indra_cogex

Release 1.0.0

Benjamin M. Gyori

Aug 30, 2023

GETTING STARTED

1 License and funding **1**

2 INDRA CoGEx modules reference **3**

2.1 INDRA CoGEx Apps . . . 3

2.1.1 INDRA CoGEx Gene List Analysis (`indra_cogex.apps.gla`) . . . 3

2.2 INDRA CoGEx Knowledge Assembly (`indra_cogex.assembly`) . . . 3

2.3 INDRA CoGEx Client . . . 4

2.3.1 Enrichment Analysis (`indra_cogex.client.enrichment`) . . . 4

2.3.2 Neo4j Client (`indra_cogex.client.neo4j_client`) . . . 13

2.3.3 The INDRA CoGEx Neo4j Client (`indra_cogex.client.queries`) . . . 21

2.3.4 Subnetwork Client (`indra_cogex.client.subnetwork`) . . . 31

2.4 Indexing of The Database . . . 33

2.4.1 Indexing The Database (`indra_cogex.indexing`) . . . 33

2.4.2 Indexing CLI (`indra_cogex.indexing.cli`) . . . 34

2.5 INDRA CoGEx Sources . . . 34

2.5.1 Source CLI (`indra_cogex.sources.cli`) . . . 34

2.5.2 INDRA CoGEx Sources Processor (`indra_cogex.sources.processor`) . . . 34

2.5.3 Bgee Processor (`indra_cogex.sources.bgee`) . . . 35

2.5.4 cBioPortal Processor (`indra_cogex.sources.cbioportal`) . . . 35

2.5.5 CellMarker Processor (`indra_cogex.sources.cellmarker`) . . . 36

2.5.6 chembl Processor (`indra_cogex.sources.chembl`) . . . 36

2.5.7 clinicaltrials Processor (`indra_cogex.sources.clinicaltrials`) . . . 37

2.5.8 goa Processor (`indra_cogex.sources.goa`) . . . 37

2.5.9 INDRA DB Processor (`indra_cogex.sources.indra_db`) . . . 38

2.5.10 INDRA Ontology Processor (`indra_cogex.sources.indra_ontology`) . . . 39

2.5.11 InterPro Processor (`indra_cogex.sources.interpro`) . . . 39

2.5.12 Odinson Processor (`indra_cogex.sources.odinson`) . . . 39

2.5.13 Pathways Processor (`indra_cogex.sources.pathways`) . . . 40

2.5.14 PubMed Processor (`indra_cogex.sources.pubmed`) . . . 40

2.5.15 Sider Processor (`indra_cogex.sources.sider`) . . . 40

2.6 INDRA CoGEx Representation (`indra_cogex.representation`) . . . 40

3 Indices and Tables **43**

Python Module Index **45**

Index **47**

CHAPTER

ONE

LICENSE AND FUNDING

INDRA CoGEx is made available under the 2-clause BSD license. The development of this project is funded under the DARPA Young Faculty Award (ARO grant W911NF2010255).

CHAPTER

TWO

INDRA COGEX MODULES REFERENCE

2.1 INDRA CoGEx Apps

INDRA CoGEx apps

2.1.1 INDRA CoGEx Gene List Analysis (`indra_cogex.apps.gla`)

2.2 INDRA CoGEx Knowledge Assembly (`indra_cogex.assembly`)

Assembly of Node objects.

class NodeAssembler(*nodes=None*)

Assembles Node objects.

Initialize a new NodeAssembler object.

Parameters
nodes (`Optional[List[Node]]`) – A list of Node objects.

add_nodes(*nodes*)

Add a list of Node objects to the assembler.

Parameters
nodes (`List[Node]`) – A list of Node objects.

assemble_nodes()

Assemble the nodes in the assembler.

Nodes with the same grounding are assembled into a single node that contains all the labels and data from all the nodes.

Returns
A list of Node objects.

Return type
nodes

get_aggregate_node(*db_ns*, *db_id*, *nodes*)

Aggregate a list of Node objects.

Parameters

- **db_ns** (`str`) – The database namespace of the nodes.
- **db_id** (`str`) – The database id of the nodes.

- **nodes** (List[*Node*]) – A list of Node objects.

Return type
Node

Returns
A Node object with all the labels and data from the input nodes.

2.3 INDRA CoGEx Client

The INDRA CoGEx Client.

2.3.1 Enrichment Analysis (indra_cogex.client.enrichment)

A module for performing enrichment analysis with the INDRA COGEX service.

Continuous Gene Enrichment Analysis (indra_cogex.client.enrichment.continuous)

A collection of analyses possible on gene lists (of HGNC identifiers) with scores.

For example, this could be applied to the log_2 fold scores from differential gene expression experiments.

> **Warning:** This module requires the optional dependency `gseapy`. Install with `pip install gseapy`.

get_human_scores(*path*, *read_csv_kwargs=None*, *gene_symbol_column_name=None*, *score_column_name=None*)

Load a differential gene expression file with human measurements.

Parameters

- **path** (Union[Path, str, DataFrame]) – Path to the file to read with `pandas.read_csv()`.
- **read_csv_kwargs** (Optional[Dict[str, Any]]) – Keyword arguments to pass to `pandas.read_csv()`
- **gene_symbol_column_name** (Optional[str]) – The name of the column with gene symbols. If none, will try and guess.
- **score_column_name** (Optional[str]) – The name of the column with scores. If none, will try and guess.

Return type
Dict[str, float]

Returns
A dictionary of human gene HGNC IDs to scores.

get_mouse_scores(*path*, *read_csv_kwargs=None*, *gene_symbol_column_name=None*, *score_column_name=None*)

Load a differential gene expression file with mouse measurements.

This function extracts the MGI gene symbols, maps them to MGI identifiers, uses PyOBO to map orthologs to HGNC, then returns the HGNC gene and scores as a dictionary.

Parameters

- **path** (Union[Path, str, DataFrame]) – Path to the file to read with pandas.read_csv().
- **read_csv_kwargs** (Optional[Dict[str, Any]]) – Keyword arguments to pass to pandas.read_csv()
- **gene_symbol_column_name** (Optional[str]) – The name of the column with gene symbols. If none, will try and guess.
- **score_column_name** (Optional[str]) – The name of the column with scores. If none, will try and guess.

Return type
Dict[str, float]

Returns
A dictionary of mapped orthologus human gene HGNC IDs to scores.

get_rat_scores(*path*, *read_csv_kwargs=None*, *gene_symbol_column_name=None*, *score_column_name=None*)

Load a differential gene expression file with rat measurements.

This function extracts the RGD gene symbols, maps them to RGD identifiers, uses PyOBO to map orthologs to HGNC, then returns the HGNC gene and scores as a dictionary.

Parameters

- **path** (Union[Path, str, DataFrame]) – Path to the file to read with pandas.read_csv().
- **read_csv_kwargs** (Optional[Dict[str, Any]]) – Keyword arguments to pass to pandas.read_csv()
- **gene_symbol_column_name** (Optional[str]) – The name of the column with gene symbols. If none, will try and guess.
- **score_column_name** (Optional[str]) – The name of the column with scores. If none, will try and guess.

