ndex2 Documentation

Release 3.5.0

Dexter Pratt, Aaron Gary & Jing Chen

Jun 28, 2022

Contents:

CHAPTER 1

Overview

The NDEx2 Python Client provides methods to access NDEx via the NDEx REST Server API. As well as methods for common operations on networks via the NiceCXNetwork class.

CHAPTER 2

Dependencies

- six
- ijson
- requests
- requests_toolbelt
- networkx
- urllib3
- pandas
- enum34 (Python < 3.4)
- numpy
- enum (Python 2.6 & 2.7)

CHAPTER 3

Compatibility

Python 3.5+

Note: Python < 3.5 may have some issues

CHAPTER 4

Installation

The NDEx2 Python Client module can be installed from the Python Package Index (PyPI) repository using PIP:

```
pip install ndex2
```

If you already have an older version of the ndex2 module installed, you can use this command instead:

```
pip install --upgrade ndex2
```

CHAPTER 5

License

See LICENSE.txt

5.1 Installation

5.1.1 From Pypi

5.1.2 Stable release

To install NDEx2 Python Client, run this command in your terminal:

```
pip install ndex2
```

If you don't have pip installed, this Python installation guide can guide you through the process.

5.1.3 From sources

The sources for NDEx2 Python Client can be downloaded from the Github repo.

You can either clone the public repository:

```
git clone git://github.com/ndexbio/ndex2-client
```

Or download the tarball:

```
curl  -OL https://github.com/ndexbio/ndex2-client/tarball/master
```

Once you have a copy of the source, you can install it with:

```
python setup.py install
```

5.2 Quick Tutorial

Below are some small, fully runnable, code blocks that show how to download, edit, and upload networks in NDEx

Note: For examples below, it is assumed that NDEx2 Python client is

5.2.1 Download network from NDEx

The code block below uses the NDEx2 Python client to download BioGRID: Protein-Protein Interactions (SARS-CoV) network from NDEx as a NiceCXNetwork.

The number of nodes and edges are then printed out and the network is converted to Networkx object.

```
import json
import ndex2


# Create NDEx2 python client
client = ndex2.client.Ndex2()

# Download BioGRID: Protein-Protein Interactions (SARS-CoV) from NDEx
# https://www.ndexbio.org/viewer/networks/669f30a3-cee6-11ea-aaef-0ac135e8bacf
client_resp = client.get_network_as_cx_stream('669f30a3-cee6-11ea-aaef-0ac135e8bacf')

# Convert downloaded network to NiceCXNetwork object
net_cx = ndex2.create_nice_cx_from_raw_cx(json.loads(client_resp.content))

# Display information about network and output 1st 100 characters of CX
print('Name: ' + net_cx.get_name())
print('Number of nodes: ' + str(len(list(net_cx.get_nodes()))))
print('Number of nodes: ' + str(len(list(net_cx.get_edges()))))
print(json.dumps(net_cx.to_cx())[0:100])

# Create Networkx network
g = net_cx.to_networkx(mode='default')

print('Name: ' + str(g))
print('Number of nodes: ' + str(g.number_of_nodes()))
print('Number of edges: ' + str(g.number_of_edges()))
print('Network annotations: ' + str(g.graph))
```

5.2.2 Upload new network to NDEx

The code block below shows how to upload a network that is a *NiceCXNetwork* object to NDEx.

```
import ndex2

# Create a test network
net_cx = ndex2.nice_cx_network.NiceCXNetwork()

# Set name of network
net_cx.set_name('Upload new network to NDEx')
```

(continues on next page)

(continued from previous page)

```
# Create two nodes and one edge
node_one_id = net_cx.create_node(node_name='foo', node_represents='representing foo')
node_two_id = net_cx.create_node(node_name='bar', node_represents='representing bar')
net_cx.create_edge(edge_source=node_one_id, edge_target=node_two_id, edge_interaction=
↪'interacts')

# Create client, be sure to replace <USERNAME> and <PASSWORD> with NDEx username &␣
↪password
client = ndex2.client.Ndex2(username='<USERNAME>', password='<PASSWORD>')

# Save network to NDEx, value returned is link to raw CX data on server.
client.save_new_network(net_cx.to_cx(), visibility='PRIVATE')

# Example return value: https://www.ndexbio.org/v2/network/4027bead-89f2-11ec-b3be-
↪0ac135e8bacf
# To view network in NDEx replace 'v2' with 'viewer' and add 's' to 'network' like so:
# https://www.ndexbio.org/viewer/networks/4027bead-89f2-11ec-b3be-0ac135e8bacf
```

Note: To update an existing network replace save_new_network() in code block above with update_cx_network() and set first argument to `net_cx.to_cx_stream()` and the second argument to str UUID of network

5.2.3 Add nodes, edges, and attributes to network

The code block below shows how to add nodes, edges and attributes to a NiceCXNetwork object

```
import ndex2

# create an empty NiceCXNetwork object
# a NiceCXNetwork could also be downloaded from NDEx or created from CX data
net_cx = ndex2.nice_cx_network.NiceCXNetwork()

# create a node, id of node is returned
node_one_id = net_cx.create_node(node_name='foo', node_represents='representing foo')

# create another node
node_two_id = net_cx.create_node(node_name='bar', node_represents='representing bar')

# create an edge connecting the nodes, id of edge is returned
edge_id = net_cx.create_edge(edge_source=node_one_id, edge_target=node_two_id, edge_
↪interaction='interacts')

# add attribute named 'altname' to 'foo' node, nothing is returned
net_cx.set_node_attribute(node_one_id, 'altname', 'alternate name for foo', type=
↪'string')

# add attribute to 'bar' node
net_cx.set_node_attribute(node_two_id, 'altname', 'alternate name for bar', type=
↪'string')

# add an edge attribute named 'weight' with value of 0.5. Set as string
# value and then set type.
net_cx.set_edge_attribute(edge_id, 'weight', '0.5', type='double')
```

(continues on next page)

(continued from previous page)

```
# Create Networkx network
g = net_cx.to_networkx(mode='default')

print('Name: ' + str(g))
print('Number of nodes: ' + str(g.number_of_nodes()))
print('Number of edges: ' + str(g.number_of_edges()))
print('Node annotations: ' + str(g.nodes.data()))
print('Edge annotations: ' + str(g.edges.data()))
```

5.2.4 Build a lookup table for node names to node ids

The code block below shows how to iterate through nodes in a NiceCXNetwork object and build a dict of node names to node ids. The network downloaded below is Multi-Scale Integrated Cell (MuSIC) v1

```
import ndex2
import json

# Create NDEx2 python client
client = ndex2.client.Ndex2()

# Download MuSIC network from NDEx
client_resp = client.get_network_as_cx_stream('7fc70ab6-9fb1-11ea-aaef-0ac135e8bacf')

# Convert downloaded network to NiceCXNetwork object
net_cx = ndex2.create_nice_cx_from_raw_cx(json.loads(client_resp.content))

node_name_dict = {}

# Build dictionary and print out all the nodes
for node_id, node_obj in net_cx.get_nodes():
    print('node_id: ' + str(node_id) + ' node_obj: ' + str(node_obj))
    node_name_dict[node_obj['n']] = node_id


# Print out dictionary
print(str(node_name_dict))
```

5.2.5 More Tutorials and Examples

- Basic Use of the NDEx2 Python Client: NDEx2 Client v2.0 Tutorial
- Working with the NiceCX Network Class: NiceCX v2.0 Tutorial

To use these tutorials or if Github isn't showing the above notebooks in the browser, clone the ndex-jupyter-notebooks repository to your local machine and start Jupyter Notebooks in the project directory.

For information on installing and using Jupyter Notebooks, go to jupyter.org

- Click here for example code to load content into NDEx

5.3 Reference

The NDEx2 Python Client can be broken into two main parts:

1. *NiceCXNetwork* provides a data model for working with NDEx networks stored in CX format
2. *Ndex2* REST client provides provides methods to interact with NDEx REST Service

5.3.1 Creating NiceCXNetwork objects

ndex2.**create_nice_cx_from_raw_cx**(*cx*)

Create a *NiceCXNetwork()* from a as a *list* of *dict* objects in CX format

Example:

```
import json
import ndex2

# cx_as_str is a str containing JSON in CX format above
net_cx = ndex2.create_nice_cx_from_raw_cx(json.loads(cx_as_str))
```

Parameters **cx** (*list*) – CX as a *list* of *dict* objects

Returns NiceCXNetwork

Return type *NiceCXNetwork()*

ndex2.**create_nice_cx_from_file**(*path*)

Create a *NiceCXNetwork()* from a file that is in the CX format

Parameters **path** (*str*) – the path of the CX file

Raises

- **Exception** – if *path* is not a file
- **OSError** – if there is an error opening the *path* file
- **JSONDecodeError** – if there is an error parsing the *path* file with json.load()

Returns NiceCXNetwork

Return type *NiceCXNetwork()*

ndex2.**create_nice_cx_from_server**(*server*, *username=None*, *password=None*, *uuid=None*, *ndex_client=None*)

Create a *NiceCXNetwork()* based on a network retrieved from NDEx, specified by its UUID.

Changed in version 3.5.0: Code refactor and **ndex_client** parameter has been added

If the network is not public, then **username** and **password** arguments (or **ndex_client** with username and password set) for an account on the server with permission to access the network must be supplied.

Example usage:

```
import ndex2

# Download BioGRID: Protein-Protein Interactions (SARS-CoV) from NDEx
# https://www.ndexbio.org/viewer/networks/669f30a3-cee6-11ea-aaef-0ac135e8bacf
net_cx = ndex2.create_nice_cx_from_server(None, uuid='669f30a3-cee6-11ea-aaef-
↪0ac135e8bacf')
```

Note: If **ndex_client** is not passed in, this function internally creates *Ndex2* using values from parameters passed into this function.

Parameters

- **server** (*str*) – the URL of the NDEx server hosting the network
- **username** (*str*) – the user name of an account with permission to access the network
- **password** (*str*) – the password of an account with permission to access the network
- **uuid** (*str*) – the UUID of the network
- **ndex_client** (*Ndex2*) – Used as NDEx REST client overriding **server**, **username** and **password** parameters if set

Raises ***NDExError*** – If uuid is not specified

Returns NiceCXNetwork

Return type *NiceCXNetwork()*

Networkx

ndex2.**create_nice_cx_from_networkx**(*G*)

Creates a *NiceCXNetwork* based on a `networkx.Graph` graph.

