AnnotationFrameworkClient

Release 1.0

Oct 20, 2020

Contents:

CHAPTER 1

Getting Started

AnnotationFramework client is a package for simplifying interactions with HTML services associated with the DynamicAnnoationFramework. Including

- pychunkedgraph (For tracking dynamic segmentations)
- NeuroglancerJsonServer (For posting/getting neuroglancer json states)
- AnnotationFrameworkInfoService (For storing dataset metadata information)
- EmAnnotationSchemas (For storing an extensible set of schemas for annotating EM data)
- AnnotatationEngine (For storing annotations on EM data)

1.1 Installation

The AnnotationFrameworkClient can be installed with pip:

```
$ pip install annotationframeworkclient
```

1.2 Assumptions

the code is setup to work flexibly with any deployment of these services, but you need to specify the server_address if that address is not https://www.dynamicannotationframework.com/ for each client when initializing it. similarly, the clients can query the info service for metadata to simplify the interaction with a dataset, but you have to specify a dataset name.

CHAPTER 2

Framework Client: one client for all services

The Dynamic Annotation Framework consists of a number of different services, each with a specific set of tasks that it can perform through REST endpoints. This module is designed to ease programmatic interaction with all of the various endpoints. Going forward, we also will be increasingly using authentication tokens for programmatic access to most if not all of the services. In order to collect a given server, dataset name, and user token together into a coherent package that can be used on multiple endpoints, we will use a FrameworkClient that can build appropriately configured clients for each of the specific services. Each of the individual services has their own specific documentation as well.

2.1 Initializing a FrameworkClient

Assuming that the services are on `www.dynamicannotationframework.com` and authentication tokens are either not being used or set up with default values (see next section), one needs only to specify the dataset name.

```
from annotationframeworkclient import FrameworkClient

dataset_name = 'pinky100'
client = FrameworkClient(dataset_name)
```

Just to confirm that this works, let's see if we can get the EM image source from the InfoService. If you get a reasonable looking path, everything is okay.

```
print(f"The image source is: {client.info.image_source()}")
```

2.1.1 Accessing specific clients

Each client can be acccessed as a property of the main client. See the documentation at left for the capabilities of each. Assuming your client is named `client`, the subclients for each service are:

- Authentication Service : `client.auth`
- AnnotationEngine : `client.annotation`

- PyChunkedGraph : `client.chunkedgraph`
- InfoService : `client.info`
- EM Annotation Schemas : `client.schemas`
- JSON Neuroglancer State Service : `client.state`

In addition, there are more complex clients that use multiple services together:

- LookupClient : Uses Cloudvolume and the ChunkedGraph to look up segmentations associated with point-like arrays or dataframes.
- ImageryClient : Uses Cloudvolume and the ChunkedGraph to look up segmentations and imagery together.

CHAPTER 3

Authentication Service

Going forward, we're going to need authentication tokens for programmatic access to our services. The AuthClient handles storing and loading your token or tokens and inserting it into requests in other clients.

We can access the auth client from `client.auth`. Once you have saved a token, you probably won't interact with this client very often, however it has some convenient features for saving new tokens the first time. Let's see if you have a token already. Probably not.

```
auth = client.auth
print(f"My current token is: {auth.token}")
```

3.1 Getting a new token

It is not yet possible to get a new token programmatically, but the function `get_new_token()` provides instructions for how to get and save it.

By default, the token is saved to `~/.cloudvolume/secrets/chunkedgraph-secret.json` as a string under the key `token`. The following steps will save a token there.

Note: I am not sure where the auth server is being hosted right now, so we are going to use a fake token for documentation purposes

```
auth.get_new_token()
```

```
new_token = 'fake_token_123'
auth.save_token(token=new_token)
print(f"My token is now: {auth.token}")
```

3.2 Loading saved tokens

Try opening `~/.cloudvolume/secrets/chunkedgraph-secret.json` to see what we just created.

If we had wanted to use a different file or a different json key, we could have specified that in auth.save_token.

Because we used the default values, this token is used automatically when we intialize a new FrameworkClient. If we wanted to use a different token file, token key, or even directly specify a token we could do so here.

```
client = FrameworkClient(dataset_name)
print(f"Now my basic token is: {client.auth.token}")

client_direct = FrameworkClient(dataset_name, auth_token='another_fake_token_678')
print(f"A directly specified token is: {client_direct.auth.token}")
```

If you use a FrameworkClient, the AuthClient and its token will be automatically applied to any other services without further use.

