

---

# **index2 Documentation**

***Release 3.3.3***

**Dexter Pratt, Aaron Gary & Jing Chen**

**Apr 22, 2021**



---

## Contents:

---

<b>1</b>	<b>Overview</b>	<b>3</b>
<b>2</b>	<b>Dependencies</b>	<b>5</b>
<b>3</b>	<b>Compatibility</b>	<b>7</b>
<b>4</b>	<b>Installation</b>	<b>9</b>
<b>5</b>	<b>License</b>	<b>11</b>
5.1	Installation . . . . .	11
5.2	Quick Tutorial . . . . .	12
5.3	Reference . . . . .	12
5.4	License . . . . .	38
5.5	History . . . . .	39
<b>6</b>	<b>Indices and tables</b>	<b>41</b>
	<b>Python Module Index</b>	<b>43</b>
	<b>Index</b>	<b>45</b>







# 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.





---

### Dependencies

---

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



## CHAPTER 3

---

### Compatibility

---

Python 2.7+

---

**Note:** Python 2.7 may have some issues, Python 3.6+ is preferred

---



## 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
```



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

### 5.2.1 Download network from NDEx

The code blocks 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
# http://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 edges: ' + 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 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_networkx(G)`

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

The resulting *NiceCXNetwork* contains the nodes, edges and their attributes from the *networkx* graph and also preserves the graph ‘pos’ attribute as a CX cartesian coordinates aspect.

The node name is taken from the *networkx* node id. Node ‘represents’ is taken from the *networkx* node attribute ‘represents’

**Parameters** *G* (*networkx graph*) – *networkx* graph

**Returns** *NiceCXNetwork*

**Return type** *NiceCXNetwork*

`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.

If only the *df* argument is provided the 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, the those and any other arguments refer to headers in the 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 it defaults to “interacts-with”

**Parameters**

- **df** – pandas dataframe to process
- **source\_field** – header name specifying the name of the source node.
- **target\_field** – header name specifying the name of the target node.
- **source\_node\_attr** – list of header names specifying attributes of the source node.
- **target\_node\_attr** – list of header names specifying attributes of the target node.
- **edge\_attr** – list of header names specifying attributes of the edge.
- **edge\_interaction** – the relationship between the source node and the target node, defaulting to “interacts-with”

**Returns** `NiceCXNetwork`

**Return type** `NiceCXNetwork()`

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

Create a `NiceCXNetwork()` based on a network retrieved from NDEx, specified by its UUID. If the network is not public, then username and password arguments for an account on the server with permission to access the network must be supplied.

**Parameters**

- **server** – the URL of the NDEx server hosting the network.
- **username** – the user name of an account with permission to access the network.
- **password** – the password of an account with permission to access the network.
- **uuid** – the UUID of the network.

**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

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()`

**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 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*

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

### 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 *upload\_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)

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 attribute 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 attribute 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

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 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:

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

**Parameters** **context** (*dict*) – dict of name, URI pairs

**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

Example:

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

**Parameters**

- **name** (*string*) – Attribute name
- **values** (*list, string, double 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.

**Parameters**

- **aspect\_name** (*string*) – Name of the aspect
- **aspect\_elements** (*list of dict*) – Aspect element

**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.

Example:

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

**Parameters**

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

**Returns** None

**Return type** None

**print\_summary** ()

Print a network summary

**Returns** Network summary

**Return type** string

**to\_cx** ()

Return the CX corresponding to the network.

**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:** For backwards compatibility *mode* is set to **legacy** but there are known bugs in this implementation 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()

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

**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()**

Export the network as a Pandas DataFrame.

Example:

```
df = nice_cx.to_pandas_dataframe() # df is now a pandas
dataframe
```

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

**Returns** Pandas dataframe

**Return type** Pandas dataframe

**update\_to(uuid, server, username, password, user\_agent=“”)**

Upload this network to the specified server to the account specified by username and password.

Example:

```
nice_cx.update_to('2ec87c51-c349-11e8-90ac-525400c25d22', 'public.ndexbio.org
↪', username, password)
```

#### Parameters

- **server** (*str*) – The NDEx server to upload the network to.
- **username** (*str*) – The username of the account to store the network.
- **password** (*str*) – The password for the account.
- **user\_agent** (*string*) – String to append to User-Agent field sent to NDEx REST service

**Returns** The UUID of the network on NDEx.

**Return type** str

**upload\_to** (*server, username, password, user\_agent=""*)

**Upload this network to the specified server to the account specified** by username and password.

Example:

```
nice_cx.upload_to('http://public.ndexbio.org', username,
password)
```

#### Parameters

- **server** (*string*) – The NDEx server to upload the network to.
- **username** (*string*) – The username of the account to store the network.
- **password** (*string*) – The password for the account.
- **user\_agent** (*string*) – String to append to User-Agent field sent to NDEx REST service

**Returns** The UUID of the network on NDEx.

**Return type** string

## Supported data types

The following data types are supported in methods that accept **type**

Example:

```
set_edge_attribute(0, 'weight', 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*)

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.

#### Parameters

- **host** (*string*) – The URL of the server.
- **username** (*string*) – The username of the [NDEx](#) account to use. (Optional)
- **password** (*string*) – The account password. (Optional)
- **update\_status** (*bool*) – If set to True tells constructor to query service for status
- **user\_agent** (*string*) – 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](#)

**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** (*basestring*) – network set id
- **networks** (*list of strings*) – networks that will be added to the set

**Returns** None

**Return type** None

**create\_networkset** (*name, description*)

Creates a new network set



**Parameters**

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

**Returns** URI of the newly created network set

**Return type** string

**delete\_network** (*network\_id*, *retry=5*)

Deletes the specified network from the server

**Parameters**

- **network\_id** (*string*) – 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** string

**delete\_networks\_from\_networkset** (*set\_id*, *networks*, *retry=5*)

Removes network(s) from a network set.

**Parameters**

- **set\_id** (*basestring*) – network set id
- **networks** (*list of strings*) – networks 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\_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** (*boolean*) – 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\_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*)

Get the network uuids owned by the user

**Parameters** **username** (*str*) – users NDEx username

**Returns** list of uuids

**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` (*basestring*) – network set id

**Returns** network set information

**Return type** dict

**get\_network\_summary** (*network\_id*)

Gets information about a network.

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

**Returns** Summary

**Return type** dict

**get\_networkset** (*set\_id*)

Gets the network set information including the list of networks

**Parameters** `set_id` (*basestring*) – network set id

**Returns** network set information

**Return type** dict

**get\_sample\_network** (*network\_id*)

Gets the sample network

**Parameters** `network_id` (*string*) – Network id

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

**Returns** Sample network

**Return type** list of dicts in cx format

**get\_task\_by\_id** (*task\_id*)

Retrieves a task by id

**Parameters** `task_id` (*string*) – Task id

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

**Returns** Task

**Return type** dict

**get\_user\_by\_username** (*username*)

Gets user information as a dict in format:

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

(continues on next page)

(continued from previous page)

```
'creationTime': 1554410138498
}
```

**Parameters** `username` (*string*) – 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.

**Parameters**

- **username** (*str*) – the username of the network owner
- **offset** (*int*) – the starting position of the network search
- **limit** –

**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** (*string*) – User name
- **network\_id** (*string*) – Network id
- **permission** (*string*) – 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** (*string*) – Group id
- **networkids** (*list*) – List of network ids
- **permission** (*string*) – 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** (*string*) – User id
- **networkids** (*list of strings*) – Network ids
- **permission** (*string (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\_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** (*string*) – Sets the visibility (PUBLIC or PRIVATE)

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

**Returns** Response data

**Return type** string or dict

**save\_new\_network** (*cx*, *visibility=None*)

Create a new network (cx) on the server

**Parameters**

- **cx** (*list of dicts*) – Network cx
- **visibility** (*string*) – Sets the visibility (PUBLIC or PRIVATE)

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

**Returns** Response data

**Return type** string 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\_cx\_network** (*cx\_stream*, *network\_id*)

Update the network specified by UUID *network\_id* using the CX stream *cx\_stream*.

