ndex2 Documentation

Release 3.2.0a3

Dexter Pratt, Aaron Gary & Jing Chen

Apr 23, 2019

Contents:

1 Reference **1**

1.1 NiceCXNetwork 1

1.2 Client access to NDEx server API 15

1.3 Constants 22

1.4 Exceptions 23

2 Indices and tables **25**

Python Module Index **27**

CHAPTER 1

Reference

NDEx2 Python Client documentation

1.1 NiceCXNetwork

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

Click here for more information about 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.

To see deprecated methods go to *Deprecated NiceCXNetwork methods*

1.1.1 Methods for building niceCX

see also this 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

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

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*)

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

1.1.2 Methods for accessing niceCX properties

see also this notebook

Node methods

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

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

Edge methods

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

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

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

1.1.3 Misc niceCX methods

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

apply_template(*server*, *uuid*, *username=None*, *password=None*)
Applies the Cytoscape visual properties of a network from the provideduuid 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: It is **highly** recommended that this mode is used

Examples:

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

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

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

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

1.1.4 Conversion of niceCX to other formats

There are several classes described below 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

Deprecated

These networkx converters are still callable, but have 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`

1.1.5 Deprecated NiceCXNetwork methods

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

add_edge(*id=None*, *source=None*, *target=None*, *interaction=None*)

> **Warning:** This method has been deprecated. Please use **create_edge()**

add_node(*id=None*, *name=None*, *represents=None*)

> **Warning:** This method has been deprecated. Please use **create_node()**

get_edge_attribute_objects(*edge*, *attribute_name*)

> **Warning:** This method has been deprecated. Please use **get_edge_attributes()**

get_node_attribute_objects(*node*, *attribute_name*)

> **Warning:** This method has been deprecated. Please use **get_node_attribute()**

get_summary()

> **Warning:** This method has been deprecated. Please use **print_summary()**

set_provenance(*provenance*)

> **Warning:** This method has been deprecated.

1.1.6 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*

1.1.7 Methods for creating niceCX from other data models

ndex2.**create_nice_cx_from_raw_cx**(*cx*)
Create a NiceCXNetwork from a CX json object. (see http://www.home.ndexbio.org/data-model)

Parameters **cx** – a valid CX document

Returns NiceCXNetwork

ndex2.**create_nice_cx_from_file**(*path*)
Create a NiceCXNetwork from a file that is in the CX format.

Parameters **path** – the path of the CX file

Returns 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. 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