CHAPTER 4

AnnotationEngine

The AnnotationClient is used to interact with the AnnotationEngine service to create tables from existing schema, upload new data, and download existing annotations. Note that annotations in the AnnotationEngine are not linked to any particular segmentation, and thus do not include any root ids. An annotation client is accessed with `client.annotation`.

4.1 Getting existing tables

A list of the existing tables for the dataset can be found at with `get_tables`.

```
all_tables = client.annotation.get_tables()
all_tables[0]
```

Each table has three main properties that can be useful to know:

- `table_name` : The table name, used to refer to it when uploading or downloading annotations. This is also passed through to the table in the Materialized database.
- `schema_name` : The name of the table's schema from EMAnnotationSchemas (see below).
- `max_annotation_id` : An upper limit on the number of annotations already contained in the table.

4.2 Downloading annotations

You can download the JSON representation of a data point through the `get_annotation` method. This can be useful if you need to look up information on unmaterialized data, or to see what a properly templated annotation looks like.

```
table_name = all_tables[0]['table_name']      # 'ais_analysis_soma'
annotation_id = 100
client.annotation.get_annotation(annotation_id=annotation_id, table_name=table_name)
```

4.3 Create a new table

One can create a new table with a specified schema with the `create_table` method:

```
client.annotation.create_table(table_name='test_table',
                               schema_name='microns_func_coreg')
```

New data can be generated as a dict or list of dicts following the schema and uploaded with `post_annotation`. For example, a `microns_func_coreg` point needs to have: * `type` set to `microns_func_coreg` * `pt` set to a dict with `position` as a key and the xyz location as a value. * `func_id` set to an integer.

The following could would create a new annotation and then upload it to the service. Note that you get back the annotation id(s) of what you uploaded.

```
new_data = {'type': 'microns_func_coreg',
            'pt': {'position': [1,2,3]},
            'func_id': 0}
client.annotation.post_annotation(table_name='test_table', data=[new_data])
```

CHAPTER 5

ChunkedGraph

The ChunkedGraph client allows one to interact with the ChunkedGraph, which stores and updates the supervoxel agglomeration graph. This is most often useful for looking up an object root id of a supervoxel or looking up supervoxels belonging to a root id. The ChunkedGraph client is at `client.chunkedgraph`.

5.1 Look up a supervoxel

Usually in Neuroglancer, one never notices supervoxel ids, but they are important for programmatic work. In order to look up the root id for a location in space, one needs to use the supervoxel segmentation to get the associated supervoxel id. The ChunkedGraph client makes this easy using the `get_root_ids` method.

```
sv_id = 104200755619042523
client.chunkedgraph.get_root_id(supervoxel_id=sv_id)
```

However, as proofreading occurs, the root id that a supervoxel belongs to can change. By default, this function returns the current state, however one can also provide a UTC timestamp to get the root id at a particular moment in history. This can be useful for reproducible analysis. Note below that the root id for the same supervoxel is different than it is now.

```
import datetime

# I looked up the UTC POSIX timestamp from a day in early 2019.
timestamp = datetime.datetime.utcfromtimestamp(1546595253)

sv_id = 104200755619042523
client.chunkedgraph.get_root_id(supervoxel_id=sv_id, timestamp=timestamp)
```

5.2 Getting supervoxels for a root id

A root id is associated with a particular agglomeration of supervoxels, which can be found with the `get_leaves` method. A new root id is generated for every new change in the chunkedgraph, so time stamps do not apply.

```
root_id = 648518346349541252
client.chunkedgraph.get_leaves(root_id)
```

CHAPTER 6

Info Service

A dataset has a number of complex paths to various data sources that together comprise a dataset. Rather than hardcode these paths, the InfoService allows one to query the location of each data source. This is also convenient in case data sources change.

