ndex2 Documentation

Release 3.3.1

Dexter Pratt, Aaron Gary & Jing Chen

Sep 23, 2019

Contents:

CHAPTER 1

NDEx2 Python Client

1.1 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.

1.2 Jupyter Notebook Tutorials

- 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

1.3 Requirements

The NDEx2 Python Client module works best with Python 3.6+ and the latest version of the PIP Python package manager for installation. Click here to download the PIP Python package.

1.4 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
```

1.5 License

See LICENSE.txt

1.6 NDEx2 Client Objects

The NDEx2 Client provides an interface to an NDEx server that is managed via a client object class. An NDEx2 Client 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 Client instance. In this example, a client object is created to access the public NDEx server.

```
import ndex2.client
anon_ndex=ndex2.client.Ndex2("http://public.ndexbio.org")
```

A client object using a specific user account can perform operations requiring authentication, such as saving networks to that account.

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

1.6.1 NDEx Client Object Methods:

Status

update_status()

- Updates the client object *status* attribute with the status of the NDEx Server.

Users

get_user_by_username(username)

- Returns a user object corresponding to the provided username
- Error if this account is not found
- If the user account has not been verified by the user yet, the returned object will contain no user UUID and the *isVerified* field will be false.

Network

save_new_network(cx)

- Creates a new network from cx, a python dict in CX format.

save_cx_stream_as_new_network(cx_stream)

- Creates a network from the byte stream cx_stream.

update_cx_network(cx_stream, network_id)

- Updates network specified by network_id with the new content from the byte stream cx_stream.
- Errors if the network_id does not correspond to an existing network on the NDEx Server which the authenticated user either owns or has WRITE permission.
- Errors if the cx_stream data is larger than the maximum size allowed by the NDEx server.

delete_network(network_id)

- Deletes the network specified by network_id.
- There is no method to undo a deletion, so care should be exercised.
- The specified network must be owned by the authenticated user.

get_network_summary(network_id)

- Retrieves a NetworkSummary JSON object from the network specified by network_id and returns it as a Python dict.
- A NetworkSummary object provides useful information about the network, a mixture of network profile information (properties expressed in special aspects of the network CX), network properties (properties expressed in the networkAttributes aspect of the network CX) and network system properties (properties expressing how the network is stored on the server, not part of the network CX).

Attribute

Description

Type

creationTme

Time at which the network was created

timeStamp

description

Text description of the network, same meaning as dc:description

string

edgeCount

The number of edge objects in the network

integer

errorMessage

If this network is not a valid CX network, this field holds the error message produced by the CX network validator.

string

externalId

UUID of the network

string

isDeleted

True if the network is marked as deleted

boolean

isReadOnly

True if the network is marked as readonly

boolean

isShowCase

True if the network is showcased

boolean

isValid

True if the network is a valid CX network

boolean

modificationTime

Time at which the network was last modified

timeStamp

name

Name or title of the network, not unique, same meaning as dc:title

string

nodeCount

The number of node objects in the network

integer

owner

The userName of the network owner

string

ownerUUID

The UUID of the networks owner

string

properties

List of NDExPropertyValuePair objects: describes properties of the networ

list

subnetworkIds

List of integers which are identifiers of subnetworks

list

uri

URI of the current network

string

version

Format is not controlled but best practice is to use a string conforming to Semantic Versioning

string

visibility

PUBLIC or PRIVATE. 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

string

warnings

List of warning messages produced by the CX network validator

list

- The **properties** attribute in the above table represents a list of attributes where each attribute is a dictionary with the following fields:

Property Object Field

Description

Type

dataType

Type of the attribute

string

predicateString

Name of the attribute.

string

value

Value of the attribute

string

subNetworkId

Subnetwork Id of the attribute

string

- Errors if the network is not found or if the authenticated user does not have READ permission for the network.

- Anonymous users can only access networks with visibility = PUBLIC.

get_network_as_cx_stream(network_id)

- Returns the network specified by network_id as a CX byte stream.
- This is performed as a monolithic operation, so it is typically advisable for applications to first use the getNetworkSummary method to check the node and edge counts for a network before retrieving the network.