Return type
Dict[str, float]

Returns
A dictionary of mapped orthologus human gene HGNC IDs to scores.

go_gsea(*scores*, *directory=None*, **, *client*, ***kwargs*)

Run GSEA with gene sets for each Gene Ontology term.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **scores** (Dict[str, float]) – A mapping from HGNC gene identifiers to floating point scores (e.g., from a differential gene expression analysis)
- **directory** (Union[None, Path, str]) – Specify the directory if the results should be saved, including both a dataframe and plots for each gen set
- **kwargs** – Remaining keyword arguments to pass through to gseapy.prerank()

Return type
DataFrame

Returns
A pandas dataframe with the GSEA results

gsea(*scores*, *gene_sets*, *directory=None*, *alpha=None*, *keep_insignificant=True*, ***kwargs*)

Run GSEA on pre-ranked data.

Parameters

- **scores** (Dict[str, float]) – A mapping from HGNC gene identifiers to floating point scores (e.g., from a differential gene expression analysis)
- **gene_sets** (Dict[Tuple[str, str], Set[str]]) – A mapping from
- **directory** (Union[None, Path, str]) – Specify the directory if the results should be saved, including both a dataframe and plots for each gen set
- **alpha** (Optional[float]) – The cutoff for significance. Defaults to 0.05
- **keep_insignificant** (bool) – If false, removes results with a p value less than alpha.
- **kwargs** – Remaining keyword arguments to pass through to `gseapy.prerank()`

Return type
DataFrame

Returns
A pandas dataframe with the GSEA results

indra_downstream_gsea(*scores*, *directory=None*, ***, *client*, *minimum_evidence_count=None*, *minimum_belief=None*, ***kwargs*)

Run GSEA for each entry in the INDRA database and the set of human genes that are upstream regulators of it.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **scores** (Dict[str, float]) – A mapping from HGNC gene identifiers to floating point scores (e.g., from a differential gene expression analysis)
- **directory** (Union[None, Path, str]) – Specify the directory if the results should be saved, including both a dataframe and plots for each gen set
- **minimum_evidence_count** (Optional[int]) – The minimum number of evidences for a relationship to count it as a regulator. Defaults to 1 (i.e., cutoff not applied.
- **minimum_belief** (Optional[float]) – The minimum belief for a relationship to count it as a regulator. Defaults to 0.0 (i.e., cutoff not applied).
- **kwargs** – Remaining keyword arguments to pass through to `gseapy.prerank()`

Return type
DataFrame

Returns
A pandas dataframe with the GSEA results

indra_upstream_gsea(*scores*, *directory=None*, ***, *client*, *minimum_evidence_count=None*, *minimum_belief=None*, ***kwargs*)

Run GSEA for each entry in the INDRA database and the set of human genes that it regulates.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **scores** (Dict[str, float]) – A mapping from HGNC gene identifiers to floating point scores (e.g., from a differential gene expression analysis)

- **directory** (Union[None, Path, str]) – Specify the directory if the results should be saved, including both a dataframe and plots for each gen set
- **minimum_evidence_count** (Optional[int]) – The minimum number of evidences for a relationship to count it as a regulator. Defaults to 1 (i.e., cutoff not applied.
- **minimum_belief** (Optional[float]) – The minimum belief for a relationship to count it as a regulator. Defaults to 0.0 (i.e., cutoff not applied).
- **kwargs** – Remaining keyword arguments to pass through to `gseapy.prerank()`

Return type
DataFrame

Returns
A pandas dataframe with the GSEA results

phenotype_gsea(*scores*, *directory=None*, *, *client*, ***kwargs*)

Run GSEA with HPO phenotype gene sets.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **scores** (Dict[str, float]) – A mapping from HGNC gene identifiers to floating point scores (e.g., from a differential gene expression analysis)
- **directory** (Union[None, Path, str]) – Specify the directory if the results should be saved, including both a dataframe and plots for each gen set
- **kwargs** – Remaining keyword arguments to pass through to `gseapy.prerank()`

Return type
DataFrame

Returns
A pandas dataframe with the GSEA results

reactome_gsea(*scores*, *directory=None*, *, *client*, ***kwargs*)

Run GSEA with Reactome gene sets.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **scores** (Dict[str, float]) – A mapping from HGNC gene identifiers to floating point scores (e.g., from a differential gene expression analysis)
- **directory** (Union[None, Path, str]) – Specify the directory if the results should be saved, including both a dataframe and plots for each gen set
- **kwargs** – Remaining keyword arguments to pass through to `gseapy.prerank()`

Return type
DataFrame

Returns
A pandas dataframe with the GSEA results

wikipathways_gsea(*scores*, *directory=None*, *, *client*, ***kwargs*)

Run GSEA with WikiPathways gene sets.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.

- **scores** (Dict[str, float]) – A mapping from HGNC gene identifiers to floating point scores (e.g., from a differential gene expression analysis)
- **directory** (Union[None, Path, str]) – Specify the directory if the results should be saved, including both a dataframe and plots for each gen set
- **kwargs** – Remaining keyword arguments to pass through to gseapy.prerank()

Return type
DataFrame

Returns
A pandas dataframe with the GSEA results

Discrete Gene Enrichment Analysis (indra_cogex.client.enrichment.discrete)

A collection of analyses possible on gene lists (of HGNC identifiers).

go_ora(*client*, *gene_ids*, *background_gene_ids=None*, ***kwargs*)

Calculate over-representation on all GO terms.

Parameters

- **client** (*Neo4jClient*) – Neo4jClient
- **gene_ids** (Iterable[str]) – List of HGNC gene identifiers
- **background_gene_ids** (Optional[Collection[str]]) – List of HGNC gene identifiers for the background gene set. If not given, all genes with HGNC IDs are used as the background.
- ****kwargs** – Additional keyword arguments to pass to _do_ora

Return type
DataFrame

Returns
DataFrame with columns: curie, name, p, q, mlp, mlq

indra_downstream_ora(*client*, *gene_ids*, *background_gene_ids=None*, ***, *minimum_evidence_count=1*, *minimum_belief=0.0*, ***kwargs*)

Calculate a p-value for each entity in the INDRA database based on the genes that are causally upstream of it and how they compare to the query gene set.

Parameters

- **client** (*Neo4jClient*) – Neo4jClient
- **gene_ids** (Iterable[str]) – List of HGNC gene identifiers
- **background_gene_ids** (Optional[Collection[str]]) – List of HGNC gene identifiers for the background gene set. If not given, all genes with HGNC IDs are used as the background.
- **minimum_evidence_count** (Optional[int]) – Minimum number of evidences to consider a causal relationship
- **minimum_belief** (Optional[float]) – Minimum belief to consider a causal relationship
- ****kwargs** – Additional keyword arguments to pass to _do_ora

Return type
DataFrame

Returns
DataFrame with columns: curie, name, p, q, mlp, mlq

indra_upstream_ora(*client*, *gene_ids*, *background_gene_ids=None*, *, *minimum_evidence_count=1*, *minimum_belief=0.0*, ***kwargs*)

Calculate a p-value for each entity in the INDRA database based on the set of genes that it regulates and how they compare to the query gene set.

Parameters

- **client** (*Neo4jClient*) – Neo4jClient
- **gene_ids** (Iterable[str]) – List of HGNC gene identifiers
- **background_gene_ids** (Optional[Collection[str]]) – List of HGNC gene identifiers for the background gene set. If not given, all genes with HGNC IDs are used as the background.
- **minimum_evidence_count** (Optional[int]) – Minimum number of evidences to consider a causal relationship
- **minimum_belief** (Optional[float]) – Minimum belief to consider a causal relationship
- ****kwargs** – Additional keyword arguments to pass to _do_ora

Return type
DataFrame

Returns
DataFrame with columns: curie, name, p, q, mlp, mlq

phenotype_ora(*gene_ids*, *background_gene_ids=None*, *, *client*, ***kwargs*)

Calculate over-representation on all HP phenotypes.

