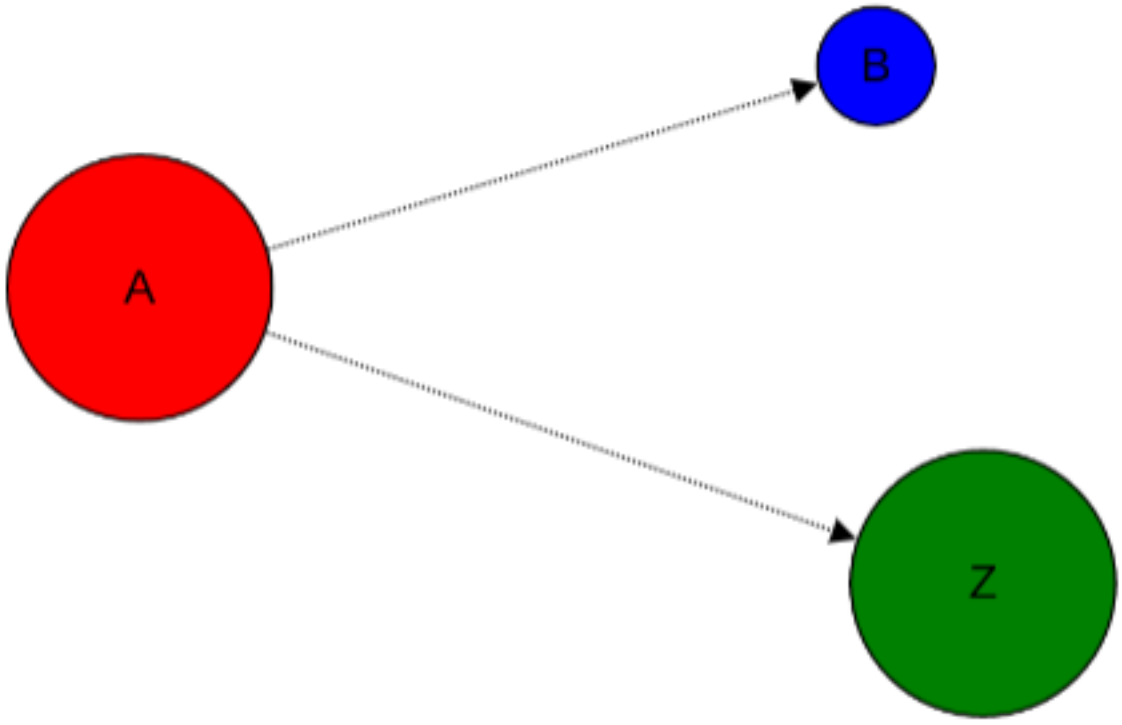


Here is another example.

```
>>> # Retrieving graph
>>> graph = graphspace.get_graph(graph_name='My Sample Graph')
>>> # Modifying the retrieved graph object
>>> graph.add_node('z', popup='sample node popup text', label='Z')
>>> graph.add_node_style('z', shape='ellipse', color='green', width=90, height=90)
>>> graph.add_edge('a', 'z', directed=True, popup='sample edge popup')
>>> graph.add_edge_style('a', 'z', directed=True, edge_style='dotted')
>>> graph.set_is_public(1)
>>> # Updating graph
>>> graph1 = graphspace.update_graph(graph)
>>> graph1.get_name()
u'My Sample Graph'
>>> graph1.get_is_public()
```

```
1
>>> graph1.nodes(data=True)
[(u'a', {u'popup': u'sample node popup text', u'name': u'a', u'id': u'a',
u'label': u'A'}), (u'b', {u'popup': u'sample node popup text', u'name':
u'b', u'id': u'b', u'label': u'B'}), (u'z', {u'popup': u'sample node
popup text', u'name': u'z', u'id': u'z', u'label': u'Z'})]
>>> graph1.edges(data=True)
[(u'a', u'b', {u'source': u'a', u'popup': u'sample edge popup',
u'is_directed': True, u'target': u'b', u'name': u'a-b'}),
(u'a', u'z', {u'source': u'a', u'popup': u'sample edge popup',
u'is_directed': True, u'target': u'z', u'name': u'a-z'})]
```

The updated graph in this case will look like this on GraphSpace:



If you also provide 'graph_name' or 'graph_id' as param then the update will be performed for that graph having the given name or id:

```
>>> graph = graphspace.update_graph(G, graph_id=29824)
```

Making a graph public on GraphSpace

You can also make a graph public using the *publish_graph()* method.

```
>>> graphspace.publish_graph(graph_name='My Sample Graph')
>>> assert graphspace.get_graph(graph_name='My Sample Graph').is_public == 1
```

You can make a graph public by id as well.

```
>>> graphspace.publish_graph(graph_id=29824)
```

You can also make a graph public by passing the graph object itself as param.

```
>>> graph = graphspace.get_graph(graph_name='My Sample Graph')
>>> graphspace.publish_graph(graph=graph)
```

Making a graph private on GraphSpace

You can also make a graph private using the *unpublish_graph()* method.

```
>>> graphspace.unpublish_graph(graph_name='My Sample Graph')
>>> assert graphspace.get_graph(graph_name='My Sample Graph').is_public == 0
```

You can make a graph private by id as well.

```
>>> graphspace.unpublish_graph(graph_id=29824)
```

You can also make a graph private by passing the graph object itself as param.

```
>>> graph = graphspace.get_graph(graph_name='My Sample Graph')
>>> graphspace.unpublish_graph(graph=graph)
```

Setting a default layout for a graph

You can set a default layout for a graph using the *set_default_graph_layout()* method.

```
>>> graph = graphspace.set_default_graph_layout(graph_name='My Sample Graph', layout_
↪id=1087)
>>> graph.default_layout_id
1087
```

You can set a default layout for a graph by graph id as well.

```
>>> graph = graphspace.set_default_graph_layout(graph_id=65930, layout_id=1087)
>>> graph.default_layout_id
1087
```

You can set a default layout for a graph by passing graph object itself as param.

```
>>> graph = graphspace.get_graph(graph_name='My Sample Graph')
>>> graph = graphspace.set_default_graph_layout(graph=graph, layout_id=1087)
>>> graph.default_layout_id
1087
```

Similarly you can use layout name instead of id.

```
>>> graph = graphspace.set_default_graph_layout(graph_id=65930, layout_name='My␣
↪Sample Layout')
>>> graph.default_layout_id
1087
```

Or you can only pass layout object provided the object has 'graph_id' attribute and layout 'name' or 'id' attribute as well.

```
>>> layout = graphspace.get_graph_layout(graph_id=65930, layout_name='My Sample Layout
↪')
>>> graph = graphspace.set_default_graph_layout(layout=layout)
>>> graph.default_layout_id
1087
```

Unset default layout for a graph

You can unset default layout for a graph using the *unset_default_graph_layout()* method.

```
>>> graph = graphspace.unset_default_graph_layout(graph_name='My Sample Graph')
>>> assert graph.default_layout_id is None
```

You can unset default layout for a graph by graph id as well.

```
>>> graph = graphspace.unset_default_graph_layout(graph_id=65930)
```

You can also pass the graph object itself as param.

```
>>> graph = graphspace.get_graph(graph_name='My Sample Graph')
>>> graph = graphspace.unset_default_graph_layout(graph=graph)
```

Deleting a graph on GraphSpace

You can also delete your graph anytime using the *delete_graph()* method.

```
>>> graphspace.delete_graph(graph_name='My Sample Graph')
u'Successfully deleted graph with id=29824'
>>> assert graphspace.get_graph(graph_name='My Sample Graph') is None
```

You can delete a graph by id as well.

```
>>> graphspace.delete_graph(graph_id=29824)
u'Successfully deleted graph with id=29824'
```

You can also delete a graph by passing the graph object itself as param.

```
>>> graph = graphspace.get_graph(graph_name='My Sample Graph')
>>> graphspace.delete_graph(graph=graph)
u'Successfully deleted graph with id=29824'
```

Creating a layout

Create an empty layout with no node positions and style properties.

```
>>> from graphspace_python.graphs.classes.gslayout import GSLayout
>>> L = GSLayout()
```

Node Positions

You can set position of one node at a time using the *set_node_position()* method.

```
>>> # Setting position of a node 'a' with y and x coordinates
>>> L.set_node_position('a', y=38.5, x=67.3)
>>> # Setting position of a node 'b' with y and x coordinates
>>> L.set_node_position('b', y=124, x=332.2)
>>> L.get_positions_json()
{'a': {'y': 38.5, 'x': 67.3}, 'b': {'y': 124, 'x': 332.2}}
```

Note: Setting position of an already present node will update its position.

Style

You can also add style for a node or an edge by using the *add_node_style()* and *add_edge_style()* methods.

```
>>> L.add_node_style('a', shape='ellipse', color='green', width=60, height=60)
>>> L.add_edge_style('a', 'b', directed=True, edge_style='dashed')
>>> L.get_style_json()
{'style': [{'style': {'border-color': '#000000', 'border-width': 1, 'height': 60,
'width': 60, 'shape': 'ellipse', 'border-style': 'solid', 'text-wrap': 'wrap',
'text-halign': 'center', 'text-valign': 'center', 'background-color': 'green'},
'selector': 'node[name="a"]'}, {'style': {'width': 1.0, 'line-color': '#000000',
'target-arrow-shape': 'triangle', 'line-style': 'dashed', 'target-arrow-fill':
'filled', 'target-arrow-color': '#000000'}, 'selector': 'edge[source="a"][target="b"]
↪'}]}
```

Layout Information

You can add more meaningful information about the layout like name, sharing status.

```
>>> L.set_name('My Sample Layout')
>>> L.get_name()
'My Sample Layout'
>>> L.set_is_shared(1)
>>> L.get_is_shared()
1
```

Saving a layout on GraphSpace

You can save your layout online using the *post_graph_layout()* method.

```
>>> layout = graphspace.post_graph_layout(L, graph_id=21722)
>>> layout.get_name()
u'My Sample Layout'
```

```
>>> layout.id
1068
```

The saved layout will look like this on GraphSpace:

You can also save your layout when graph name is known.

```
>>> layout = graphspace.post_graph_layout(L, graph_name='My Sample Graph')
```

You can also save your layout by passing graph object as param.

```
>>> graph = graphspace.get_graph(graph_name='My Sample Graph')
>>> layout = graphspace.post_graph_layout(L, graph=graph)
```

Fetching a layout from GraphSpace

You can retrieve your saved layout anytime from GraphSpace using the *get_graph_layout()* method.

```
>>> layout = graphspace.get_graph_layout(layout_name='My Sample Layout', graph_
↪id=21722)
>>> layout.get_name()
u'My Sample Layout'
>>> layout.id
1068
>>> layout.get_is_shared()
1
>>> layout.get_positions_json()
{u'a': {u'y': 38.5, u'x': 67.3}, u'b': {u'y': 124, u'x': 332.2}}
>>> layout.get_style_json()
{u'style': [{u'style': {u'border-color': u'#000000', u'border-width': 1, u'height':
60, u'shape': u'ellipse', u'width': 60, u'border-style': u'solid', u'text-wrap':
u'wrap', u'text-halign': u'center', u'text-valign': u'center', u'background-color':
u'green'}, u'selector': u'node[name="a"]'}, {u'style': {u'line-color': u'#000000',
u'target-arrow-shape': u'triangle', u'target-arrow-fill': u'filled', u'width': 1.0,
u'line-style': u'dashed', u'target-arrow-color': u'#000000'}, u'selector':
u'edge[source="a"][target="b"]'}]}
```

You can retrieve a layout by id as well.

```
>>> layout = graphspace.get_graph_layout(layout_id=1068, graph_id=21722)
```

You can also retrieve a layout by providing graph name instead of id.

```
>>> layout = graphspace.get_graph_layout(layout_id=1068, graph_name='My Sample Graph')
```

You can also retrieve a layout by passing the graph object as param.

```
>>> graph = graphspace.get_graph(graph_name='My Sample Graph')
>>> layout = graphspace.get_graph_layout(layout_id=1068, graph=graph)
```

Updating a layout on GraphSpace

You can also update your layout anytime using the *update_graph_layout()* method.

```
>>> L = GSLayout()
>>> L.set_node_position('b', y=38.5, x=67.3)
>>> L.set_node_position('a', y=102, x=238.1)
>>> L.add_node_style('a', shape='octagon', color='green', width=60, height=60)
>>> L.add_edge_style('a', 'b', directed=True, edge_style='solid')
>>> L.set_name('My Sample Layout')
>>> L.set_is_shared(1)
>>> layout = graphspace.update_graph_layout(L, graph_id=21722)
>>> layout.get_name()
u'My Sample Layout'
>>> layout.get_is_shared()
1
>>> layout.get_positions_json()
{u'a': {u'y': 102, u'x': 238.1}, u'b': {u'y': 38.5, u'x': 67.3}}
>>> layout.get_style_json()
{u'style': [{u'style': {u'border-color': u'#000000', u'border-width': 1, u'height':
60, u'shape': u'octagon', u'width': 60, u'border-style': u'solid', u'text-wrap':
u'wrap', u'text-halign': u'center', u'text-valign': u'center', u'background-color':
u'green'}, u'selector': u'node[name="a"]'}, {u'style': {u'line-color': u'#000000',
u'target-arrow-shape': u'triangle', u'target-arrow-fill': u'filled', u'width': 1.0,
u'line-style': u'solid', u'target-arrow-color': u'#000000'}, u'selector':
u'edge[source="a"][target="b"]'}]}
```

The updated layout will look like this on GraphSpace:

Here is another example.

```
>>> # Retrieving layout
>>> layout = graphspace.get_graph_layout(graph_id=21722, name='My Sample Layout')
>>> # Modifying the retrieved layout object
>>> layout.set_node_position('b', y=30, x=67)
>>> layout.set_node_position('a', y=30, x=211)
>>> layout.add_node_style('a', shape='roundrectangle', color='green', width=45,␣
↪height=45)
>>> layout.add_edge_style('a', 'b', directed=True, edge_style='solid')
>>> # Updating layout
>>> layout1 = graphspace.update_graph_layout(layout)
>>> layout1.get_positions_json()
{u'a': {u'y': 30, u'x': 211}, u'b': {u'y': 30, u'x': 67}}
```

The updated layout in this case will look like this on GraphSpace:

If you also provide 'layout_name' or 'layout_id' as param then the update will be performed for that layout having the given name or id:

```
>>> layout = graphspace.update_graph_layout(L, layout_id=1068, graph_id=21722)
```

Deleting a layout on GraphSpace

You can also delete your layout anytime using the *delete_graph_layout()* method.

```
>>> graphspace.delete_graph_layout(layout_name='My Sample Layout', graph_id=21722)
u'Successfully deleted layout with id=1068'
>>> assert graphspace.get_graph_layout(graph_id=21722, name='My Sample Layout') is␣
↪None
```

You can delete a layout by id as well.

```
>>> graphspace.delete_graph_layout(layout_id=1068, graph_id=21722)
u'Successfully deleted layout with id=1068'
```

You can also delete a layout by passing only the layout object as param provided the object has 'graph_id' attribute and layout 'name' or 'id' attribute.

```
>>> layout = graphspace.get_graph_layout(layout_name='My Sample Layout', graph_
↪id=21722)
>>> graphspace.delete_graph_layout(layout=layout)
u'Successfully deleted layout with id=1068'
```

You can also use graph name instead of id.

```
>>> graphspace.delete_graph_layout(layout_id=1068, graph_name='My Sample Graph')
u'Successfully deleted layout with id=1068'
```

Or you can also pass the graph object as param.

```
>>> graph = graphspace.get_graph(graph_name='My Sample Graph')
>>> graphspace.delete_graph_layout(layout_id=1068, graph=graph)
u'Successfully deleted layout with id=1068'
```

Creating a group

Create a group providing the name and description.

```
>>> from graphspace_python.graphs.classes.gsgroup import GSGroup
>>> group = GSGroup(name='My first group', description='sample group')
```

You can also set name and description of the group using the *set_name()* and *set_description()* methods.

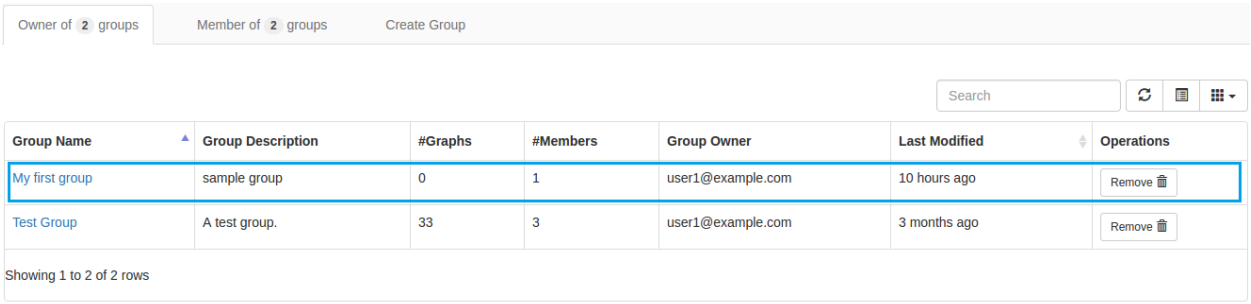
```
>>> group = GSGroup()
>>> group.set_name('My first group')
>>> group.get_name()
'My first group'
>>> group.set_description('sample group')
>>> group.get_description()
'sample group'
```

Saving a group on GraphSpace

You can save your group online using the *post_group()* method.

```
>>> group1 = graphspace.post_group(group)
>>> group1.get_name()
u'My first group'
>>> group1.id
318
```

You can also view your saved group on GraphSpace.



