Download PDF

NDEx Python Client Tutorial v3.0

Overview

The NDEx Python Client is a module that simplifies access to the NDEx Server API and provides convenience methods for common operations on networks. This tutorial is composed of 2 sections: section 1 shows how to install the NDEx Python Client module and perform some basic operations on networks stored in an NDEx server. Section 2 provides details about all the API functions available.

NDEx Client Objects

The NDEx Python Client provides an interface to an NDEx server that is managed via a client object class. An NDEx 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 instance of the NDEx Client.

Requirements

The **NDEx Python Client 3.0** requires Python 2.7.9 and the latest version of the PIP Python package manager for installation. Click here (https://pypi.python.org/pypi/pip) to download the PIP Python package.

The **NDEx Python Client 3.0** is undergoing beta testing for Python 3 compatibility: we encourage you to use it in Python 3 and report any issues (https://github.com/ndexbio/ndex-python) you may find.

Finally, this tutorials requires that you first create a personal account on the NDEx Public Server (http://www.ndexbio.org).

Section 1 - Installing the NDEx Python Client Module

The Python NDEx 3.0 module can be installed from the Python Package Index (PyPI) repository using PIP. This tutorial requires the 3.0 release (or higher) of the ndex module. To install this module to your computer using PIP:

```
>>> pip install ndex
```

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

```
>>> pip install --upgrade ndex
```

Setting up NDEx Clients

In this section you will configure two client objects to access the public NDEx server. The first will enable you to make anonymous requests. The second will enable you to perform operations requiring authentication, such as saving networks to your account.

With the ndex module installed, start Python and import ndex.client:

```
>>> python
```

```
>>> import ndex.client as nc
```

Anonymous Clients

The following code creates an NDEx client object to access the NDEx public server anonymously, then tests the client by getting the current server status.

```
>>> anon_ndex=nc.Ndex("http://public.ndexbio.org")

>>> anon_ndex.update_status()

>>> networks = anon_ndex.status.get("networkCount")

>>> users = anon_ndex.status.get("userCount")

>>> groups = anon_ndex.status.get("groupCount")

>>> print("anon client: %s networks, %s users, %s groups" % (networks, users, groups))

anon client: 3097 networks, 672 users, 26 groups

>>>
```

Personal Clients

A personal client enables you to perform operations requiring authentication, such as saving networks to your account. The following code creates an NDEx client object to access your account on the NDEx Public Server, then tests the client by getting the current server status.

Note: you must first create an account on the NDEx Public Server in order to create a personal client object.

For convenience and clarity, this example uses the variables ‘my_account’ and ‘my_password’ to hold the strings for your account name and password. Substitute the values for your account and password.

```
>>> my_account="your account"

>>> my_password="your password"

>>> my_ndex=nc.Ndex("http://public.ndexbio.org", my_account, my_password)

>>> my_ndex.update_status()

>>> networks = my_ndex.status.get("networkCount")

>>> users = my_ndex.status.get("userCount")

>>> groups = my_ndex.status.get("groupCount")

>>> print("my_ndex client: %s networks, %s users, %s groups" % (networks, users, groups))

my_ndex client: 3097 networks, 672 users, 26 groups

>>>
```

Working with NDEx Networks Using the Anonymous Client

Get Network Information by Accession

You can access a network by its accession ID, which is a universally unique identifier (UUID) assigned to the network by the NDEx server. All networks have a UUID and they are unique across all servers - no two networks will share the same UUID.

In this step, you will get basic information about the network, retrieving a NetworkSummary structure. The ‘Metabolism of RNA’ network is in the NDEx Tutorials account on the public NDEx server; its UUID is ‘9ed0cd55-9ac0-11e4-9499-000c29202374’

Method: get_network_summary(network_id)

Returns: a NetworkSummary as a dict

```
>>> ns = anon_ndex.get_network_summary('9ed0cd55-9ac0-11e4-9499-000c29202374')

>>> print("network name is %s." % ns.get('name'))

network name is Metabolism of RNA.
```

```
>>> print("network has %s edges and %s nodes." % (ns.get('edgeCount'), ns.get('nodeCount')))

network has 4344 edges and 361 nodes.

>>>
```

Find Networks by Search

You can search for networks by the text in their name and description as well as the names and controlled vocabulary terms associated with their nodes. The input is a search string that conforms to Lucene search string syntax, but in its simplest form is one or more search terms separated by spaces.

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

Returns: list of NetworkSummary dicts

Search and print the number of networks found:

```
>>> metabolic_networks=anon_ndex.search_networks('metabo*')

>>> print("%s networks found." % (len(metabolic_networks['networks'])))

68 networks found.

>>>
```

The search can also be limited to a specific account and to a number of search results:

```
>>> metabolic_networks=anon_ndex.search_networks('metabo* userAdmin:ndextutorials', size=2)

>>> print("%s networks found" % (len(metabolic_networks['networks'])))

2 networks found
```

```
>>>for ns in metabolic_networks['networks']: print(" %s" % ns.get('name'))

Metabolism

Metabolism of proteins
```

Get a Network

You can obtain an entire network as a CX stream, which is an NDEx format that is optimized for streaming networks. This is performed as a monolithic operation, so care should be taken when requesting very large networks. Applications can use the get_network_summary method to check the node and edge counts for a network before attempting to use get_network_as_cx_stream. The stream is contained in a Response object from the Python requests library (http://docs.python-requests.org/en/master/).