Parameters

- **gene_ids** (Iterable[str]) – List of HGNC gene identifiers
- **background_gene_ids** (Optional[Collection[str]]) – List of HGNC gene identifiers for the background gene set. If not given, all genes with HGNC IDs are used as the background.
- **client** (*Neo4jClient*) – Neo4jClient
- ****kwargs** – Additional keyword arguments to pass to _do_ora

Return type
DataFrame

Returns
DataFrame with columns: curie, name, p, q, mlp, mlq

reactome_ora(*client*, *gene_ids*, *background_gene_ids=None*, ***kwargs*)

Calculate over-representation on all Reactome pathways.

Parameters

- **client** (*Neo4jClient*) – Neo4jClient
- **gene_ids** (Iterable[str]) – List of HGNC gene identifiers
- **background_gene_ids** (Optional[Collection[str]]) – List of HGNC gene identifiers for the background gene set. If not given, all genes with HGNC IDs are used as the background.

- ****kwargs** – Additional keyword arguments to pass to _do_ora

Return type
`DataFrame`

Returns
DataFrame with columns: curie, name, p, q, mlp, mlq

wikipathways_ora(*client*, *gene_ids*, *background_gene_ids=None*, ***kwargs*)

Calculate over-representation on all WikiPathway pathways.

Parameters

- **client** (*Neo4jClient*) – Neo4jClient
- **gene_ids** (Iterable[str]) – List of HGNC gene identifiers
- **background_gene_ids** (Optional[Collection[str]]) – List of HGNC gene identifiers for the background gene set. If not given, all genes with HGNC IDs are used as the background.
- ****kwargs** – Additional keyword arguments to pass to _do_ora

Return type
`DataFrame`

Returns
DataFrame with columns: curie, name, p, q, mlp, mlq

Signed Gene Enrichment Analysis (indra_cogex.client.enrichment.signed)

A collection of analyses possible on pairs of gene lists (of HGNC identifiers).

main()

Demonstrate signed gene list functions.

reverse_causal_reasoning(*positive_hgnc_ids*, *negative_hgnc_ids*, *minimum_size=4*, *alpha=None*, *keep_insignificant=True*, ***, *client*, *minimum_evidence_count=None*, *minimum_belief=None*)

Implement the Reverse Causal Reasoning algorithm from [catlett2013].

Parameters

- **client** (*Neo4jClient*) – A neo4j client
- **positive_hgnc_ids** (Iterable[str]) – A list of positive-signed HGNC gene identifiers (e.g., up-regulated genes in a differential gene expression analysis)
- **negative_hgnc_ids** (Iterable[str]) – A list of negative-signed HGNC gene identifiers (e.g., down-regulated genes in a differential gene expression analysis)
- **minimum_size** (int) – The minimum number of entities marked as downstream of an entity for it to be usable as a hyp
- **alpha** (Optional[float]) – The cutoff for significance. Defaults to 0.05
- **keep_insignificant** (bool) – If false, removes results with a p value less than alpha.
- **minimum_evidence_count** (Optional[int]) – The minimum number of evidences for a relationship to count it as a regulator. Defaults to 1 (i.e., cutoff not applied).
- **minimum_belief** (Optional[float]) – The minimum belief for a relationship to count it as a regulator. Defaults to 0.0 (i.e., cutoff not applied).

Return type
`DataFrame`

Returns

- A pandas DataFrame with results for each entity in the graph database
- **.. [catlett2013] Catlett, N. L., *et al.* (2013). `Reverse causal reasoning** (*applying*) – qualitative causal knowledge to the interpretation of high-throughput data <https://doi.org/10.1186/1471-2105-14-340>`_. BMC Bioinformatics, **14**(1), 340.

Gene Enrichment Analysis Utilities (`indra_cogex.client.enrichment.utils`)

Utility functions for gene enrichment analysis.

Utilities for getting gene sets.

`collect_gene_sets`(*query*, *, *client*, *background_gene_ids=None*, *include_ontology_children=False*, *cache_file=None*)

Collect gene sets based on the given query.

Parameters

- **`query`** (`str`) – A cypher query
- **`client`** (*`Neo4jClient`*) – The Neo4j client.
- **`background_gene_ids`** (`Optional`[`Iterable`[`str`]]) – List of HGNC gene identifiers for the background gene set. If not given, all genes with HGNC IDs are used as the background.
- **`include_ontology_children`** (`bool`) – If True, extend the gene set associations with associations from child terms using the indra ontology
- **`cache_file`** (`Optional`[`Path`]) – The path to the cache file.

Return type
`Dict`[`Tuple`[`str`, `str`], `Set`[`str`]]

Returns
A dictionary whose keys that are 2-tuples of CURIE and name of each queried item and whose values are sets of HGNC gene identifiers (as strings)

`get_entity_to_regulators`(*, *client*, *background_gene_ids=None*, *minimum_evidence_count=1*, *minimum_belief=0.0*)

Get a mapping from each entity in the INDRA database to the set of human genes that are causally upstream of it.

Parameters

- **`client`** (*`Neo4jClient`*) – The Neo4j client.
- **`background_gene_ids`** (`Optional`[`Iterable`[`str`]]) – List of HGNC gene identifiers for the background gene set. If not given, all genes with HGNC IDs are used as the background.
- **`minimum_evidence_count`** (`Optional`[`int`]) – The minimum number of evidences for a relationship to count it as a regulator. Defaults to 1 (i.e., cutoff not applied.
- **`minimum_belief`** (`Optional`[`float`]) – The minimum belief for a relationship to count it as a regulator. Defaults to 0.0 (i.e., cutoff not applied).

Return type
`Dict`[`Tuple`[`str`, `str`], `Set`[`str`]]

Returns
A dictionary whose keys that are 2-tuples of CURIE and name of each entity and whose values are sets of HGNC gene identifiers (as strings)

`get_entity_to_targets`(*, *client*, *background_gene_ids=None*, *minimum_evidence_count=1*, *minimum_belief=0.0*)

Get a mapping from each entity in the INDRA database to the set of human genes that it regulates.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **background_gene_ids** (Optional[Iterable[str]]) – List of HGNC gene identifiers for the background gene set. If not given, all genes with HGNC IDs are used as the background.
- **minimum_evidence_count** (Optional[int]) – The minimum number of evidences for a relationship to count it as a regulator. Defaults to 1 (i.e., cutoff not applied.
- **minimum_belief** (Optional[float]) – The minimum belief for a relationship to count it as a regulator. Defaults to 0.0 (i.e., cutoff not applied).

Return type
Dict[Tuple[str, str], Set[str]]

Returns
A dictionary whose keys that are 2-tuples of CURIE and name of each entity and whose values are sets of HGNC gene identifiers (as strings)

`get_go`(*, *background_gene_ids=None*, *client*)

Get GO gene sets.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **background_gene_ids** (Optional[Iterable[str]]) – List of HGNC gene identifiers for the background gene set. If not given, all genes with HGNC IDs are used as the background.