Changed in version 3.5.0: Major refactor to fix multiple bugs #83, #84, #90

```
import ndex2
import networkx as nx

G = nx.Graph()
G.add_node(1, someval=1.5, name='node 1')
G.add_node(2, someval=2.5, name='node 2')
G.add_edge(1, 2, weight=5)

print(ndex2.create_nice_cx_from_networkx(G).to_cx())
```

The resulting *NiceCXNetwork* contains the nodes, edges and their attributes from the `networkx.Graph` graph and also preserves the graph 'pos' attribute as a CX cartesian coordinates aspect *CARTESIAN_LAYOUT_ASPECT* with the values of *Y* inverted

Description of how conversion is performed:

Network:

- Network name is set value of `G.graph.get('name')` or to `created from networkx by ndex2.create_nice_cx_networkx()` if *name* is `None` or not present

Nodes:

- Node id is value of `n` from this for loop: `for n, d G.nodes(data=True):` if `n` is **NOT** an `int`, new ids starting from `0` are used
- Node name is value of *name* attribute on the node or is set to id of node if *name* is not present.
- Node *represents* is value of *represents* attribute on the node or set is to node *name* if `None` or not present

Edges:

- Interaction is value of *interaction* attribute on the edge or is set to `neighbor-of` if `None` or not present

---

Note: Data types are inferred by using `isinstance()` and converted to corresponding CX data types. For list items, only the 1st item is examined to determine type

---

Parameters **G** (networkx.Graph) – Graph to convert

Raises **Exception** – if **G** parameter is None or there is another error in conversion

Returns Converted network

Return type *NiceCXNetwork*

Pandas

ndex2.**create_nice_cx_from_pandas**(*df*, *source_field=None*, *target_field=None*, *source_node_attr=[]*, *target_node_attr=[]*, *edge_attr=[]*, *edge_interaction=None*, *source_represents=None*, *target_represents=None*)

Create a *NiceCXNetwork()* from a pandas.DataFrame in which each row specifies one edge in the network.

Changed in version 3.5.0: Removed print statements showing progress and network name is now being set

If only the **df** argument is provided the pandas.DataFrame is treated as 'SIF' format, where the first two columns specify the source and target node ids of the edge and all other columns are ignored. The edge interaction is defaulted to "interacts-with"

If both the source_field and target_field arguments are provided, then those and any other arguments refer to headers in the pandas.DataFrame, controlling the mapping of columns to the attributes of nodes, and edges in the resulting *NiceCXNetwork()*.

If a header is not mapped, the corresponding column is ignored.

If the edge_interaction is not specified, interaction is set to "interacts-with"

```
import ndex2
import pandas as pd

data = {'source': ['Node 1','Node 2'],
        'target': ['Node 2','Node 3'],
        'interaction': ['helps', 'hurts']}
df = pd.DataFrame.from_dict(data)

net = ndex2.create_nice_cx_from_pandas(df, source_field='source',
                                       target_field='target',
                                       edge_interaction='interaction')

print(net.get_nodes())
print(net.get_edges())
```

Note: The datatype for everything added to the network is the CX string type

Parameters

- **df** (pandas.DataFrame) – Pandas dataframe to process
- **source_field** (*str*) – header name specifying the name of the source node.
- **target_field** (*str*) – header name specifying the name of the target node.
- **source_node_attr** (*list*) – list of header names specifying attributes of the source node.

- **target_node_attr** (*list*) – list of header names specifying attributes of the target node.
- **edge_attr** (*list*) – list of header names specifying attributes of the edge.
- **edge_interaction** (*str*) – the relationship between the source node and the target node, defaulting to "interacts-with"
- **source_represents** (*str*) –
- **target_represents** (*str*) –

Returns NiceCXNetwork

Return type *NiceCXNetwork()*

5.3.2 Converting NiceCXNetwork objects to other formats

Below are converters that facilitate conversion of *NiceCXNetwork* object to other types (such as NetworkX)

Networkx

class ndex2.nice_cx_network.**DefaultNetworkXFactory**(*legacymode=False*)

Converts *NiceCXNetwork* to networkx.Graph object or one of its subtypes

For details on implementation see *get_graph()*

Constructor

Note: the parameters in the constructor change behavior of *get_graph()*

Parameters **legacymode** (*bool*) – If set to True then *get_graph()* behaves like NDEx2 Python client version 3.1 and earlier in that this method returns a networkx.Graph object. see *get_graph()* for more information

Raises ***NDExError*** – If invalid value is set in *legacymode* parameter

get_graph(*nice_cx_network*, *networkx_graph=None*)

Creates a networkx.Graph, or a subtype, object from *nice_cx_network* passed in.

> **Warning:** Converting large networks (10,000+ edges or nodes) may take a long time and consume lots of memory.

The conversion is done as follows:

Any network attributes are copied to the networkx.Graph in manner described here: *add_network_attributes_from_nice_cx_network()*

For nodes:

All nodes are added with the node id set to the id or *NODE_ID* of input network nodes.

A node attribute named 'name' is set for each node with its value set to the value of the 'name' attribute from the input network.

If 'r' exists on node, the value is added as a node attribute named 'represents' (unless *legacymode* is set to *True* in constructor)

All other node attributes are added using the same attribute name as found in the input network. The value is directly set as it was found in input network (could be single object or list)

For edges:

Each edge is added setting the source to the value of *EDGE_SOURCE* attribute and target set as *EDGE_TARGET* attribute of input network.

Any edge attributes named *EDGE_INTERACTION* are renamed 'interaction' and stored as an attribute for the edge

Changed in version 3.5.0: If the value of an edge attribute is a list then the value is set directly in the graph as is as opposed to being converted into a comma delimited string

Coordinates are copied in manner described here: *copy_cartesian_coords_into_graph()*

Warning: If **legacymode** is set to True in constructor then:

- networkx.Graph created by this method does **NOT** support multiple edges between the same nodes. Extra edges encountered are **ignored** and not converted.
- In addition, the 'r' attribute in the node dict is **NOT** copied to the resulting networkx.Graph object.
- *networkx_graph* parameter is ignored

Parameters

- **nice_cx_network** (*NiceCXNetwork*) – Network to extract graph from
- **networkx_graph** (networkx.Graph or subtype) – Empty networkx graph to populate which is **IGNORED** if *legacymode* is set to True in constructor. If unset and *legacymode* is False in constructor then a networkx.MultiDiGraph is created

Raises ***NDExError*** – if input network is None

Returns Input network converted to networkx Graph

Return type networkx.Graph if legacymode is set to True in constructor otherwise networkx.MultiDiGraph unless *networkx_graph* is set in which case *networkx_graph* is returned

This networkx converter is still callable, but has been deprecated

class ndex2.nice_cx_network.**LegacyNetworkXVersionTwoPlusFactory**

Deprecated since version 3.2.0: This implementation contains errors, but is left for backwards compatibility of NiceCXNetwork.to_networkx()

Converts *NiceCXNetwork* to networkx.Graph object following logic in legacy NDEx2 Python client when networkx 2.0+ is installed.

Warning: This implementation assumes networkx 2.0+ is installed and will fail with older versions.

For conversion details see *get_graph()*

Constructor

get_graph(*nice_cx_network*, *networkx_graph=None*)

Creates a networkx.Graph object from *nice_cx_network* passed in.

Deprecated since version 3.2.0: This implementation contains errors, but is left for backwards compatibility of NiceCXNetwork.to_networkx()

> **Warning:** Converting large networks (10,000+ edges or nodes) may take a long time and consume lots of memory.
>
> This implementation uses node name as ID for nodes, which is problematic if multiple nodes share the same name and results in invalid mapping of node positions
>
> networkx.Graph created by this method does NOT support multiple edges between the same nodes. Extra edges encountered are **ignored** and not converted.

The conversion is done as follows:

Any network attributes are copied to the networkx.Graph in manner described here: *add_network_attributes_from_nice_cx_network()*

For nodes:

All nodes are added with the node id set to value of 'n' on node. For multiple nodes with same 'n' value behavior is unknown

A node attribute named 'name' is set for each node with its value set to the value of the 'name' attribute from the input network.

If 'r' exists on node, the value is added as a node attribute named 'represents'

All other node attributes are added using the same attribute name as found in the input network. The value is directly set name as found in the input network. The value is directly set as it was found in input network (could be single object or list)

For edges:

Each edge is added setting the source to the value of 's' attribute and target set as 't' attribute of input network.

Any edge attributes named 'i' are renamed 'interaction' and stored as an attribute for the edge

If the value of an edge attribute is a list then the list values are turned into a string separated by a comma and then enclosed by double quotes.

Coordinates are copied in manner described here: *copy_cartesian_coords_into_graph()*

Parameters

- **nice_cx_network** (*NiceCXNetwork*) – Network to extract graph from
- **networkx_graph** (networkx.Graph or subtype) – ignored by this implementation

Returns Input network converted to networkx Graph

Return type networkx.Graph

Base class for 'Networkx <https://networkx.org/>'__ converters above

class ndex2.nice_cx_network.**NetworkXFactory**

Base class for subclasses that implement a factory that creates networkx.Graph objects and contains a couple utility methods used by implementing factory classes

add_edge(*networkx_graph*, *source_node*, *target_node*, *attribute_dict*)

Adds edge to *graph* dealing with differences between networkx 1.x and 2.x+

Parameters

- **networkx_graph** (networkx.Graph or one of its subtypes) – networkx graph to add node to
- **source_node** – id of source node

- **target_node** – id of target node
- **attribute_dict** (*dict*) – dictionary of edge attributes

Returns None

add_network_attributes_from_nice_cx_network(*nice_cx_network*, *networkx_graph*)
Iterates through network attributes of input *nice_cx_network* appending the attributes to the graph object passed in setting the values like so:

```
networkx_graph.graph[attribute_name] = attribute_value
```

If the value of a network attribute is of type list then the values are converted to strings and concatenated into a single string separated by commas.

Parameters

- **nice_cx_network** (*NiceCXNetwork*) – Network to extract network attributes from
- **networkx_graph** (networkx.Graph) – networkx Graph object, should work with any of the types of Graphs ie MultiGraph etc..