Method: get_network_as_cx_stream(network_id)

Returns: Response object (from Python requests library)

```
>>> response=anon_ndex.get_network_as_cx_stream('9ed0cd55-9ac0-11e4-9499-000c29202374')

>>> print("Received %s characters of CX" % len(response.content))

Received 665511 characters of CX
```

Query a Network – Neighborhood Query

You can retrieve a 'neighborhood' subnetwork of a network as CX stream. The query finds the subnetwork by first identifying nodes that are associated with identifiers in the search_string, then traversing search_depth number of steps from those nodes. The search_depth parameter controls the search, defaults to 1 edge and can be no more than 3 edges.

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

Returns: Subnetwork of a network as a CX Python dict

```
>>> query_result_cx=anon_ndex.get_neighborhood('9ed0cd55-9ac0-11e4-9499-000c29202374', 'XRN1')

>>>

>>> def getNumberOfNodesAndEdgesFromCX(cx):

... numberOfEdges = numberOfNodes = 0;

... for aspect in cx:

... if 'metaData' in aspect:

... metaData = aspect['metaData']

... for element in metaData:

... if (('name' in element) and (element['name'] == 'nodes')):

... numberOfNodes = element['elementCount']

... if (('name' in element) and (element['name'] == 'edges')):

... numberOfEdges = element['elementCount']

... break

... return numberOfNodes, numberOfEdges

...

>>>

>>> nodes, edges = getNumberOfNodesAndEdgesFromCX(query_result_cx)

>>>

>>> print("Query result network contains %s nodes and %s edges." % (nodes, edges))

Query result network contains 20 nodes and 26 edges.

>>>
```

Working with the NDEx Network Using Your Personal Client

Create a Network

You can create a new network on an NDEx server if you have a CX stream. The network is created in the user account associated with the client object. All methods that create or modify content on the NDEx server require authentication, so you will use the my_ndex client object that you set up at the start of the tutorial and will create a network in your account.

In the previous section, your neighborhood query retrieved a small network (26 edges) which was bound to the variable **query_result_cx**.

We will now save this network to your account using **save_new_network(network)** and receive the URI for the new network. The URI includes the network UUID.

Method: save_new_network(network)

Returns: URI of new network

```
>>> uri = my_ndex.save_new_network(query_result_cx)

>>> uuid = uri.rpartition('/')[-1]

>>> print("URI of the newly created network %s is %s" % (uuid, uri))

URI of the newly created network 62ccfcce-042d-11e7-aba2-0ac135e8bacf is http://public.ndexbio.org/v2/network/62ccfcce-042d-11e7-aba2-0ac135e8bacf
```

Update Network Profile

With the network UUID, you can update the name, description and version of the new network using the method **update_network_profile(network_id, network_profile)**.

Method: update_network_profile(network_id, network_profile)

Returns: nothing (empty string)

```
>>> >>> network_profile={"name":"Renamed Network", "description":"New Description", "version":"2.0"}

>>> my_ndex.update_network_profile(uuid, network_profile)

''

>>> new_summary = my_ndex.get_network_summary(uuid)

>>> print("new name = %s" % new_summary.get('name'))

new name = Renamed Network

>>> print("new description = %s" % new_summary.get('description'))

new description = New Description

>>> print("new version = %s" % new_summary.get('version'))

new version = 2.0

>>>
```

Set Read-Only

The new network can be set to read-only using the **set_read_only(network_id, boolean)** method, preventing unintended modification. The read-only networks can also be retrieved more quickly by others. Please note that a read only network cannot be edited or deleted, so you will need to revert it to its original state prior to proceeding with the final step in this tutorial.

Method: set_read_only(network_id, boolean)

Returns: Nothing (empty string)

```
>>> my_ndex.set_read_only(uuid, True)

''

>>> new_summary = my_ndex.get_network_summary(uuid)

>>> print("The read only status is %s" % new_summary.get('isReadOnly'))

The read only status is True

>>>
```

The network is then reverted to read/write thus enabling modification.

```
>>> my_ndex.set_read_only(uuid, False)

''

>>> new_summary = my_ndex.get_network_summary(uuid)

>>> print("The read only status has been reverted to %s" % new_summary.get('isReadOnly'))

The read only status has been reverted to False

>>>
```

Delete a Network

Finally, the query result network is deleted using **delete_network(networkId)**. You can only delete networks that you own. **Be careful... There is no method to undo a deletion!**

Method: delete_network(networkId)

Returns: Nothing (empty string)

You can now delete the network that we saved to your account earlier on:

```
>>> my_ndex.delete_network(uuid)

''
```

Section 2 - NDEx Python Client v3.0 API

User

get_user_by_username(username)

- Returns the user 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)

- ets 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 (../network-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.