Return type
Dict[Tuple[str, str], Set[str]]

Returns
A dictionary whose keys that are 2-tuples of CURIE and name of each GO term and whose values are sets of HGNC gene identifiers (as strings)

`get_phenotype_gene_sets`(*, *background_gene_ids=None*, *client*)

Get HPO phenotype gene sets.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **background_gene_ids** (Optional[Iterable[str]]) – List of HGNC gene identifiers for the background gene set. If not given, all genes with HGNC IDs are used as the background.

Return type
Dict[Tuple[str, str], Set[str]]

Returns
A dictionary whose keys that are 2-tuples of CURIE and name of each phenotype gene set and whose values are sets of HGNC gene identifiers (as strings)

get_reactome(*, *background_gene_ids=None*, *client*)

Get Reactome gene sets.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **background_gene_ids** (Optional[Iterable[str]]) – List of HGNC gene identifiers for the background gene set. If not given, all genes with HGNC IDs are used as the background.

Return type
Dict[Tuple[str, str], Set[str]]

Returns
A dictionary whose keys that are 2-tuples of CURIE and name of each Reactome pathway and whose values are sets of HGNC gene identifiers (as strings)

get_wikipathways(*, *background_gene_ids=None*, *client*)

Get WikiPathways gene sets.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **background_gene_ids** (Optional[Iterable[str]]) – List of HGNC gene identifiers for the background gene set. If not given, all genes with HGNC IDs are used as the background.

Return type
Dict[Tuple[str, str], Set[str]]

Returns
A dictionary whose keys that are 2-tuples of CURIE and name of each WikiPathway pathway and whose values are sets of HGNC gene identifiers (as strings)

2.3.2 Neo4j Client (indra_cogex.client.neo4j_client)

Neo4j client module.

class Neo4jClient(*url=None*, *auth=None*)

A client to communicate with an INDRA CogEx neo4j instance

Parameters

- **url** (Optional[str]) – The bolt URL to the neo4j instance to override INDRA_NEO4J_URL set as an environment variable or set in the INDRA config file.
- **auth** (Optional[Tuple[str, str]]) – A tuple consisting of the user name and password for the neo4j instance to override INDRA_NEO4J_USER and INDRA_NEO4J_PASSWORD set as environment variables or set in the INDRA config file.

Initialize the Neo4j client.

add_node(*node*)

Merge a single node into the graph.

add_nodes(*nodes*)

Merge a set of graph nodes (create or update).

add_relations(*relations*)

Merge a set of graph relations (create or update).

close_session()

Close the session if it exists.

create_nodes(*nodes*)

Create a set of new graph nodes.

create_single_property_node_index(*index_name*, *label*, *property_name*, *exist_ok=False*)

Create a single property node index.

Reference: https://neo4j.com/docs/cypher-manual/4.4/indexes-for-search-performance/#administration-indexes-create-a-single-property-b-tree-index-only-if-it-does-not-already-exist

Parameters

- **index_name** (str) – The name of the index.
- **label** (str) – The label of the node.
- **property_name** (str) – The property name to index.
- **exist_ok** (bool) – If True, ignore the indexes that already exist. If False, raise error if index already exists. Default: False.

create_single_property_relationship_index(*index_name*, *rel_type*, *property_name*)

Create a single property relationship index.

NOTE: Relationship indexes can only be created once, and there is no IF NOT EXISTS option to silently ignore if the index already exists.

Reference: https://neo4j.com/docs/cypher-manual/4.4/indexes-for-search-performance/#administration-indexes-create-a-single-property-b-tree-index-for-relationships

Parameters

- **index_name** (str) – The name of the index.
- **rel_type** (str) – The relationship type to index a property on
- **property_name** (str) – The property name to index.

create_tx(*query*, *query_params=None*)

Run a transaction which writes to the neo4j instance.

Parameters

- **query** (str) – The query string to be executed.
- **query_params** (Optional[Mapping[str, Any]]) – Parameters associated with the query.

delete_all()

Delete everything in the neo4j database.

get_all_relations(*node*, *relation=None*, *node_type=None*, *other_type=None*)

Get relations that connect sources and targets with the given node.

Parameters

- **node** (Tuple[str, str]) – Node namespace and identifier.
- **relation** (Optional[str]) – Relation type.
- **node_type** (Optional[str]) – Type constraint on the queried node itself
- **other_type** (Optional[str]) – Type constraint on the other node in the relation

Returns
A list of relations matching the constraints.

Return type
rels

`get_common_sources`(*targets*, *relation*, *source_type=None*, *target_type=None*)

Return the common source nodes related to all the given targets via a given relation type.

Parameters

- **targets** (List[Tuple[str, str]]) – The target nodes' IDs.
- **relation** (str) – The relation label to constrain to when finding sources.
- **source_type** (Optional[str]) – A constraint on the source type
- **target_type** (Optional[str]) – A constraint on the target type

Returns
A list of source nodes.

Return type
sources

`get_common_targets`(*sources*, *relation*, *source_type=None*, *target_type=None*)

Return the common target nodes related to all the given sources via a given relation type.

Parameters

- **sources** (List[Tuple[str, str]]) – Source namespace and identifier.
- **relation** (str) – The relation label to constrain to when finding targets.
- **source_type** (Optional[str]) – A constraint on the source type
- **target_type** (Optional[str]) – A constraint on the target type

Returns
A list of target nodes.

Return type
targets

`get_predecessors`(*target*, *relations*, *source_type=None*, *target_type=None*)

Return the nodes that precede the given node via the given relation types.

Parameters

- **target** (Tuple[str, str]) – The target node's ID.
- **relations** (Iterable[str]) – The relation labels to constrain to when finding predecessors.
- **source_type** (Optional[str]) – A constraint on the source type
- **target_type** (Optional[str]) – A constraint on the target type

Returns
A list of predecessor nodes.

Return type
predecessors

static get_property_from_relations(*relations*, *prop*)

Return the set of property values on given relations.

Parameters

- **relations** (List[*Relation*]) – The relations, each of which may or may not contain a value for the given property.
- **prop** (str) – The key/name of the property to look for on each relation.

Returns
A set of the values of the given property on the given list of relations.

Return type
props

get_relations(*source=None*, *target=None*, *relation=None*, *source_type=None*, *target_type=None*, *limit=None*, *bidirectional=False*)

Return relations based on source, target and type constraints.

This is a generic function for getting relations, all of its parameters are optional, though at least a source or a target needs to be provided.

Parameters

- **source** (Optional[Tuple[str, str]]) – Surce namespace and ID.
- **target** (Optional[Tuple[str, str]]) – Target namespace and ID.
- **relation** (Optional[str]) – Relation type.
- **source_type** (Optional[str]) – A constraint on the source type
- **target_type** (Optional[str]) – A constraint on the target type
- **limit** (Optional[int]) – A limit on the number of relations returned.
- **bidirectional** (Optional[bool]) – If True, return both directions of relationships between the source and target.

Returns
A list of relations matching the constraints.

Return type
rels

get_session(*renew=False*)

Return an existing session or create one if needed.

Parameters
renew (Optional[bool]) – If True, a new session is created. Default: False

Returns
A neo4j session.

Return type
session

get_source_agents(*target*, *relation*)

Return the nodes related to the target via a given relation type as INDRA Agents.

Parameters

- **target** (Tuple[str, str]) – Target namespace and identifier.

- **relation** (str) – The relation label to constrain to when finding sources.

Returns
A list of source nodes as INDRA Agents.

Return type
sources

get_source_relations(*target*, *relation=None*, *target_type=None*, *source_type=None*)

Get relations that connect sources to the given target.