set_network_system_properties(network_id, network_system_properties)

- Sets the system properties specified in network_system_properties data for the network specified by network_id.
- Network System properties describe the network's status on the NDEx server but are not part of the corresponding CX network object.
- As of NDEx V2.0 the supported system properties are:
  - readOnly: boolean
  - visibility: PUBLIC or PRIVATE.
  - showcase: boolean. Controls whether the network will display on the homepage of the authenticated user. Returns an error if the user does not have explicit permission to the network.
  - network_system_properties format: {property: value, . . . }, such as:
    - {"readOnly": True}
    - {"visibility": "PUBLIC"}
    - {"showcase": True}
    - {"readOnly": True, "visibility": "PRIVATE", "showcase": False}.

make_network_private(network_id)

- Sets visibility of the network specified by network_id to private.
- This is a shortcut for setting the visibility of the network to PRIVATE with the set_network_system_properties method:
  - set_network_system_properties(network_id, {"visibility": "PRIVATE"}).

make_network_public(network_id)

- Sets visibility of the network specified by network_id to public
- This is a shortcut for setting the visibility of the network to PUBLIC with the set_network_system_properties method:
  - set_network_system_properties(network_id, {"visibility": "PUBLIC"}).

set_read_only(network_id, value)

- Sets the read-only flag of the network specified by network_id to value.
- The type of value is boolean (True or False).
- This is a shortcut for setting readOnly for the network by the set_network_system_properties method:
  - set_network_system_properties(network_id, {"readOnly": True})
  - set_network_system_properties(network_id, {"readOnly": False}).

update_network_group_permission(group_id, network_id, permission)

- Updates the permission of a group specified by group_id for the network specified by network_id.
- The permission is updated to the value specified in the permission parameter, either READ, WRITE, or ADMIN.
- Errors if the authenticated user making the request does not have WRITE or ADMIN permissions to the specified network.
- Errors if network_id does not correspond to an existing network.
- Errors if the operation would leave the network without any user having ADMIN permissions: NDEx does not permit networks to become 'orphans' without any owner.

grant_networks_to_group(group_id, network_ids, permission="READ")

- Updates the permission of a group specified by group_id for all the networks specified in network_ids list
- For each network, the permission is updated to the value specified in the permission parameter. permission parameter is READ, WRITE, or ADMIN; default value is READ.
- Errors if the authenticated user making the request does not have WRITE or ADMIN permissions to each network.
- Errors if any of the network_ids does not correspond to an existing network.
- Errors if it would leave any network without any user having ADMIN permissions: NDEx does not permit networks to become 'orphans' without any owner.

update_network_user_permission(user_id, network_id, permission)

- Updates the permission of the user specified by user_id for the network specified by network_id.
- The permission is updated to the value specified in the permission parameter. permission parameter is READ, WRITE, or ADMIN.
- Errors if the authenticated user making the request does not have WRITE or ADMIN permissions to the specified network.
- Errors if network_id does not correspond to an existing network.
- Errors if it would leave the network without any user having ADMIN permissions: NDEx does not permit networks to become 'orphans' without any owner.

grant_network_to_user_by_username(username, network_id, permission)

- Updates the permission of a user specified by username for the network specified by network_id.
- This method is equivalent to getting the user_id via get_user_by_name(username), and then calling update_network_user_permission with that user_id.

grant_networks_to_user(user_id, network_ids, permission="READ")

- Updates the permission of a user specified by user_id for all the networks specified in network_ids list.

update_network_profile(network_id, network_profile)

- Updates the profile information of the network specified by network_id based on a network_profile object specifying the attributes to update.
- 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'.

set_network_properties(network_id, network_properties)

- Updates the NetworkAttributes aspect the network specified by network_id based on the list of NdexPropertyValuePair objects specified in network_properties.
- **This method requires careful use**:
  - Many networks in NDEx have no subnetworks and in those cases the subNetworkId attribute of every NdexPropertyValuePair should **not** be set.
  - Some networks, including some saved from Cytoscape have one subnetwork. In those cases, every NdexPropertyValuePair should have the **subNetworkId attribute set to the id of that subNetwork**.