Fetching a group from GraphSpace

You can retrieve your saved group anytime from GraphSpace using the *get_group()* method.

```
>>> group = graphspace.get_group(group_name='My first group')
>>> group.get_name()
u'My first group'
>>> group.id
318
>>> group.get_description()
u'sample group'
```

You can retrieve a group by id as well.

```
>>> group = graphspace.get_group(group_id=318)
```

Updating a group on GraphSpace

You can also update your group anytime using the *update_group()* method.

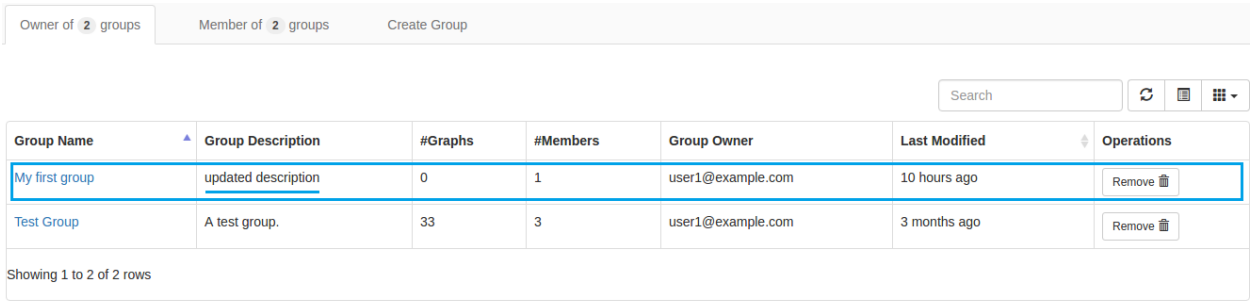
```
>>> group = GSGroup(name='My first group', description='updated description')
>>> group1 = graphspace.update_group(group)
>>> group1.get_description()
u'updated description'
```

Here is another example.

```
>>> group = graphspace.get_group(name='My first group')
>>> group.set_description('updated description')
>>> group1 = graphspace.update_group(group)
```

```
>>> group1.get_description()
u'updated description'
```

You can also view your updated group on GraphSpace.



If you also provide 'group_name' or 'group_id' as param then the update will be performed for that group having the given name or id:

```
>>> group1 = graphspace.update_group(group, group_id=198)
```

Fetching members of a group from GraphSpace

You can retrieve the members of your group anytime using the *get_group_members()* method.

```
>>> members = graphspace.get_group_members(group_name='My first group')
>>> members[0].email
u'user1@example.com'
```

You can retrieve group members by group_id as well.

```
>>> members = graphspace.get_group_members(group_id=318)
>>> members[0].email
u'user1@example.com'
```

You can also retrieve members of a group by passing the group object itself as param.

```
>>> group = graphspace.get_group(group_name='My first group')
>>> members = graphspace.get_group_members(group=group)
>>> members[0].email
u'user1@example.com'
```

Adding a member to a group on GraphSpace

You can add a member to your group anytime using the *add_group_member()* method.

```
>>> response = graphspace.add_group_member(member_email='user3@example.com', group_
↪name='My first group')
>>> response['user_id']
2
```

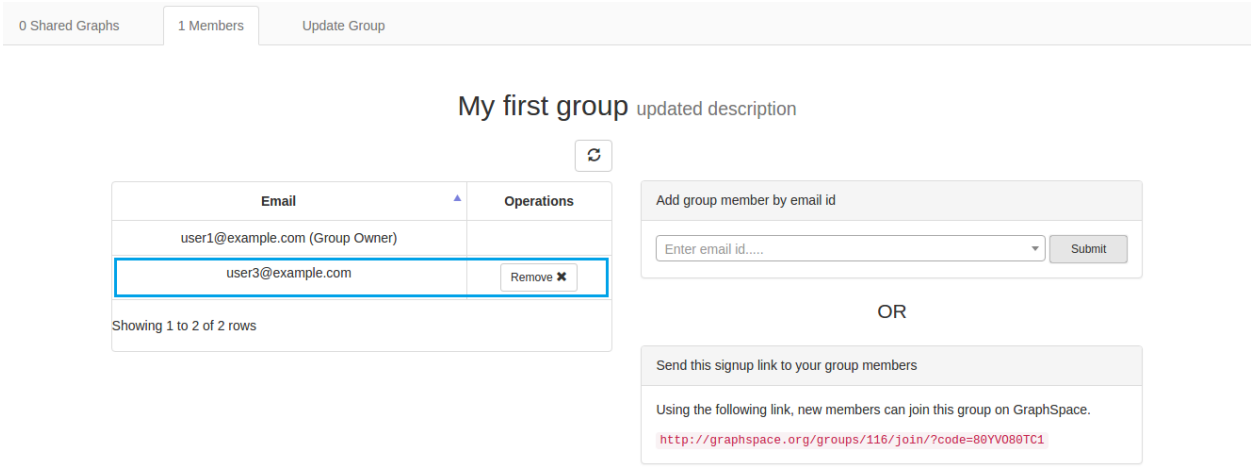
You can add a group member by group_id as well.

```
>>> graphspace.add_group_member(member_email='user3@example.com', group_id=318)
{u'group_id': u'318', u'user_id': 2}
```

You can also add a group member to a group by passing the group object as param.

```
>>> group = graphspace.get_group(group_name='My first group')
>>> graphspace.add_group_member(member_email='user3@example.com', group=group)
{u'group_id': u'318', u'user_id': 2}
```

You can also view the added member on GraphSpace.



Deleting a member from a group on GraphSpace

You can delete a member from your group anytime using the *delete_group_member()* method.

```
>>> graphspace.delete_group_member(member_id=2, group_name='My first group')
u'Successfully deleted member with id=2 from group with id=318'
```

You can delete a group member by group_id as well.

```
>>> graphspace.delete_group_member(member_id=2, group_id=318)
u'Successfully deleted member with id=2 from group with id=318'
```

You can also delete a group member by passing the group object as param.

```
>>> group = graphspace.get_group(group_name='My first group')
>>> graphspace.delete_group_member(member_id=2, group=group)
u'Successfully deleted member with id=2 from group with id=318'
```

Or you can also pass the member object directly.

```
>>> members = graphspace.get_group_members(group_name='My first group')
>>> graphspace.delete_group_member(member=members[0], group_name='My first group')
u'Successfully deleted member with id=2 from group with id=318'
```

Fetching graphs shared with a group

You can retrieve the graphs shared with your group anytime using the *get_group_graphs()* method.

```
>>> graphs = graphspace.get_group_graphs(group_name='My first group')
>>> graphs[0].get_name()
u'My Sample Graph'
```

You can retrieve graphs shared with a group by group_id as well.

```
>>> graphs = graphspace.get_group_graphs(group_id=318)
>>> graphs[0].get_name()
u'My Sample Graph'
```

You can also retrieve the shared graphs by passing the group object itself as param.

```
>>> group = graphspace.get_group(group_name='My first group')
>>> graphs = graphspace.get_group_graphs(group=group)
>>> graphs[0].get_name()
u'My Sample Graph'
```

Sharing a graph with a group

You can share a graph with your group anytime using the *share_graph()* method.

```
>>> response = graphspace.share_graph(graph_id=34786, group_name='My first group')
>>> response['graph_id']
34786
```

You can share a graph with a group by group_id as well.

```
>>> graphspace.share_graph(graph_id=34786, group_id=318)
{u'created_at': u'2017-07-20T18:40:36.267052', u'group_id': u'318', u'graph_id':
34786, u'updated_at': u'2017-07-20T18:40:36.267052'}
```

You can also share a graph with a group by passing the group object as param.

```
>>> group = graphspace.get_group(group_name='My first group')
>>> graphspace.share_graph(graph_id=34786, group=group)
{u'created_at': u'2017-07-20T18:40:36.267052', u'group_id': u'318', u'graph_id':
34786, u'updated_at': u'2017-07-20T18:40:36.267052'}
```

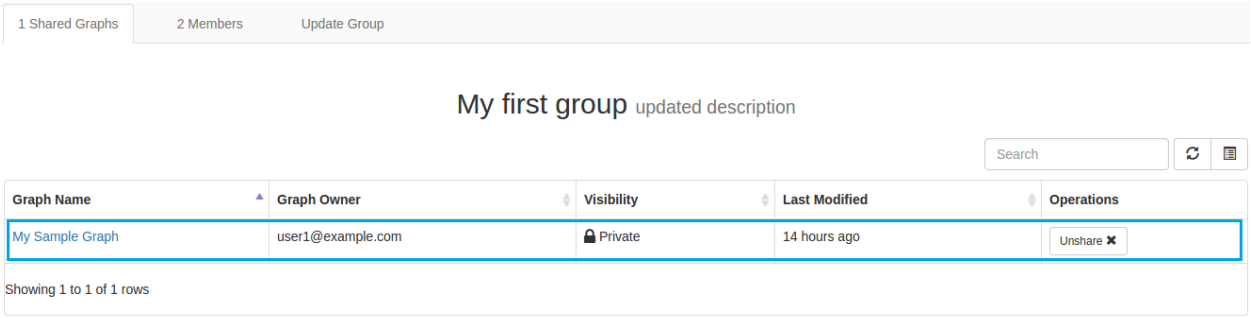
You can also provide the graph name instead of id for sharing.

```
>>> graphspace.share_graph(graph_name='My Sample Graph', group_id=318)
{u'created_at': u'2017-07-20T18:40:36.267052', u'group_id': u'318', u'graph_id':
34786, u'updated_at': u'2017-07-20T18:40:36.267052'}
```

Or you can provide the graph object itself as param.

```
>>> graph = graphspace.get_graph(graph_name='My Sample Graph')
>>> graphspace.share_graph(graph=graph, group_id=318)
{u'created_at': u'2017-07-20T18:40:36.267052', u'group_id': u'318', u'graph_id':
34786, u'updated_at': u'2017-07-20T18:40:36.267052'}
```

You can also view the shared graph on GraphSpace.



Unsharing a graph with a group

You can unshare a graph with your group anytime using the *unshare_graph()* method.

```
>>> graphspace.unshare_graph(graph_id=34786, group_name='My first group')
u'Successfully deleted graph with id=34786 from group with id=318'
```

You can unshare a graph with a group by group_id as well.

```
>>> graphspace.unshare_graph(graph_id=34786, group_id=318)
u'Successfully deleted graph with id=34786 from group with id=318'
```

You can also unshare a graph with a group by passing the group object as param.

```
>>> group = graphspace.get_group(group_name='My first group')
>>> graphspace.unshare_graph(graph_id=34786, group=group)
u'Successfully deleted graph with id=34786 from group with id=318'
```

You can also provide the graph name instead of id for unsharing.

```
>>> graphspace.unshare_graph(graph_name='My Sample Graph', group_id=318)
u'Successfully deleted graph with id=34786 from group with id=318'
```

Or you can provide the graph object itself as param.

```
>>> graph = graphspace.get_graph(graph_name='My Sample Graph')
>>> graphspace.unshare_graph(graph=graph, group_id=318)
u'Successfully deleted graph with id=34786 from group with id=318'
```

Deleting a group on GraphSpace

You can also delete your group anytime using the *delete_group()* method.

```
>>> graphspace.delete_group(group_name='My first group')
u'Successfully deleted group with id=318'
>>> assert graphspace.get_group(group_name='My first group') is None
```

You can delete a group by id as well.

```
>>> graphspace.delete_group(group_id=318)
u'Successfully deleted group with id=318'
```

You can also delete a group by passing the group object itself as param.

```
>>> group = graphspace.get_group(group_name='My first group')
>>> graphspace.delete_group(group=group)
u'Successfully deleted group with id=318'
```

Responses

Responses from the API are parsed into the respective object types.

Graphs endpoint responses

When response is a single `Graph` object:

```
>>> graph = graphspace.get_graph('My Sample Graph')
>>> graph.get_name()
u'My Sample Graph'
```

When response is a list of `Graph` objects:

```
>>> graphs = graphspace.get_my_graphs()
>>> graphs
[<Graph 1>, <Graph 2>, ...]
>>> graphs[0].get_name()
u'My Sample Graph'
```

Layouts endpoint responses

When response is a single `Layout` object:

```
>>> layout = graphspace.get_graph_layout(graph_id=21722, name='My Sample Layout')
>>> layout.get_name()
u'My Sample Layout'
```

When response is a list of `Layout` objects:

```
>>> layouts = graphspace.get_my_graph_layouts(graph_id=21722)
>>> layouts
[<Layout 1>, <Layout 2>, ...]
>>> layouts[0].get_name()
u'My Sample Layout'
```

Groups endpoint responses

When response is a single `Group` object:

```
>>> group = graphspace.get_group(name='My first group')
>>> group.get_name()
u'My first group'
```

When response is a list of `Group` objects:

```
>>> groups = graphspace.get_my_groups()
>>> groups
[<Group 1>, <Group 2>, ...]
>>> groups[0].get_name()
u'My first group'
```

Groups member response

Group member response is a list of `Member` objects.

```
>>> members = graphspace.get_group_members(name='My first group')
>>> members
[<Member 1>, <Member 2>, ...]
>>> members[0].email
u'user1@example.com'
```

CHAPTER 4

Demos

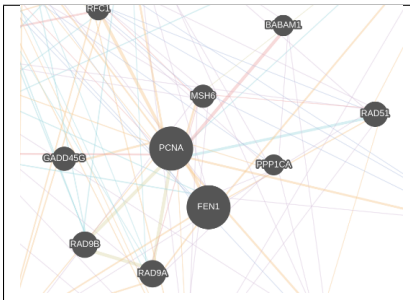


Fig. 4.1: Gene-gene Graph [GitHub]

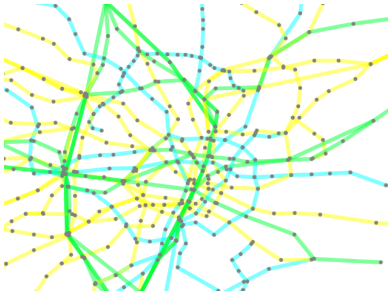


Fig. 4.2: Tokyo Railways [GitHub]

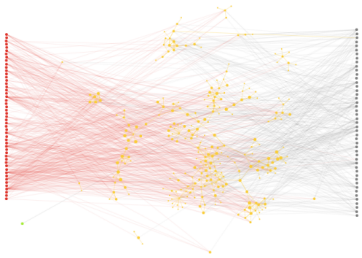


Fig. 4.3: Wine & Cheese [GitHub]

Fig. 4.4: Popup [GitHub]

Fig. 4.5: Circle Layout [GitHub]

Fig. 4.6: Concentric Layout [GitHub]

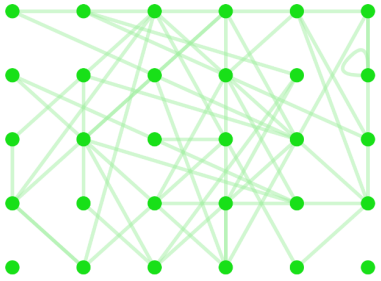


Fig. 4.7: Grid Layout [GitHub]

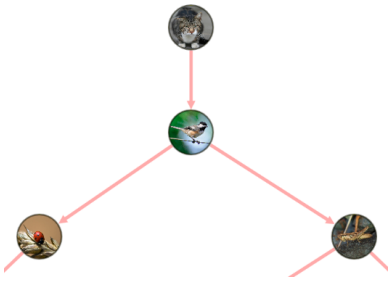


Fig. 4.8: Image Graph [GitHub]

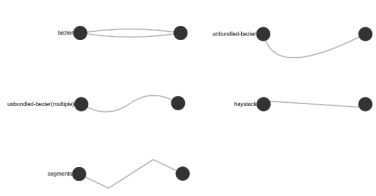
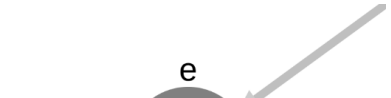


Fig. 4.9: Edge Types [GitHub]



CHAPTER 5

API Reference

GraphSpace Client

class graphspace_python.api.client.**GraphSpace**(*username*, *password*)

Bases: object

GraphSpace client class.

Connects the user with GraphSpace.

Makes request to the GraphSpace REST APIs and returns the response.

Parameters

- **username** (*str*) – Email of the user.
- **password** (*str*) – Password of the user.

Example

Initialising a GraphSpace client can be done in the following way:

```
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
```

auth_token
str – Base64 encoded username and password.

username
str – Email of user.

api_host
str – Host address of GraphSpace REST API.

set_api_host(*host*)
Manually set host address of GraphSpace REST APIs.

Parameters **host** (*str*) – Host address of GraphSpace APIs.

Example

```
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> graphspace.set_api_host('localhost:8000')
```

Graphs Endpoint Class

class graphspace_python.api.endpoint.graphs.**Graphs**(*client*)

Bases: object

Graphs endpoint class.

Provides methods for graph related operations such as saving, fetching, updating and deleting graphs on GraphSpace.

delete_graph(*graph_name=None*, *graph_id=None*, *graph=None*)

Delete a graph from GraphSpace provided the graph_name, graph_id or the graph object itself.