Parameters

- **target** (Tuple[str, str]) – Target namespace and identifier.
- **relation** (Optional[str]) – Relation type.
- **target_type** (Optional[str]) – A constraint on the target node type.
- **source_type** (Optional[str]) – A constraint on the source node type.

Returns
A list of relations matching the constraints.

Return type
rels

get_sources(*target*, *relation=None*, *source_type=None*, *target_type=None*)

Return the nodes related to the target via a given relation type.

Parameters

- **target** (Tuple[str, str]) – The target node's ID.
- **relation** (Optional[str]) – The relation label to constrain to when finding sources.
- **source_type** (Optional[str]) – A constraint on the source type
- **target_type** (Optional[str]) – A constraint on the target type

Returns
A list of source nodes.

Return type
sources

get_successors(*source*, *relations*, *source_type=None*, *target_type=None*)

Return the nodes that precede the given node via the given relation types.

Parameters

- **source** (Tuple[str, str]) – The source node's ID.
- **relations** (Iterable[str]) – The relation labels to constrain to when finding successors.
- **source_type** (Optional[str]) – A constraint on the source type
- **target_type** (Optional[str]) – A constraint on the target type

Returns
A list of successors nodes.

Return type
predecessors

get_target_agents(*source*, *relation*, *source_type=None*)

Return the nodes related to the source via a given relation type as INDRA Agents.

Parameters

- **source** (Tuple[str, str]) – Source namespace and identifier.
- **relation** (str) – The relation label to constrain to when finding targets.
- **source_type** (Optional[str]) – A constraint on the source type

Returns
A list of target nodes as INDRA Agents.

Return type
targets

get_target_relations(*source*, *relation=None*, *source_type=None*, *target_type=None*)

Get relations that connect targets from the given source.

Parameters

- **source** (Tuple[str, str]) – Source namespace and identifier.
- **relation** (Optional[str]) – Relation type.
- **source_type** (Optional[str]) – A constraint on the source node type.
- **target_type** (Optional[str]) – A constraint on the target node type.

Returns
A list of relations matching the constraints.

Return type
rels

get_targets(*source*, *relation=None*, *source_type=None*, *target_type=None*)

Return the nodes related to the source via a given relation type.

Parameters

- **source** (Tuple[str, str]) – Source namespace and identifier.
- **relation** (Optional[str]) – The relation label to constrain to when finding targets.
- **source_type** (Optional[str]) – A constraint on the source type
- **target_type** (Optional[str]) – A constraint on the target type

Returns
A list of target nodes.

Return type
targets

has_relation(*source*, *target*, *relation*, *source_type=None*, *target_type=None*)

Return True if there is a relation between the source and the target.

Parameters

- **source** (Tuple[str, str]) – Source namespace and identifier.
- **target** (Tuple[str, str]) – Target namespace and identifier.
- **relation** (str) – Relation type.
- **source_type** (Optional[str]) – A constraint on the source type

- **target_type** (Optional[str]) – A constraint on the target type

Returns
True if there is a relation of the given type, otherwise False.

Return type
related

static neo4j_to_node(*neo4j_node*)

Return a Node from a neo4j internal node.

Parameters
neo4j_node (Node) – A neo4j internal node using its internal data structure and identifier scheme.

Returns
A Node object with the INDRA standard identifier scheme.

Return type
node

classmethod neo4j_to_relation(*neo4j_path*)

Return a Relation from a neo4j internal single-relation path.

Parameters
neo4j_path (Path) – A neo4j internal single-edge path using its internal data structure and identifier scheme.

Returns
A Relation object with the INDRA standard identifier scheme.

Return type
relation

static neo4j_to_relations(*neo4j_path*)

Return a list of Relations from a neo4j internal multi-relation path.

Parameters
neo4j_path (Path) – A neo4j internal single-edge path using its internal data structure and identifier scheme.

Return type
List[*Relation*]

Returns
A list of Relation objects with the INDRA standard identifier scheme.

static node_to_agent(*node*)

Return an INDRA Agent from a Node.

Parameters
node (*Node*) – A Node object.

Returns
An INDRA Agent with standardized name and expanded/standardized db_refs.

Return type
agent

query_dict(*query*, ***query_params*)

Run a read-only query that generates a dictionary.

Return type
`Dict`

query_dict_value_json(*query*, ***query_params*)

Run a read-only query that generates a dictionary.

Return type
`Dict`

query_nodes(*query*, ***query_params*)

Run a read-only query for nodes.

Parameters

- **query** (`str`) – The query string to be executed.
- **query_params** – Query parameters to pass to cypher

Returns
A list of `Node` instances corresponding to the results of the query

Return type
values

query_relations(*query*, ***query_params*)

Run a read-only query for relations.

Parameters

- **query** (`str`) – The query string to be executed. Must have a `RETURN` with a single element `p` where in the `MATCH` part of the query it has something like `p=(h)-[r]->(t)`.
- **query_params** – Query parameters to pass to query transaction function that will fill out the placeholders in the cypher query

Returns
A list of `Relation` instances corresponding to the results of the query

Return type
values

query_tx(*query*, *squeeze=False*, ***query_params*)

Run a read-only query and return the results.

Parameters

- **query** (`str`) – The query string to be executed.
- **squeeze** (`bool`) – If true, unpacks the 0-indexed element in each value returned. Useful when only returning value per row of the results.
- **query_params** – kwargs to pass to query

Returns
A list of results where each result is a list of one or more objects (typically neo4j nodes or relations).

Return type
values

session: Optional[Session]

The session

autoclient(*, *cache=False*, *maxsize=128*)

Wrap a function that takes a client for easier usage.

Parameters

- **cache** (bool) – Should the result be cached using functools.lru_cache()? Is False by default.
- **maxsize** (Optional[int]) – If cache is True, this is the value passed to the maxsize argument of functools.lru_cache(). Set to None for unlimited caching, but beware that this can potentially use a lot of memory and isn't a good idea for queries that can take a lot of different kinds of input over time.

Returns

A decorator object that will wrap the function

Examples

Not appropriate for caching (i.e., many possible inputs, especially in a web app scenario):

```
@autoclient()
def get_tissues_for_gene(gene: Tuple[str, str], *, client: Neo4jClient):
    return client.get_targets(
        gene,
        relation="expressed_in",
        source_type="BioEntity",
        target_type="BioEntity",
    )
```

Appropriate for caching (e.g., doen't take inputs at all):

```
@autoclient(cache=True, maxsize=1)
def get_node_count(*, client: Neo4jClient) -> Counter:
    return Counter(
        {
            label[0]: client.query_tx(f"MATCH (n:{label[0]}) RETURN count(*)")[0][0]
            for label in client.query_tx("call db.labels();")
        }
    )
```

2.3.3 The INDRA CoGEx Neo4j Client (indra_cogex.client.queries)

get_diseases_for_trial(*trial*, *, *client*)

Return the diseases for the given trial.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **trial** (Tuple[str, str]) – The trial to query.

Return type

Iterable[*Node*]

Returns

The diseases for the given trial.

get_drugs_for_side_effect(*side_effect*, *, *client*)

Return the drugs for the given side effect.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **side_effect** (Tuple[str, str]) – The side effect to query.

Return type
Iterable[*Node*]

Returns
The drugs for the given side effect.

get_drugs_for_target(*target*, *, *client*)

Return the drugs targeting the given protein.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **target** (Tuple[str, str]) – The target to query.