  - Other networks originating in Cytoscape Desktop correspond to Cytoscape "collections" and may have multiple subnetworks. Each subnetwork may have NdexPropertyValuePairs associated with it and these will be visible in the Cytoscape network viewer. The collection itself may have NdexPropertyValuePairs associated with it and these are not visible in the Cytoscape network viewer but may be set or read by specific Cytoscape Apps. In these cases, **we strongly recommend that you edit these network attributes in Cytoscape** rather than via this API unless you are very familiar with the Cytoscape data model.
- NdexPropertyValuePair object has these attributes:

Attribute

Description

Type

subNetworkId

Optional identifier of the subnetwork to which the property applies.

string

predicateString

Name of the attribute.

string

dataType

Data type of this property. Its value has to be one of the attribute data types that CX supports.

string

value

A string representation of the property value.

string

- Errors if the authenticated user does not have ADMIN permissions to the specified network.
- Errors if network_id does not correspond to an existing network.

get_provenance(network_id)

- Returns the Provenance aspect of the network specified by network_id.
- See the document NDEx Provenance History for a detailed description of this structure and best practices for its use.
- Errors if network_id does not correspond to an existing network.
- The returned value is a Python dict corresponding to a JSON ProvenanceEntity object:
  - A provenance history is a tree structure containing ProvenanceEntity and ProvenanceEvent objects. It is serialized as a JSON structure by the NDEx API.
  - The root of the tree structure is a ProvenanceEntity object representing the current state of the network.
  - Each ProvenanceEntity may have a single ProvenanceEvent object that represents the immediately prior event that produced the ProvenanceEntity. In turn, linked to network of ProvenanceEvent and ProvenanceEntity objects representing the workflow history that produced the current state of the Network.
  - The provenance history records significant events as Networks are copied, modified, or created, incorporating snapshots of information about "ancestor" networks.
  - Attributes in ProvenanceEntity:
    - *uri* : URI of the resource described by the ProvenanceEntity. This field will not be set in some cases, such as a file upload or an algorithmic event that generates a network without a prior network as input
    - *creationEvent* : ProvenanceEvent. has semantics of PROV:wasGeneratedBy properties: array of SimplePropertyValuePair objects
  - Attributes in ProvenanceEvent:
    - *endedAtTime* : timestamp. Has semantics of PROV:endedAtTime
    - *startedAtTime* : timestamp. Has semantics of PROV:endedAtTime
    - *inputs* : array of ProvenanceEntity objects. Has semantics of PROV:used.
    - *properties*: array of SimplePropertyValuePair.

set_provenance(network_id, provenance)

- Updates the Provenance aspect of the network specified by network_id to be the ProvenanceEntity object specified by provenance argument.
- The provenance argument is intended to represent the current state and history of the network and to contain a tree-structure of ProvenanceEvent and ProvenanceEntity objects that describe the networks provenance history.
- Errors if the authenticated user does not have ADMIN permissions to the specified network.
- Errors if network_id does not correspond to an existing network.

Search

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

- Returns a SearchResult object which contains:
  - Array of NetworkSummary objects (networks)
  - the total hit count of the search (numFound)
  - Position of the returned elements (start)
- Search_string parameter specifies the search string.
- **DEPRECATED**: the account_name is optional, but has been superseded by the search string field **userAdmin:account_name** If it is provided, the the search will be constrained to networks owned by that account.
- The start and size parameter are optional. The default values are start = 0 and size = 100.
- The optional include_groups argument defaults to false. It enables search to return a network where a group has permission to access the network and the user is a member of the group. if include_groups is true, the search will also return networks based on permissions from the authenticated user's group memberships.
- The method find_networks is a deprecated alternate name for search_networks.

find_networks(search_string="", account_name=None, start=0, size=100)

- This method is deprecated; search_networks should be used instead.

get_network_summaries_for_user(account_name)

- Returns a SearchResult object which contains:
  - Array of NetworkSummary objects (networks)
  - The total hit count of the search (numFound)
  - Position of the returned elements (start) for user specified by acount_name argument.
- The number of found NetworkSummary objects is limited to (will not exceed) 1000.
- This function will not return networks where a group has permission to access the network and account_name is a member of the group.
- This function is equivalent to calling search_networks("", account_name, size=1000).

get_network_ids_for_user(account_name)

- Returns a list of network Ids for the user specified by acount_name argument. The number of found network Ids is limited to (will not exceed) 1000.