Parameters

- **graph_name** (*str, optional*) – Name of the graph to be deleted. Defaults to None.
- **graph_id** (*int, optional*) – ID of the graph to be deleted. Defaults to None.
- **graph** (GSGraph *or Graph, optional*) – Object having graph details, such as name, graph_json, style_json, is_public, tags. Defaults to None.

Returns Success/Error Message from GraphSpace.

Return type str

Raises

- Exception – If both 'graph_name' and 'graph_id' are None and graph object has no 'name' or 'id' attribute; or if graph doesnot exist.
- GraphSpaceError – If error response is received from the GraphSpace API.

Examples

Deleting a graph by name:

```
>>> # Connecting to GraphSpace
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> # Deleting a graph
>>> graphspace.delete_graph(graph_name='My Sample Graph')
u'Successfully deleted graph with id=65930'
```

Deleting a graph by id:

```
>>> graphspace.delete_graph(graph_id=65930)
u'Successfully deleted graph with id=65930'
```

Deleting a graph by passing graph object itself as param:

```
>>> graph = graphspace.get_graph(graph_name='My Sample Graph')
>>> graphspace.delete_graph(graph=graph)
u'Successfully deleted graph with id=65930'
```

Note: Refer to the tutorial for more about deleting graphs.

get_graph(*graph_name=None*, *graph_id=None*, *owner_email=None*)
Get a graph with the given graph_name or graph_id.

Parameters

- **graph_name** (*str, optional*) – Name of the graph to be fetched. Defaults to None.
- **graph_id** (*int, optional*) – ID of the graph to be fetched. Defaults to None.
- **owner_email** (*str, optional*) – Email of the owner of the graph. Defaults to None.

Returns Graph object, if graph with the given 'graph_name' or 'graph_id' exists; otherwise None.

Return type Graph or None

Raises

- `Exception` – If both 'graph_name' and 'graph_id' are None.
- `GraphSpaceError` – If error response is received from the GraphSpace API.

Examples

Getting a graph by name:

```
>>> # Connecting to GraphSpace
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> # Fetching a graph
>>> graph = graphspace.get_graph(graph_name='My Sample Graph')
>>> graph.get_name()
u'My Sample Graph'
```

Getting a graph by id:

```
>>> graph = graphspace.get_graph(graph_id=65930)
>>> graph.get_name()
u'My Sample Graph'
```

Note: Refer to the tutorial for more about fetching graphs.

get_my_graphs(*tags=None*, *limit=20*, *offset=0*)
Get graphs created by the requesting user.

Parameters

- **tags** (*List[str], optional*) – Search for graphs with the given given list of tag names. In order to search for graphs with given tag as a substring, wrap the name of the tag with percentage symbol. For example, %xyz% will search for all graphs with 'xyz' in their tag names. Defaults to None.
- **offset** (*int, optional*) – Offset the list of returned entities by this number. Defaults to 0.
- **limit** (*int, optional*) – Number of entities to return. Defaults to 20.

Returns List of graphs owned by the requesting user.

Return type List[Graph]

Raises GraphSpaceError – If error response is received from the GraphSpace API.

Examples

Getting your graphs:

```
>>> # Connecting to GraphSpace
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> # Fetching my graphs
>>> graphs = graphspace.get_my_graphs(limit=5)
>>> graphs[0].get_name()
u'test'
```

Getting your graphs by tags:

```
>>> graphs = graphspace.get_my_graphs(tags=['Kegg-networks'], limit=5)
>>> graphs[0].get_name()
u'KEGG-Wnt-signaling-pathway'
```

get_public_graphs(*tags=None*, *limit=20*, *offset=0*)

Get public graphs.

Parameters

- **tags** (*List[str], optional*) – Search for graphs with the given given list of tag names. In order to search for graphs with given tag as a substring, wrap the name of the tag with percentage symbol. For example, %xyz% will search for all graphs with 'xyz' in their tag names. Defaults to None.
- **offset** (*int, optional*) – Offset the list of returned entities by this number. Defaults to 0.
- **limit** (*int, optional*) – Number of entities to return. Defaults to 20.

Returns List of public graphs.

Return type List[Graph]

Raises GraphSpaceError – If error response is received from the GraphSpace API.

Examples

Getting public graphs:

```
>>> # Connecting to GraphSpace
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> # Fetching public graphs
>>> graphs = graphspace.get_public_graphs(limit=5)
>>> graphs[0].get_name()
u'Wnt-Pathway-Reconstruction'
```

Getting public graphs by tags:

```
>>> graphs = graphspace.get_public_graphs(tags=['Kegg-networks'], limit=5)
>>> graphs[0].get_name()
u'KEGG-Wnt-signaling-pathway-with-ranks'
```

get_shared_graphs(*tags=None*, *limit=20*, *offset=0*)

Get graphs shared with the groups where requesting user is a member.

Parameters

- **tags** (*List[str], optional*) – Search for graphs with the given given list of tag names. In order to search for graphs with given tag as a substring, wrap the name of the tag with percentage symbol. For example, %xyz% will search for all graphs with 'xyz' in their tag names. Defaults to None.
- **offset** (*int, optional*) – Offset the list of returned entities by this number. Defaults to 0.
- **limit** (*int, optional*) – Number of entities to return. Defaults to 20.

Returns List of shared graphs.

Return type List[Graph]

Raises GraphSpaceError – If error response is received from the GraphSpace API.

Examples

Getting shared graphs:

```
>>> # Connecting to GraphSpace
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> # Fetching shared graphs
>>> graphs = graphspace.get_shared_graphs(limit=5)
>>> graphs[0].get_name()
u'KEGG-Wnt-signaling-pathway'
```

Getting shared graphs by tags:

```
>>> graphs = graphspace.get_shared_graphs(tags=['Kegg-networks'], limit=5)
>>> graphs[0].get_name()
u'KEGG-Wnt-signaling-pathway'
```

post_graph(*graph*)

Posts NetworkX graph to the requesting users account on GraphSpace.

Parameters **graph** (*GSGraph or Graph*) – Object having graph details, such as name, graph_json, style_json, is_public, tags.

Returns Saved graph on GraphSpace.

Return type Graph

Raises `GraphSpaceError` – If error response is received from the GraphSpace API.

Example

```
>>> # Connecting to GraphSpace
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> # Creating a graph
>>> from graphspace_python.graphs.classes.gsgraph import GSGraph
>>> G = GSGraph()
>>> G.set_name('My Sample Graph')
>>> G.set_tags(['sample'])
>>> G.add_node('a', popup='sample node popup text', label='A')
>>> G.add_node('b', popup='sample node popup text', label='B')
>>> G.add_edge('a', 'b', directed=True, popup='sample edge popup')
>>> G.add_edge_style('a', 'b', directed=True, edge_style='dotted')
>>> # Saving graph on GraphSpace
>>> graphspace.post_graph(G)
```

Note: Refer to the tutorial for more about posting graphs.

publish_graph(*graph_name=None*, *graph_id=None*, *graph=None*)

Makes a graph publicly viewable.

Parameters

- **graph_name** (*str, optional*) – Name of the graph. Defaults to None.
- **graph_id** (*int, optional*) – ID of the graph. Defaults to None.
- **graph** (*GSGraph or Graph, optional*) – Object having graph details, such as name, graph_json, style_json, is_public, tags. Defaults to None.

Returns Updated graph on GraphSpace.

Return type Graph

Raises

- `Exception` – If both 'graph_name' and 'graph_id' are None and graph object has no 'name' or 'id' attribute; or if graph doesnot exist.
- `GraphSpaceError` – If error response is received from the GraphSpace API.

Examples

Make graph public by name:

```
>>> # Connecting to GraphSpace
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> # Publishing the graph
>>> graph = graphspace.publish_graph(graph_name='My Sample Graph')
```

```
>>> graph.get_is_public()
1
```

Make graph public by id:

```
>>> graph = graphspace.publish_graph(graph_id=65930)
>>> graph.get_is_public()
1
```

Make graph public by passing graph object itself as param:

```
>>> graph = graphspace.get_graph(graph_name='My Sample Graph')
>>> graphspace.publish_graph(graph=graph)
```

Note: Refer to the tutorial for more about making a graph public.

---

set_default_graph_layout(*graph_name=None*, *graph_id=None*, *graph=None*, *layout_name=None*, *layout_id=None*, *layout=None*)

Set a default layout (provided the layout_name, layout_id or layout object) for a graph with given graph_name, graph_id or graph object itself.

Parameters

- **graph_name** (*str, optional*) – Name of the graph. Defaults to None.
- **graph_id** (*int, optional*) – ID of the graph. Defaults to None.
- **graph** (*GSGraph or Graph, optional*) – Object having graph details, such as name, graph_json, style_json, is_public, tags. Defaults to None.
- **layout_name** (*str, optional*) – Name of the layout. Defaults to None.
- **layout_id** (*int, optional*) – ID of the layout. Defaults to None.
- **layout** (*GSLayout or Layout*) – Object having layout details, such as name, is_shared, style_json, positions_json.

Returns Updated graph on GraphSpace.

Return type Graph

Raises

- `Exception` – If both 'layout_name' and 'layout_id' are None and layout object has no 'name' or 'id' attribute; or if both 'graph_name' and 'graph_id' are None and neither graph object has 'name' or 'id' attribute nor layout object has 'graph_id' attribute; or if graph or layout doesnot exist.
- `GraphSpaceError` – If error response is received from the GraphSpace API.

Examples

Setting a default layout when graph name is known:

```
>>> # Connecting to GraphSpace
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> # Setting default layout for the graph
```

```
>>> graph = graphspace.set_default_graph_layout(graph_name='My Sample Graph',␣
↪layout_id=1087)
>>> graph.default_layout_id
1087
```

Setting a default layout when graph id is known:

```
>>> graph = graphspace.set_default_graph_layout(graph_id=65930, layout_
↪id=1087)
>>> graph.default_layout_id
1087
```

Setting a default layout when graph object is passed as param:

```
>>> graph = graphspace.get_graph(graph_name='My Sample Graph')
>>> graph = graphspace.set_default_graph_layout(graph=graph, layout_id=1087)
>>> graph.default_layout_id
1087
```

Setting a default layout by passing layout name:

```
>>> graph = graphspace.set_default_graph_layout(graph_id=65930, layout_name=
↪'My Sample Layout')
>>> graph.default_layout_id
1087
```

Setting a default layout by only passing layout object as param:

```
>>> layout = graphspace.get_graph_layout(graph_id=65930, layout_name='My␣
↪Sample Layout')
>>> graph = graphspace.set_default_graph_layout(layout=layout)
>>> graph.default_layout_id
1087
```

Note: Refer to the tutorial for more about setting default graph layout.

---

unpublish_graph(*graph_name=None*, *graph_id=None*, *graph=None*)

Makes a graph privately viewable.

Parameters

- **graph_name** (*str, optional*) – Name of the graph. Defaults to None.
- **graph_id** (*int, optional*) – ID of the graph. Defaults to None.
- **graph** (*GSGraph or Graph, optional*) – Object having graph details, such as name, graph_json, style_json, is_public, tags. Defaults to None.

Returns Updated graph on GraphSpace.

Return type Graph

Raises

- `Exception` – If both 'graph_name' and 'graph_id' are None and graph object has no 'name' or 'id' attribute; or if graph doesnot exist.
- `GraphSpaceError` – If error response is received from the GraphSpace API.

Examples

Make graph private by name:

```
>>> # Connecting to GraphSpace
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> # Unpublishing the graph
>>> graph = graphspace.unpublish_graph(graph_name='My Sample Graph')
>>> graph.get_is_public()
0
```

Make graph private by id:

```
>>> graph = graphspace.unpublish_graph(graph_id=65930)
>>> graph.get_is_public()
0
```

Make graph private by passing graph object itself as param:

```
>>> graph = graphspace.get_graph(graph_name='My Sample Graph')
>>> graphspace.unpublish_graph(graph=graph)
```

Note: Refer to the tutorial for more about making a graph private.

unset_default_graph_layout(*graph_name=None*, *graph_id=None*, *graph=None*)

Unsets the current default layout of a graph provided the graph_name, graph_id or graph object itself.

Parameters

- **graph_name** (*str, optional*) – Name of the graph. Defaults to None.
- **graph_id** (*int, optional*) – ID of the graph. Defaults to None.
- **graph** (*GSGraph or Graph, optional*) – Object having graph details, such as name, graph_json, style_json, is_public, tags. Defaults to None.

Returns Updated graph on GraphSpace.

Return type Graph

Raises

- Exception – If both 'graph_name' and 'graph_id' are None and graph object has no 'name' or 'id' attribute; or if graph doesnot exist.
- GraphSpaceError – If error response is received from the GraphSpace API.

Examples

Unset default graph layout by graph name:

```
>>> # Connecting to GraphSpace
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> # Unset the default graph layout
```

```
>>> graph = graphspace.unset_default_graph_layout(graph_name='My Sample Graph
↪')
>>> assert graph.default_layout_id is None
```

Unset default graph layout by graph id:

```
>>> graph = graphspace.unset_default_graph_layout(graph_id=65930)
```

Unset default graph layout by graph object:

```
>>> graph = graphspace.get_graph(graph_name='My Sample Graph')
>>> graph = graphspace.unset_default_graph_layout(graph=graph)
```

Note: Refer to the tutorial for more about unsetting a default graph layout.

update_graph(*graph*, *graph_name=None*, *graph_id=None*, *owner_email=None*)

Update a graph on GraphSpace with the provided graph object. If graph_name or graph_id is also provided then the update will be performed for that graph having the given name or id.

Parameters

- **graph** (*GSGraph or Graph*) – Object having graph details, such as name, graph_json, style_json, is_public, tags.
- **graph_name** (*str, optional*) – Name of the graph to be updated. Defaults to None.
- **graph_id** (*int, optional*) – ID of the graph to be updated. Defaults to None.
- **owner_email** (*str, optional*) – Email of owner of the graph. Defaults to None.

Returns Updated graph on GraphSpace.

Return type Graph

Raises

- Exception – If both 'graph_name' and 'graph_id' are None and graph object has no 'name' or 'id' attribute; or if graph doesnot exist.
- GraphSpaceError – If error response is received from the GraphSpace API.

Examples

Updating a graph by creating a new graph and replacing the existing graph:

```
>>> # Connecting to GraphSpace
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> # Creating the new graph
>>> G = GSGraph()
>>> G.add_node('a', popup='sample node popup text', label='A updated')
>>> G.add_node('b', popup='sample node popup text', label='B updated')
>>> G.add_edge('a', 'b', directed=True, popup='sample edge popup')
>>> G.add_edge_style('a', 'b', directed=True, edge_style='dotted')
>>> G.set_name('My Sample Graph')
>>> G.set_is_public(1)
>>> # Updating to replace the existing graph
>>> graphspace.update_graph(G)
```

Another way of updating a graph by fetching and editing the existing graph:

```
>>> # Fetching the graph
>>> graph = graphspace.get_graph(graph_name='My Sample Graph')
>>> # Modifying the fetched graph
>>> graph.add_node('z', popup='sample node popup text', label='Z')
>>> graph.add_node_style('z', shape='ellipse', color='green', width=90,␣
↪height=90)
>>> graph.add_edge('a', 'z', directed=True, popup='sample edge popup')
>>> graph.set_is_public(1)
>>> # Updating graph
>>> graphspace.update_graph(graph)
```

If you also provide 'graph_name' or 'graph_id' as param then the update will be performed for that graph having the given name or id:

```
>>> graphspace.update_graph(G, graph_id=65930)
```

Note: Refer to the tutorial for more about updating graphs.

Layouts Endpoint Class

class graphspace_python.api.endpoint.layouts.**Layouts**(*client*)

Bases: object

Layouts endpoint class.

Provides methods for layout related operations such as saving, fetching, updating and deleting layouts on GraphSpace.

delete_graph_layout(*layout_name=None*, *layout_id=None*, *layout=None*, *graph_name=None*, *graph_id=None*, *graph=None*)

Delete a layout with given layout_name, layout_id or layout object itself from the graph with the given graph_name, graph_id or the graph object.

Parameters

- **layout_name** (*str, optional*) – Name of the layout to be deleted. Defaults to None.
- **layout_id** (*int, optional*) – ID of the layout to be deleted. Defaults to None.
- **layout** (*GSLayout or Layout*) – Object having layout details, such as name, is_shared, style_json, positions_json.
- **graph_name** (*str, optional*) – Name of the graph. Defaults to None.
- **graph_id** (*int, optional*) – ID of the graph. Defaults to None.
- **graph** (*GSGraph or Graph, optional*) – Object having graph details, such as name, graph_json, style_json, is_public, tags. Defaults to None.

Returns Success/Error Message from GraphSpace.

Return type str

Raises

- `Exception` – If both 'layout_name' and 'layout_id' are None and layout object has no 'name' or 'id' attribute; or if both 'graph_name' and 'graph_id' are None and neither graph object has 'name' or 'id' attribute nor layout object has 'graph_id' attribute; or if graph or layout doesnot exist.
- `GraphSpaceError` – If error response is received from the GraphSpace API.