**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** (*string*) – Group id
- **networkid** (*string*) – Network id
- **permission** (*string*) – 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": "",
```

(continues on next page)



(continued from previous page)

```

        "predicateString": "string",
        "dataType": "string",
        "value": "string"
    }
]
}

```

**Parameters**

- **network\_id**(*string*) – 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**(*string*) – User id
- **networkid**(*string*) – Network id
- **permission**(*string*) – Network permission

Returns Result

Return type dict

### 5.3.5 Constants

Contains constants used by the NDEx2 Python Client

`ndex2.constants.CARTESIAN_LAYOUT_ASPECT = 'cartesianLayout'`

Name of opaque aspect containing coordinates of nodes

`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.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.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.VALID_ATTRIBUTE_DATATYPES = ['boolean', 'double', 'integer', 'long', 'string']
    List of valid attribute data types

```

### 5.3.6 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

Copyright © 2013-2019, The Regents of the University of California, The Cytoscape Consortium. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the

following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL <COPYRIGHT HOLDER> BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## 5.5 History

### 5.5.1 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](#)

### 5.5.2 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](#)

### 5.5.3 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*

### 5.5.4 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](#)

### 5.5.5 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](#)

### 5.5.6 3.1.0a1 (2019-03-20)

- Add method to ndex2 python client to apply style from one `NiceCXNetwork` to another `NiceCXNetwork` [Issue #43](#)

### 5.5.7 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` parameter in `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](#)

### 5.5.8 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`



### n

`ndex2`, [13](#)

`ndex2.constants`, [37](#)





## A

add\_edge() (*ndex2.nice\_cx\_network.NetworkXFactory method*), 17

add\_network\_attributes\_from\_nice\_cx\_network() (*ndex2.nice\_cx\_network.NetworkXFactory method*), 17

add\_networks\_to\_networkset() (*ndex2.client.Ndex2 method*), 28

add\_node() (*ndex2.nice\_cx\_network.NetworkXFactory method*), 17

apply\_style\_from\_network() (*ndex2.nice\_cx\_network.NiceCXNetwork method*), 24

apply\_template() (*ndex2.nice\_cx\_network.NiceCXNetwork method*), 25

## C

CARTESIAN\_LAYOUT\_ASPECT (in module *ndex2.constants*), 37

copy\_cartesian\_coords\_into\_graph() (*ndex2.nice\_cx\_network.NetworkXFactory method*), 18

create\_edge() (*ndex2.nice\_cx\_network.NiceCXNetwork method*), 21

create\_networkset() (*ndex2.client.Ndex2 method*), 28

create\_nice\_cx\_from\_file() (in module *ndex2*), 13

create\_nice\_cx\_from\_networkx() (in module *ndex2*), 13

create\_nice\_cx\_from\_pandas() (in module *ndex2*), 13

create\_nice\_cx\_from\_raw\_cx() (in module *ndex2*), 13

create\_nice\_cx\_from\_server() (in module *ndex2*), 14

create\_node() (*ndex2.nice\_cx\_network.NiceCXNetwork method*), 19

## D

DefaultNetworkXFactory (class in *ndex2.nice\_cx\_network*), 14

delete\_network() (*ndex2.client.Ndex2 method*), 29

delete\_networks\_from\_networkset() (*ndex2.client.Ndex2 method*), 29

delete\_networkset() (*ndex2.client.Ndex2 method*), 29

## E

EDGE\_ID (in module *ndex2.constants*), 37

EDGE\_INTERACTION (in module *ndex2.constants*), 37

EDGE\_SOURCE (in module *ndex2.constants*), 37

EDGE\_TARGET (in module *ndex2.constants*), 37

## G

get\_context() (*ndex2.nice\_cx\_network.NiceCXNetwork method*), 23

get\_edge\_attribute() (*ndex2.nice\_cx\_network.NiceCXNetwork method*), 21

get\_edge\_attribute\_value() (*ndex2.nice\_cx\_network.NiceCXNetwork method*), 21