- This function is equivalent to calling get_network_summaries_for_user("", account_name, size=1000), and then building a list of network Ids returned by the call to get_network_summaries_for_user.

get_neighborhood_as_cx_stream(network_id, search_string, search_depth=1, edge_limit=2500)

- Returns a network CX byte stream that is a subset (neighborhood) of the network specified by network_id.
- The subset is determined by a traversal search from nodes identified by search_string to a depth specified by search_depth.
- edge_limit specifies the maximum number of edges that this query can return.
- Server will return an error if the number of edges in the result is larger than the edge_limit parameter.

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

- The arguments and behavior are the same as get_neighborhood_as_cx_stream but returns a Python dict corresponding to a network CX JSON object.

Task

get_task_by_id(task_id)

- Returns a JSON task object for the task specified by task_id.
- Errors if no task found or if the authenticated user does not own the specified task.

1.7 NiceCX Objects

1.7.1 Nodes

create_node(name, represents=None)

Create a new node in the network, specifying the node's name and optionally the id of the entity that it represents.

- **name**: Name for the node
- **represents**: The ID of the entity represented by the node. Best practice is to use IDs from standard namespaces and to define namespace prefixes in the network context.

add_node(node)

Add a node object to the network.

- **node**: A node object (ndexncxModel.cx.aspects.NodesElement)

set_node_attribute(node, attribute_name, values, type=None, subnetwork=None)

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

- **node**: node object or node id

- **attribute_name**: attribute name
- **values**: A value or list of values of the attribute
- **type**: the datatype of the attribute values, defaults to the python datatype of the values.
- **subnetwork**: the id of the subnetwork to which this attribute applies.

get_node_attribute(node, attribute_name, subnetwork=None)

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.

- **node**: node object or node id
- **attribute_name**: attribute name
- **subnetwork**: the id of the subnetwork (if any) to which this attribute applies.

get_node_attribute_objects(node, attribute_name)

Get the attribute objects for a node attribute name, where the node may be specified by its id or passed in as an object. The node attribute objects include datatype and subnetwork information. An example of networks that include subnetworks are Cytoscape collections stored in NDEx.

- **node**: node object or node id
- **attribute_name**: attribute name

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.

- **node**: node object or node id

get_nodes()

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

1.7.2 Edges

create_edge(source, target, interaction)

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

- **source**: The source node this edge, either its id or the node object itself.
- **target**: The target node this edge, either its id or the node object itself.
- **interaction**: The interaction that describes the relationship between the source and target nodes

add_edge(edge)

Add an edge object to the network.

- **edge**: An edge object (nicecxModel.cx.aspects.EdgesElement)

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

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

- **name**: attribute name
- **values**: the values of the attribute
- **type**: the datatype of the attribute values, defaults to the python datatype of the values.
- **subnetwork**: the id of the subnetwork to which this attribute applies.

get_edge_attribute(edge, attribute_name, subnetwork=None)

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.

- **edge**: edge object or edge id
- **attribute_name**: attribute name
- **subnetwork**: the id of the subnetwork (if any) to which this attribute was applied.

get_edge_attribute_objects(edge, attribute_name)

Get the attribute objects for an edge attribute name, where the edge may be specified by its id or passed in as an object. The edge attribute objects include datatype and subnetwork information. An example of networks that include subnetworks are Cytoscape collections stored in NDEx.

- **edge**: node object or node id
- **attribute_name**: attribute name

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.

- **edge**: edge object or edge id

get_edges()

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

1.7.3 Network

get_name()

Get the network name

set_name(network_name)

Set the network name

getSummary()

Get a network summary

set_network_attribute(name=None, values=None, type=None, subnetwork_id=None)

Set an attribute of the network

- **name**: attribute name
- **values**: the values of the attribute
- **type**: the datatype of the attribute values
- **subnetwork**: the id of the subnetwork (if any) to which this attribute applies.

get_network_attribute(attribute_name, subnetwork_id=None)

Get the value of a network attribute

- **attribute_name**: attribute name
- **subnetwork**: the id of the subnetwork (if any) to which this attribute was applied.