Examples

Deleting a layout by layout name:

```
>>> # Connecting to GraphSpace
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> # Deleting a layout
>>> graphspace.delete_graph_layout(layout_name='My Sample Layout', graph_
↪id=65390)
u'Successfully deleted layout with id=1087'
```

Deleting a layout by layout id:

```
>>> graphspace.delete_graph_layout(layout_id=1087, graph_id=65930)
u'Successfully deleted layout with id=1087'
```

Deleting a layout by passing layout object as param:

```
>>> layout = graphspace.get_graph_layout(layout_name='My Sample Layout',␣
↪graph_id=65390)
>>> graphspace.delete_graph_layout(layout=layout)
u'Successfully deleted layout with id=1087'
```

Deleting a layout by passing graph name instead of id:

```
>>> graphspace.delete_graph_layout(layout_id=1087, graph_name='My Sample Graph
↪')
u'Successfully deleted layout with id=1087'
```

Deleting a layout by passing graph object as param:

```
>>> graph = graphspace.get_graph(graph_name='My Sample Graph')
>>> graphspace.delete_graph_layout(layout_id=1087, graph=graph)
u'Successfully deleted layout with id=1087'
```

Note: Refer to the tutorial for more about deleting layouts.

---

get_graph_layout(*layout_name=None*, *layout_id=None*, *graph_name=None*, *graph_id=None*, *graph=None*, *owner_email=None*)

Get a layout with given layout_id or layout_name for the graph with given graph_id, graph_name or graph object.

Parameters

- **layout_name** (*str, optional*) – Name of the layout to be fetched. Defaults to None.
- **layout_id** (*int, optional*) – ID of the layout to be fetched. Defaults to None.

- **graph_name** (*str, optional*) – Name of the graph. Defaults to None.
- **graph_id** (*int, optional*) – ID of the graph. Defaults to None.
- **graph** (*GSGraph or Graph, optional*) – Object having graph details, such as name, graph_json, style_json, is_public, tags. Defaults to None.
- **owner_email** (*str, optional*) – Email of owner of layout. Defaults to None.

Returns Layout object, if layout with the given 'layout_name' or 'layout_id' exists; otherwise None.

Return type Layout or None

Raises

- `Exception` – If both 'layout_name' and 'layout_id' are None; or if both 'graph_name' and 'graph_id' are None and graph object has no 'name' or 'id' attribute; or if graph doesnot exist.
- `GraphSpaceError` – If error response is received from the GraphSpace API.

Examples

Getting a layout by name:

```
>>> # Connecting to GraphSpace
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> # Fetching a layout
>>> layout = graphspace.get_graph_layout(layout_name='My Sample Layout',␣
↪graph_id=65390)
>>> layout.get_name()
u'My Sample Layout'
```

Getting a layout by id:

```
>>> layout = graphspace.get_graph_layout(layout_id=1087, graph_id=65390)
>>> layout.get_name()
u'My Sample Layout'
```

Getting a layout by providing graph name:

```
>>> layout = graphspace.get_graph_layout(layout_id=1087, graph_name='My␣
↪Sample Graph')
>>> layout.get_name()
u'My Sample Layout'
```

Getting a layout by providing graph object as param:

```
>>> graph = graphspace.get_graph(graph_name='My Sample Graph')
>>> layout = graphspace.get_graph_layout(layout_id=1087, graph=graph)
>>> layout.get_name()
u'My Sample Layout'
```

Note: Refer to the tutorial for more about fetching layouts.

get_my_graph_layouts(*graph_name=None*, *graph_id=None*, *graph=None*, *limit=20*, *offset=0*)
Get layouts created by the requesting user for the graph with given graph_name, graph_id or graph object.

Parameters

- **graph_name** (*str, optional*) – Name of the graph. Defaults to None.
- **graph_id** (*int, optional*) – ID of the graph. Defaults to None.
- **graph** (*GSGraph or Graph, optional*) – Object having graph details, such as name, graph_json, style_json, is_public, tags. Defaults to None.
- **offset** (*int, optional*) – Offset the list of returned entities by this number. Defaults to 0.
- **limit** (*int, optional*) – Number of entities to return. Defaults to 20.

Returns List of layouts owned by the requesting user.

Return type List[Layout]

Raises

- `Exception` – If both 'graph_name' and 'graph_id' are None and graph object has no 'name' or 'id' attribute; or if graph doesnot exist.
- `GraphSpaceError` – If error response is received from the GraphSpace API.

Examples

Getting your graph layouts by graph id:

```
>>> # Connecting to GraphSpace
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> # Fetching my graph layouts
>>> layouts = graphspace.get_my_graph_layouts(graph_id=65390, limit=5)
>>> layouts[0].get_name()
u'My Sample Layout'
```

Getting your graph layouts by graph name:

```
>>> layouts = graphspace.get_my_graph_layouts(graph_name='My Sample Graph',␣
↪limit=5)
```

Getting your graph layouts by graph object itself:

```
>>> graph = graphspace.get_graph(graph_name='My Sample Graph')
>>> layouts = graphspace.get_my_graph_layouts(graph=graph, limit=5)
```

get_shared_graph_layouts(*graph_name=None*, *graph_id=None*, *graph=None*, *limit=20*, *offset=0*)
Get layouts shared with the requesting user for the graph with given graph_name, graph_id or graph object.

Parameters

- **graph_name** (*str, optional*) – Name of the graph. Defaults to None.
- **graph_id** (*int, optional*) – ID of the graph. Defaults to None.
- **graph** (*GSGraph or Graph, optional*) – Object having graph details, such as name, graph_json, style_json, is_public, tags. Defaults to None.

- **offset** (*int, optional*) – Offset the list of returned entities by this number. Defaults to 0.
- **limit** (*int, optional*) – Number of entities to return. Defaults to 20.

Returns List of layouts shared with the requesting user.

Return type List[Layout]

Raises

- `Exception` – If both 'graph_name' and 'graph_id' are None and graph object has no 'name' or 'id' attribute; or if graph doesnot exist.
- `GraphSpaceError` – If error response is received from the GraphSpace API.

Examples

Getting shared graph layouts by graph id:

```
>>> # Connecting to GraphSpace
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> # Fetching shared graph layouts
>>> layouts = graphspace.get_shared_graph_layouts(graph_id=65390, limit=5)
>>> layouts[0].get_name()
u'My Sample Layout'
```

Getting shared graph layouts by graph name:

```
>>> layouts = graphspace.get_shared_graph_layouts(graph_name='My Sample Graph
↪', limit=5)
```

Getting shared graph layouts by graph object itself:

```
>>> graph = graphspace.get_graph(graph_name='My Sample Graph')
>>> layouts = graphspace.get_shared_graph_layouts(graph=graph, limit=5)
```

post_graph_layout(*layout*, *graph_name=None*, *graph_id=None*, *graph=None*)

Create a layout for a graph provided the graph_name, graph_id or the graph object itself.

Parameters

- **layout** (*GSLayout or Layout*) – Object having layout details, such as name, is_shared, style_json, positions_json.
- **graph_name** (*str, optional*) – Name of the graph. Defaults to None.
- **graph_id** (*int, optional*) – ID of the graph. Defaults to None.
- **graph** (*GSGraph or Graph, optional*) – Object having graph details, such as name, graph_json, style_json, is_public, tags. Defaults to None.

Returns Saved layout on GraphSpace.

Return type Layout

Raises

- `Exception` – If both 'graph_name' and 'graph_id' are None and graph object has no 'name' or 'id' attribute; or if graph doesnot exist.

- `GraphSpaceError` – If error response is received from the GraphSpace API.

Examples

Saving a layout when graph id is known:

```
>>> # Connecting to GraphSpace
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> # Creating a layout
>>> from graphspace_python.graphs.classes.gslayout import GSLayout
>>> L = GSLayout()
>>> L.set_node_position('a', y=38.5, x=67.3)
>>> L.add_node_style('a', shape='ellipse', color='green', width=60, height=60)
>>> L.set_name('My Sample Layout')
>>> L.set_is_shared(1)
>>> # Saving layout on GraphSpace
>>> graphspace.post_graph_layout(L, graph_id=65390)
```

Saving a layout when graph name is known:

```
>>> graphspace.post_graph_layout(L, graph_name='My Sample Graph')
```

Saving a layout by passing graph object itself as param:

```
>>> graph = graphspace.get_graph(graph_name='My Sample Graph')
>>> graphspace.post_graph_layout(L, graph=graph)
```

Note: Refer to the tutorial for more about posting layouts.

update_graph_layout(*layout*, *layout_name=None*, *layout_id=None*, *graph_name=None*, *graph_id=None*, *graph=None*, *owner_email=None*)

Update a layout with given layout_id or layout_name for a graph with the given graph_name, graph_id or the graph object itself.

Parameters

- **layout** (*GSLayout or Layout*) – Object having layout details, such as name, is_shared, style_json, positions_json.
- **layout_name** (*str, optional*) – Name of the layout to be updated. Defaults to None.
- **layout_id** (*int, optional*) – ID of the layout to be updated. Defaults to None.
- **graph_name** (*str, optional*) – Name of the graph. Defaults to None.
- **graph_id** (*int, optional*) – ID of the graph. Defaults to None.
- **graph** (*GSGraph or Graph, optional*) – Object having graph details, such as name, graph_json, style_json, is_public, tags. Defaults to None.
- **owner_email** (*str, optional*) – Email of owner of layout. Defaults to None.

Returns Updated layout on GraphSpace.

Return type Layout

Raises

- `Exception` – If both 'layout_name' and 'layout_id' are None and layout object has no 'name' or 'id' attribute; or if both 'graph_name' and 'graph_id' are None and neither graph object has 'name' or 'id' attribute nor layout object has 'graph_id' attribute; or if graph or layout doesnot exist.
- `GraphSpaceError` – If error response is received from the GraphSpace API.

Examples

Updating a layout by creating a new layout and replacing the existing layout:

```
>>> # Connecting to GraphSpace
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> # Creating the new layout
>>> L = GSLayout()
>>> L.set_node_position('a', y=102, x=238.1)
>>> L.add_node_style('a', shape='octagon', color='green', width=60, height=60)
>>> L.set_name('My Sample Layout')
>>> L.set_is_shared(1)
>>> # Updating to replace the existing layout
>>> graphspace.update_graph_layout(L, graph_id=65390)
```

Another way of updating a layout by fetching and editing the existing layout:

```
>>> # Fetching the layout
>>> layout = graphspace.get_graph_layout(name='My Sample Layout')
>>> # Modifying the fetched layout
>>> layout.set_node_position('a', y=30, x=211)
>>> layout.add_node_style('a', shape='roundrectangle', color='green',␣
↪width=45, height=45)
>>> layout.set_is_shared(0)
>>> # Updating layout
>>> graphspace.update_graph_layout(layout)
```

If you also provide 'layout_name' or 'layout_id' as param then the update will be performed for that layout having the given name or id:

```
>>> graphspace.update_graph_layout(L, layout_id=1087, graph_id=65390)
```

Note: Refer to the tutorial for more about updating layouts.

Groups Endpoint Class

class `graphspace_python.api.endpoint.groups.`**`Groups`**(*client*)

Bases: `object`

Groups endpoint class.

Provides methods for group related operations such as saving, fetching, updating, deleting groups on GraphSpace.

Also provides methods for group member and group graph related operations such as fetching all members or graphs of the group, adding or deleting new member or graph to the group.

add_group_member(*member_email*, *group_name=None*, *group_id=None*, *group=None*)
Add a member to a group provided the group_name, group_id or the group object itself.

Parameters

- **member_email** (*str*) – Email of the member to be added to the group.
- **group_name** (*str, optional*) – Name of the group. Defaults to None.
- **group_id** (*int, optional*) – ID of the group. Defaults to None.
- **group** (*GSGroup or Group, optional*) – Object having group details, such as name, description. Defaults to None.

Returns Dict containing 'group_id' and 'user_id' of the added member.

Return type dict

Raises

- `Exception` – If both 'group_name' and 'group_id' are None and group object has no 'name' or 'id' attribute; or if group doesnot exist.
- `GraphSpaceError` – If error response is received from the GraphSpace API.

Examples

Adding a member to a group when group name is known:

```
>>> # Connecting to GraphSpace
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> # Adding a member to group
>>> graphspace.add_group_member(member_email='user2@example.com', group_name=
↪'My Sample Group')
{u'group_id': u'198', u'user_id': 2}
```

Adding a member to a group when group id is known:

```
>>> graphspace.add_group_member(member_email='user2@example.com', group_
↪id=198)
{u'group_id': u'198', u'user_id': 2}
```

Adding a member to a group by passing group object itself as param:

```
>>> group = graphspace.get_group(group_name='My Sample Group')
>>> graphspace.add_group_member(member_email='user2@example.com', group=group)
{u'group_id': u'198', u'user_id': 2}
```

Note: Refer to the tutorial for more about adding group members.

delete_group(*group_name=None*, *group_id=None*, *group=None*)
Delete a group from GraphSpace provided the group_name, group_id or the group object itself, where the requesting user is the owner.

Parameters

- **group_name** (*str, optional*) – Name of the group to be deleted. Defaults to None.
- **group_id** (*int, optional*) – ID of the group to be deleted. Defaults to None.

- **group** (*GSGroup or Group, optional*) – Object having group details, such as name, description. Defaults to None.

Returns Success/Error Message from GraphSpace.

Return type str

Raises

- Exception – If both 'group_name' and 'group_id' are None and group object has no 'name' or 'id' attribute; or if group doesnot exist.
- GraphSpaceError – If error response is received from the GraphSpace API.

Examples

Deleting a group by name:

```
>>> # Connecting to GraphSpace
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> # Deleting a group
>>> graphspace.delete_group(group_name='My Sample Group')
u'Successfully deleted group with id=198'
```

Deleting a group by id:

```
>>> graphspace.delete_group(group_id=198)
u'Successfully deleted group with id=198'
```

Deleting a group by passing group object itself as param:

```
>>> group = graphspace.get_group(group_name='My Sample Group')
>>> graphspace.delete_group(group=group)
u'Successfully deleted group with id=198'
```

Note: Refer to the tutorial for more about deleting groups.

delete_group_member(*member_id=None*, *member=None*, *group_name=None*, *group_id=None*, *group=None*)

Delete a member with given member_id or member object from a group provided the group_name, group_id or the group object itself.

Parameters

- **member_id** (*int, optional*) – ID of the member to be deleted from the group. Defaults to None.
- **member** (*Member, optional*) – Object having member details, such as id, email. Defaults to None.
- **group_name** (*str, optional*) – Name of the group. Defaults to None.
- **group_id** (*int, optional*) – ID of the group. Defaults to None.
- **group** (*GSGroup or Group, optional*) – Object having group details, such as name, description. Defaults to None.

Returns Success/Error Message from GraphSpace.

Return type str

Raises

- `Exception` – If both 'group_name' and 'group_id' are None and group object has no 'name' or 'id' attribute; or if 'member_id' is None and member object has no 'id' attribute; or if group doesnot exist.
- `GraphSpaceError` – If error response is received from the GraphSpace API.

Examples

Deleting a member from a group when group name is known:

```
>>> # Connecting to GraphSpace
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> # Deleting a member from group
>>> graphspace.delete_group_member(member_id=2, group_name='My Sample Group')
u'Successfully deleted member with id=2 from group with id=198'
```

Deleting a member from a group when group id is known:

```
>>> graphspace.delete_group_member(member_id=2, group_id=198)
u'Successfully deleted member with id=2 from group with id=198'
```

Deleting a member from a group by passing group object itself as param:

```
>>> group = graphspace.get_group(group_name='My Sample Group')
>>> graphspace.delete_group_member(member_id=2, group=group)
u'Successfully deleted member with id=2 from group with id=198'
```

Deleting a member from a group by passing member object as param:

```
>>> members = graphspace.get_group_members(group_name='My Sample Group')
>>> graphspace.delete_group_member(member=members[0], group_name='My Sample␣
↪Group')
u'Successfully deleted member with id=2 from group with id=198'
```

Note: Refer to the tutorial for more about deleting group members.

get_all_groups (*limit=20*, *offset=0*)

Get groups where the requesting user is a member.

Parameters

- **offset** (*int, optional*) – Offset the list of returned entities by this number. Defaults to 0.
- **limit** (*int, optional*) – Number of entities to return. Defaults to 20.

Returns List of groups where the requesting user is a member.

Return type List[Group]

Raises `GraphSpaceError` – If error response is received from the GraphSpace API.

Example

```
>>> # Connecting to GraphSpace
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> # Fetching all groups
>>> groups = graphspace.get_all_groups(limit=5)
>>> groups[0].get_name()
u'Test Group'
```

get_group(*group_name=None*, *group_id=None*)

Get a group with the given group_name or group_id, where the requesting user is a member.

Parameters

- **group_name** (*str, optional*) – Name of the group to be fetched. Defaults to None.
- **group_id** (*int, optional*) – ID of the group to be fetched. Defaults to None.

Returns Group object, if group with the given 'group_name' or 'group_id' exists; otherwise None.

Return type Group or None

Raises

- `Exception` – If both 'group_name' and 'group_id' are None.
- `GraphSpaceError` – If error response is received from the GraphSpace API.

Examples

Getting a group by name:

```
>>> # Connecting to GraphSpace
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> # Fetching a group
>>> group = graphspace.get_group(group_name='My Sample Group')
>>> group.get_name()
u'My Sample Group'
```

Getting a group by id:

```
>>> group = graphspace.get_group(group_id=198)
>>> group.get_name()
u'My Sample Graph'
```

Note: Refer to the tutorial for more about fetching groups.

get_group_graphs(*group_name=None*, *group_id=None*, *group=None*)

Get graphs shared with a group provided the group_name, group_id or the group object itself.