Raises ***NDExError*** – If either input parameter is None

Returns None

add_node(*networkx_graph*, *nodeid*, *node_attributes*, *name=None*, *represents=None*)
Adds node to *graph* dealing with differences between networkx 1.x and 2.x+

Parameters

- **networkx_graph** (networkx.Graph or one of its subtypes) – networkx graph to add node to
- **nodeid** – node identifier can be string, int etc.
- **node_attributes** (*dict*) – dictionary of key => value data to set set node attributes with
- **name** (*string*) – name of node that is set as attribute with key 'name' on node
- **represents** – represents value for node that is set as attribute with key 'represents' on node

Returns None

copy_cartesian_coords_into_graph(*nice_cx_network*, *networkx_graph*)
Examines the *nice_cx_network* extracting the content of the opaque aspect *CARTESIAN_LAYOUT_ASPECT*

Changed in version 3.5.0: code now inverts value of y coordinate so position is correct in networkx

If data is found in above aspect, then this method iterates through the list of values which is assumed to be a dictionary of node ids with coordinates as seen here:

```
[
 { 'node': <id>, 'x': <x coord>, 'y': <y coord},
 { 'node': <id>, 'x': <x coord>, 'y': <y coord},
 .
 .
]
```

These values (as seen in example above) are stored in the *networkx_graph* object as tuples with id of node set as key like so:

```
networkx_graph.pos[<id from above>] = (<x coord>, <y coord>)
```

Parameters

- **nice_cx_network** (*NiceCXNetwork*) – Input network
- **networkx_graph** (networkx.Graph) – Network to append coordinates to

Raises ***NDExError*** – If either input parameter is None

Returns None

Pandas

For conversion to Pandas see `to_pandas_dataframe()`

5.3.3 NiceCXNetwork

The *NiceCXNetwork* class provides a data model for working with NDEx networks that are stored in CX format

Note: The term **niceCX** is CX with no duplicate aspects.

Methods are provided to add nodes, edges, node attributes, edge attributes, etc. Once a *NiceCXNetwork* object is populated it can be saved to the NDEx server by calling either `upload_to()` to create a new network or `update_to()` to update an existing network.

Methods

Example usage of the methods below can be found in the Jupyter notebook links here:

Tutorial Notebook Navigating NiceCXNetwork Notebook

Node methods

class ndex2.nice_cx_network.**NiceCXNetwork**(***attr*)

create_node(*node_name=None*, *node_represents=None*)

Creates a new node with the corresponding name and represents (external id)

Warning: Version *3.3.1* and prior of this library had a bug that caused this method to behave incorrectly. Please upgrade to 3.3.2 or greater.

Example:

```
my_node = create_node(node_name='MAPK1, node_represents='1114208')
```

Parameters

- **node_name** (*str*) – Name of the node
- **node_represents** (*str*) – Representation of the node (alternate identifier)

Returns Node ID

Return type int

get_node_attribute(*node*, *attribute_name*)
Get the node attribute of a node, where the node may be specified by its id or passed in as an object.

Example:

```
get_node_attribute(my_node, 'Pathway')   # returns: {'@id': 0,
'n': 'diffusion-heat', 'v': 0.832, 'd': 'double'}
```

Parameters

- **node** (*int or node dict with @id attribute*) – node object or node id
- **attribute_name** – attribute name

Returns the node attibute object or None if the attribute doesn't exist

Return type dict

get_node_attribute_value(*node*, *attribute_name*)
Get the value(s) of an attribute of a node, where the node may be specified by its id or passed in as an object.

Example:

```
get_node_attribute_value(my_node, 'Pathway') # returns: 'Signal
Transduction / Growth Regulation'
```

Parameters

- **node** (*int or node dict with @id attribute*) – node object or node id
- **attribute_name** – attribute name

Returns the value of the attibute or None if the attribute doesn't exist

Return type string

get_node_attributes(*node*)
Get the attribute objects of a node, where the node may be specified by its id or passed in as an object.

Example:

```
get_node_attributes(my_node)        # returns: [{'po': 0, 'n':
'Pathway', 'v': 'Signal Transduction / Growth Regulation'}]
```

Parameters **node** (*int or node dict with @id attribute*) – node object or node id

Returns node attributes

Return type list

get_nodes()
Returns an iterator over node ids as keys and node objects as values.

Example:

```
for id, node in nice_cx.get_nodes():

node_name = node.get('n')

node_represents = node.get('r')
```

Returns iterator over nodes

Return type iterator

set_node_attribute(*node*, *attribute_name*, *values*, *type=None*, *overwrite=False*)

Set an attribute of a node, where the node may be specified by its id or passed in as a node dict.

Example:

```
set_node_attribute(my_node, 'Pathway', 'Signal Transduction /
Growth Regulation')
```

or

```
set_node_attribute(my_node, 'Mutation Frequency', 0.007,
type='double')
```

Parameters

- **node** (*int or node dict with @id attribute*) – Node to add the attribute to
- **attribute_name** (*string*) – attribute name
- **values** (*list, string, int or double*) – A value or list of values of the attribute
- **type** (*str*) – The datatype of the attribute values, defaults is string. See *Supported data types*
- **overwrite** (*bool True means to overwrite node attribute named attribute_name*) – If True node attribute matching 'attribute_name' is removed first otherwise code blindly adds attribute

Returns None

Return type None

Edge methods

class ndex2.nice_cx_network.**NiceCXNetwork**(**attr*)

create_edge(*edge_source=None*, *edge_target=None*, *edge_interaction=None*)

Create a new edge in the network by specifying source-interaction-target

> **Warning:** Version *3.3.1* and prior of this library had a bug that caused this method to behave incorrectly. Please upgrade to 3.3.2 or greater.

Example:

```
my_edge = create_edge(edge_source=my_node, edge_target=my_node2,
edge_interaction='up-regulates')
```

Parameters

- **edge_source** (int, dict (with *EDGE_ID* property)) – The source node of this edge, either its id or the node object itself.
- **edge_target** (int, dict (with *EDGE_ID* property)) – The target node of this edge, either its id or the node object itself.
- **edge_interaction** (*string*) – The interaction that describes the relationship between the source and target nodes

Returns Edge ID

Return type int

get_edge_attribute(*edge*, *attribute_name*)

Get the edge attributes of an edge, where the edge may be specified by its id or passed in as an object.

Example:

```
get_edge_attribute(my_edge, 'weight')

# returns:  {'@id':  0, 'n':  'weight', 'v':  0.849, 'd':
'double'}
```

Parameters

- **edge** (*int or edge dict with @id attribute*) – Edge object or edge id
- **attribute_name** – Attribute name

Returns Edge attribute object

Return type list, string, int or double

get_edge_attribute_value(*edge*, *attribute_name*)

Get the value(s) of an attribute of an edge, where the edge may be specified by its id or passed in as an object.

Example:

```
get_edge_attribute_value(my_edge, 'weight')

# returns:  0.849
```

Parameters

- **edge** (*int or edge dict with @id attribute*) – Edge object or edge id
- **attribute_name** – Attribute name

Returns Edge attribute value(s)

Return type list, string, int or double

get_edge_attributes(*edge*)

Get the attribute objects of an edge, where the edge may be specified by its id or passed in as an object.

Example:

```
get_edge_attributes(my_edge)

# returns:  [{'@id':  0, 'n':  'weight', 'v':  0.849, 'd':
'double'}, {'@id':  0, 'n':  'Type', 'v':  'E1'}]
```

Parameters **edge** (*int or edge dict with @id attribute*) – Edge object or edge id

Returns Edge attribute objects

Return type list of edge dict

get_edges()

Returns an iterator over edge ids as keys and edge objects as values.

Example:

```
for edge_id, edge_obj in nice_cx.get_edges():
print(edge_obj.get('i')) # print interaction
print(edge_obj.get('s')) # print source node id
```

Returns Edge iterator

Return type iterator

set_edge_attribute(*edge*, *attribute_name*, *values*, *type=None*)

Set the value(s) of attribute of an edge, where the edge may be specified by its id or passed in an object.

Example:

```
set_edge_attribute(0, 'weight', 0.5, type='double')
```

or

```
set_edge_attribute(my_edge, 'Disease', 'Atherosclerosis')
```

Parameters

- **edge** (*int or edge dict with @id attribute*) – Edge to add the attribute to
- **attribute_name** (*str*) – Attribute name
- **values** (*list*) – A value or list of values of the attribute
- **type** (*str*) – The datatype of the attribute values, defaults to the python datatype of the values. See *Supported data types*

Returns None

Return type None

Network methods

class ndex2.nice_cx_network.**NiceCXNetwork**(**attr*)

get_context()

Get the @context information of the network. This information maps namespace prefixes to their defining URIs

Example:

```
{'pmid':  'https://www.ncbi.nlm.nih.gov/pubmed/'}
```

Returns context object

Return type dict

get_name()
Get the network name

Returns Network name

Return type string

get_network_attribute(*attribute_name*)
Get the value of a network attribute

Parameters **attribute_name** (*string*) – Attribute name

Returns Network attribute object

Return type dict

get_network_attribute_names()
Creates a a generator that gets network attribute names.

Returns attribute name via a generator

Return type string

get_opaque_aspect(*aspect_name*)
Get the elements of the aspect specified by aspect_name

Parameters **aspect_name** (*string*) – the name of the aspect to retrieve.

Returns Opaque aspect

Return type list of aspect elements

set_context(*context*)
Set the @context information of the network. This information maps namespace prefixes to their defining URIs

Example:

```
from ndex2.nice_cx_network import NiceCXNetwork

net = NiceCXNetwork()
net.set_context({'pmid': 'https://www.ncbi.nlm.nih.gov/pubmed/'})
```

Parameters **context** (*dict or list*) – dict where key is name and value is URI or list of those dict objects

Raises ***NDExError*** – If **context** is not of type `list` or `dict`

Returns None

Return type none

set_name(*network_name*)
Set the network name

Example:

```
set_name('P38 Signaling')
```

Parameters **network_name** (*string*) – Network name

Returns None

Return type None

set_network_attribute(*name*, *values=None*, *type=None*)
Set an attribute of the network

```
from ndex2.nice_cx_network import NiceCXNetwork

net = NiceCXNetwork()
net.set_network_attribute(name='networkType', values='Genetic interactions')
```

Parameters

- **name** (*str*) – Attribute name
- **values** (*list, str, float, or int*) – The values of the attribute
- **type** (*str*) – The datatype of the attribute values. See *Supported data types*

Returns None

Return type none

set_opaque_aspect(*aspect_name*, *aspect_elements*)
Set the aspect specified by aspect_name to the list of aspect elements. If aspect_elements is *None*, the aspect is removed.