An InfoClient is accessed at `client.info`.

```
client = FrameworkClient(dataset_name)
print(f"This is an info client for {client.info.dataset_name} on {client.info.server_
↪address}")
```

6.1 Accessing dataset information

All of the information accessible for the dataset can be seen as a dict using `get_dataset_info()`.

```
info.get_dataset_info()
```

Individual entries can be found as well. Use tab autocomplete to see the various possibilities.

```
info.graphene_source()
```

6.2 Adjusting formatting

Because of the way neuroglancer looks up data versus cloudvolume, sometimes one needs to convert between `gs://` style paths to `https://storage.googleapis.com/` stype paths. All of the path sources in the info client accept a `format_for` argument that can handle this, and correctly adapts to graphene vs precomputed data sources.

```
neuroglancer_style_source = info.image_source(format_for='neuroglancer')
print(f"With gs-style: { neuroglancer_style_source }")
```

(continues on next page)

(continued from previous page)

```
cloudvolume_style_source = info.image_source(format_for='cloudvolume')
print(f"With https-style: { cloudvolume_style_source }")
```

CHAPTER 7

EMAnnotationSchemas

The EMAnnotationSchemas client lets one look up the available schemas and how they are defined. This is mostly used for programmatic interactions between services, but can be useful when looking up schema definitions for new tables.

7.1 Get the list of schema

One can get the list of all available schema with the `schema` method. Currently, new schema have to be generated on the server side, although we aim to have a generic set available to use.

```
client.schema.schema()
```

7.2 View a specific schema

The details of each schema can be viewed with the `schema_definition` method, formatted as per JSONSchema.

```
example_schema = client.schema.schema_definition('microns_func_coreg')
example_schema
```

This is mostly useful for programmatic interaction between services at the moment, but can also be used to inspect the expected form of an annotation by digging into the format.

```
example_schema['definitions']['FunctionalCoregistration']
```

CHAPTER 8

JSON Neuroglancer State Service

We store the JSON description of a Neuroglancer state in a simple database at the JSON Service. This is a convenient way to build states to distribute to people, or pull states to parse work by individuals. The JSON Client is at `client.state`

```
client.state
```

8.1 Retrieving a state

JSON states are found simply by their ID, which you get when uploading a state. You can download a state with `get_state_json`.

```
example_id = 4845531975188480
example_state = client.state.get_state_json(test_id)
example_state['layers'][0]
```

8.2 Uploading a state

You can also upload states with `upload_state_json`. If you do this, the state id is returned by the function. Note that there is no easy way to query what you uploaded later, so be VERY CAREFUL with this state id if you wish to see it again.

Note: If you are working with a Neuroglancer Viewer object or similar, in order to upload, use viewer.state.to_json() to generate this representation.

```
example_state['layers'][0]['name'] = 'example_name'
new_id = client.state.upload_state_json(example_state)
```

```
test_state = client.state.get_state_json(new_id)
test_state['layers'][0]['name']
```

CHAPTER 9

LookupClient

The LookupClient simplifies the process of looking up supervoxel ids and root ids for a list of points in space. While this is less efficient than using the materialized database, it can be useful for small-scale scenarios like checking a few hundred annotations before uploading them to the AnnotationEngine or prototyping an analysis.

9.1 Initializing a LookupClient

The LookupClient combines multiple services and can't be created from a server address and dataset name alone, unlike the single service clients. Thus instead of being part of a FrameworkClient object, we instead use a `client` to initialize a LookupClient. To generate a default client, you don't need any arguments.

```
lookup = client.make_lookup_client()
```

If you need more complex features, like setting a timestamp for `root_id` queries, setting a segmentation mip level, or changing the default voxel resolution, these are all possible. For example,

```
import datetime
timestamp = datetime.datetime.utcfromtimestamp(1546595253)
lookup = client.make_lookup_client(timestamp=timestamp,
                                   segmentation_mip=1,
                                   voxel_resolution=[8,8,40])
```

Will use the mip-1 segmentation level to map points to supervoxels, will expect points given in a resolution of 8x8x40 nm, and will query root ids at the specified timestamp.

9.2 Looking up points

The LookupClient links two actions: Finding the supervoxels associated with a point in space, and looking up the root ids for those supervoxels. Supervoxel lookup uses only Cloudvolume and root id lookup uses only the ChunkedGraph. However, for simplicity we can call each here.

```
import numpy as np
pts = [[1,2,3], [3,4,5]]
supervoxel_ids = lookup.lookup_supervoxels(pts)
root_ids = lookup.lookup_root_ids(supervoxel_ids)
```

However, because these two actions are often called as part of one pipeline, we can simplify the process in a single command:

```
root_ids, supervoxel_ids = lookup.lookup_points(pts)
```

Note that one can override the default mip, voxel resolution, and timestamp options for each call. See the method documentation for details.