get\_edge\_attributes() (*ndex2.nice\_cx\_network.NiceCXNetwork method*), 22

get\_edges() (*ndex2.nice\_cx\_network.NiceCXNetwork method*), 22

get\_graph() (*ndex2.nice\_cx\_network.DefaultNetworkXFactory method*), 15

get\_graph() (*ndex2.nice\_cx\_network.LegacyNetworkXVersionTwoPlus method*), 16

get\_name() (*ndex2.nice\_cx\_network.NiceCXNetwork method*), 23

get\_neighborhood() (*ndex2.client.Ndex2 method*), 29

get\_neighborhood\_as\_cx\_stream() (*ndex2.client.Ndex2 method*), 30

[get\\_network\\_as\\_cx\\_stream\(\)](#)  
     ([ndex2.client.Ndex2 method](#)), 30  
[get\\_network\\_aspect\\_as\\_cx\\_stream\(\)](#)  
     ([ndex2.client.Ndex2 method](#)), 30  
[get\\_network\\_attribute\(\)](#)  
     ([ndex2.nice\\_cx\\_network.NiceCXNetwork](#)  
     [method](#)), 23  
[get\\_network\\_attribute\\_names\(\)](#)  
     ([ndex2.nice\\_cx\\_network.NiceCXNetwork](#)  
     [method](#)), 23  
[get\\_network\\_ids\\_for\\_user\(\)](#)  
     ([ndex2.client.Ndex2 method](#)), 30  
[get\\_network\\_set\(\)](#) ([ndex2.client.Ndex2 method](#)),  
     30  
[get\\_network\\_summary\(\)](#) ([ndex2.client.Ndex2](#)  
     [method](#)), 31  
[get\\_networkset\(\)](#) ([ndex2.client.Ndex2 method](#)), 31  
[get\\_node\\_attribute\(\)](#)  
     ([ndex2.nice\\_cx\\_network.NiceCXNetwork](#)  
     [method](#)), 19  
[get\\_node\\_attribute\\_value\(\)](#)  
     ([ndex2.nice\\_cx\\_network.NiceCXNetwork](#)  
     [method](#)), 19  
[get\\_node\\_attributes\(\)](#)  
     ([ndex2.nice\\_cx\\_network.NiceCXNetwork](#)  
     [method](#)), 19  
[get\\_nodes\(\)](#) ([ndex2.nice\\_cx\\_network.NiceCXNetwork](#)  
     [method](#)), 20  
[get\\_opaque\\_aspect\(\)](#)  
     ([ndex2.nice\\_cx\\_network.NiceCXNetwork](#)  
     [method](#)), 23  
[get\\_sample\\_network\(\)](#) ([ndex2.client.Ndex2](#)  
     [method](#)), 31  
[get\\_task\\_by\\_id\(\)](#) ([ndex2.client.Ndex2 method](#)), 31  
[get\\_user\\_by\\_username\(\)](#) ([ndex2.client.Ndex2](#)  
     [method](#)), 31  
[get\\_user\\_network\\_summaries\(\)](#)  
     ([ndex2.client.Ndex2 method](#)), 32  
[grant\\_network\\_to\\_user\\_by\\_username\(\)](#)  
     ([ndex2.client.Ndex2 method](#)), 32  
[grant\\_networks\\_to\\_group\(\)](#)  
     ([ndex2.client.Ndex2 method](#)), 32  
[grant\\_networks\\_to\\_user\(\)](#) ([ndex2.client.Ndex2](#)  
     [method](#)), 32

## L

[LAYOUT\\_NODE](#) (in module [ndex2.constants](#)), 37  
[LAYOUT\\_X](#) (in module [ndex2.constants](#)), 37  
[LAYOUT\\_Y](#) (in module [ndex2.constants](#)), 37  
[LegacyNetworkXVersionTwoPlusFactory](#)  
     (class in [ndex2.nice\\_cx\\_network](#)), 16

## M

[make\\_network\\_private\(\)](#) ([ndex2.client.Ndex2](#)

[method](#)), 33  
[make\\_network\\_public\(\)](#) ([ndex2.client.Ndex2](#)  
[method](#)), 33