Changed in version 3.5.0: Fixed bug where passing *None* in *aspect_elements* did **NOT** remove aspect. Code also now raises *NDExError* if input values are invalid

```
from ndex2.nice_cx_network import NiceCXNetwork
net = NiceCXNetwork()

# to set an opaque aspect
net.set_opaque_aspect('foo', [{'data': 'val'}])

# to remove an opaque aspect named 'foo'
net.set_opaque_aspect('foo', None)
```

Parameters

- **aspect_name** (*str*) – Name of the aspect
- **aspect_elements** (*list of dict or dict*) – Aspect element

Raises ***NDExError*** – If *aspect_name* is *None*, or if *aspect_elements* is not *None*, dict, or list

Returns None

Return type none

Miscellaneous methods

class ndex2.nice_cx_network.**NiceCXNetwork**(**attr*)

apply_style_from_network(*nicecxnetwork*)
Applies Cytoscape visual properties from the network passed into this method. The style is pulled from VISUAL_PROPERTIES or CY_VISUAL_PROPERTIES

Parameters **nicecxnetwork** (*NiceCXNetwork*) – Network to extract style from

Raises

- **TypeError** – If object passed in is NOT a *NiceCXNetwork* object or if object is None
- ***NDExError*** – If *NiceCXNetwork* does not have any visual styles

Returns None

Return type None

apply_template(*server*, *uuid*, *username=None*, *password=None*)

Applies the Cytoscape visual properties of a network from the provided uuid to this network.

This allows the use of networks formatted in Cytoscape as templates to apply visual styles to other networks.

Changed in version 3.5.0: Fixed bug where style from template was appended instead of replacing the existing style. In most cases, method now raises *NDExError* and subclasses instead of more generic Exception

```
from ndex2.nice_cx_network import NiceCXNetwork

nice_cx = NiceCXNetwork()
nice_cx.apply_template('public.ndexbio.org',
                       '51247435-1e5f-11e8-b939-0ac135e8bacf')
```

Parameters

- **server** (*str*) – server host name (i.e. public.ndexbio.org)
- **username** (*str*) – username (optional - used when accessing private networks)
- **password** (*str*) – password (optional - used when accessing private networks)
- **uuid** (*str*) – uuid of the styled network

Raises

- ***NDExError*** – Raised if *server* or *uuid* not set or if metaData is not found in the network specified by *uuid* or some other server error
- ***NDExUnauthorizedError*** – If credentials not authorized to access network specified by *uuid*
- ***NDExNotFoundError*** – If network with *uuid* not found

Returns None

Return type None

print_summary()

Print a network summary

Returns Network summary

Return type string

to_cx(*log_to_stdout=True*)

Return the CX corresponding to the network.

Changed in version 3.5.0: Added **log_to_stdout** param which lets caller silence print statement *Generating CX*

Parameters **log_to_stdout** (*bool*) – If `True` then code will output to standard out *Generating CX*

Returns CX representation of the network

Return type CX (list of dict aspects)

to_cx_stream()
Returns a stream of the CX corresponding to the network. Can be used to post to endpoints that can accept streaming inputs

Returns The CX stream representation of this network.

Return type io.BytesIO

to_networkx(*mode='legacy'*)
Returns a NetworkX `Graph()` object or one of its subclasses based on the network. The *mode* parameter dictates how the translation occurs.

This method currently supports the following mode values:

> **Warning:** **legacy** mode has known bugs when networkx 2.0+ or greater is installed.
>
> See the description on **legacy** mode below for more information.

Modes:

legacy:

If mode set to **legacy** then this method will behave as it has for all versions of NDEx2 Python Client 3.1.0 and earlier which varies depending on version of networkx installed as described here:

For networkx 2.0 and greater: (see *LegacyNetworkXVersionTwoPlusFactory*)

For older versions of networkx the following class is used with the *legacymode* parameter set to *True*: (see *DefaultNetworkXFactory*)

default:

If mode is **default** or None then this method uses *DefaultNetworkXFactory* regardless of networkx installed with *legacymode* set to *False*

Note: `default` mode is the preferred mode to use

Examples:

```
# returns networkx graph using improved converter
graph = nice_cx.to_networkx(mode='default')

# returns networkx graph using legacy implementation
graph = nice_cx.to_networkx(mode='legacy)
```

Parameters **mode** (*string*) – Since translation to networkx can be done in many ways this mode lets the caller dictate the method.

Raises ***NDExError*** – If *mode* is not None, 'legacy', or 'default'

Returns Networkx graph

Return type `networkx.Graph` or `networkx.MultiGraph`

to_pandas_dataframe(*dataconverter=<ndex2.util.PandasDataConverter object>*, *include_attributes=False*)
Network edges exported as a `pandas.DataFrame`

Changed in version 3.5.0: Added **include_attributes** and **dataconverter** parameters

The following columns will be added to the `pandas.DataFrame`:

- **source** - Name of edge source node
- **interaction** - Interaction between source and target node
- **target** - Name of edge target node

If **include_attributes** parameter is set to `True` then:

All edge attributes will be also added as separate columns with same name.

Attributes on **source** node will be added as a columns with `source_` prefixed to name.

Attributes on **target** node will be added as columns with `target_` prefixed to name.

Note: Values will converted based on CX data types. See *PandasDataConverter* for information on how conversion is performed

---

```
from ndex2.nice_cx_network import NiceCXNetwork

net = NiceCXNetwork()
node_one = net.create_node('node1')
node_two = net.create_node('node2')

net.set_node_attribute(node_one, 'weight', 0.5, type='double')
net.set_node_attribute(node_two, 'weight', 0.2, type='double')

edge_one = net.create_edge(edge_source=node_one, edge_target=node_two,
                           edge_interaction='binds')

net.set_edge_attribute(edge_one, 'edgelabel', 'an edge')
df = net.to_pandas_dataframe(include_attributes=True) # df is now a pandas␣
↪dataframe

print(df.head())
```

Output from above code block:

```
   source interaction target edgelabel  target_weight  source_weight
0  node1       binds  node2   an edge            0.2            0.5
```

Note: This method only processes nodes, edges, node attributes and edge attributes, but not network attributes or other aspects

---

Parameters

- **dataconverter** (*DataConverter*) – Object that converts CX data values to native data types. Default is *PandasDataConverter*
- **include_attributes** (*bool*) – If *True* then edge attributes are added to `pandas.DataFrame`, otherwise only **source**, **target**, and **interaction** are added

Raises ***NDExInvalidParameterError*** – If **include_attributes** is not `None` or a `bool`

Returns Edge table with attributes

Return type pandas.DataFrame

update_to(*uuid*, *server=None*, *username=None*, *password=None*, *user_agent=''*, *client=None*)
Replace the network on NDEx server with matching NDEx *uuid* with this network.

Changed in version 3.4.0: This method was switched to named arguments and the server and account credentials can be passed in one of two ways.

Option 1) Set **username** and **password** parameters.

Option 2) Set **client** parameter with valid *Ndex2* object

Note: If **client** parameter is set, **username**, **password**, and **server** parameters are ignored.

Example:

```
import ndex2
nice_cx = ndex2.nice_cx_network.NiceCXNetwork()
nice_cx.create_node('foo')

# using production NDEx server
nice_cx.update_to('2ec87c51-c349-11e8-90ac-525400c25d22',
                  username=user_var,
                  password=password_var)

# if one needs to use alternate NDEx server
nice_cx.update_to('2ec87c51-c349-11e8-90ac-525400c25d22',
                  server='public.ndexbio.org',
                  username=username_var,
                  password=password_var)

# Option 2, create Ndex2 client object
ndex_client = ndex2.client.Ndex2(username=username_var,
                                 password=password_var)

# using NDEx client object for connection
nice_cx.update_to('2ec87c51-c349-11e8-90ac-525400c25d22',
                  client=ndex_client)
```

Parameters

- **uuid** (*str*) – UUID of the network on NDEx.
- **server** (*str*) – The NDEx server to upload the network to. Leaving unset or *None* will use production.
- **username** (*str*) – The username of the account to store the network.
- **password** (*str*) – The password for the account.
- **user_agent** (*str*) – String to append to User-Agent field sent to NDEx REST service
- **client** (*Ndex2*) – NDEx2 object with valid credentials. If set **server**, **username**, and **password** parameters will be ignored.

Returns Empty string

Return type str

upload_to(*server=None*, *username=None*, *password=None*, *user_agent=''*, *client=None*)
Upload this network as a new network on NDEx server.

Changed in version 3.4.0: This method was switched to named arguments and the server and account credentials can be passed in one of two ways.

Option 1) Set **username** and **password** parameters.

Option 2) Set **client** parameter with valid *Ndex2* object

Note: If **client** parameter is set, **username**, **password**, and **server** parameters are ignored

---

Example:

```
import ndex2
nice_cx = ndex2.nice_cx_network.NiceCXNetwork()
nice_cx.create_node('foo')

# using production NDEx server
nice_cx.update_to(username=user_var,
                  password=password_var)

# if one needs to use alternate NDEx server
nice_cx.update_to(server='public.ndexbio.org',
                  username=username_var,
                  password=password_var)

# Option 2, create Ndex2 client object
ndex_client = ndex2.client.Ndex2(username=username_var,
                                 password=password_var)

# using NDEx client object for connection
nice_cx.update_to(client=ndex_client)
```

Parameters

- **server** (*str*) – The NDEx server to upload the network to. Leaving unset or *None* will use production
- **username** (*str*) – The username of the account to store the network.
- **password** (*str*) – The password for the account.
- **user_agent** (*str*) – String to append to User-Agent field sent to NDEx REST service
- **client** (*Ndex2*) – NDEx2 object with valid credentials. If set **server**, **username**, and **password** parameters will be ignored.

Returns The UUID of the network on NDEx.

Return type str

Supported data types

The following CX Data Types are supported in methods that accept **type**

Example:

```
import ndex2
net = ndex2.nice_cx_network.NiceCXNetwork()
node_id = net.create_node('hi')
net.set_node_attribute(node_id, 'somevalue', 0.5, type='double')
```

- string
- double
- boolean
- integer
- long
- list_of_string
- list_of_double
- list_of_boolean
- list_of_integer
- list_of_long

These constants are defined here: *VALID_ATTRIBUTE_DATATYPES*

5.3.4 Ndex2 REST client

The Ndex2 class provides methods to interface with the NDEx REST Server API The *Ndex2* object can be used to access an NDEx server either anonymously or using a specific user account. For each NDEx server and user account that you want to use in your script or application, you create an *Ndex2* instance.