9.3 Looking up DataFrames

Most of the annotations we work with in the DynamicAnnotationFramework live naturally in tabular DataFrames. In particular, materialized point data follows a schema where each point in a spatial annotation has a location, a supervoxel id, and a root id, as well as whatever other associated metadata there may be (e.g. cell type). To make the LookupClient produce output in a similar format, we have a handy method where you specify one or more point columns in a dataframe.

```
import pandas as pd
#Intialize a dataframe
df = pd.DataFrame(data={'cell_type':['e', 'i'], 'pt':pts})

df_lookup = lookup.lookup_dataframe(point_column='pt', data=df)
```

The resulting `df_lookup` no longer has a column called `pt`, but rather three new columns:

- *pt_positon* : The original point column data
- *pt_supervoxel_id* : The supervoxel id for that point
- *pt_root_id* : The root id for that point

If the segmentation is flat, *pt_supervoxel_id* is omitted since supervoxels and root ids are the same. Each of the suffixes (`_position`, `_supervoxel_id`, and `_root_id`) can be set as optional parameters. The `point_column` argument can also take a list of point column names if more than one point is stored per annotation.

CHAPTER 10

API

10.1 annotationframeworkclient package

10.1.1 Submodules

10.1.2 annotationengine module

10.1.3 auth module

class annotationframeworkclient.auth.**AuthClient**(*token_file='~/.cloudvolume/secrets/chunkedgraph-secret.json'*, *token_key='token'*, *token=None*, *server_address='https://global.daf-apis.com'*)

Bases: object

Client to find and use auth tokens to access the dynamic annotation framework services.

Parameters

- **token_file** (*str, optional*) – Path to a JSON key:value file holding your auth token. By default, "~/.cloudvolume/secrets/chunkedgraph-secret.json"
- **token_key** (*str, optional*) – Key for the token in the token_file. By default, "token"
- **token** (*str or None, optional*) – Direct entry of the token as a string. If provided, overrides the files. If None, attempts to use the file paths.
- **server_address** (*str, optional,*) – URL to the auth server. By default, uses a default server address.

get_new_token(*open=False*)

Currently, returns instructions for getting a new token based on the current settings and saving it to the local environment. New OAuth tokens are currently not able to be retrieved programmatically.

Parameters **open** (*bool, optional*) – If True, opens a web browser to the web page where you can generate a new token.

get_token (*token_key=None*)
Load a token with a given key the specified token file

Parameters **token_key** (*str or None, optional*) – key in the token file JSON, by default None. If None, uses 'token'.

request_header
Formatted request header with the specified token

save_token (*token=None, token_key='token', overwrite=False, token_file=None, switch_token=True*)
Conveniently save a token in the correct format.

After getting a new token by following the instructions in *authclient.get_new_token()*, you can save it with a fully default configuration by running:

token = 'my_shiny_new_token'

authclient.save_token(token=token)

Now on next load, authclient=AuthClient() will make an authclient instance using this token. If you would like to specify more information about the json file where the token will be stored, see the parameters below.

Parameters

- **token** (*str, optional*) – New token to save, by default None
- **token_key** (*str, optional*) – Key for the token in the token_file json, by default "token"
- **overwrite** (*bool, optional*) – Allow an existing token to be changed, by default False
- **token_file** (*str, optional*) – Path to the token file, by default None. If None, uses the default file location specified above.
- **switch_token** (*bool, optional*) – If True, switch the auth client over into using the new token, by default True

token
Secret token used to authenticate yourself to the Dynamic Annotation Framework services.

10.1.4 chunkedgraph module

10.1.5 emannotationschemas module

annotationframeworkclient.emannotationschemas.**SchemaClient** (*server_address=None, auth_client=None, api_version='latest'*)

class annotationframeworkclient.emannotationschemas.**SchemaClientLegacy** (*server_address, auth_header, api_version, endpoints, server_name*)

Bases: annotationframeworkclient.base.ClientBase

schema()
Get the available schema types

Returns List of schema types available on the Schema service.