## N

[Ndex2](#) (class in [ndex2.client](#)), 28  
[ndex2](#) (module), 13  
[ndex2.constants](#) (module), 37  
[NDExError](#) (class in [ndex2.exceptions](#)), 38  
[NDExInvalidCXError](#) (class in [ndex2.exceptions](#)),  
     38  
[NDExInvalidParameterError](#) (class in  
     [ndex2.exceptions](#)), 38  
[NDExNotFoundError](#) (class in [ndex2.exceptions](#)), 38  
[NDExUnauthorizedError](#) (class in  
     [ndex2.exceptions](#)), 38  
[NDExUnsupportedCallError](#) (class in  
     [ndex2.exceptions](#)), 38  
[NET\\_ATTR\\_NAME](#) (in module [ndex2.constants](#)), 37  
[NET\\_ATTR\\_VALUE](#) (in module [ndex2.constants](#)), 38  
[NetworkXFactory](#) (class in [ndex2.nice\\_cx\\_network](#)),  
     17  
[NiceCXNetwork](#) (class in [ndex2.nice\\_cx\\_network](#)), 19,  
     21, 23, 24  
[NODE\\_ATTR\\_DATATYPE](#) (in module [ndex2.constants](#)),  
     38  
[NODE\\_ATTR\\_NAME](#) (in module [ndex2.constants](#)), 38  
[NODE\\_ATTR\\_PROPERTYOF](#) (in module  
     [ndex2.constants](#)), 38  
[NODE\\_ATTR\\_VALUE](#) (in module [ndex2.constants](#)), 38  
[NODE\\_ID](#) (in module [ndex2.constants](#)), 38  
[NODE\\_NAME](#) (in module [ndex2.constants](#)), 38  
[NODE\\_REPRESENTS](#) (in module [ndex2.constants](#)), 38

## P

[print\\_summary\(\)](#) ([ndex2.nice\\_cx\\_network.NiceCXNetwork](#)  
[method](#)), 25

## S

[save\\_cx\\_stream\\_as\\_new\\_network\(\)](#)  
     ([ndex2.client.Ndex2 method](#)), 33  
[save\\_new\\_network\(\)](#) ([ndex2.client.Ndex2 method](#)),  
     33  
[search\\_networks\(\)](#) ([ndex2.client.Ndex2 method](#)),  
     33  
[set\\_context\(\)](#) ([ndex2.nice\\_cx\\_network.NiceCXNetwork](#)  
     [method](#)), 23  
[set\\_edge\\_attribute\(\)](#)  
     ([ndex2.nice\\_cx\\_network.NiceCXNetwork](#)  
     [method](#)), 22  
[set\\_name\(\)](#) ([ndex2.nice\\_cx\\_network.NiceCXNetwork](#)  
     [method](#)), 23

```

set_network_attribute()
    (ndex2.nice_cx_network.NiceCXNetwork
     method), 24
set_network_properties() (ndex2.client.Ndex2
    method), 34
set_network_system_properties()
    (ndex2.client.Ndex2 method), 34
set_node_attribute()
    (ndex2.nice_cx_network.NiceCXNetwork
     method), 20
set_opaque_aspect()
    (ndex2.nice_cx_network.NiceCXNetwork
     method), 24
set_read_only() (ndex2.client.Ndex2 method), 35

```

## T

```

to_cx()      (ndex2.nice_cx_network.NiceCXNetwork
    method), 25
to_cx_stream() (ndex2.nice_cx_network.NiceCXNetwork
    method), 25
to_networkx() (ndex2.nice_cx_network.NiceCXNetwork
    method), 25
to_pandas_dataframe()
    (ndex2.nice_cx_network.NiceCXNetwork
     method), 26

```

## U

```

update_cx_network()      (ndex2.client.Ndex2
    method), 36
update_network_group_permission()
    (ndex2.client.Ndex2 method), 36
update_network_profile() (ndex2.client.Ndex2
    method), 36
update_network_user_permission()
    (ndex2.client.Ndex2 method), 37
update_to() (ndex2.nice_cx_network.NiceCXNetwork
    method), 26
upload_to() (ndex2.nice_cx_network.NiceCXNetwork
    method), 27

```

## V

```

VALID_ATTRIBUTE_DATATYPES    (in module
    ndex2.constants), 38

```