Example creating anonymous connection:

```
import ndex2.client
anon_ndex=ndex2.client.Ndex2()
```

Example creating connection with username and password:

```
import ndex2.client
my_account="your account"
my_password="your password"
my_ndex=ndex2.client.Ndex2("http://public.ndexbio.org", my_account,␣
↪my_password)
```

class ndex2.client.**Ndex2**(*host=None*, *username=None*, *password=None*, *update_status=False*, *debug=False*, *user_agent=''*, *timeout=30*, *skip_version_check=False*)

A class to facilitate communication with an NDEx server.

If host is not provided it will default to the NDEx public server. UUID is required

Creates a connection to a particular NDEx server.

New in version 3.5.0: *skip_version_check* parameter added

Parameters

- **host** (*str*) – The URL of the server.
- **username** (*str*) – The username of the NDEx account to use. (Optional)
- **password** (*str*) – The account password. (Optional)
- **update_status** (*bool*) – If set to True tells constructor to query service for status
- **user_agent** (*str*) – String to append to User-Agent header sent with all requests to server

- **timeout** (*float or tuple(float, float)*) – The timeout in seconds value for requests to server. This value is passed to Request calls Click here for more information
- **skip_version_check** (*bool*) – If `True`, it is assumed NDEx server supports **v2** endpoints, otherwise NDEx server is queried to see if **v2** endpoints are supported

add_networks_to_networkset(*set_id*, *networks*)
Add networks to a network set. User must have visibility of all networks being added

Parameters

- **set_id** (*str*) – network set id
- **networks** (*list*) – networks (ids as str) that will be added to the set

Returns None

Return type None

create_networkset(*name*, *description*)
Creates a new network set

Parameters

- **name** (*str*) – Network set name
- **description** (*str*) – Network set description

Returns URI of the newly created network set

Return type str

delete_network(*network_id*, *retry=5*)
Deletes the specified network from the server

Parameters

- **network_id** (*str*) – Network id
- **retry** (*int*) – Number of times to retry if deleting fails

Raises ***NDExUnauthorizedError*** – If credentials are invalid or not set

Returns Error json if there is an error. Blank

Return type str

delete_networks_from_networkset(*set_id*, *networks*, *retry=5*)
Removes network(s) from a network set.

Parameters

- **set_id** (*str*) – network set id
- **networks** (*list*) – networks (ids as str) that will be removed from the set
- **retry** (*int*) – Number of times to retry

Returns None

Return type None

delete_networkset(*networkset_id*)
Deletes the network set, requires credentials

Parameters **networkset_id** (*str*) – networkset UUID id

Raises

- ***NDExInvalidParameterError*** – for invalid networkset id parameter
- ***NDExUnauthorizedError*** – If no credentials or user is not authorized
- ***NDExNotFoundError*** – If no networkset with id passed in found
- ***NDExError*** – For any other error with contents of error in message

Returns None upon success

get_id_for_user(*username*)

Gets NDEx user Id for user

New in version 3.4.0.

```
import ndex2.client
my_ndex = ndex2.client.Ndex2()
my_ndex.get_id_for_user('nci-pid')
```

Parameters **username** (*str*) – Name of user on NDEx. If `None` user set in constructor of this client will be used.

Raises

- ***NDExError*** – If there was an error on the server.
- ***NDExInvalidParameterError*** – If username is empty string or is of type other then str.

Returns Id of user on NDEx server.

Return type str

get_neighborhood(*network_id*, *search_string*, *search_depth=1*, *edge_limit=2500*)

Get the CX for a subnetwork of the network specified by UUID network_id and a traversal of search_depth steps around the nodes found by search_string.

Parameters

- **network_id** (*str*) – The UUID of the network.
- **search_string** (*str*) – The search string used to identify the network neighborhood.
- **search_depth** (*int*) – The depth of the neighborhood from the core nodes identified.
- **edge_limit** (*int*) – The maximum size of the neighborhood.

Returns The CX json object.

Return type response object

get_neighborhood_as_cx_stream(*network_id*, *search_string*, *search_depth=1*, *edge_limit=2500*, *error_when_limit=True*)

Get a CX stream for a subnetwork of the network specified by UUID network_id and a traversal of search_depth steps around the nodes found by search_string.

Parameters

- **network_id** (*str*) – The UUID of the network.
- **search_string** (*str*) – The search string used to identify the network neighborhood.
- **search_depth** (*int*) – The depth of the neighborhood from the core nodes identified.
- **edge_limit** (*int*) – The maximum size of the neighborhood.

- **error_when_limit** (*bool*) – Default value is true. If this value is true the server will stop streaming the network when it hits the edgeLimit, add success: false and error: "EdgeLimitExceeded" in the status aspect and close the CX stream. If this value is set to false the server will return a subnetwork with edge count up to edgeLimit. The status aspect will be a success, and a network attribute {"EdgeLimitExceeded": "true"} will be added to the returned network only if the server hits the edgeLimit..

Returns The response.

Return type

response object

get_network_as_cx2_stream(*network_id*, *access_key=None*)

Get the existing network with UUID network_id from the NDEx connection as CX2 stream contained within a `requests.Response` object

New in version 3.5.0.

Example usage:

```
from ndex2.client import Ndex2
client = Ndex2(skip_version_check=True)

# 7fc.. is UUID MuSIC v1 network: http://doi.org/10.1038/s41586-021-04115-9
client_resp = client.get_network_as_cx2_stream('7fc70ab6-9fb1-11ea-aaef-
↪0ac135e8bacf')

# for HTTP status code, 200 means success
print(client_resp.status_code)

# for smaller networks one can get the CX2 by calling:
print(client_resp.json())
```

Note: For retrieving larger networks see `requests.Response.iter_content()`

This method sets *stream=True* in the request to avoid loading response into memory.

Parameters

- **network_id** – The UUID of the network
- **access_key** – Optional access key UUID

Raises ***NDExError*** – If there was an error

Returns Requests library response with CX2 in content and status code of 200 upon success

Return type `requests.Response`

get_network_as_cx_stream(*network_id*)

Get the existing network with UUID network_id from the NDEx connection as a CX stream.

Parameters **network_id** (*str*) – The UUID of the network.

Returns The response.

Return type

response object

get_network_aspect_as_cx_stream(*network_id*, *aspect_name*)
Get the specified aspect of the existing network with UUID network_id from the NDEx connection as a CX stream.

For a list of aspect names look at **Core Aspects** section of CX Data Model Documentation

Parameters

- **network_id** (*str*) – The UUID of the network.
- **aspect_name** – The aspect NAME.

Returns The response.

Return type

response object

get_network_ids_for_user(*username*, *offset=0*, *limit=1000*)
Get the network UUIDs owned by the user as well as any networks shared with the user. As set via **limit** parameter only the first `1,000` ids are returned. The **offset** parameter combined with **limit** provides pagination support.

Changed in version 3.4.0: **offset** and **limit** parameters added.

Parameters

- **username** (*str*) – NDEx username
- **offset** (*int*) – Starting position of the query. If set, **limit** parameter must be set to a positive value.
- **limit** (*int*) – Number of summaries to return starting from **offset** If set to `None` or `0` all summaries will be returned.

Raises ***NDExInvalidParameterError*** – If **offset/limit** parameters are not of type int. If **offset** parameter is set to positive number and **limit** is `0` or negative.

Returns List of uuids as str

Return type list

get_network_set(*set_id*)
Gets the network set information including the list of networks

Deprecated since version 3.2.0: Use *get_networkset()* instead.

Parameters **set_id** (*str*) – network set id

Returns network set information

Return type dict

get_network_summary(*network_id*)
Gets information and status of a network

Example usage:

```
from ndex2.client import Ndex2
client = Ndex2(skip_version_check=True)

# 7fc.. is UUID MuSIC v1 network: http://doi.org/10.1038/s41586-021-04115-9
net_sum = client.get_network_summary('7fc70ab6-9fb1-11ea-aaef-0ac135e8bacf')

print(net_sum)
```

Example result:


```
{
  "ownerUUID": "daa09f36-8cdd-11e7-a10d-0ac135e8bacf",
  "isReadOnly": true,
  "subnetworkIds": [],
  "isValid": true,
  "warnings": [],
  "isShowcase": true,
  "doi": "10.18119/N9188W",
  "isCertified": true,
  "indexLevel": "ALL",
  "hasLayout": true,
  "hasSample": false,
  "cxFileSize": 82656,
  "cx2FileSize": 68979,
  "visibility": "PUBLIC",
  "nodeCount": 70,
  "edgeCount": 87,
  "completed": true,
  "version": "1.0",
  "owner": "yue",
  "description": "<div><br/></div><div>Two central approaches for mapping␣
↪cellular structure - protein fluorescent imaging and protein biophysical␣
↪association - each generate extensive datasets but of distinct qualities␣
↪and resolutions that are typically treated separately. The MuSIC map is␣
↪designed to address this challenge, by integrating immunofluorescent images␣
↪in the Human Protein Atlas with ongoing affinity purification experiments␣
↪from the BioPlex resource. The result is a unified hierarchical map of␣
↪eukaryotic cell architecture. In the MuSIC hierarchy, nodes represent␣
↪systems and arrows indicate containment of the lower system by the upper.␣
↪Node color indicates known (gold) or putative novel (purple) systems. The␣
↪size of each circle is based on the number of proteins in the system. The␣
↪relative height of each system in the layout is determined based on the␣
↪predicted diameter of the system in MuSIC.<br/></div>",
  "name": "Multi-Scale Integrated Cell (MuSIC) v1",
  "properties": [
    {
      "subNetworkId": null,
      "predicateString": "author",
      "dataType": "string",
      "value": "Yue Qin"
    },
    {
      "subNetworkId": null,
      "predicateString": "rights",
      "dataType": "string",
      "value": "MIT license (MIT)"
    },
    {
      "subNetworkId": null,
      "predicateString": "rightsHolder",
      "dataType": "string",
      "value": "Yue Qin"
    },
    {
      "subNetworkId": null,
      "predicateString": "reference",
```


(continues on next page)