Parameters

- **group_name** (*str, optional*) – Name of the group. Defaults to None.
- **group_id** (*int, optional*) – ID of the group. Defaults to None.

- **group** (*GSGroup or Group, optional*) – Object having group details, such as name, description. Defaults to None.

Returns List of graphs belonging to the group.

Return type List[Graph]

Raises

- `Exception` – If both 'group_name' and 'group_id' are None and group object has no 'name' or 'id' attribute; or if group doesnot exist.
- `GraphSpaceError` – If error response is received from the GraphSpace API.

Examples

Getting graphs of a group when group name is known:

```
>>> # Connecting to GraphSpace
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> # Fetching group graphs
>>> graphs = graphspace.get_group_graphs(group_name='My Sample Group')
>>> graphs[0].get_name()
u'My Sample Graph'
```

Getting graphs of a group when group id is known:

```
>>> graphs = graphspace.get_group_graphs(group_id=198)
>>> graphs[0].get_name()
u'My Sample Graph'
```

Getting graphs of a group by passing group object itself as param:

```
>>> group = graphspace.get_group(group_name='My Sample Group')
>>> graphs = graphspace.get_group_graphs(group=group)
>>> graphs[0].get_name()
u'My Sample Graph'
```

Note: Refer to the tutorial for more about fetching graphs of a group.

get_group_members(*group_name=None*, *group_id=None*, *group=None*)

Get members of a group provided the group_name, group_id or the group object itself.

Parameters

- **group_name** (*str, optional*) – Name of the group. Defaults to None.
- **group_id** (*int, optional*) – ID of the group. Defaults to None.
- **group** (*GSGroup or Group, optional*) – Object having group details, such as name, description. Defaults to None.

Returns List of members belonging to the group.

Return type List[Member]

Raises

- `Exception` – If both 'group_name' and 'group_id' are None and group object has no 'name' or 'id' attribute; or if group doesnot exist.
- `GraphSpaceError` – If error response is received from the GraphSpace API.

Examples

Getting members of a group when group name is known:

```
>>> # Connecting to GraphSpace
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> # Fetching group members
>>> members = graphspace.get_group_members(group_name='My Sample Group')
>>> members[0].email
u'user1@example.com'
```

Getting members of a group when group id is known:

```
>>> members = graphspace.get_group_members(group_id=198)
>>> members[0].email
u'user1@example.com'
```

Getting members of a group by passing group object itself as param:

```
>>> group = graphspace.get_group(group_name='My Sample Group')
>>> members = graphspace.get_group_members(group=group)
>>> members[0].email
u'user1@example.com'
```

Note: Refer to the tutorial for more about fetching group members.

---

get_my_groups(*limit=20*, *offset=0*)

Get groups created by the requesting user.

Parameters

- **offset** (*int, optional*) – Offset the list of returned entities by this number. Defaults to 0.
- **limit** (*int, optional*) – Number of entities to return. Defaults to 20.

Returns List of groups owned by the requesting user.

Return type List[Group]

Raises `GraphSpaceError` – If error response is received from the GraphSpace API.

Example

```
>>> # Connecting to GraphSpace
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> # Fetching my groups
>>> groups = graphspace.get_my_groups(limit=5)
```

```
>>> groups[0].get_name()
u'Test Group'
```

post_group(*group*)

Create a group for the requesting user.

Parameters **group** (*GSGroup or Group*) – Object having group details, such as name, description.

Returns Saved group on GraphSpace.

Return type Group

Raises GraphSpaceError – If error response is received from the GraphSpace API.

Example

```
>>> # Connecting to GraphSpace
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> # Creating a group
>>> from graphspace_python.graphs.classes.gsgroup import GSGroup
>>> group = GSGroup(name='My Sample Group', description='sample group')
>>> # Saving group on GraphSpace
>>> graphspace.post_graph(group)
```

Note: Refer to the tutorial for more about posting groups.

share_graph(*graph_name=None*, *graph_id=None*, *graph=None*, *group_name=None*, *group_id=None*, *group=None*)

Share a graph with a group by providing any of graph_name, graph_id or graph object along with any of group_name, group_id or group object.

Parameters

- **graph_name** (*str, optional*) – Name of the graph. Defaults to None.
- **graph_id** (*int, optional*) – ID of the graph. Defaults to None.
- **graph** (*GSGraph or Graph, optional*) – Object having graph details, such as name, graph_json, style_json, is_public, tags. Defaults to None.
- **group_name** (*str, optional*) – Name of the group. Defaults to None.
- **group_id** (*int, optional*) – ID of the group. Defaults to None.
- **group** (*GSGroup or Group, optional*) – Object having group details, such as name, description. Defaults to None.

Returns Dict containing 'group_id', 'graph_id', 'created_at', 'updated_at' details of the shared graph.

Return type dict

Raises

- `Exception` – If both 'group_name' and 'group_id' are None and group object has no 'name' or 'id' attribute; or if both 'graph_name' and 'graph_id' are None and graph object has no 'name' or 'id' attribute; or if the group or graph doesnot exist.
- `GraphSpaceError` – If error response is received from the GraphSpace API.

Examples

Sharing a graph with a group when group name is known:

```
>>> # Connecting to GraphSpace
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> # Sharing a graph with a group
>>> graphspace.share_graph(graph_id=65390, group_name='My Sample Group')
{u'created_at': u'2017-07-20T18:40:36.267052', u'group_id': u'198', u'graph_id
↪':
65390, u'updated_at': u'2017-07-20T18:40:36.267052'}
```

Sharing a graph with a group when group id is known:

```
>>> graphspace.share_graph(graph_id=65390, group_id=198)
{u'created_at': u'2017-07-20T18:40:36.267052', u'group_id': u'198', u'graph_id
↪':
65390, u'updated_at': u'2017-07-20T18:40:36.267052'}
```

Sharing a graph with a group by passing group object itself as param:

```
>>> group = graphspace.get_group(group_name='My Sample Group')
>>> graphspace.share_graph(graph_id=65390, group=group)
{u'created_at': u'2017-07-20T18:40:36.267052', u'group_id': u'198', u'graph_id
↪':
65390, u'updated_at': u'2017-07-20T18:40:36.267052'}
```

Sharing a graph with a group by providing graph name:

```
>>> graphspace.share_graph(graph_name='My Sample Graph', group_id=198)
{u'created_at': u'2017-07-20T18:40:36.267052', u'group_id': u'198', u'graph_id
↪':
65390, u'updated_at': u'2017-07-20T18:40:36.267052'}
```

Sharing a graph with a group by providing graph object:

```
>>> graph = graphspace.get_graph(graph_name='My Sample Graph')
>>> graphspace.share_graph(graph=graph, group_id=198)
{u'created_at': u'2017-07-20T18:40:36.267052', u'group_id': u'198', u'graph_id
↪':
65390, u'updated_at': u'2017-07-20T18:40:36.267052'}
```

Note: Refer to the tutorial for more about adding graph to a group.

---

unshare_graph(*graph_name=None*, *graph_id=None*, *graph=None*, *group_name=None*, *group_id=None*, *group=None*)

Unshare a graph with a group by providing any of graph_name, graph_id or graph object along with any of group_name, group_id or group object.

Parameters

- **graph_name** (*str, optional*) – Name of the graph. Defaults to None.
- **graph_id** (*int, optional*) – ID of the graph. Defaults to None.
- **graph** (*GSGraph or Graph, optional*) – Object having graph details, such as name, graph_json, style_json, is_public, tags. Defaults to None.
- **group_name** (*str, optional*) – Name of the group. Defaults to None.
- **group_id** (*int, optional*) – ID of the group. Defaults to None.
- **group** (*GSGroup or Group, optional*) – Object having group details, such as name, description. Defaults to None.

Returns Success/Error Message from GraphSpace.

Return type str

Raises

- `Exception` – If both 'group_name' and 'group_id' are None and group object has no 'name' or 'id' attribute; or if both 'graph_name' and 'graph_id' are None and graph object has no 'name' or 'id' attribute; or if the group or graph doesnot exist.
- `GraphSpaceError` – If error response is received from the GraphSpace API.

Examples

Unshare a graph with a group when group name is known:

```
>>> # Connecting to GraphSpace
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> # Unsharing a graph with a group
>>> graphspace.unshare_graph(graph_id=65390, group_name='My Sample Group')
u'Successfully deleted graph with id=65390 from group with id=198'
```

Unshare a graph with a group when group id is known:

```
>>> graphspace.unshare_graph(graph_id=65390, group_id=198)
u'Successfully deleted graph with id=65390 from group with id=198'
```

Unshare a graph with a group by passing group object itself as param:

```
>>> group = graphspace.get_group(group_name='My Sample Group')
>>> graphspace.unshare_graph(graph_id=65390, group=group)
u'Successfully deleted graph with id=65390 from group with id=198'
```

Unshare a graph with a group by providing graph name:

```
>>> graphspace.unshare_graph(graph_name='My Sample Graph', group_id=198)
u'Successfully deleted graph with id=65390 from group with id=198'
```

Unshare a graph with a group by providing graph object:

```
>>> graph = graphspace.get_graph(graph_name='My Sample Graph')
>>> graphspace.unshare_graph(graph=graph, group_id=198)
u'Successfully deleted graph with id=65390 from group with id=198'
```

Note: Refer to the tutorial for more about deleting graph from a group.

update_group(*group*, *group_name=None*, *group_id=None*)

Update a group on GraphSpace with the provided group object. If group_name or group_id is also provided then the update will be performed for that group having the given name or id.

Parameters

- **group** (*GSGroup or Group*) – Object having group details, such as name, description.
- **group_name** (*str, optional*) – Name of the group to be updated. Defaults to None.
- **group_id** (*int, optional*) – ID of the group to be updated. Defaults to None.

Returns Updated group on GraphSpace.

Return type Group

Raises

- `Exception` – If both 'group_name' and 'group_id' are None and group object has no 'name' or 'id' attribute; or if group doesnot exist.
- `GraphSpaceError` – If error response is received from the GraphSpace API.

Examples

Updating a group by creating a new group and replacing the existing group:

```
>>> # Connecting to GraphSpace
>>> from graphspace_python.api.client import GraphSpace
>>> graphspace = GraphSpace('user1@example.com', 'user1')
>>> # Creating the new group
>>> group = GSGroup(name='My Sample Group', description='updated sample group
↪')
>>> # Updating to replace the existing group
>>> group = graphspace.update_group(group)
>>> group.get_description()
u'updated sample group'
```

Another way of updating a group by fetching and editing the existing group:

```
>>> # Fetching the group
>>> group = graphspace.get_group(group_name='My Sample Group')
>>> # Modifying the fetched group
>>> group.set_description('updated sample group')
>>> # Updating group
>>> group = graphspace.update_group(group)
>>> group.get_description()
u'updated sample group'
```

If you also provide 'group_name' or 'group_id' as param then the update will be performed for that group having the given name or id:

```
>>> graphspace.update_group(group, group_id=198)
```

Note: Refer to the tutorial for more about updating groups.

GSGraph Class

class graphspace_python.graphs.classes.gsgraph.**GSGraph**(**args*, ***kwargs*)

Bases: networkx.classes.digraph.DiGraph

GSGraph class.

A GSGraph stores the details of a graph that is understood by GraphSpace.

It stores nodes and edges of a graph with some data attributes in an organised json structure.

It also stores the style attributes of the respective nodes and edges in an organised json structure.

It holds the information about the graph such as name, tags and viewability status.

It provides methods to define, modify and delete the details of the graph.

name
str – Name of graph.

is_public
int – Visibility status of graph. Has value 0 if graph is private, 1 if graph is public.

style_json
dict – Json representation for graph style.

graph_json
dict – Json representation for graph structure.

tags
List[str] – Tags of graph.

data
dict – Metadata of graph.

node
dict – Json representation for nodes of graph.

edge
dict – Json representation for edges of graph.

ALLOWED_ARROW_FILL = ['filled', 'hollow']

ALLOWED_ARROW_SHAPES = ['tee', 'triangle', 'triangle-tee', 'triangle-backcurve', 'square', 'circle', 'diamond', 'none']

ALLOWED_EDGE_STYLES = ['solid', 'dotted', 'dashed']

ALLOWED_NODE_BACKGROUND_REPEAT = ['no-repeat', 'repeat-x', 'repeat-y', 'repeat']

ALLOWED_NODE_BORDER_STYLES = ['solid', 'dotted', 'dashed', 'double']

ALLOWED_NODE_SHAPES = ['rectangle', 'roundrectangle', 'ellipse', 'triangle', 'pentagon', 'hexagon', 'heptagon', 'octago

ALLOWED_NODE_TEXT_TRANSFORM = ['none', 'uppercase', 'lowercase']

ALLOWED_NODE_TEXT_WRAP = ['none', 'wrap']

ALLOWED_TEXT_BACKROUND_SHAPE = ['rectangle', 'roundrectangle']

ALLOWED_TEXT_HALIGN = ['left', 'center', 'right']

ALLOWED_TEXT_VALIGN = ['top', 'center', 'bottom']

ALLOWED_TEXT_WRAP = ['wrap', 'none']

EDGE_COLOR_ATTRIBUTES = ['line-color', 'source-arrow-color', 'mid-source-arrow-color', 'target-arrow-color', 'mid-t

NODE_COLOR_ATTRIBUTES = ['background-color', 'border-color', 'color', 'text-outline-color', 'text-shadow-color', 'tex

add_edge(*source*, *target*, *attr_dict=None*, *directed=False*, *popup=None*, *k=None*, ***attr*)

Add an edge to the graph.

Parameters

- **source** (*str*) – Source node.
- **target** (*str*) – Target node.
- **attr_dict** (*dict, optional*) – Json representation of edge data. Defaults to None.
- **directed** (*bool, optional*) – True if edge is directed, else False. Defaults to False.
- **popup** (*str, optional*) – A string that will be displayed in a popup window when the user clicks the edge. This string can be HTML-formatted information, e.g., Gene Ontology annotations and database links for a protein; or types, mechanism, and database sources for an interaction.
- **k** (*int, optional*) – An integer-valued attribute for the edge, which denotes a rank. Through this attribute, GraphSpace allows the user to filter nodes and edges in a network visualization.
- ****attr** – Arbitrary keyword arguments.

Example

```
>>> from graphspace_python.graphs.classes.gsgraph import GSGraph
>>> G = GSGraph()
>>> G.add_node('a', popup='sample node popup text', label='A')
>>> G.add_node('b', popup='sample node popup text', label='B')
>>> G.add_edge('a', 'b', directed=True, popup='sample edge popup')
>>> G.edges(data=True)
[('a', 'b', {'source': 'a', 'popup': 'sample edge popup',
'is_directed': True, 'target': 'b'})]
```

add_edge_style(*source*, *target*, *attr_dict=None*, *directed=False*, *color='#000000'*, *width=1.0*, *arrow_shape='triangle'*, *edge_style='solid'*, *arrow_fill='filled'*)

Add styling for an edge whose source and target nodes are provided.

Parameters

- **source** (*str*) – Unique ID of the source node.
- **target** (*str*) – Unique ID of the target node.
- **attr_dict** (*dict, optional*) – Json representation of style of edge. Defaults to None.
- **color** (*str, optional*) – Hexadecimal representation of the color (e.g., #000000), or the color name. Defaults to black.
- **directed** (*bool, optional*) – If True, draw the edge as directed. Defaults to False.
- **width** (*float, optional*) – Width of the edge. Defaults to 1.0.

- **arrow_shape** (*str, optional*) – Shape of arrow head. Defaults to 'triangle'. See *ALLOWED_ARROW_SHAPES* for more details.
- **edge_style** (*str, optional*) – Style of edge. Defaults to 'solid'. See *ALLOWED_EDGE_STYLES* for more details.
- **arrow_fill** (*str, optional*) – Fill of arrow. Defaults to 'filled'. See *ALLOWED_ARROW_FILL* for more details.

Examples

```
>>> from graphspace_python.graphs.classes.gsgraph import GSGraph
>>> G = GSGraph()
>>> G.add_edge_style('a', 'b', directed=True, edge_style='dotted')
>>> G.add_edge_style('b', 'c', arrow_shape='tee', arrow_fill='hollow')
>>> G.get_style_json()
{'style': [{'style': {'width': 1.0, 'line-color': '#000000', 'target-arrow-
↪shape':
'triangle', 'line-style': 'dotted', 'target-arrow-fill': 'filled', 'target-
↪arrow-color':
'#000000'}, 'selector': 'edge[source="a"][target="b"]'}, {'style': {'width':␣
↪1.0,
'line-color': '#000000', 'target-arrow-shape': 'none', 'line-style': 'solid',
'target-arrow-fill': 'hollow', 'target-arrow-color': '#000000'}, 'selector':
'edge[source="b"][target="c"]'}]}
```

add_node (*node_name*, *attr_dict=None*, *parent=None*, *label=None*, *popup=None*, *k=None*, ***attr*)

Add a node to the graph.