Return type
Iterable[Agent]

Returns
The drugs targeting the given protein.

get_drugs_for_targets(*targets*, *, *client*)

Return the drugs targeting each of the given targets.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **targets** (Iterable[Tuple[str, str]]) – The targets to query.

Return type
Mapping[str, Iterable[Agent]]

Returns
A mapping of targets to the drugs targeting each of the given targets.

get_drugs_for_trial(*trial*, *, *client*)

Return the drugs for the given trial.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **trial** (Tuple[str, str]) – The trial to query.

Return type
Iterable[*Node*]

Returns
The drugs for the given trial.

get_edge_counter(*, *client*)

Get a count of each edge type.

Return type
Counter

get_evidences_for_mesh(*mesh_term*, *include_child_terms=True*, *, *client*)

Return the evidence objects for the given MESH term.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **mesh_term** (Tuple[str, str]) – The MESH ID to query.
- **include_child_terms** (bool) – If True, also match against the child MESH terms of the given MESH ID

Return type
Dict[int, List[Evidence]]

Returns
The evidence objects for the given MESH ID grouped into a dict by statement hash.

get_evidences_for_stmt_hash(*stmt_hash*, *, *client*, *limit=None*, *offset=0*, *remove_medscan=True*)

Return the matching evidence objects for the given statement hash.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **stmt_hash** (int) – The statement hash to query, accepts both string and integer.
- **limit** (Optional[int]) – The maximum number of results to return.
- **offset** (int) – The number of results to skip before returning the first result.
- **remove_medscan** (bool) – If True, remove the MedScan evidence from the results.

Return type
Iterable[Evidence]

Returns
The evidence objects for the given statement hash.

get_evidences_for_stmt_hashes(*stmt_hashes*, *, *client*, *limit=None*, *remove_medscan=True*)

Return the matching evidence objects for the given statement hashes.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **stmt_hashes** (Iterable[int]) – The statement hashes to query, accepts integers and strings.
- **limit** (Optional[str]) – The optional maximum number of evidences returned for each statement hash
- **remove_medscan** (bool) – If True, remove the MedScan evidence from the results.

Return type
Dict[int, List[Evidence]]

Returns
A mapping of stmt hash to a list of evidence objects for the given statement hashes.

get_genes_for_go_term(*go_term*, *include_indirect=False*, *, *client*)

Return the genes associated with the given GO term.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.

- **go_term** (Tuple[str, str]) – The GO term to query. Example: `("GO", "GO:0006915")`
- **include_indirect** (bool) – Should ontological children of the given GO term be queried as well? Defaults to False.

Return type
Iterable[*Node*]

Returns
The genes associated with the given GO term.

get_genes_for_pathway(*pathway*, *, *client*)

Return the genes for the given pathway.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **pathway** (Tuple[str, str]) – The pathway to query.

Return type
Iterable[*Node*]

Returns
The genes for the given pathway.

get_genes_in_tissue(*tissue*, *, *client*)

Return the genes in the given tissue.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **tissue** (Tuple[str, str]) – The tissue to query.

Return type
Iterable[*Node*]

Returns
The genes expressed in the given tissue.

get_go_terms_for_gene(*gene*, *include_indirect=False*, *, *client*)

Return the GO terms for the given gene.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **gene** (Tuple[str, str]) – The gene to query.
- **include_indirect** (bool) – If True, also return indirect GO terms.

Return type
Iterable[*Node*]

Returns
The GO terms for the given gene.

get_mesh_ids_for_pmid(*pmid_term*, *, *client*)

Return the MESH terms for the given PubMed ID.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **pmid_term** (Tuple[str, str]) – The PubMed ID to query.

Return type
`Iterable[Node]`

Returns
The MESH terms for the given PubMed ID.

`get_mutated_genes(cell_line, *, client)`

Return the list of genes that are mutated in a given cell line.

Parameters client:

The Neo4j client.

cell_line :
The cell line to query.

Return type
`List[Node]`

Returns
The list of genes that are mutated in the given cell line.

`get_node_counter(*, client)`

Get a count of each entity type.

Parameters
client (*Neo4jClient*) – The Neo4j client.

Return type
`Counter`

Returns

A Counter of the entity types.

> **Warning:** This code assumes all nodes only have one label, as in `label[0]`

`get_ontology_child_terms(term, *, client)`

Return the child terms of the given term.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **term** (`Tuple[str, str]`) – The term to query.

Return type
`Iterable[Node]`

Returns
The child terms of the given term.

`get_ontology_parent_terms(term, *, client)`

Return the parent terms of the given term.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **term** (`Tuple[str, str]`) – The term to query.

Return type
Iterable[*Node*]

Returns
The parent terms of the given term.

get_pathways_for_gene(*gene*, *, *client*)

Return the pathways for the given gene.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **gene** (Tuple[str, str]) – The gene to query.

Return type
Iterable[*Node*]

Returns
The pathways for the given gene.

get_pmids_for_mesh(*mesh_term*, *include_child_terms=True*, *, *client*)

Return the PubMed IDs for the given MESH term.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **mesh_term** (Tuple[str, str]) – The MESH term to query.
- **include_child_terms** (bool) – If True, also match against the child MESH terms of the given MESH term.

Return type
Iterable[*Node*]

Returns
The PubMed IDs for the given MESH term and, optionally, its child terms.

get_schema_graph(*, *client*)

Get a NetworkX graph reflecting the schema of the Neo4j graph.

Generate a PDF diagram (works with PNG and SVG too) with the following:

```
>>> from networkx.drawing.nx_agraph import to_agraph
>>> client = ...
>>> graph = get_schema_graph(client=client)
>>> to_agraph(graph).draw("~/Desktop/cogex_schema.pdf", prog="dot")
```

Return type
MultiDiGraph

get_shared_pathways_for_genes(*genes*, *, *client*)

Return the shared pathways for the given list of genes.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **genes** (Iterable[Tuple[str, str]]) – The list of genes to query.

Return type
Iterable[*Node*]

Returns
The pathways for the given gene.

`get_side_effects_for_drug`(*drug*, *, *client*)

Return the side effects for the given drug.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **drug** (Tuple[str, str]) – The drug to query.

Return type
Iterable[*Node*]

Returns
The side effects for the given drug.

`get_stmts_for_mesh`(*mesh_term*, *include_child_terms=True*, *, *client*, ***kwargs*)

Return the statements with evidence for the given MESH ID.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **mesh_term** (Tuple[str, str]) – The MESH ID to query.
- **include_child_terms** (bool) – If True, also match against the children of the given MESH ID.
- **kwargs** – Additional keyword arguments to forward to *get_stmts_for_stmt_hashes()*

Return type
Iterable[Statement]

Returns
The statements for the given MESH ID.

`get_stmts_for_paper`(*paper_term*, *, *client*, ***kwargs*)

Return the statements with evidence from the given PubMed ID.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **paper_term** (Tuple[str, str]) – The term to query. Can be a PubMed ID, PMC id, TRID, or DOI

Return type
List[Statement]

Returns
The statements for the given PubMed ID.

`get_stmts_for_pubmeds`(*pubmeds*, *, *client*, ***kwargs*)

Return the statements with evidence from the given PubMed ID.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **pubmeds** (List[Union[str, int]]) – The PMIDs to query

Return type
List[Statement]

Returns
The statements for the given PubMed identifiers.