(continued from previous page)


```
      "dataType": "string",
      "value": "Yue Qin, Edward L. Huttlin, Casper F. Winsnes, Maya L.␣
↪Gosztyla, Ludivine Wacheul, Marcus R. Kelly, Steven M. Blue, Fan Zheng,␣
↪Michael Chen, Leah V. Schaffer, Katherine Licon, Anna Bäckström, Laura␣
↪Pontano Vaites, John J. Lee, Wei Ouyang, Sophie N. Liu, Tian Zhang, Erica␣
↪Silva, Jisoo Park, Adriana Pitea, Jason F. Kreisberg, Steven P. Gygi,␣
↪Jianzhu Ma, J. Wade Harper, Gene W. Yeo, Denis L. J. Lafontaine, Emma␣
↪Lundberg, Trey Ideker<br><strong>A multi-scale map of cell structure fusing␣
↪protein images and interactions</strong><br><i>Nature 600, 536-542 (2021).</
↪i>, (2021)<br><a href="http://doi.org/10.1038/s41586-021-04115-9"  target="_
↪blank">10.1038/s41586-021-04115-9</a>"
    }
  ],
  "externalId": "7fc70ab6-9fb1-11ea-aaef-0ac135e8bacf",
  "isDeleted": false,
  "modificationTime": 1630270298717,
  "creationTime": 1590539529001
}
```


Note: **isvalid** is a boolean to denote that the network was inspected, not that it is actually valid.

errorMessage Will be in result if there was an error parsing network

completed is set to `True` after all server tasks have completed and network is ready to be used

Parameters **network_id** (*str*) – The UUID of the network

Returns Summary information about network

Return type dict

get_networkset(*set_id*)

Gets the network set information including the list of networks

Parameters **set_id** (*str*) – network set id

Returns network set information

Return type dict

get_networksets_for_user_id(*user_id*, *summary_only=True*, *showcase=False*, *offset=0*, *limit=0*)

Gets a list of Network Set objects owned by the user identified by **user_id**

New in version 3.4.0.

Example when **summary_only** is `True` or if Network Set does not contain any networks:


```
[
 {'name': 'test networkset',
  'description': ' ',
  'ownerId': '4f0a6356-ed4a-49df-bd81-098fee90b448',
  'showcased': False,
  'properties': {},
  'externalId': '956e31e8-f25c-471f-8596-2cae8348dcad',
  'isDeleted': False,
  'modificationTime': 1568844043868,
  'creationTime': 1568844043868
```


(continues on next page)

(continued from previous page)

```
  }
]
```

When **summary_only** is `False` and Network Set does contain networks there will be an additional property named `networks`:

```
'networks': ['face63b6-aba7-11eb-9e72-0ac135e8bacf',
             'fae4d1e8-aba7-11eb-9e72-0ac135e8bacf']
```

Parameters

- **user_id** (*str*) – Id of user on NDEx. To get Id of user see *get_id_for_user()*
- **summary_only** (*bool*) – When `True`, the server will not return the list of network IDs in this Network Set
- **showcase** (*bool*) – When `True`, only showcased Network Sets are returned
- **offset** (*int*) – Index to first object to return. If `0`/`None` no offset will be applied. If this parameter is set to a positive value then **limit** parameter must be set to a positive value or this offset will be ignored.
- **limit** (*int*) – Number of objects to retrieve. If `0`, `None`, or negative all results will be returned.

Raises

- ***NDExInvalidParameterError*** – If **user_id** parameter is not of type str. If **offset/limit** parameters are not `None` or of type int. If **offset** parameter is set to positive number and **limit** is `0`, `None`, or negative.
- ***NDExError*** – If there is an error from server

Returns list with dict objects containing Network Sets

Return type list

get_sample_network(*network_id*)

Gets the sample network

Parameters **network_id** (*str*) – Network id

Raises ***NDExUnauthorizedError*** – If credentials are invalid or not set

Returns Sample network in CX format

Return type list

get_task_by_id(*task_id*)

Retrieves a task by id

Parameters **task_id** (*str*) – Task id

Raises ***NDExUnauthorizedError*** – If credentials are invalid or not set

Returns Task

Return type dict

get_user_by_id(*user_id*)

Gets user matching id from NDEx server.

New in version 3.4.0.

Result is a dict in format:

```
{'properties': {},
 'isIndividual': True,
 'userName': 'bsmith',
 'isVerified': True,
 'firstName': 'bob',
 'lastName': 'smith',
 'emailAddress': 'bob.smith@ndexbio.org',
 'diskQuota': 10000000000,
 'diskUsed': 3971183103,
 'externalId': 'f2c3a7ef-b0d9-4c61-bf31-4c9fcabe4173',
 'isDeleted': False,
 'modificationTime': 1554410147104,
 'creationTime': 1554410138498
}
```

Parameters **user_id** (*str*) – Id of user on NDEx server

Raises

- ***NDExError*** – If there was an error on the server
- ***NDExInvalidParameterError*** – If user_id is not of type str or if empty str

Returns user object. *externalId* is Id of user on NDEx server

Return type dict

get_user_by_username(*username*)

Gets user information from NDEx.

Example user information:

```
{'properties': {},
 'isIndividual': True,
 'userName': 'bsmith',
 'isVerified': True,
 'firstName': 'bob',
 'lastName': 'smith',
 'emailAddress': 'bob.smith@ndexbio.org',
 'diskQuota': 10000000000,
 'diskUsed': 3971183103,
 'externalId': 'f2c3a7ef-b0d9-4c61-bf31-4c9fcabe4173',
 'isDeleted': False,
 'modificationTime': 1554410147104,
 'creationTime': 1554410138498
}
```

Parameters **username** (*str*) – User name

Returns User information as dict

Return type dict

get_user_network_summaries(*username*, *offset=0*, *limit=1000*)

Get a list of network summaries for networks owned by specified user. It returns not only the networks that the user owns but also the networks that are shared with them directly. As set via **limit** parameter only the first 1,000 ids are returned. The **offset** parameter combined with **limit** parameter provides pagination support.

Parameters

- **username** (*str*) – Username of the network owner
- **offset** (*int*) – Starting position of the network search
- **limit** (*int*) – Number of summaries to return starting from *offset*

Returns List of uuids

Return type list

grant_network_to_user_by_username(*username*, *network_id*, *permission*)
Grants permission to network for the given user name

Parameters

- **username** (*str*) – User name
- **network_id** (*str*) – Network id
- **permission** (*str*) – Network permission

Returns Result

Return type dict

grant_networks_to_group(*groupid*, *networkids*, *permission='READ'*)
Set group permission for a set of networks

Parameters

- **groupid** (*str*) – Group id
- **networkids** (*list*) – List of network ids
- **permission** (*str*) – Network permission

Returns Result

Return type dict

grant_networks_to_user(*userid*, *networkids*, *permission='READ'*)
Gives read permission to specified networks for the provided user

Parameters

- **userid** (*str*) – User id
- **networkids** (*list*) – Network ids as str
- **permission** (*str (default is READ)*) – Network permissions

Returns None

Return type None

make_network_private(*network_id*)
Makes the network specified by the **network_id** private by invoking *set_network_system_properties()* with

```
{'visibility': 'PRIVATE'}
```

Parameters **network_id** (*str*) – The UUID of the network.

Raises

- ***NDExUnauthorizedError*** – If credentials are invalid or not set
- **requests.exception.HTTPError** – If there is some other error

Returns empty string upon success

Return type str

make_network_public(*network_id*)

Makes the network specified by the **network_id** public by invoking *set_network_system_properties()* with

```
{'visibility': 'PUBLIC'}
```

Parameters **network_id** (*str*) – The UUID of the network.

Raises

- ***NDExUnauthorizedError*** – If credentials are invalid or not set
- **requests.exception.HTTPError** – If there is some other error

Returns empty string upon success

Return type str

save_cx2_stream_as_new_network(*cx_stream*, *visibility=None*)

Create a new network from a CX2 stream

New in version 3.5.0.

```
import io
import json
from ndex2.client import Ndex2
from ndex2.exceptions import NDExError

client = Ndex2(username=<NDEx USER NAME>,
               password=<NDEx PASSWORD>,
               skip_version_check=True)

# cx is set to an empty CX2 network
cx = [{"CXVersion":"2.0","hasFragments":false},
      {"status":[{"success":true}]}]

try:
    cx_stream = io.BytesIO(json.dumps(cx,
                                      cls=DecimalEncoder).encode('utf-8'))
    net_url = client.save_cx2_stream_as_new_network(cx_stream,
                                                    visibility='PUBLIC')
    print('Network URL: ' + str(net_url))
except NDExError as ne:
    print('Caught error: ' + str(ne))
```

Parameters

- **cx_stream** (*BytesIO like object*) – IO stream of cx2
- **visibility** (*str*) – Sets the visibility (PUBLIC or PRIVATE)

Raises

- ***NDExUnauthorizedError*** – If credentials are invalid or not set
- ***NDExError*** – if there is an error saving the network

Returns Full URL to newly created network (ie http://ndexbio.org/v3/networks/XXXX)

Return type str

save_cx_stream_as_new_network(*cx_stream*, *visibility=None*)
Create a new network from a CX stream.

Parameters

- **cx_stream** (*BytesIO*) – IO stream of cx
- **visibility** (*str*) – Sets the visibility (PUBLIC or PRIVATE)

Raises ***NDExUnauthorizedError*** – If credentials are invalid or not set

Returns Response data

Return type str or dict

save_new_cx2_network(*cx*, *visibility=None*)
Create a new network (CX2) on the server

New in version 3.5.0.

```
from ndex2.client import Ndex2
from ndex2.exceptions import NDExError

client = Ndex2(username=<NDEx USER NAME>,
               password=<NDEx PASSWORD>,
               skip_version_check=True)

# cx is set to an empty CX2 network
cx = [{"CXVersion":"2.0","hasFragments":false},
      {"status":[{"success":true}]}]

try:
    net_url = client.save_new_cx2_network(cx, visibility='PRIVATE')
    print('URL of new network: ' + str(net_url))
except NDExError as ne:
    print('Caught error: ' + str(ne))
```

Parameters

- **cx** (*list*) – Network CX2 which is a list of dict objects
- **visibility** (*str*) – Sets the visibility (PUBLIC or PRIVATE) If `None` sets visibility to PRIVATE

Raises

- ***NDExUnauthorizedError*** – If credentials are invalid or not set
- ***NDExInvalidCXError*** – if **cx** is `None`, not a list, or is an empty list
- ***NDExError*** – if there is an error saving the network

Returns Full URL to newly created network (ie http://ndexbio.org/v3/networks/XXXX)

Return type str

save_new_network(*cx*, *visibility=None*)
Create a new network (CX) on the server

Parameters

- **cx** (*list*) – Network CX which is a list of dict objects
- **visibility** (*str*) – Sets the visibility (PUBLIC or PRIVATE)

Raises ***NDExInvalidCXError*** – For invalid CX data

Returns Response data

Return type str or dict

search_networks(*search_string=''*, *account_name=None*, *start=0*, *size=100*, *include_groups=False*)

Search for networks based on the search_text, optionally limited to networks owned by the specified account_name.