get_network_attribute_objects(attribute_name)

Get the attribute objects for the network. The attribute objects include datatype and subnetwork information. An example of networks that include subnetworks are Cytoscape collections stored in NDEx.

get_network_attributes()

Get the attribute objects of the network.

get_metadata()

- Get the network metadata

set_metadata()

- Set the network metadata

getProvenance()

- Get the network provenance as a Python dictionary having the CX provenance schema.

set_provenance(provenance)

- Set the network provenance

get_context(context)

Get the @context aspect of the network, the aspect that maps namespace prefixes to their defining URIs

set_context()

Set the @context aspect of the network, the aspect that maps namespace prefixes to their defining URIs

get_opaque_aspect(aspect_name)

Get the elements of the aspect specified by aspect_name (nicecxModel.cx.aspects.AspectElement)

- **aspect_name**: the name of the aspect to retrieve.

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. (nicecxModel.cx.aspects.AspectElement)

get_opaque_aspect_names()

- Get the names of all opaque aspects

1.7.4 I/O

to_cx()

- Return the CX corresponding to the network.

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

to_networkx()

Return a NetworkX graph based on the network. Elements in the CartesianCoordinates aspect of the network are transformed to the NetworkX **pos** attribute.

to_pandas_dataframe()

Export the network as a Pandas DataFrame.

Example: my_niceCx.upload_to(uuid='34f29fd1-884b-11e7-a10d-0ac135e8bacf', server='http://test.ndexbio.org', username='myusername', password='mypassword')

upload(ndex_server, username, password, update_uuid=None)

Upload the network to the specified NDEx server to the account specified by username and password, return the UUID of the network on NDEx.

Example: my_niceCx.upload_to('http://test.ndexbio.org', 'myusername', 'mypassword')

- server: The NDEx server to upload the network to.
- username: The username of the account to store the network
- password: The password for the account.
- update_uuid: Instead of creating a new network, update the network that has this UUID with the content of this NiceCX object.

apply_template(server, username, password, uuid)

Get a network from NDEx, copy its cytoscapeVisualProperties aspect to this network.

- **server**: The ndex server host of the network from which the layout will be copied
- **username**: Optional username to enable access to a private network
- **password**: Optional password to enable access to a private network
- **uuid**: The unique identifier of the network from which the layout will be copied

to be undocumented…

**any method that works with CX JSON will be an undocumented function for internal use

addNode(json_obj=None)

Used to add a node to the network.

- **name**: Name for the node
- **represents**: The representation for the node. This can be used to store the normalized id for the node
- **json_obj**: The cx representation of a node

add_edge_element(json_obj=None, edge) Low level function

- **json_obj**: The cx representation of an edge

addNetworkAttribute(json_obj=None)

CHAPTER 2

Reference

NDEx2 Python Client documentation

2.1 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*

2.1.1 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

2.1.2 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

2.1.3 Misc niceCX 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 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

2.1.4 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`

2.1.5 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.

2.1.6 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*

2.1.7 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 co-ordinates 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

2.2 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

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_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@ndexbio.org',
 'diskQuota': 10000000000,
 'diskUsed': 3971183103,
 'externalId': 'f2c3a7ef-b0d9-4c61-bf31-4c9fcabe4173',
 'isDeleted': False,
 'modificationTime': 1554410147104,
 '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.

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.

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*)
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*, *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'.

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

2.3 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

2.4 Exceptions

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

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

CHAPTER 3

License

CHAPTER 4

History

4.1 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*

4.2 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

4.3 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

4.4 3.1.0a1 (2019-03-20)

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

4.5 3.0.0a1 (2019-02-11)

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

4.6 2.0.1 (2019-01-03)

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

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

n

Index

L

M

N

P

S

T

U

V