Return type list

schema_definition(*schema_type*)
Get the definition of a specified schema_type

Parameters **schema_type** (*str*) – Name of a schema_type

Returns Schema definition

Return type json

10.1.6 frameworkclient module

10.1.7 imagery module

10.1.8 infoservice module

annotationframeworkclient.infoservice.**InfoServiceClient**(*server_address=None*, *dataset_name=None*, *auth_client=None*, *api_version='latest'*)

class annotationframeworkclient.infoservice.**InfoServiceClientLegacy**(*server_address*, *auth_header*, *api_version*, *endpoints*, *server_name*, *dataset_name*)

Bases: annotationframeworkclient.base.ClientBaseWithDataset

annotation_endpoint(*dataset_name=None*, *use_stored=True*)
AnnotationEngine endpoint for a dataset.

Parameters

- **dataset_name** (*str or None, optional*) – Name of the dataset to look up. If None, uses the value specified by the client. Default is None.
- **use_stored** (*bool, optional*) – If True, uses the cached value if available. If False, re-queries the InfoService. Default is True.

Returns Location of the AnnotationEngine

Return type str

flat_segmentation_source(*dataset_name=None*, *use_stored=True*, *format_for='raw'*)
Cloud path to the flat segmentation for the dataset

Parameters

- **dataset_name** (*str or None, optional*) – Name of the dataset to look up. If None, uses the value specified by the client. Default is None.
- **use_stored** (*bool, optional*) – If True, uses the cached value if available. If False, re-queries the InfoService. Default is True.

- **format_for** (*'raw', 'cloudvolume', or 'neuroglancer', optional*) – Formats the path for different uses. If 'raw' (default), the path in the InfoService is passed along. If 'cloudvolume', a "precomputed://gs://" type path is converted to a full https URL. If 'neuroglancer', a full https URL is converted to a "precomputed://gs://" type path.

Returns Formatted cloud path to the flat segmentation

Return type str

get_dataset_info(*dataset_name=None*, *use_stored=True*)
Gets the info record for a dataset

Parameters

- **dataset_name** (*str, optional*) – Dataset to look up. If None, uses the one specified by the client. By default None
- **use_stored** (*bool, optional*) – If True and the information has already been queried for that dataset, then uses the cached version. If False, re-queries the infromation. By default True

Returns The complete info record for the dataset

Return type dict or None

get_datasets()
Query which datasets are available at the info service

Returns List of dataset names

Return type list

graphene_source(*dataset_name=None*, *use_stored=True*, *format_for='raw'*)
Cloud path to the chunkgraph-backed Graphene segmentation for a dataset

Parameters

- **dataset_name** (*str or None, optional*) – Name of the dataset to look up. If None, uses the value specified by the client. Default is None.
- **use_stored** (*bool, optional*) – If True, uses the cached value if available. If False, re-queries the InfoService. Default is True.
- **format_for** (*'raw', 'cloudvolume', or 'neuroglancer', optional*) – Formats the path for different uses. If 'raw' (default), the path in the InfoService is passed along. If 'cloudvolume', a "graphene://https://" type path is used If 'neuroglancer', a "graphene://https://" type path is used, as needed by Neuroglancer.

Returns Formatted cloud path to the Graphene segmentation

Return type str

image_source(*dataset_name=None*, *use_stored=True*, *format_for='raw'*)
Cloud path to the imagery for the dataset

Parameters

- **dataset_name** (*str or None, optional*) – Name of the dataset to look up. If None, uses the value specified by the client. Default is None.
- **use_stored** (*bool, optional*) – If True, uses the cached value if available. If False, re-queries the InfoService. Default is True.

- **format_for** (*'raw', 'cloudvolume', or 'neuroglancer', optional*) – Formats the path for different uses. If 'raw' (default), the path in the InfoService is passed along. If 'cloudvolume', a "precomputed://gs://" type path is converted to a full https URL. If 'neuroglancer', a full https URL is converted to a "precomputed://gs://" type path.