Parameters

- **node_name** (*str*) – Name of node.
- **attr_dict** (*dict, optional*) – Json representation of node data. Defaults to None.
- **parent** (*str, optional*) – Parent of the node, if any (for compound nodes). Defaults to None.
- **label** (*str, optional*) – Label of node. Defaults to None.
- **popup** (*str, optional*) – A string that will be displayed in a popup window when the user clicks the node. This string can be HTML-formatted information, e.g., Gene Ontology annotations and database links for a protein; or types, mechanism, and database sources for an interaction.
- **k** (*int, optional*) – An integer-valued attribute for the node, which denotes a rank. Through this attribute, GraphSpace allows the user to filter nodes and edges in a network visualization.
- ****attr** – Arbitrary keyword arguments.

Example

```
>>> from graphspace_python.graphs.classes.gsgraph import GSGraph
>>> G = GSGraph()
>>> G.add_node('a', popup='sample node popup text', label='A')
>>> G.add_node('b', popup='sample node popup text', label='B')
>>> G.nodes(data=True)
```

```
[('a', {'id': 'a', 'popup': 'sample node popup text', 'name': 'a',
'label': 'A'}), ('b', {'id': 'b', 'popup': 'sample node popup text',
'name': 'b', 'label': 'B'})]
```

add_node_style(*node_name*, *attr_dict=None*, *content=None*, *shape='ellipse'*, *color='#FFFFFF'*, *height=None*, *width=None*, *bubble=None*, *valign='center'*, *halign='center'*, *style='solid'*, *border_color='#000000'*, *border_width=1*)

Add styling for a node belonging to the graph.

Parameters

- **node_name** (*str*) – Name of node.
- **attr_dict** (*dict, optional*) – Json representation of style of node. Defaults to None.
- **shape** (*str, optional*) – Shape of node. Defaults to 'ellipse'. See *ALLOWED_NODE_SHAPES* for more details.
- **color** (*str, optional*) – Hexadecimal representation of the color (e.g., #FFFFFF) or color name. Defaults to white.
- **height** (*int, optional*) – Height of the node's body, or None to determine height from the number of lines in the label. Defaults to None.
- **width** (*int, optional*) – Width of the node's body, or None to determine width from length of label. Defaults to None.
- **bubble** (*str, optional*) – Color of the text outline. Using this option gives a "bubble" effect; see the bubbleeffect() function. Defaults to None.
- **valign** (*str, optional*) – Vertical alignment. Defaults to 'center'. See *ALLOWED_TEXT_VALIGN* for more details.
- **halign** (*str, optional*) – Horizontal alignment. Defaults to 'center'. See *ALLOWED_TEXT_HALIGN* for more details.
- **style** (*str, optional*) – Style of border. Defaults to 'solid'. If 'bubble' is specified, then style is overwritten. See *ALLOWED_NODE_BORDER_STYLES* for more details.
- **border_color** (*str, optional*) – Color of border. Defaults to '#000000'. If 'bubble' is specified, then style is overwritten.
- **border_width** (*int, optional*) – Width of border. Defaults to 1. If 'bubble' is specified, then style is overwritten.

Examples

```
>>> from graphspace_python.graphs.classes.gsgraph import GSGraph
>>> G = GSGraph()
>>> G.add_node('a', popup='sample node popup text', label='A')
>>> G.add_node_style('a', shape='ellipse', color='red', width=90, height=90)
>>> G.add_node('b', popup='sample node popup text', label='B')
>>> G.add_node_style('b', color='blue', width=90, height=90, border_color='
↪#4f4f4f')
>>> G.get_style_json()
{'style': [{'style': {'border-color': '#000000', 'border-width': 1, 'height':␣
↪90,
'width': 90, 'shape': 'ellipse', 'border-style': 'solid', 'text-wrap': 'wrap',
'text-halign': 'center', 'text-valign': 'center', 'background-color': 'red'},
```

```
'selector': 'node[name="a"]'}, {'style': {'border-color': '#4f4f4f', 'border-
↪width': 1,
'height': 90, 'width': 90, 'shape': 'ellipse', 'border-style': 'solid', 'text-
↪wrap':
'wrap', 'text-halign': 'center', 'text-valign': 'center', 'background-color':
↪'blue'},
'selector': 'node[name="b"]'}]}
```

add_style(*selector*, *style_dict*)

Add styling for a given selector, for e.g., 'nodes', 'edges', etc.

Parameters

- **selector** (*str*) – A selector functions similar to a CSS selector on DOM elements, but here it works on collections of graph elements.
- **style_dict** (*dict*) – Key-value pair of style attributes and their values.

Examples

```
>>> from graphspace_python.graphs.classes.gsgraph import GSGraph
>>> G = GSGraph()
>>> G.add_style('node', {'background-color': '#bbb', 'opacity': 0.8})
>>> G.add_style('edge', {'line-color': 'green'})
>>> G.get_style_json()
{'style': [{'style': {'opacity': 0.8, 'background-color': '#bbb'}, 'selector':
'node'}, {'style': {'line-color': 'green'}, 'selector': 'edge'}]}
```

static **check_color_hex**(*color_code*)

Check the validity of the hexadecimal code of various node and edge color related attributes.

This function returns an error if the hexadecimal code is not of the format '#XXX' or '#XXXXXX', i.e. hexadecimal color code is not valid.

Parameters **color_code** (*str*) – Hex code of color or color name.

Returns None, if color is valid; error message if color is invalid.

Return type None or str

get_data()

Computes the metadata of the graph and returns it.

Returns Metadata of the graph.

Return type dict

Examples

```
>>> from graphspace_python.graphs.classes.gsgraph import GSGraph
>>> G = GSGraph()
>>> G.get_data()
{'name': 'Graph 12:10PM on July 20, 2017', 'tags': []}
>>> G.set_name('My sample graph')
>>> G.set_tags(['sample','tutorial'])
>>> G.get_data()
{'name': 'My sample graph', 'tags': ['sample', 'tutorial']}
```

get_graph_json()
Computes the json representation for the graph structure from the graph nodes and edges and returns it.

Returns Json representation of graph structure.

Return type dict

Examples

```
>>> from graphspace_python.graphs.classes.gsgraph import GSGraph
>>> G = GSGraph()
>>> G.get_graph_json()
{'elements': {'nodes': [], 'edges': []}, 'data': {'name':
'Graph 12:10PM on July 20, 2017', 'tags': []}}
>>> G.set_name('My sample graph')
>>> G.add_node('a', popup='sample node popup text', label='A')
>>> G.add_node('b', popup='sample node popup text', label='B')
>>> G.add_edge('a', 'b', directed=True, popup='sample edge popup')
>>> G.get_graph_json()
{'elements': {'nodes': [{'data': {'id': 'a', 'popup': 'sample node popup text
↪',
'name': 'a', 'label': 'A'}}, {'data': {'id': 'b', 'popup': 'sample node popup␣
↪text',
'name': 'b', 'label': 'B'}}], 'edges': [{'data': {'source': 'a', 'popup':
'sample edge popup', 'is_directed': True, 'target': 'b'}}]}, 'data': {'name':
'My sample graph', 'tags': []}}
```

get_is_public()
Get visibility status of the graph.

Returns Visibility status of graph. Either 0 or 1.

Return type int

Examples

```
>>> from graphspace_python.graphs.classes.gsgraph import GSGraph
>>> G = GSGraph()
>>> G.get_is_public()
0
>>> G.set_is_public(1)
>>> G.get_is_public()
1
```

get_name()
Get the name of graph.

Returns Name of graph.

Return type str

Examples

```
>>> from graphspace_python.graphs.classes.gsgraph import GSGraph
>>> G = GSGraph()
>>> G.get_name()
'Graph 01:22PM on July 20, 2017'
>>> G.set_name('My sample graph')
>>> G.get_name()
'My sample graph'
```

get_node_position(*node_name*)
Get the x,y position of a node.

Parameters **node_name** (*str*) – Name of the node.

Returns Dict of x,y co-ordinates of the node, if node position is defined; otherwise None.

Return type dict or None

get_style_json()
Get the json representation for the graph style.

Returns Json representation of graph style.

Return type dict

Examples

```
>>> from graphspace_python.graphs.classes.gsgraph import GSGraph
>>> G = GSGraph()
>>> G.get_style_json()
{'style': []}
>>> G.add_node_style('a', shape='ellipse', color='red', width=90, height=90)
>>> G.get_style_json()
{'style': [{'style': {'border-color': '#000000', 'border-width': 1, 'height':␣
↪90,
'width': 90, 'shape': 'ellipse', 'border-style': 'solid', 'text-wrap': 'wrap',
'text-halign': 'center', 'text-valign': 'center', 'background-color': 'red'},
'selector': 'node[name="a"]'}]}
```

get_tags()
Get the tags for the graph.

Returns List of tags of graph.

Return type List[str]

Examples

```
>>> from graphspace_python.graphs.classes.gsgraph import GSGraph
>>> G = GSGraph()
>>> G.get_tags()
[]
>>> G.set_tags(['sample', 'tutorial'])
>>> G.get_tags()
['sample', 'tutorial']
```

json()
Get the json representation of graph details.

Returns Json representation of graph details.

Return type dict

Examples

```
>>> from graphspace_python.graphs.classes.gsgraph import GSGraph
>>> G = GSGraph()
>>> G.json()
{'is_public': 0, 'style_json': {'style': []}, 'tags': [], 'name':
'Graph 12:10PM on July 20, 2017', 'graph_json': {'elements': {'nodes': [],
'edges': []}, 'data': {'name': 'Graph 12:10PM on July 20, 2017', 'tags': []}}}
>>> G.set_name('My sample graph')
>>> G.add_node('a', popup='sample node popup text', label='A')
>>> G.add_node_style('a', shape='ellipse', color='red', width=90, height=90)
>>> G.json()
{'is_public': 0, 'style_json': {'style': [{'style': {'border-color': '#000000
↪',
'border-width': 1, 'height': 90, 'width': 90, 'shape': 'ellipse', 'border-
↪style':
'solid', 'text-wrap': 'wrap', 'text-halign': 'center', 'text-valign': 'center
↪',
'background-color': 'red'}, 'selector': 'node[name="a"]'}]}, 'tags': [], 'name
↪':
'My sample graph', 'graph_json': {'elements': {'nodes': [{'data': {'id': 'a',
'popup': 'sample node popup text', 'name': 'a', 'label': 'A'}}], 'edges': []},
'data': {'name': 'My sample graph', 'tags': []}}}
```

remove_node_position(*node_name*)

Remove the x,y position of a node.

Parameters **node_name** (*str*) – Name of the node.

Raises Exception – If node positions are undefined.

set_data(*data*)

Set the metadata of the graph.

Parameters **data** (*dict*) – Key-value pairs describing the graph.

Example

```
>>> from graphspace_python.graphs.classes.gsgraph import GSGraph
>>> G = GSGraph()
>>> G.set_name('My sample graph')
>>> G.set_tags(['sample'])
>>> G.set_data({'description': 'A sample graph for demonstration purpose.'})
>>> G.get_data()
{'description': 'A sample graph for demonstration purpose.', 'name':
'My sample graph', 'tags': ['sample']}
```

static set_edge_color_property(*edge_properties*, *color*)

Add a edge color to the edge_properties.

Color both the line and the target arrow; if the edge is undirected, then the target arrow color doesn't matter. If it's directed, then the arrow color will match the line color.

Color can be a name (e.g., 'black') or an HTML string (e.g., #00000).

Parameters

- **edge_properties** (*dict*) – Dictionary of edge attributes. Key-value pairs will be used to set data associated with the edge.
- **color** (*str*) – Hexadecimal representation of the color (e.g., #FFFFFF) or color name.

Returns Dictionary of edge attributes.

Return type dict

Raises `Exception` – If the color is improperly formatted.

static set_edge_directionality_property(*edge_properties*, *directed*, *arrow_shape='triangle'*)

Sets a target arrow shape.

Parameters

- **edge_properties** (*dict*) – Dictionary of edge attributes. Key-value pairs will be used to set data associated with the edge.
- **directed** (*bool*) – If True, draw the edge as directed.
- **arrow_shape** (*str*) – Shape of arrow. Defaults to 'triangle'. See ALLOWED_ARROW_SHAPES.

Returns Dictionary of edge attributes.

Return type dict

static set_edge_line_style_property(*edge_properties*, *style*)

Adds the edge line style to edge.

Parameters

- **edge_properties** (*dict*) – Dictionary of edge attributes. Key-value pairs will be used to set data associated with the edge.
- **style** (*str*) – Style of line.

Returns Dictionary of edge attributes.

Return type dict

static set_edge_target_arrow_fill(*edge_properties*, *fill*)

Adds the arrowhead fill to edge.

Parameters

- **edge_properties** (*dict*) – Dictionary of edge attributes. Key-value pairs will be used to set data associated with the edge.
- **fill** (*str*) – Fill of arrowhead.

Returns Dictionary of edge attributes.

Return type dict

static set_edge_target_arrow_shape_property(*edge_properties*, *arrow_shape*)

Assigns an arrow shape to edge.

Parameters

- **edge_properties** (*dict*) – Dictionary of edge attributes. Key-value pairs will be used to set data associated with the edge.

- **arrow_shape** (*str*) – Shape of arrow. See ALLOWED_ARROW_SHAPES.

Returns Dictionary of edge attributes.

Return type dict

static set_edge_width_property(*edge_properties*, *width*)
Sets the width property of the edge.

Parameters

- **edge_properties** (*dict*) – Dictionary of edge attributes. Key-value pairs will be used to set data associated with the edge.
- **width** (*float*) – Width of the edge.

Returns Dictionary of edge attributes.

Return type dict

set_graph_json(*graph_json*)
Set the json representation for the graph structure.

Parameters **graph_json** (*dict*) – Json representation for the graph structure.

Example

```
>>> from graphspace_python.graphs.classes.gsgraph import GSGraph
>>> G = GSGraph()
>>> graph_json = {
...     'elements': {
...             'nodes': [
...                     {
...                             'data': {
...                                     'id': 'a',
...                                     'popup': 'sample node popup text',
...                                     'name': 'a',
...                                     'label': 'A'
...                             }
...                     }
...             ],
...             'edges': []
...     },
...     'data': {
...             'name': 'My sample graph',
...             'tags': ['sample', 'tutorial']
...     }
... }
>>> G.set_graph_json(graph_json)
>>> G.get_graph_json()
{'elements': {'nodes': [{'data': {'id': 'a', 'popup': 'sample node popup text
↪',
'name': 'a', 'label': 'A'}}], 'edges': []}, 'data': {'name': 'My sample graph
↪',
'tags': ['sample', 'tutorial']}}
```

set_is_public(*is_public=1*)
Set visibility status of the graph.

Parameters **is_public** (*int, optional*) – Visibility status of graph. Defaults to 1.

Raises `Exception` – If 'is_public' is neither 0 nor 1.

Examples

```
>>> from graphspace_python.graphs.classes.gsgraph import GSGraph
>>> G = GSGraph()
>>> G.set_is_public() # By default takes param 'is_public' as 1.
>>> G.get_is_public()
1
>>> G.set_is_public(0)
>>> G.get_is_public()
0
```

set_name(*name*)
Set the name of the graph.

Parameters **name** (*str*) – Name of graph.

Example

```
>>> from graphspace_python.graphs.classes.gsgraph import GSGraph
>>> G = GSGraph()
>>> G.set_name('My sample graph')
>>> G.get_name()
'My sample graph'
```

static set_node_border_color_property(*node_properties*, *border_color*)
Set the border_color in node_properties.

Parameters

- **node_properties** (*dict*) – Dictionary of node attributes. Key-value pairs will be used to set data associated with the node.
- **border_color** (*str*) – Hexadecimal representation of the border color (e.g., #FFFFFF) or a color name.

Returns Dictionary of node attributes.

Return type dict

Raises `Exception` – If the border_color is improperly formatted.

static set_node_border_style_property(*node_properties*, *border_style*)
Set the border width in node_properties.

Parameters

- **node_properties** (*dict*) – Dictionary of node attributes. Key-value pairs will be used to set data associated with the node.
- **border_style** (*str*) – Style of border.

Returns Dictionary of node attributes.

Return type dict

Raises `Exception` – If the border_style parameter is not one of the allowed border styles. See ALLOWED_NODE_BORDER_STYLES for more details.

static set_node_border_width_property(*node_properties*, *border_width*)
Set the border width in node_properties.

Parameters

- **node_properties** (*dict*) – Dictionary of node attributes. Key-value pairs will be used to set data associated with the node.
- **border_width** (*int*) – Width of border.

Returns Dictionary of node attributes.

Return type dict

static set_node_bubble_effect_property(*node_properties*, *color*, *whitetext=False*)
Add a "bubble effect" to the node by making the border color the same as the text outline color.

Parameters

- **node_properties** (*dict*) – Dictionary of node attributes. Key-value pairs will be used to set data associated with the node.
- **color** (*str*) – Hexadecimal representation of the text outline color (e.g., #FFFFFF) or a color name.
- **whitetext** (*bool, optional*) – If True, text is colored white instead of black. Defaults to False.

Returns Dictionary of node attributes.

Return type dict

static set_node_color_property(*node_properties*, *color*)
Add a background color to the node_properties. Color can be a name (e.g., 'black') or an HTML string (e.g., #00000).

Parameters

- **node_properties** (*dict*) – Dictionary of node attributes. Key-value pairs will be used to set data associated with the node.
- **color** (*str*) – Hexadecimal representation of the color (e.g., #FFFFFF) or color name.