Example

```
from indra_cogex.client.queries import get_stmts_for_pubmeds

pubmeds = [20861832, 19503834]
stmts = get_stmts_for_pubmeds(pubmeds)
```

get_stmts_for_stmt_hashes(*stmt_hashes*, *, *evidence_map=None*, *client*, *evidence_limit=None*, *return_evidence_counts=False*, *subject_prefix=None*, *object_prefix=None*)

Return the statements for the given statement hashes.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **stmt_hashes** (Iterable[int]) – The statement hashes to query.
- **evidence_map** (Optional[Dict[int, List[Evidence]]]) – Optionally provide a mapping of stmt hash to a list of evidence objects
- **evidence_limit** (Optional[int]) – An optional maximum number of evidences to return

Return type
Union[List[Statement], Tuple[List[Statement], Mapping[int, int]]]

Returns
The statements for the given statement hashes.

get_stmts_meta_for_stmt_hashes(*stmt_hashes*, *, *client*)

Return the metadata and statements for a given list of hashes

Parameters

- **stmt_hashes** (Iterable[int]) – The list of statement hashes to query.
- **client** (*Neo4jClient*) – The Neo4j client.

Return type
Iterable[*Relation*]

Returns
A dict of statements with their metadata

get_targets_for_drug(*drug*, *, *client*)

Return the proteins targeted by the given drug.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **drug** (Tuple[str, str]) – The drug to query.

Return type
Iterable[Agent]

Returns
The proteins targeted by the given drug.

get_targets_for_drugs(*drugs*, *, *client*)

Return the proteins targeted by each of the given drugs

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **drugs** (Iterable[Tuple[str, str]]) – A list of drugs to get the targets for.

Return type
Mapping[str, Iterable[Agent]]

Returns
A mapping from each drug to the proteins targeted by that drug.

get_tissues_for_gene(*gene*, *, *client*)

Return the tissues the gene is expressed in.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **gene** (Tuple[str, str]) – The gene to query.

Return type
Iterable[*Node*]

Returns
The tissues the gene is expressed in.

get_trials_for_disease(*disease*, *, *client*)

Return the trials for the given disease.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **disease** (Tuple[str, str]) – The disease to query.

Return type
Iterable[*Node*]

Returns
The trials for the given disease.

get_trials_for_drug(*drug*, *, *client*)

Return the trials for the given drug.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **drug** (Tuple[str, str]) – The drug to query.

Return type
Iterable[*Node*]

Returns
The trials for the given drug.

is_drug_target(*drug*, *target*, *, *client*)

Return True if the drug targets the given protein.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.

- **drug** (Tuple[str, str]) – The drug to query.
- **target** (Tuple[str, str]) – The target to query.

Return type
bool

Returns
True if the drug targets the given protein.

`is_gene_in_pathway`(*gene*, *pathway*, *, *client*)

Return True if the gene is in the given pathway.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **gene** (Tuple[str, str]) – The gene to query.
- **pathway** (Tuple[str, str]) – The pathway to query.

Return type
bool

Returns
True if the gene is in the given pathway.

`is_gene_in_tissue`(*gene*, *tissue*, *, *client*)

Return True if the gene is expressed in the given tissue.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **gene** (Tuple[str, str]) – The gene to query.
- **tissue** (Tuple[str, str]) – The tissue to query.

Return type
bool

Returns
True if the gene is expressed in the given tissue.

`is_gene_mutated`(*gene*, *cell_line*, *, *client*)

Return True if the gene is mutated in the given cell line.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **gene** (Tuple[str, str]) – The gene to query.
- **cell_line** (Tuple[str, str]) – The cell line to query.

Return type
bool

Returns
True if the gene is mutated in the given cell line.

`is_go_term_for_gene`(*gene*, *go_term*, *, *client*)

Return True if the given GO term is associated with the given gene.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **gene** (Tuple[str, str]) – The gene to query.
- **go_term** (Tuple[str, str]) – The GO term to query.

Return type
bool

Returns
True if the given GO term is associated with the given gene.

is_side_effect_for_drug(*drug*, *side_effect*, *, *client*)

Return True if the given side effect is associated with the given drug.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **drug** (Tuple[str, str]) – The drug to query.
- **side_effect** (Tuple[str, str]) – The side effect to query.

Return type
bool

Returns
True if the given side effect is associated with the given drug.

isa_or_partof(*term*, *parent*, *, *client*)

Return True if the given term is a child of the given parent.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **term** (Tuple[str, str]) – The term to query.
- **parent** (Tuple[str, str]) – The parent to query.

Return type
bool

Returns
True if the given term is a child term of the given parent.

2.3.4 Subnetwork Client (indra_cogex.client.subnetwork)

Queries that generate statement subnetworks.

indra_mediated_subnetwork(*nodes*, *, *client*)

Return the INDRA Statement subnetwork induced pairs of statements between the given nodes.

For example, if gene A and gene B are given as the query, find statements mediated by X such that A -> X -> B.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **nodes** (Iterable[Tuple[str, str]]) – The nodes to query.

Return type
List[Statement]

Returns
The subnetwork induced by the given nodes.

`indra_subnetwork`(*nodes*, ***, *client*)

Return the INDRA Statement subnetwork induced by the given nodes.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **nodes** (Iterable[Tuple[str, str]]) – The nodes to query.

Return type
List[Statement]

Returns
The subnetwork induced by the given nodes.

`indra_subnetwork_go`(*go_term*, ***, *client*, *include_indirect=False*, *mediated=False*, *upstream_controllers=False*, *downstream_targets=False*)

Return the INDRA Statement subnetwork induced by the given GO term.

Parameters

- **go_term** (Tuple[str, str]) – The GO term to query. Example: `("GO", "GO:0006915")`
- **client** (*Neo4jClient*) – The Neo4j client.
- **include_indirect** (bool) – Should ontological children of the given GO term be queried as well? Defaults to False.
- **mediated** (bool) – Should relations A->X->B be included for X not associated to the given GO term? Defaults to False.
- **upstream_controllers** (bool) – Should relations A<-X->B be included for upstream controller X not associated to the given GO term? Defaults to False.
- **downstream_targets** (bool) – Should relations A->X<-B be included for downstream target X not associated to the given GO term? Defaults to False.

Return type
List[Statement]

Returns
The INDRA statement subnetwork induced by GO term.

`indra_subnetwork_relations`(*nodes*, ***, *client*)

Return the subnetwork induced by the given nodes as a set of Relations.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **nodes** (Iterable[Tuple[str, str]]) – The nodes to query.

Return type
List[*Relation*]

Returns
The subnetwork induced by the given nodes represented as Relation objects.

`indra_subnetwork_tissue`(*nodes*, *tissue*, ***, *client*)

Return the INDRA Statement subnetwork induced by the given nodes and expressed in the given tissue.

Parameters

- **client** (*Neo4jClient*) – The Neo4j client.
- **nodes** (List[Tuple[str, str]]) – The nodes to query.
- **tissue** (Tuple[str, str]) – The tissue to query.

Return type
List[Statement]

Returns
The subnetwork induced by the given nodes and expressed in the given tissue.