Parameters

- **search_string** (*str*) – The text to search for.
- **account_name** (*str*) – The account to search
- **start** (*int*) – The number of blocks to skip. Usually zero, but may be used to page results.
- **size** (*int*) – The size of the block.
- **include_groups** –

Returns The response.

Return type

response object

set_network_properties(*network_id*, *network_properties*)

Updates properties of network

Starting with version 2.5 of NDEx, any network properties not in the *network_properties* parameter are left unchanged.

> **Warning:** `name, description, version` network attributes/properties cannot be updated by this method. Please use *update_network_profile()* to update these values.

The format of *network_properties* should be a `list()` of `dict()` objects in this format:

The `predicateString` field above is the network attribute/property name.

The `dataType` field above must be one of the following types

Regardless of `dataType`, `value` should be converted to `str()` or `list()` of `str()`

For more information please visit the underlying REST call documentation

Example to add two network properties (`foo`, `bar`):

Parameters

- **network_id** (*str*) – Network id
- **network_properties** (*list or str*) – List of NDEx property value pairs aka network properties to set on the network. This can also be a `str()` in JSON format

Raises

- **Exception** – If *network_properties* is not a `str()` or `list()`
- ***NDExUnauthorizedError*** – If credentials are invalid or not set
- **requests.HTTPError** – If there is an error with the request or if `name, version, description` is set in *network_properties* as a value to `predicateString`

Returns Empty string or `1`

Return type str or int

set_network_system_properties(*network_id*, *network_properties*, *skipvalidation=False*)
Set network system properties on network with UUID specified by **network_id**

The network properties should be a `dict()` or a json string of a `dict()` in this format:

```
{'showcase': (boolean True or False),
 'visibility': (str 'PUBLIC' or 'PRIVATE'),
 'index_level': (str  'NONE', 'META', or 'ALL'),
 'readOnly': (boolean True or False)
}
```

Note: Omit any values from `dict()` that you do NOT want changed

Definition of **showcase** values:

`True` - means network will display in her home page for other users and `False` hides the network for other users. where other users includes anonymous users

Definition of **visibility** values:

'PUBLIC' - means it can be found or read by anyone, including anonymous users

'PRIVATE' - is the default, means that it can only be found or read by users according to their permissions

Definition of **index_level** values:

'NONE' - no index

'META' - only index network attributes

'ALL' - full index on the network

Definition of **readOnly** values:

`True` - means network is only readonly, False is NOT readonly

This method will validate **network_properties** matches above `dict()` unless **skipvalidation** is set to `True` in which case the code only verifies the **network_properties** is valid JSON

Parameters

- **network_id** (*str*) – Network id
- **network_properties** (*dict or str*) – Network properties as `dict()` or a JSON string of `dict()` adhering to structure above.
- **skipvalidation** – If `True`, only verify **network_properties** can be parsed/converted to valid JSON

Raises

- ***NDExUnsupportedCallError*** – If version of NDEx server is < 2
- ***NDExUnauthorizedError*** – If credentials are invalid or not set
- ***NDExInvalidParameterError*** – If invalid data is set in **network_properties** parameter
- **requests.exception.HTTPError** – If there is some other error

Returns empty string upon success

Return type str

set_read_only(*network_id*, *value*)
Sets the read only flag to **value** on the network specified by **network_id**

Parameters

- **network_id** (*str*) – Network id
- **value** (*bool*) – Must `True` for read only, `False` otherwise

Raises

- ***NDExUnauthorizedError*** – If credentials are invalid or not set
- ***NDExInvalidParameterError*** – If non bool is set in **valid** parameter
- **requests.exception.HTTPError** – If there is some other error

Returns empty string upon success

Return type str

update_cx2_network(*cx_stream*, *network_id*)
Update the network specified by UUID network_id using the CX2 stream **cx_stream** passed in

New in version 3.5.0.

```
import io
import json
from ndex2.client import Ndex2
from ndex2.exceptions import NDExError

client = Ndex2(username=<NDEx USER NAME>,
               password=<NDEx PASSWORD>,
               skip_version_check=True)

# cx is set to an empty CX2 network
cx = [{"CXVersion":"2.0","hasFragments":false},
      {"status":[{"success":true}]}]

try:
    cx_stream = io.BytesIO(json.dumps(cx,
                                      cls=DecimalEncoder).encode('utf-8'))
    client.update_cx2_network(cx_stream, <UUID OF NETWORK TO UPDATE>)
    print('Success')
except NDExError as ne:
    print('Caught error: ' + str(ne))
```

Parameters

- **cx_stream** – The network stream.
- **network_id** (*str*) – The UUID of the network.

Raises

- ***NDExUnauthorizedError*** – If credentials are invalid or not set
- ***NDExError*** – If there is an error updating the network

Returns Nothing is returned. To check status call *get_network_summary()*

update_cx_network(*cx_stream*, *network_id*)
Update the network specified by UUID network_id using the CX stream **cx_stream** passed in

Parameters

- **cx_stream** – The network stream.
- **network_id** (*str*) – The UUID of the network.

Raises ***NDExUnauthorizedError*** – If credentials are invalid or not set

Returns The response.

Return type

response object

update_network_group_permission(*groupid*, *networkid*, *permission*)
Updated group permissions

Parameters

- **groupid** (*str*) – Group id
- **networkid** (*str*) – Network id
- **permission** (*str*) – Network permission

Returns Result

Return type dict

update_network_profile(*network_id*, *network_profile*)
Updates the network profile Any profile attributes specified will be updated but attributes that are not specified will have no effect - omission of an attribute does not mean deletion of that attribute. The network profile attributes that can be updated by this method are: 'name', 'description' and 'version'.

```
{
  "name": "string",
  "description": "string",
  "version": "string",
  "visibility": "string",
  "properties": [
    {
      "subNetworkId": "",
      "predicateString": "string",
      "dataType": "string",
      "value": "string"
    }
  ]
}
```

Parameters

- **network_id** (*str*) – Network id
- **network_profile** (*dict*) – Network profile

Raises ***NDExUnauthorizedError*** – If credentials are invalid or not set

Returns

Return type

update_network_user_permission(*userid*, *networkid*, *permission*)
Updated network user permission

Parameters

- **userid** (*str*) – User id
- **networkid** (*str*) – Network id
- **permission** (*str*) – Network permission

Returns Result

Return type dict

5.3.5 Constants

Contains constants used by the NDEx2 Python Client

ndex2.constants.**BOOLEAN_DATATYPE = 'boolean'**
Boolean data type for CX

ndex2.constants.**CARTESIAN_LAYOUT_ASPECT = 'cartesianLayout'**
Name of opaque aspect containing coordinates of nodes

```
"cartesianLayout": [
    {
        "node": 0,
        "x": 25.0,
        "y": 50.0
    }, {
        "node": 1,
        "x": -10.0,
        "y": -200.0
    }
]
```

Note: Although the name implies a cartesian coordinate system, that is actually wrong. The Y access is inverted so lower values of Y are rendered higher on a graph. 0,0 is considered upper left corner, but negative values are allowed

ndex2.constants.**DOUBLE_DATATYPE = 'double'**
Double data type for CX

ndex2.constants.**EDGE_ID = '@id'**
Key for id of edge

ndex2.constants.**EDGE_INTERACTION = 'i'**
Key for edge interaction

ndex2.constants.**EDGE_SOURCE = 's'**
Key for edge source

ndex2.constants.**EDGE_TARGET = 't'**
Key for edge target

ndex2.constants.**INTEGER_DATATYPE = 'integer'**
Integer data type for CX

ndex2.constants.**LAYOUT_NODE = 'node'**
Key for node id in *CARTESIAN_LAYOUT_ASPECT* opaque aspect

ndex2.constants.**LAYOUT_X = 'x'**
Key for X coordinate in *CARTESIAN_LAYOUT_ASPECT* opaque aspect

ndex2.constants.**LAYOUT_Y = 'y'**
Key for Y coordinate in *CARTESIAN_LAYOUT_ASPECT* opaque aspect

ndex2.constants.**LIST_OF_BOOLEAN = 'list_of_boolean'**
List of Boolean data type for CX

ndex2.constants.**LIST_OF_DOUBLE = 'list_of_double'**
List of Double data type for CX

ndex2.constants.**LIST_OF_INTEGER = 'list_of_integer'**
List of Integer data type for CX

ndex2.constants.**LIST_OF_LONG = 'list_of_long'**
List of Long data type for CX

ndex2.constants.**LIST_OF_STRING = 'list_of_string'**
List of String data type for CX

ndex2.constants.**LONG_DATATYPE = 'long'**
Long data type for CX

ndex2.constants.**NET_ATTR_NAME = 'n'**
Key for network attribute name

ndex2.constants.**NET_ATTR_VALUE = 'v'**
Key for network attribute value

ndex2.constants.**NODE_ATTR_DATATYPE = 'd'**
Key for node attribute data type

ndex2.constants.**NODE_ATTR_NAME = 'n'**
Key for node attribute name

ndex2.constants.**NODE_ATTR_PROPERTYOF = 'po'**
Key for node property of

ndex2.constants.**NODE_ATTR_VALUE = 'v'**
Key for node attribute value

ndex2.constants.**NODE_ID = '@id'**
Key for id of node

ndex2.constants.**NODE_NAME = 'n'**
Key for node name

ndex2.constants.**NODE_REPRESENTS = 'r'**
Key for node represents

ndex2.constants.**STRING_DATATYPE = 'string'**
String data type for CX

ndex2.constants.**VALID_ATTRIBUTE_DATATYPES = ['boolean', 'double', 'integer', 'long', 'strin**
List of valid attribute data types

5.3.6 Miscellaneous

class ndex2.util.**DataConverter**
Base class for subclasses that convert CX data types to/from native data types

convert_value(*value=None*, *datatype=None*)
Defines method to converts *value* from CX to native data type using *datatype* as a guide

Parameters

- **value** (*object*) – Value to convert
- **datatype** (*str*) – CX data type which is one of the following: *ndex2.constants.VALID_ATTRIBUTE_DATATYPES*

Raises **NotImplementedError** – Always raises this error cause subclasses should implement

Returns Always raises NotImplementedError

class ndex2.util.**PandasDataConverter**
Converts CX values to native Python data types via *PandasDataConverter.convert_value()* method

New in version 3.5.0.

convert_value(*value=None*, *datatype=None*)
Converts *value* parameter passed in to type based on value of *datatype* parameter passed in. This is used in data conversion by to_pandas_dataframe()

New in version 3.5.0.