Returns Formatted cloud path to the flat segmentation

Return type str

pychunkedgraph_endpoint(*dataset_name=None*, *use_stored=True*)

pychunkedgraph_segmentation_source(*dataset_name=None*, *use_stored=True*, *format_for='raw'*)

pychunkedgraph_viewer_source(***kwargs*)

pychunkgraph_endpoint(***kwargs*)

pychunkgraph_segmentation_source(***kwargs*)

refresh_stored_data()
Reload the stored info values from the server.

supervoxel_source(*dataset_name=None*, *use_stored=True*, *format_for='raw'*)
Cloud path to the supervoxel segmentation for a dataset

Parameters

- **dataset_name** (*str or None, optional*) – Name of the dataset to look up. If None, uses the value specified by the client. Default is None.
- **use_stored** (*bool, optional*) – If True, uses the cached value if available. If False, re-queries the InfoService. Default is True.
- **format_for** (*'raw', 'cloudvolume', or 'neuroglancer', optional*) – Formats the path for different uses. If 'raw' (default), the path in the InfoService is passed along. If 'cloudvolume', a "precomputed://gs://" type path is converted to a full https URL. If 'neuroglancer', a full https URL is converted to a "precomputed://gs://" type path.

Returns Formatted cloud path to the supervoxel segmentation

Return type str

synapse_segmentation_source(*dataset_name=None*, *use_stored=True*, *format_for='raw'*)
Cloud path to the synapse segmentation for a dataset

Parameters

- **dataset_name** (*str or None, optional*) – Name of the dataset to look up. If None, uses the value specified by the client. Default is None.
- **use_stored** (*bool, optional*) – If True, uses the cached value if available. If False, re-queries the InfoService. Default is True.
- **format_for** (*'raw', 'cloudvolume', or 'neuroglancer', optional*) – Formats the path for different uses. If 'raw' (default), the path in the InfoService is passed along. If 'cloudvolume', a "precomputed://gs://" type path is converted to a full https URL. If 'neuroglancer', a full https URL is converted to a "precomputed://gs://" type path.

Returns Formatted cloud path to the synapse segmentation

Return type str

10.1.9 jsonservice module

annotationframeworkclient.jsonservice.**JSONService**(*server_address=None*, *auth_client=None*, *api_version='latest'*)

Client factory to interface with the JSON state service.

Parameters

- **server_address** (*str, optional*) – URL to the JSON state server. If None, set to the default global server address. By default None.
- **auth_client** (*An Auth client, optional*) – An auth client with a token for the same global server, by default None
- **api_version** (*int or 'latest', optional*) – Which endpoint API version to use or 'latest'. By default, 'latest' tries to ask the server for which versions are available, if such functionality exists, or if not it defaults to the latest version for which there is a client. By default 'latest'

class annotationframeworkclient.jsonservice.**JSONServiceLegacy**(*server_address*, *auth_header*, *api_version*, *endpoints*, *server_name*)

Bases: annotationframeworkclient.base.ClientBase

build_neuroglancer_url(*state_id*, *ngl_url*)

Build a URL for a Neuroglancer deployment that will automatically retrieve specified state.

Parameters

- **state_id** (*int*) – State id to retrieve
- **ngl_url** (*str*) – Base url of a neuroglancer deployment. For example, 'https://neuromancer-seung-import.appspot.com'.

Returns The full URL requested

Return type str

get_state_json(*state_id*)

Download a Neuroglancer JSON state

Parameters **state_id** (*int*) – ID of a JSON state uploaded to the state service.

Returns JSON specifying a Neuroglancer state.

Return type dict

upload_state_json(*json_state*)

Upload a Neuroglancer JSON state

Parameters **json_state** (*dict*) – JSON-formatted Neuroglancer state

Returns state_id of the uploaded JSON state

Return type int

class annotationframeworkclient.jsonservice.**JSONServiceV1**(*server_address*, *auth_header*, *api_version*, *endpoints*, *server_name*)

Bases: annotationframeworkclient.base.ClientBase

build_neuroglancer_url(*state_id*, *ngl_url*)
Build a URL for a Neuroglancer deployment that will automatically retrieve specified state.

Parameters

- **state_id** (*int*) – State id to retrieve
- **ngl_url** (*str*) – Base url of a neuroglancer deployment. For example, 'https://neuromancer-seung-import.appspot.com'.

Returns The full URL requested

Return type str

get_state_json(*state_id*)
Download a Neuroglancer JSON state

Parameters **state_id** (*int*) – ID of a JSON state uploaded to the state service.

Returns JSON specifying a Neuroglancer state.

Return type dict

upload_state_json(*json_state*)
Upload a Neuroglancer JSON state

Parameters **json_state** (*dict*) – JSON-formatted Neuroglancer state

Returns state_id of the uploaded JSON state

Return type int

10.1.10 lookup module

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

R

S

T

U