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

1.2 Client access to NDEx server API

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("http://public.ndexbio.org")
```

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

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_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

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_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 the user id by user name

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

Returns User id

Return type string

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.

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

Returns The response.

Return type

response object

make_network_public(*network_id*)
Makes the network specified by the network_id public.

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

Returns The response.

Return type

response object

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*)
Sets network properties

Parameters

- **network_id** (*string*) – Network id
- **network_properties** (*list*) – List of NDEx property value pairs

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

Returns

Return type

set_network_system_properties(*network_id*, *network_properties*)
Set network system properties

Parameters

- **network_id** (*string*) – Network id
- **network_properties** (*dict of NDEx network property value pairs*) – Network properties

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

Returns Result

Return type dict

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

Parameters

- **network_id** (*string*) – Network id
- **value** (*bool*) – Read only value

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

Returns Result

Return type dict

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'.

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

1.3 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', 'stri**
List of valid attribute data types

1.4 Exceptions

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

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

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

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

n

ndex2, 14
ndex2.constants, 22

Index

A

add_edge() (*ndex2.nice_cx_network.NiceCXNetwork method*), 13
add_networks_to_networkset() (*ndex2.client.Ndex2 method*), 16
add_node() (*ndex2.nice_cx_network.NiceCXNetwork method*), 13
apply_template() (*ndex2.nice_cx_network.NiceCXNetwork method*), 7

C

CARTESIAN_LAYOUT_ASPECT (*in module ndex2.constants*), 22
create_edge() (*ndex2.nice_cx_network.NiceCXNetwork method*), 2
create_networkset() (*ndex2.client.Ndex2 method*), 16
create_nice_cx_from_file() (*in module ndex2*), 14
create_nice_cx_from_networkx() (*in module ndex2*), 14
create_nice_cx_from_pandas() (*in module ndex2*), 14
create_nice_cx_from_raw_cx() (*in module ndex2*), 14
create_nice_cx_from_server() (*in module ndex2*), 15
create_node() (*ndex2.nice_cx_network.NiceCXNetwork method*), 1

D

DefaultNetworkXFactory (*class in ndex2.nice_cx_network*), 10
delete_network() (*ndex2.client.Ndex2 method*), 16
delete_networks_from_networkset() (*ndex2.client.Ndex2 method*), 16

E

EDGE_ID (*in module ndex2.constants*), 22
EDGE_INTERACTION (*in module ndex2.constants*), 22
EDGE_SOURCE (*in module ndex2.constants*), 22
EDGE_TARGET (*in module ndex2.constants*), 22

G

get_context() (*ndex2.nice_cx_network.NiceCXNetwork method*), 7
get_edge_attribute() (*ndex2.nice_cx_network.NiceCXNetwork method*), 5
get_edge_attribute_objects() (*ndex2.nice_cx_network.NiceCXNetwork method*), 13
get_edge_attribute_value() (*ndex2.nice_cx_network.NiceCXNetwork method*), 6
get_edge_attributes() (*ndex2.nice_cx_network.NiceCXNetwork method*), 6
get_edges() (*ndex2.nice_cx_network.NiceCXNetwork method*), 6
get_graph() (*ndex2.nice_cx_network.DefaultNetworkXFactory method*), 10
get_graph() (*ndex2.nice_cx_network.LegacyNetworkXVersionTwoPlus method*), 12
get_name() (*ndex2.nice_cx_network.NiceCXNetwork method*), 7
get_neighborhood() (*ndex2.client.Ndex2 method*), 17
get_neighborhood_as_cx_stream() (*ndex2.client.Ndex2 method*), 17
get_network_as_cx_stream() (*ndex2.client.Ndex2 method*), 17
get_network_attribute() (*ndex2.nice_cx_network.NiceCXNetwork method*), 7
get_network_attribute_names() (*ndex2.nice_cx_network.NiceCXNetwork method*), 7

get_network_ids_for_user() (*ndex2.client.Ndex2 method*), 17
get_network_set() (*ndex2.client.Ndex2 method*), 17
get_network_summary() (*ndex2.client.Ndex2 method*), 18
get_node_attribute() (*ndex2.nice_cx_network.NiceCXNetwork method*), 4
get_node_attribute_objects() (*ndex2.nice_cx_network.NiceCXNetwork method*), 13
get_node_attribute_value() (*ndex2.nice_cx_network.NiceCXNetwork method*), 5
get_node_attributes() (*ndex2.nice_cx_network.NiceCXNetwork method*), 5
get_nodes() (*ndex2.nice_cx_network.NiceCXNetwork method*), 5
get_opaque_aspect() (*ndex2.nice_cx_network.NiceCXNetwork method*), 7
get_sample_network() (*ndex2.client.Ndex2 method*), 18
get_summary() (*ndex2.nice_cx_network.NiceCXNetwork method*), 13
get_task_by_id() (*ndex2.client.Ndex2 method*), 18
get_user_by_username() (*ndex2.client.Ndex2 method*), 18
get_user_network_summaries() (*ndex2.client.Ndex2 method*), 18
grant_network_to_user_by_username() (*ndex2.client.Ndex2 method*), 18
grant_networks_to_group() (*ndex2.client.Ndex2 method*), 19
grant_networks_to_user() (*ndex2.client.Ndex2 method*), 19

L

LAYOUT_NODE (*in module ndex2.constants*), 22
LAYOUT_X (*in module ndex2.constants*), 22
LAYOUT_Y (*in module ndex2.constants*), 22
LegacyNetworkXVersionTwoPlusFactory (*class in ndex2.nice_cx_network*), 11

M

make_network_private() (*ndex2.client.Ndex2 method*), 19
make_network_public() (*ndex2.client.Ndex2 method*), 19

N

Ndex2 (*class in ndex2.client*), 15
ndex2 (*module*), 14
ndex2.constants (*module*), 22
NDExError (*class in ndex2.exceptions*), 23
NDExInvalidCXError (*class in ndex2.exceptions*), 23
NDExUnauthorizedError (*class in ndex2.exceptions*), 23
NET_ATTR_NAME (*in module ndex2.constants*), 22
NET_ATTR_VALUE (*in module ndex2.constants*), 22
NiceCXNetwork (*class in ndex2.nice_cx_network*), 1–5, 7, 13
NODE_ATTR_DATATYPE (*in module ndex2.constants*), 22
NODE_ATTR_NAME (*in module ndex2.constants*), 22
NODE_ATTR_PROPERTYOF (*in module ndex2.constants*), 22
NODE_ATTR_VALUE (*in module ndex2.constants*), 22
NODE_ID (*in module ndex2.constants*), 22
NODE_NAME (*in module ndex2.constants*), 22
NODE_REPRESENTS (*in module ndex2.constants*), 23

P

print_summary() (*ndex2.nice_cx_network.NiceCXNetwork method*), 8

S

save_cx_stream_as_new_network() (*ndex2.client.Ndex2 method*), 19
save_new_network() (*ndex2.client.Ndex2 method*), 20
search_networks() (*ndex2.client.Ndex2 method*), 20
set_context() (*ndex2.nice_cx_network.NiceCXNetwork method*), 3
set_edge_attribute() (*ndex2.nice_cx_network.NiceCXNetwork method*), 3
set_name() (*ndex2.nice_cx_network.NiceCXNetwork method*), 3
set_network_attribute() (*ndex2.nice_cx_network.NiceCXNetwork method*), 4
set_network_properties() (*ndex2.client.Ndex2 method*), 20
set_network_system_properties() (*ndex2.client.Ndex2 method*), 20
set_node_attribute() (*ndex2.nice_cx_network.NiceCXNetwork method*), 2
set_opaque_aspect() (*ndex2.nice_cx_network.NiceCXNetwork method*), 4
set_provenance() (*ndex2.nice_cx_network.NiceCXNetwork method*), 13

set_read_only() (*ndex2.client.Ndex2 method*), 21

T

to_cx() (*ndex2.nice_cx_network.NiceCXNetwork method*), 8
to_cx_stream() (*ndex2.nice_cx_network.NiceCXNetwork method*), 8
to_networkx() (*ndex2.nice_cx_network.NiceCXNetwork method*), 8
to_pandas_dataframe() (*ndex2.nice_cx_network.NiceCXNetwork method*), 9

U

update_cx_network() (*ndex2.client.Ndex2 method*), 21
update_network_group_permission() (*ndex2.client.Ndex2 method*), 21
update_network_profile() (*ndex2.client.Ndex2 method*), 21
update_network_user_permission() (*ndex2.client.Ndex2 method*), 21
update_to() (*ndex2.nice_cx_network.NiceCXNetwork method*), 9
upload_to() (*ndex2.nice_cx_network.NiceCXNetwork method*), 10

V

VALID_ATTRIBUTE_DATATYPES (*in module ndex2.constants*), 23