Returns Dictionary of node attributes.

Return type dict

Raises Exception – If the color is improperly formatted.

static set_node_height_property(*node_properties*, *height*)
Add a node height property to the node_properties. If the height is 'None', then the height of the node is determined by the number of newlines in the label that will be displayed.

Parameters

- **node_properties** (*dict*) – Dictionary of node attributes. Key-value pairs will be used to set data associated with the node.
- **height** (*int or None*) – Height of the node's body, or None to determine height from the number of lines in the label.

Returns Dictionary of node attributes.

Return type dict

static set_node_horizontal_alignment_property(*node_properties*, *halign*)
Set the horizontal alignment of label in node_properties.

Parameters

- **node_properties** (*dict*) – Dictionary of node attributes. Key-value pairs will be used to set data associated with the node.
- **halign** (*str*) – Horizontal alignment of text.

Returns Dictionary of node attributes.

Return type dict

static set_node_label_property(*node_properties*, *label*)
Set 'label' to 'node_properties' dict and return the 'node_properties' dict. The label is stored under 'content' in the node information. Also set wrap = 'wrap' so newlines are interpreted.

Parameters

- **node_properties** (*dict*) – Dictionary of node attributes. Key-value pairs will be used to set data associated with the node.
- **label** (*str*) – Text to display on node. Newline sequence will be interpreted as a line break.

Returns Dictionary of node attributes.

Return type dict

set_node_position(*node_name*, *y*, *x*)
Set the x,y position of a node.

Parameters

- **node_name** (*str*) – Name of the node.
- **y** (*float*) – y co-ordinate of node.
- **x** (*float*) – x co-ordinate of node.

static set_node_shape_property(*node_properties*, *shape*)
Add a shape property "shape" to the node_properties.

Parameters

- **node_properties** (*dict*) – Dictionary of node attributes. Key-value pairs will be used to set data associated with the node.
- **shape** (*str*) – Shape of node.

Returns Dictionary of node attributes.

Return type dict

Raises Exception – If the shape is not one of the allowed node shapes. See ALLOWED_NODE_SHAPES global variable.

static set_node_vertical_alignment_property(*node_properties*, *valign*)
Set the vertical alignment of label in node_properties.

Parameters

- **node_properties** (*dict*) – Dictionary of node attributes. Key-value pairs will be used to set data associated with the node.
- **valign** (*str*) – Vertical alignment of text.

Returns Dictionary of node attributes.

Return type dict

static set_node_width_property(*node_properties*, *width*)
Add a node width property to the node_properties. If the width is 'None', then the width of the node is determined by the length of the label.

Parameters

- **node_properties** (*dict*) – Dictionary of node attributes. Key-value pairs will be used to set data associated with the node.
- **width** (*int or None*) – Width of the node's body, or None to determine width from length of label.

Returns Dictionary of node attributes.

Return type dict

static set_node_wrap_property(*node_properties*, *wrap*)
Adding node wrap allows the newline sequence to be interpreted as a line break for the node.

Parameters

- **node_properties** (*dict*) – Dictionary of node attributes. Key-value pairs will be used to set data associated with the node.
- **wrap** (*str*) – String denoting the type of wrap: one of "wrap" or "none".

Returns Dictionary of node attributes.

Return type dict

Raises Exception – If the wrap parameter is not one of the allowed wrap styles. See ALLOWED_NODE_TEXT_WRAP for more details.

set_style_json(*style_json*)
Set the json representation for the graph style.

Parameters **style_json** (*dict*) – Json representation for the graph style.

Example

```
>>> from graphspace_python.graphs.classes.gsgraph import GSGraph
>>> G = GSGraph()
>>> style_json = {
...     'style': [
...             {
...                     'style': {
...                             'border-color': '#000000',
...                             'border-width': 1,
...                             'height': 90,
...                             'width': 90,
...                             'shape': 'ellipse',
...                             'border-style': 'solid',
...                             'text-wrap': 'wrap',
...                             'text-halign': 'center',
...                             'text-valign': 'center',
...                             'background-color': 'red'
...                     },
...                     'selector': 'node[name="a"]'
...             }
...     ]
... }
```

```
>>> G.set_style_json(style_json)
>>> G.get_style_json()
{'style': [{'style': {'border-color': '#000000', 'border-width': 1, 'height':␣
↪90,
'width': 90, 'shape': 'ellipse', 'border-style': 'solid', 'text-wrap': 'wrap',
'text-halign': 'center', 'text-valign': 'center', 'background-color': 'red'},
'selector': 'node[name="a"]'}]}
```

set_tags (*tags*)
Set the tags for the graph.

Parameters **tags** (*List[str]*) – List of tags of graph.

Example

```
>>> from graphspace_python.graphs.classes.gsgraph import GSGraph
>>> G = GSGraph()
>>> G.set_tags(['sample', 'tutorial'])
>>> G.get_tags()
['sample', 'tutorial']
```

static validate_edge_data_properties (*data_properties*, *nodes_list*)
Validates the data properties.

Parameters

- **data_properties** (*dict*) – Dict of edge data properties.
- **nodes_list** (*List[str]*) – List of nodes.

Raises Exception – If properties are invalid.

static validate_node_data_properties (*data_properties*, *nodes_list*)
Validates the data properties.

Parameters

- **data_properties** (*dict*) – Dict of node data properties
- **nodes_list** (*List[str]*) – List of nodes.

Raises Exception – If properties are invalid.

static validate_property (*element*, *element_selector*, *property_name*, *valid_property_values*)
Goes through array to see if property is contained in the array.

Parameters

- **element** (*dict*) – Element to search for in network.
- **element_selector** (*str*) – Selector for element in the network.
- **property_name** (*str*) – Name of the property.
- **valid_property_values** (*List[str]*) – List of valid properties.

Returns None, if the property is valid or does not exist; error message if property is invalid.

Return type None or str

static validate_style_json (*style_json*)
Validates the json representation of style of graph.

Parameters **style_json** (*dict*) – Json representation for graph style.

Raises Exception – If properties are invalid.

static validate_style_properties(*style_properties*, *selector*)
Validates the style properties.

Parameters

- **style_properties** (*dict*) – Dict of elements style properties.
- **selector** (*str*) – Selector for the element.

Returns None, if properties are valid.

Return type None

Raises Exception – If properties are invalid.

Note: Refer to http://js.cytoscape.org/#selectors for selectors.

GSLayout Class

class graphspace_python.graphs.classes.gslayout.**GSLayout**
Bases: object

GSLayout class.

A GSLayout stores the details of a layout that is understood by GraphSpace.

It stores the X,Y positions of nodes of a graph in an organised json structure.

It also stores the style attributes of the respective nodes and edges in an organised json structure.

It holds the information about the layout such as name and sharing status.

It provides methods to define, modify and delete the details of the layout.

name
str – Name of layout.

is_shared
int – Sharing status of layout. Has value 0 if layout is private, 1 if layout is shared.

style_json
dict – Json representation for layout style.

positions_json
dict – Json representation for layout node positions.

add_edge_style(*source*, *target*, *attr_dict=None*, *directed=False*, *color='#000000'*, *width=1.0*, *arrow_shape='triangle'*, *edge_style='solid'*, *arrow_fill='filled'*)
Add styling for an edge whose source and target nodes are provided.

Parameters

- **source** (*str*) – Unique ID of the source node.
- **target** (*str*) – Unique ID of the target node.
- **attr_dict** (*dict, optional*) – Json representation of style of edge. Defaults to None.

- **color** (*str, optional*) – Hexadecimal representation of the color (e.g., #000000), or the color name. Defaults to black.
- **directed** (*bool, optional*) – If True, draw the edge as directed. Defaults to False.
- **width** (*float, optional*) – Width of the edge. Defaults to 1.0.
- **arrow_shape** (*str, optional*) – Shape of arrow head. Defaults to 'triangle'. See ALLOWED_ARROW_SHAPES for more details.
- **edge_style** (*str, optional*) – Style of edge. Defaults to 'solid'. See ALLOWED_EDGE_STYLES for more details.
- **arrow_fill** (*str, optional*) – Fill of arrow. Defaults to 'filled'. See ALLOWED_ARROW_FILL for more details.

Examples

```
>>> from graphspace_python.graphs.classes.gslayout import GSLayout
>>> L = GSLayout()
>>> L.add_edge_style('a', 'b', directed=True, edge_style='dotted')
>>> L.add_edge_style('b', 'c', arrow_shape='tee', arrow_fill='hollow')
>>> L.get_style_json()
{'style': [{'style': {'width': 1.0, 'line-color': '#000000', 'target-arrow-
↪shape':
'triangle', 'line-style': 'dotted', 'target-arrow-fill': 'filled', 'target-
↪arrow-color':
'#000000'}, 'selector': 'edge[source="a"][target="b"]'}, {'style': {'width':␣
↪1.0,
'line-color': '#000000', 'target-arrow-shape': 'none', 'line-style': 'solid',
'target-arrow-fill': 'hollow', 'target-arrow-color': '#000000'}, 'selector':
'edge[source="b"][target="c"]'}]}
```

add_node_style(*node_name*, *attr_dict=None*, *content=None*, *shape='ellipse'*, *color='#FFFFFF'*, *height=None*, *width=None*, *bubble=None*, *valign='center'*, *halign='center'*, *style='solid'*, *border_color='#000000'*, *border_width=1*)

Add styling for a node belonging to the graph.

Parameters

- **node_name** (*str*) – Name of node.
- **attr_dict** (*dict, optional*) – Json representation of style of node. Defaults to None.
- **shape** (*str, optional*) – Shape of node. Defaults to 'ellipse'. See ALLOWED_NODE_SHAPES for more details.
- **color** (*str, optional*) – Hexadecimal representation of the color (e.g., #FFFFFF) or color name. Defaults to white.
- **height** (*int, optional*) – Height of the node's body, or None to determine height from the number of lines in the label. Defaults to None.
- **width** (*int, optional*) – Width of the node's body, or None to determine width from length of label. Defaults to None.
- **bubble** (*str, optional*) – Color of the text outline. Using this option gives a "bubble" effect; see the bubbleeffect() function. Defaults to None.

- **valign** (*str, optional*) – Vertical alignment. Defaults to 'center'. See ALLOWED_TEXT_VALIGN for more details.
- **halign** (*str, optional*) – Horizontal alignment. Defaults to 'center'. See ALLOWED_TEXT_HALIGN for more details.
- **style** (*str, optional*) – Style of border. Defaults to 'solid'. If 'bubble' is specified, then style is overwritten. See ALLOWED_NODE_BORDER_STYLES for more details.
- **border_color** (*str, optional*) – Color of border. Defaults to '#000000'. If 'bubble' is specified, then style is overwritten.
- **border_width** (*int, optional*) – Width of border. Defaults to 1. If 'bubble' is specified, then style is overwritten.

Examples

```
>>> from graphspace_python.graphs.classes.gslayout import GSLayout
>>> L = GSLayout()
>>> L.add_node_style('a', shape='ellipse', color='red', width=90, height=90)
>>> L.add_node_style('b', color='blue', width=90, height=90, border_color='
↪#4f4f4f')
>>> L.get_style_json()
{'style': [{'style': {'border-color': '#000000', 'border-width': 1, 'height':␣
↪90,
'width': 90, 'shape': 'ellipse', 'border-style': 'solid', 'text-wrap': 'wrap',
'text-halign': 'center', 'text-valign': 'center', 'background-color': 'red'},
'selector': 'node[name="a"]'}, {'style': {'border-color': '#4f4f4f', 'border-
↪width': 1,
'height': 90, 'width': 90, 'shape': 'ellipse', 'border-style': 'solid', 'text-
↪wrap':
'wrap', 'text-halign': 'center', 'text-valign': 'center', 'background-color':
↪'blue'},
'selector': 'node[name="b"]'}]}
```

add_style(*selector*, *style_dict*)

Add styling for a given selector, for e.g., 'nodes', 'edges', etc.

Parameters

- **selector** (*str*) – A selector functions similar to a CSS selector on DOM elements, but here it works on collections of graph elements.
- **style_dict** (*dict*) – Key-value pair of style attributes and their values.

Examples

```
>>> from graphspace_python.graphs.classes.gsgraph import GSGraph
>>> G = GSGraph()
>>> G.add_style('node', {'background-color': '#bbb', 'opacity': 0.8})
>>> G.add_style('edge', {'line-color': 'green'})
>>> G.get_style_json()
{'style': [{'style': {'opacity': 0.8, 'background-color': '#bbb'}, 'selector':
'node'}, {'style': {'line-color': 'green'}, 'selector': 'edge'}]}
```

get_is_shared()

Get sharing status of the layout.

Returns Sharing status of layout. Either 0 or 1.

Return type int

Examples

```
>>> from graphspace_python.graphs.classes.gslayout import GSLayout
>>> L = GSLayout()
>>> L.get_is_shared()
0
>>> L.set_is_shared(1)
>>> L.get_is_shared()
1
```

get_name()
Get the name of layout.

Returns Name of layout.

Return type str

Examples

```
>>> from graphspace_python.graphs.classes.gslayout import GSLayout
>>> L = GSLayout()
>>> L.get_name()
'Layout 03:42PM on July 20, 2017'
>>> L.set_name('My Sample Layout')
>>> L.get_name()
'My Sample Layout'
```

get_node_position(*node_name*)
Get the x,y position of a node.

Parameters **node_name** (*str*) – Name of the node.

Returns Dict of x,y co-ordinates of the node, if node position is defined; otherwise None.

Return type dict or None

Example

```
>>> from graphspace_python.graphs.classes.gslayout import GSLayout
>>> L = GSLayout()
>>> L.set_node_position('a', y=38.5, x=67.3)
>>> L.get_node_position('a')
{'y': 38.5, 'x': 67.3}
```

get_positions_json()
Get the json representation for the layout node postitions.

Returns Json representation of layout node postitions.

Return type dict

Examples

```
>>> from graphspace_python.graphs.classes.gslayout import GSLayout
>>> L = GSLayout()
>>> L.get_positions_json()
{}
>>> L.set_node_position('a', y=38.5, x=67.3)
>>> L.get_positions_json()
{'a': {'y': 38.5, 'x': 67.3}}
```

get_style_json()

Get the json representation for the layout style.

Returns Json representation of layout style.

Return type dict

Examples

```
>>> from graphspace_python.graphs.classes.gslayout import GSLayout
>>> L = GSLayout()
>>> L.get_style_json()
{'style': []}
>>> L.add_node_style('a', shape='ellipse', color='green', width=60, height=60)
>>> L.get_style_json()
{'style': [{'style': {'border-color': '#000000', 'border-width': 1, 'height':␣
↪60,
'width': 60, 'shape': 'ellipse', 'border-style': 'solid', 'text-wrap': 'wrap',
'text-halign': 'center', 'text-valign': 'center', 'background-color': 'green'}
↪,
'selector': 'node[name="a"]'}]}
```

json()

Get the json representation of layout details.

Returns Json representation of layout details.

Return type dict

Examples

```
>>> from graphspace_python.graphs.classes.gslayout import GSLayout
>>> L = GSLayout()
>>> L.json()
{'style_json': {'style': []}, 'positions_json': {}, 'name':
'Layout 03:42PM on July 20, 2017', 'is_shared': 0}
>>> L.set_node_position('a', y=38.5, x=67.3)
>>> L.add_node_style('a', shape='ellipse', color='green', width=60, height=60)
>>> L.set_name('My Sample Layout')
>>> L.json()
{'style_json': {'style': [{'style': {'border-color': '#000000', 'border-width
↪': 1,
'height': 60, 'width': 60, 'shape': 'ellipse', 'border-style': 'solid', 'text-
↪wrap':
'wrap', 'text-halign': 'center', 'text-valign': 'center', 'background-color':
↪'green'},
```

```
'selector': 'node[name="a"]'}]}, 'positions_json': {'a': {'y': 38.5, 'x': 67.
↪3}},
'name': 'My Sample Layout', 'is_shared': 0}
```

remove_node_position(*node_name*)
Remove the x,y position of a node.

Parameters **node_name** (*str*) – Name of the node.

Raises Exception – If node positions are undefined.

Example

```
>>> from graphspace_python.graphs.classes.gslayout import GSLayout
>>> L = GSLayout()
>>> L.set_node_position('a', y=38.5, x=67.3)
>>> L.get_positions_json()
{'a': {'y': 38.5, 'x': 67.3}}
>>> L.remove_node_position('a')
>>> L.get_positions_json()
{}
```

set_is_shared(*is_shared=1*)
Set sharing status of the layout.

Parameters **is_shared** (*int, optional*) – Sharing status of layout. Defaults to 1.

Raises Exception – If 'is_shared' is neither 0 nor 1.

Examples

```
>>> from graphspace_python.graphs.classes.gslayout import GSLayout
>>> L = GSLayout()
>>> L.set_is_shared() # By default takes param 'is_shared' as 1.
>>> L.get_is_shared()
1
>>> L.set_is_shared(0)
>>> L.get_is_shared()
0
```

set_name(*name*)
Set the name of the layout.

Parameters **name** (*str*) – Name of layout.

Example

```
>>> from graphspace_python.graphs.classes.gslayout import GSLayout
>>> L = GSLayout()
>>> L.set_name('My Sample Layout')
>>> L.get_name()
'My Sample Layout'
```

set_node_position (*node_name*, *y*, *x*)
Set the x,y position of a node.