Conversion rules for different values of *datatype* parameter:

- *STRING_DATATYPE* or None
  - *value* is converted by str and returned
- *BOOLEAN_DATATYPE*
  - If *value* is of type bool it is just returned. If *value* is of type str and equals *true* (ignore case) or *1* then True is returned and if value equals *false* (ignore case) or *0* is then False is returned. Otherwise value is converted by bool and returned. If conversion fails *NDExError* is raised
- *DOUBLE_DATATYPE*
  - *value* is converted by float and returned. If conversion fails *NDExError* is raised
- *INTEGER_DATATYPE* or *LONG_DATATYPE*
  - *value* is converted by int and returned. If conversion fails *NDExError* is raised
- *LIST_OF_STRING*, *LIST_OF_BOOLEAN* *LIST_OF_DOUBLE*, *LIST_OF_INTEGER*, *LIST_OF_LONG*
  - If *value* is **NOT** of type list then the *value* converted as if it's datatype is *STRING_DATATYPE*. If *value* is a list, each element is converted as if it's datatype is *STRING_DATATYPE* and values of list are converted to a comma delimited str

Example usage:

```
from ndex2.util import PandasDataConverter

converter = PandasDataConverter()

# converts number to type str
```

(continues on next page)

(continued from previous page)

```
res = converter.convert_value(123, 'string')

# would output <class 'str'>
print(type(res))
```

Parameters

- **value** (*object*) – Value to convert
- **datatype** (*str*) – CX data type which is one of the following: *ndex2.constants.VALID_ATTRIBUTE_DATATYPES*

Raises ***NDExError*** – If there is an error with conversion

Returns Converted value

Return type list, str, int, float, or bool

5.3.7 Exceptions

class ndex2.exceptions.**NDExError**
Base Exception for all NDEx2 Python Client Exceptions

> **Warning:** Many methods in this code base still incorrectly raise errors not derived from this base class

class ndex2.exceptions.**NDExNotFoundError**
Raised if resource requested was not found

class ndex2.exceptions.**NDExUnauthorizedError**
Raised if unable to authenticate, either due to lack of or invalid credentials.

class ndex2.exceptions.**NDExInvalidParameterError**
Raised if invalid parameter is passed in

class ndex2.exceptions.**NDExInvalidCXError**
Raised due to invalid CX

class ndex2.exceptions.**NDExUnsupportedCallError**
Raised if call is unsupported, for example a method that is only supported in 2.0+ of NDEx server is attempted against a server running 1.0

5.4 License

5.5 History

5.5.1 3.5.0 (2022-06-28)

- **Enhancements**
  - Added **skip_version_check** parameter to `Ndex2()` constructor to let caller optionally bypass NDEx server call to see if **v2** endpoint is supported
  - Added the following *CX2* methods to `Ndex2()` client: `get_network_as_cx2_stream()`, `get_network_aspect_as_cx2_stream()`, `save_cx2_stream_as_new_network()`, `save_new_cx2_network()`, and `update_cx2_network()` Issue #87
  - In `Ndex2()` client, methods that raise `NDExError` exceptions from calls to NDEx server will now raise the more specific `NDExUnauthorizedError` subclass when the response from NDEx server is a 401 aka unauthorized.
  - Added new parameters **dataconverter** and **include_attributes** to `NiceCXNetwork.to_pandas_dataframe()`. **dataconverter** parameter specifies data type conversion and **include_attributes** parameter lets caller specify whether all node/edge attributes are added to the resulting DataFrame
  - Added new parameter to `ndex2.create_nice_cx_from_server()` named **ndex_client** that lets caller specify `NDex2()` client object to use.
  - Passing `None` for the **server** positional parameter into `ndex2.create_nice_cx_from_server(None, uuid='XXXX')` will default to the production NDEx server
- **Bug fixes**
  - Fixed bug where creation of *NiceCXNetwork* from networkx via `ndex2.create_nice_cx_from_networkx()` incorrectly set the data type for boolean values to integer. Issue #83
  - Fixed bug where converting *NiceCXNetwork* to networkx and back does not handle name attribute correctly. Issue #84
  - Fixed bug where *@context* was lost if it was set as aspect in CX format and loaded into NiceCXNetwork object. Issue #88
  - Fixed bug where creation of *NiceCXNetwork* from networkx via `ndex2.create_nice_cx_from_networkx()` incorrectly set the data type for empty list to string. Issue #90
  - Fixed bug where Y coordinates of nodes would be inverted when converting to/from networkx from *NiceCXNetwork*. This was due to differences in coordinate systems between networkx and *NiceCXNetwork*

  - *DefaultNetworkXFactory* networkx converter (used by `NiceCXNetwork.to_networkx(mode='default')`) no longer converts edge attributes that are of type list into strings delimited by commas
  - Fixed bug where `ndex2.create_nice_cx_from_server()` failed on networks with *provenanceHistory* aspect

- **Removals**
  - Removed unused test methods from internal class *NiceCXBuilder*: `load_aspect()`, `stream_aspect()`, `stream_aspect_raw()`
  - Removed the following deprecated methods from *NiceCXNetwork*: `add_node()`, `add_edge()`, `get_edge_attribute_object()`, `get_node_attribute_objects()`, `get_edge_attribute_objects()`, `add_metadata()`, `get_provenance()`, `set_provenance()`, `__merge_node_attributes()`, `create_from_pandas()`, `create_from_networkx()`, `create_from_server()`, `upload_new_network_stream()`, & `create_from_cx()`

5.5.2 3.4.0 (2021-05-06)

- Added **offset** and **limit** parameters to *Ndex2.get_network_ids_for_user()* to enable retrieval of all networks for a user. Issue #78
- Switched *NiceCXNetwork.upload_to()* to named arguments and added **client** parameter. Issue #80
- Switched *NiceCXNetwork.update_to()* to named arguments and added **client** parameter. Issue #81
- Fixed documentation *NiceCXNetwork.update_to()* to correctly state method returns empty string upon success. Issue #82
- Fixed bug in *NiceCXNetwork.set_opaque_aspect()* where passing *None* in the **aspect_elements** parameter raised an error instead of removing the aspect
- Added *Ndex2.get_user_by_id()* method to get user information by NDEx user Id.
- Added *Ndex2.get_id_for_user()* method to get NDEx user Id by username.
- Added *Ndex2.get_networksets_for_user_id()* to get Network Sets for a given user Id. Issue #61
- Improved documentation by adding intersphinx to provide links to python documentation for python objects.

5.5.3 3.3.3 (2021-04-22)

- Fixed bug where *NiceCXNetwork.to_networkx()* fails with *ValueError* when installed networkx version has X.Y.Z format (example: 2.5.1) Issue #79

5.5.4 3.3.2 (2021-04-13)

- Fixed bug where *NiceCXNetwork.create_node()* and *.create_edge()* overwrote existing nodes/edges. Issue #60
- Fixed bug where *enum34* package would be unnecessarily installed on versions of Python 3.4 and newer. Issue #76
- Improved documentation for *Ndex2.set_network_properties()* method. Issue #77

5.5.5 3.3.1 (2019-09-23)

- Added *MANIFEST.in* file to include *README.rst, HISTORY.rst, and LICENSE.txt* files as well as documentation and tests so *python setup.py install* will work properly on distribution of this client on PyPI. Thanks to Ben G. for catching this. Issue #62
- Minor updates to *README.rst*

5.5.6 3.3.0 (2019-09-11)

- Fixed bug where if server version is not 2.0 exactly then *Ndex2()* object incorrectly falls back to version of 1.3 of REST calls Issue #40
- Fixed bug in *NiceCXNetwork.add_network_attribute()* method where type not properly reset when adding duplicate attribute Issue #50
- Added *delete_networksets()* method to Ndex2 client to allow deletion of networksets Issue #59

5.5.7 3.2.0 (2019-04-23)

- Verify consistent conversion of CX for networkx 1.11 and 2.0+ Issue #30
- *NiceCXNetwork.get_nodes()*, *NiceCXNetwork.get_edges()*, *NiceCXNetwork.get_metadata()* needs to make correct iterator call in Python 2 Issue #44
- Add *NiceCXNetwork.get_network_attribute_names()* function enhancement Issue #45
- *NiceCXNetwork.create_edge()* fails to correctly create edge when node dict passed in Issue #46

5.5.8 3.1.0a1 (2019-03-20)

- Add method to ndex2 python client to apply style from one NiceCXNetwork to another NiceCXNetwork Issue #43

5.5.9 3.0.0a1 (2019-02-11)

- In NiceCXNetwork class ability to add to User-Agent for calls to NDEx service Issue #36
- Methods in *ndex2/client.py* should raise an NDExError for invalid credentials Issue #39
- Add timeout flag to all web request calls Issue #33
- Update *User-Agent* to reflect actual version of software Issue #35
- *NiceCXNetwork.set_node_attribute()* incorrectly handles duplicate attributes Issue #41
- *NiceCXNetwork.set_node_attribute()* fails if node object passed to it Issue #42
- Passing None to user_agent parameterin *Ndex2()* constructor raises TypeError Issue #34
- *Ndex2()* constructor does not properly handle invalid json from server Issue #28
- Eliminate circular import between ndex2 and ndex2cx/nice_cx_builder.py Issue #31
- Replace print statements with logging calls in *ndex2/client.py* Issue #32

5.5.10 2.0.1 (2019-01-03)

- Fixed bug where logs directory is created within the package installation directory. Issue #26

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

n

Index

I

L

M

N

P

S

U

V