Parameters

- **node_name** (*str*) – Name of the node.
- **y** (*float*) – y co-ordinate of node.
- **x** (*float*) – x co-ordinate of node.

Examples

```
>>> from graphspace_python.graphs.classes.gslayout import GSLayout
>>> L = GSLayout()
>>> L.set_node_position('a', y=38.5, x=67.3)
>>> L.get_positions_json()
{'a': {'y': 38.5, 'x': 67.3}}
>>> L.set_node_position('a', y=45, x=176) # Overwrites the position of 'a'.
>>> L.get_positions_json()
{'a': {'y': 45, 'x': 176}}
```

set_positions_json (*positions_json*)
Set the json representation for the layout node postitions.

Parameters **positions_json** (*dict*) – Json representation of layout node positions.

Example

```
>>> from graphspace_python.graphs.classes.gslayout import GSLayout
>>> L = GSLayout()
>>> positions_json = {
...     'a': {
...             'y': 38.5,
...             'x': 67.3
...     },
...     'b': {
...             'y': 124,
...             'x': 332.2
...     }
... }
>>> L.set_positions_json(positions_json)
>>> L.get_positions_json()
{'a': {'y': 38.5, 'x': 67.3}, 'b': {'y': 124, 'x': 332.2}}
```

set_style_json (*style_json*)
Set the json representation for the layout style.

Parameters **style_json** (*dict*) – Json representation of layout style.

Example

```
>>> from graphspace_python.graphs.classes.gslayout import GSLayout
>>> L = GSLayout()
>>> style_json = {
```

```
...     'style': [
...             {
...                     'style': {
...                             'border-color': '#000000',
...                             'border-width': 1,
...                             'height': 60,
...                             'width': 60,
...                             'shape': 'ellipse',
...                             'border-style': 'solid',
...                             'text-wrap': 'wrap',
...                             'text-halign': 'center',
...                             'text-valign': 'center',
...                             'background-color': 'green'
...                     },
...                     'selector': 'node[name="a"]'
...             }
...     ]
... }
>>> L.set_style_json(style_json)
>>> L.get_style_json()
{'style': [{'style': {'border-color': '#000000', 'border-width': 1, 'height':␣
↪60,
'width': 60, 'shape': 'ellipse', 'border-style': 'solid', 'text-wrap': 'wrap',
'text-halign': 'center', 'text-valign': 'center', 'background-color': 'green'}
↪,
'selector': 'node[name="a"]'}]}
```

GSGroup Class

class graphspace_python.graphs.classes.gsgroup.**GSGroup**(*name=None*, *description=None*)

Bases: object

GSGroup class.

A GSGroup stores the details of a group that is understood by GraphSpace.

It holds the information about the group such as name and description.

It provides methods to define, modify and delete the details of the group.

name
str – Name of group.

description
str – Description of group.

get_description()
Get description of the group.

Returns Description of group.

Return type str

Example

```
>>> from graphspace_python.graphs.classes.gsgroup import GSGroup
>>> group = GSGroup(name='My sample group', description='a sample group for␣
↪demo')
>>> group.get_description()
'a sample group for demo'
```

get_name()
Get the name of group.

Returns Name of group.

Return type str

Example

```
>>> from graphspace_python.graphs.classes.gsgroup import GSGroup
>>> group = GSGroup(name='My sample group', description='a sample group for␣
↪demo')
>>> group.get_name()
'My sample group'
```

json()
Get the json representation of group details.

Returns Json representation of group details.

Return type dict

Example

```
>>> from graphspace_python.graphs.classes.gsgroup import GSGroup
>>> group = GSGroup(name='My sample group', description='a sample group for␣
↪demo')
>>> group.json()
{'name': 'My sample group', 'description': 'a sample group for demo'}
```

set_description(*description*)
Set description of the group.

Parameters **description** (*str*) – Description of group.

Example

```
>>> from graphspace_python.graphs.classes.gsgroup import GSGroup
>>> group = GSGroup()
>>> group.set_description('a sample group for demo')
>>> group.get_description()
'a sample group for demo'
```

set_name(*name*)
Set the name of the group.

Parameters **name** (*str*) – Name of group.

Example

```
>>> from graphspace_python.graphs.classes.gsgroup import GSGroup
>>> group = GSGroup()
>>> group.set_name('My sample group')
>>> group.get_name()
'My sample group'
```

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

graphspace_python.api.client, 29
graphspace_python.api.endpoint.graphs, 30
graphspace_python.api.endpoint.groups, 45
graphspace_python.api.endpoint.layouts, 39
graphspace_python.graphs.classes.gsgraph, 56
graphspace_python.graphs.classes.gsgroup, 78
graphspace_python.graphs.classes.gslayout, 71

Index

A

add_edge() (graphspace_python.graphs.classes.gsgraph.GSGraph method), 57
add_edge_style() (graphspace_python.graphs.classes.gsgraph.GSGraph method), 57
add_edge_style() (graphspace_python.graphs.classes.gslayout.GSLayout method), 71
add_group_member() (graphspace_python.api.endpoint.groups.Groups method), 45
add_node() (graphspace_python.graphs.classes.gsgraph.GSGraph method), 58
add_node_style() (graphspace_python.graphs.classes.gsgraph.GSGraph method), 59
add_node_style() (graphspace_python.graphs.classes.gslayout.GSLayout method), 72
add_style() (graphspace_python.graphs.classes.gsgraph.GSGraph method), 60
add_style() (graphspace_python.graphs.classes.gslayout.GSLayout method), 73
ALLOWED_ARROW_FILL (graphspace_python.graphs.classes.gsgraph.GSGraph attribute), 56
ALLOWED_ARROW_SHAPES (graphspace_python.graphs.classes.gsgraph.GSGraph attribute), 56
ALLOWED_EDGE_STYLES (graphspace_python.graphs.classes.gsgraph.GSGraph attribute), 56
ALLOWED_NODE_BACKGROUND_REPEAT (graphspace_python.graphs.classes.gsgraph.GSGraph attribute), 56
ALLOWED_NODE_BORDER_STYLES (graphspace_python.graphs.classes.gsgraph.GSGraph attribute), 56
ALLOWED_NODE_SHAPES (graphspace_python.graphs.classes.gsgraph.GSGraph attribute), 56
ALLOWED_NODE_TEXT_TRANSFORM (graphspace_python.graphs.classes.gsgraph.GSGraph attribute), 56
ALLOWED_NODE_TEXT_WRAP (graphspace_python.graphs.classes.gsgraph.GSGraph attribute), 56
ALLOWED_TEXT_BACKROUND_SHAPE (graphspace_python.graphs.classes.gsgraph.GSGraph attribute), 56
ALLOWED_TEXT_HALIGN (graphspace_python.graphs.classes.gsgraph.GSGraph attribute), 56
ALLOWED_TEXT_VALIGN (graphspace_python.graphs.classes.gsgraph.GSGraph attribute), 56
ALLOWED_TEXT_WRAP (graphspace_python.graphs.classes.gsgraph.GSGraph attribute), 57
api_host (graphspace_python.api.client.GraphSpace attribute), 29
auth_token (graphspace_python.api.client.GraphSpace attribute), 29

C

check_color_hex() (graphspace_python.graphs.classes.gsgraph.GSGraph static method), 60

D

data (graphspace_python.graphs.classes.gsgraph.GSGraph attribute), 56
delete_graph() (graphspace_python.api.endpoint.graphs.Graphs method), 30
delete_graph_layout() (graphspace_python.api.endpoint.layouts.Layouts method), 39
delete_group() (graphspace_python.api.endpoint.groups.Groups method), 46
delete_group_member() (graphspace_python.api.endpoint.groups.Groups method), 47
description (graphspace_python.graphs.classes.gsgroup.GSGroup attribute), 78

E

edge (graphspace_python.graphs.classes.gsgraph.GSGraph attribute), 56
EDGE_COLOR_ATTRIBUTES (graphspace_python.graphs.classes.gsgraph.GSGraph attribute), 57

G

get_all_groups() (graphspace_python.api.endpoint.groups.Groups method), 48
get_data() (graphspace_python.graphs.classes.gsgraph.GSGraph method), 60
get_description() (graphspace_python.graphs.classes.gsgroup.GSGroup method), 78
get_graph() (graphspace_python.api.endpoint.graphs.Graphs method), 31
get_graph_json() (graphspace_python.graphs.classes.gsgraph.GSGraph method), 60
get_graph_layout() (graphspace_python.api.endpoint.layouts.Layouts method), 40
get_group() (graphspace_python.api.endpoint.groups.Groups method), 49
get_group_graphs() (graphspace_python.api.endpoint.groups.Groups method), 49
get_group_members() (graphspace_python.api.endpoint.groups.Groups method), 50
get_is_public() (graphspace_python.graphs.classes.gsgraph.GSGraph method), 61
get_is_shared() (graphspace_python.graphs.classes.gslayout.GSLayout method), 73
get_my_graph_layouts() (graphspace_python.api.endpoint.layouts.Layouts method), 41
get_my_graphs() (graphspace_python.api.endpoint.graphs.Graphs method), 31
get_my_groups() (graphspace_python.api.endpoint.groups.Groups method), 51
get_name() (graphspace_python.graphs.classes.gsgraph.GSGraph method), 61
get_name() (graphspace_python.graphs.classes.gsgroup.GSGroup method), 79
get_name() (graphspace_python.graphs.classes.gslayout.GSLayout method), 74
get_node_position() (graphspace_python.graphs.classes.gsgraph.GSGraph method), 62
get_node_position() (graphspace_python.graphs.classes.gslayout.GSLayout method), 74
get_positions_json() (graphspace_python.graphs.classes.gslayout.GSLayout method), 74
get_public_graphs() (graphspace_python.api.endpoint.graphs.Graphs method), 32
get_shared_graph_layouts() (graphspace_python.api.endpoint.layouts.Layouts method), 42
get_shared_graphs() (graphspace_python.api.endpoint.graphs.Graphs method), 33
get_style_json() (graphspace_python.graphs.classes.gsgraph.GSGraph method), 62
get_style_json() (graphspace_python.graphs.classes.gslayout.GSLayout method), 75
get_tags() (graphspace_python.graphs.classes.gsgraph.GSGraph method), 62
graph_json (graphspace_python.graphs.classes.gsgraph.GSGraph attribute), 56
Graphs (class in graphspace_python.api.endpoint.graphs), 30
GraphSpace (class in graphspace_python.api.client), 29
graphspace_python.api.client (module), 29
graphspace_python.api.endpoint.graphs (module), 30
graphspace_python.api.endpoint.groups (module), 45
graphspace_python.api.endpoint.layouts (module), 39
graphspace_python.graphs.classes.gsgraph (module), 56
graphspace_python.graphs.classes.gsgroup (module), 78
graphspace_python.graphs.classes.gslayout (module), 71
Groups (class in graphspace_python.api.endpoint.groups), 45
GSGraph (class in graphspace_python.graphs.classes.gsgraph), 56
GSGroup (class in graphspace_python.graphs.classes.gsgroup), 78
GSLayout (class in graphspace_python.graphs.classes.gslayout), 71

I

is_public (graphspace_python.graphs.classes.gsgraph.GSGraph attribute), 56
is_shared (graphspace_python.graphs.classes.gslayout.GSLayout attribute), 71

J

json() (graphspace_python.graphs.classes.gsgraph.GSGraph method), 62
json() (graphspace_python.graphs.classes.gsgroup.GSGroup method), 79
json() (graphspace_python.graphs.classes.gslayout.GSLayout method), 75

L

Layouts (class in graphspace_python.api.endpoint.layouts), 39

N

name (graphspace_python.graphs.classes.gsgraph.GSGraph attribute), 56
name (graphspace_python.graphs.classes.gsgroup.GSGroup attribute), 78
name (graphspace_python.graphs.classes.gslayout.GSLayout attribute), 71

node (graphspace_python.graphs.classes.gsgraph.GSGraph attribute), 56
NODE_COLOR_ATTRIBUTES (graphspace_python.graphs.classes.gsgraph.GSGraph attribute), 57

P

positions_json (graphspace_python.graphs.classes.gslayout.GSLayout attribute), 71
post_graph() (graphspace_python.api.endpoint.graphs.Graphs method), 33
post_graph_layout() (graphspace_python.api.endpoint.layouts.Layouts method), 43
post_group() (graphspace_python.api.endpoint.groups.Groups method), 52
publish_graph() (graphspace_python.api.endpoint.graphs.Graphs method), 34

R

remove_node_position() (graphspace_python.graphs.classes.gsgraph.GSGraph method), 63
remove_node_position() (graphspace_python.graphs.classes.gslayout.GSLayout method), 76

S

set_api_host() (graphspace_python.api.client.GraphSpace method), 29
set_data() (graphspace_python.graphs.classes.gsgraph.GSGraph method), 63
set_default_graph_layout() (graphspace_python.api.endpoint.graphs.Graphs method), 35
set_description() (graphspace_python.graphs.classes.gsgroup.GSGroup method), 79
set_edge_color_property() (graphspace_python.graphs.classes.gsgraph.GSGraph static method), 63
set_edge_directionality_property() (graphspace_python.graphs.classes.gsgraph.GSGraph static method), 64
set_edge_line_style_property() (graphspace_python.graphs.classes.gsgraph.GSGraph static method), 64
set_edge_target_arrow_fill() (graphspace_python.graphs.classes.gsgraph.GSGraph static method), 64
set_edge_target_arrow_shape_property() (graphspace_python.graphs.classes.gsgraph.GSGraph static method), 64
set_edge_width_property() (graphspace_python.graphs.classes.gsgraph.GSGraph static method), 65
set_graph_json() (graphspace_python.graphs.classes.gsgraph.GSGraph method), 65
set_is_public() (graphspace_python.graphs.classes.gsgraph.GSGraph method), 65
set_is_shared() (graphspace_python.graphs.classes.gslayout.GSLayout method), 76
set_name() (graphspace_python.graphs.classes.gsgraph.GSGraph method), 66
set_name() (graphspace_python.graphs.classes.gsgroup.GSGroup method), 79
set_name() (graphspace_python.graphs.classes.gslayout.GSLayout method), 76
set_node_border_color_property() (graphspace_python.graphs.classes.gsgraph.GSGraph static method), 66
set_node_border_style_property() (graphspace_python.graphs.classes.gsgraph.GSGraph static method), 66
set_node_border_width_property() (graphspace_python.graphs.classes.gsgraph.GSGraph static method), 66
set_node_bubble_effect_property() (graphspace_python.graphs.classes.gsgraph.GSGraph static method), 67
set_node_color_property() (graphspace_python.graphs.classes.gsgraph.GSGraph static method), 67
set_node_height_property() (graphspace_python.graphs.classes.gsgraph.GSGraph static method), 67
set_node_horizontal_alignment_property() (graphspace_python.graphs.classes.gsgraph.GSGraph static method), 67
set_node_label_property() (graphspace_python.graphs.classes.gsgraph.GSGraph static method), 68
set_node_position() (graphspace_python.graphs.classes.gsgraph.GSGraph method), 68
set_node_position() (graphspace_python.graphs.classes.gslayout.GSLayout method), 76
set_node_shape_property() (graphspace_python.graphs.classes.gsgraph.GSGraph static method), 68
set_node_vertical_alignment_property() (graphspace_python.graphs.classes.gsgraph.GSGraph static method), 68
set_node_width_property() (graphspace_python.graphs.classes.gsgraph.GSGraph static method), 68
set_node_wrap_property() (graphspace_python.graphs.classes.gsgraph.GSGraph static method), 69
set_positions_json() (graphspace_python.graphs.classes.gslayout.GSLayout method), 77
set_style_json() (graphspace_python.graphs.classes.gsgraph.GSGraph method), 69

set_style_json() (graphspace_python.graphs.classes.gslayout.GSLayout method), 77
set_tags() (graphspace_python.graphs.classes.gsgraph.GSGraph method), 70
share_graph() (graphspace_python.api.endpoint.groups.Groups method), 52
style_json (graphspace_python.graphs.classes.gsgraph.GSGraph attribute), 56
style_json (graphspace_python.graphs.classes.gslayout.GSLayout attribute), 71

T

tags (graphspace_python.graphs.classes.gsgraph.GSGraph attribute), 56

U

unpublish_graph() (graphspace_python.api.endpoint.graphs.Graphs method), 36
unset_default_graph_layout() (graphspace_python.api.endpoint.graphs.Graphs method), 37
unshare_graph() (graphspace_python.api.endpoint.groups.Groups method), 53
update_graph() (graphspace_python.api.endpoint.graphs.Graphs method), 38
update_graph_layout() (graphspace_python.api.endpoint.layouts.Layouts method), 44
update_group() (graphspace_python.api.endpoint.groups.Groups method), 55
username (graphspace_python.api.client.GraphSpace attribute), 29

V

validate_edge_data_properties() (graphspace_python.graphs.classes.gsgraph.GSGraph static method), 70
validate_node_data_properties() (graphspace_python.graphs.classes.gsgraph.GSGraph static method), 70
validate_property() (graphspace_python.graphs.classes.gsgraph.GSGraph static method), 70
validate_style_json() (graphspace_python.graphs.classes.gsgraph.GSGraph static method), 70
validate_style_properties() (graphspace_python.graphs.classes.gsgraph.GSGraph static method), 71