2.4 Indexing of The Database

Once the database is built, it can be indexed using the following command:

```
python -m indra_cogex.indexing
```

or from the root directory of the repository:

```
./build_extra_indexes.sh
```

2.4.1 Indexing The Database (indra_cogex.indexing)

A collection of functions for indexing on the database.

index_evidence_on_stmt_hash(*client*, *exist_ok=False*)

Index all Evidence nodes on the stmt_hash property

Parameters

- **client** (*Neo4jClient*) – Neo4jClient instance to the graph database to be indexed
- **exist_ok** (bool) – If False, raise an exception if the index already exists. Default: False.

index_indra_rel_on_stmt_hash(*client*)

Index all indra_rel relationships on stmt_hash property

Parameters
client (*Neo4jClient*) – Neo4jClient instance to the graph database to be indexed

index_nodes_on_id(*client*, *exist_ok=False*)

Index all nodes on the id property

Parameters

- **client** (*Neo4jClient*) – Neo4jClient instance to the graph database to be indexed
- **exist_ok** (bool) – If False, raise an exception if the index already exists. Default: False.

2.4.2 Indexing CLI (indra_cogex.indexing.cli)

indra_cogex indexing

Build indexes on the database.

```
indra_cogex indexing [OPTIONS]
```

Options

--all
Build all indexes

--index-nodes
Index all nodes on the id property.

--index-evidence-nodes
Index the Evidence nodes on the stmt_hash property.

--index-indra-relations
Index the INDRA relations on the stmt_hash property.

--exist-ok
If set, skip already set indices silently, otherwise an exception is raised if attempting to set an index that already exists.

2.5 INDRA CoGEx Sources

2.5.1 Source CLI (indra_cogex.sources.cli)

2.5.2 INDRA CoGEx Sources Processor (indra_cogex.sources.processor)

Base classes for processors.

class Processor
A processor creates nodes and iterables to upload to Neo4j.

classmethod cli()
Run the CLI for this processor.

Return type
None

dump()
Dump the contents of this processor to CSV files ready for use in `neo4-admin import`.

Return type
Tuple[Path, List[*Node*], Path]

classmethod get_cli()
Get the CLI for this processor.

Return type
Command

abstract get_nodes()

Iterate over the nodes to upload.

Return type
Iterable[*Node*]

abstract get_relations()

Iterate over the relations to upload.

Return type
Iterable[*Relation*]

2.5.3 Bgee Processor (indra_cogex.sources.bgee)

Processor for Bgee.

class BgeeProcessor(*path=None*)

Bases: *Processor*

Processor for Bgee.

Initialize the Bgee processor.

Parameters
path (Union[None, Path, str]) – The path to the Bgee dump pickle. If none given, will look in the default location.

get_nodes()

Iterate over the nodes to upload.

Return type
Iterable[*Node*]

get_relations()

Iterate over the relations to upload.

Return type
Iterable[*Relation*]

2.5.4 cBioPortal Processor (indra_cogex.sources.cbioportal)

class CcleCnaProcessor(*path=None*)

Bases: *Processor*

get_nodes()

Iterate over the nodes to upload.

get_relations()

Iterate over the relations to upload.

class CcleDrugResponseProcessor(*path=None*)

Bases: *Processor*

get_nodes()

Iterate over the nodes to upload.

get_relations()

Iterate over the relations to upload.

class CcleMutationsProcessor(*path=None*)

Bases: *Processor*

get_nodes()

Iterate over the nodes to upload.

get_relations()

Iterate over the relations to upload.

2.5.5 CellMarker Processor (indra_cogex.sources.cellmarker)

Processor for the CellMarker database.

See also:

- Website: http://xteam.xbio.top/CellMarker/
- Publication: https://doi.org/10.1093/nar/gky900

class CellMarkerProcessor(*df=None*)

Processor for the CellMarker database.

Initialize the CellMarker processor.

get_nodes()

Get cell, tissue, and gene nodes.

get_relations()

Iterate over the relations to upload.

2.5.6 chembl Processor (indra_cogex.sources.chembl)

Processor for ChEMBL.

class ChemblIndicationsProcessor(*version=None*)

Bases: *Processor*

A processor for ChEMBL indications.

get_nodes()

Iterate over ChEMBL chemicals and indications

Return type
Iterable[*Node*]

get_relations()

Iterate over ChEMBL indication annotations.

Return type
Iterable[*Relation*]

MOLECULE_SQL = '\nSELECT DISTINCT\n MOLECULE_DICTIONARY.chembl_id,\n MOLECULE_DICTIONARY.pref_name\nFROM MOLECULE_DICTIONARY\nJOIN DRUG_INDICATION ON MOLECULE_DICTIONARY.molregno == DRUG_INDICATION.molregno\n'

SQL for ChEMBL to get molecules that have indications

SQL = '\nSELECT\n MOLECULE_DICTIONARY.chembl_id,\n DRUG_INDICATION.mesh_id,\n DRUG_INDICATION.max_phase_for_ind\nFROM MOLECULE_DICTIONARY\nJOIN DRUG_INDICATION ON MOLECULE_DICTIONARY.molregno == DRUG_INDICATION.molregno\n'

SQL for ChEMBL to get indications

2.5.7 clinicaltrials Processor (indra_cogex.sources.clinicaltrials)

This module implements input for ClinicalTrials.gov data

NOTE: ClinicalTrials.gov are working on a more modern API that is currently in Beta: https://beta.clinicaltrials.gov/data-about-studies/learn-about-api Once this API is released, we should switch to using it. The instructions for using the current/old API are below.

To obtain the custom download for ingest, do the following

1. Go to https://clinicaltrials.gov/api/gui/demo/simple_study_fields
2. Enter the following in the form:

expr= fields=NCTId,BriefTitle,Condition,ConditionMeshTerm,ConditionMeshId,InterventionName,InterventionType,InterventionMesh min_rnk=1 max_rnk=500000 # or any number larger than the current number of studies fmt=csv

3. Send Request

4. Enter the captcha characters into the text box and then press enter (make sure to use the enter key and not press any buttons).

5. The website will display "please wait… " for a couple of minutes, finally, the Save to file button will be active.

6. Click the Save to file button to download the response as a txt file.

7. Rename the txt file to clinical_trials.csv and then compress it as gzip clinical_trials.csv to get clinical_trials.csv.gz, then place this file into <pystow home>/indra/cogex/clinicaltrials/

class ClinicaltrialsProcessor(*path=None*)

Bases: *Processor*

get_nodes()

Iterate over the nodes to upload.

get_relations()

Iterate over the relations to upload.

2.5.8 goa Processor (indra_cogex.sources.goa)

Processor for the Gene Ontology Associations (GOA) database.

class GoaProcessor

Bases: *Processor*

Processor for the Gene Ontology Associations (GOA) database.

Initialize the GOA processor.

get_nodes()

Iterate over the nodes to upload.

get_relations()

Iterate over the relations to upload.

load_goa(*url*)

Get the Gene Ontology Annotations database as a dataframe.

Parameters
url (str) – The URL to the GOA database file.

Return type
DataFrame

Returns
The GOA database as a dataframe

2.5.9 INDRA DB Processor (indra_cogex.sources.indra_db)

Processor for the INDRA database.

class DbProcessor(*dir_path=None*)

Bases: *Processor*

Processor for the INDRA database.

Initialize the INDRA database processor.

Parameters
dir_path (Union[None, Path, str]) – The path to the directory containing unique and grounded statements as a *.tsv.gz file, source counts as a pickle file and belief scores as a pickle file.

get_nodes()

Iterate over the nodes to upload.

get_relations(*max_complex_members=3*)

Iterate over the relations to upload.

class EvidenceProcessor

Bases: *Processor*

Initialize the Evidence processor

get_nodes(*num_rows=None*)

Get INDRA Evidence and Publication nodes

Return type
Iterable[*Node*]

get_relations()

Iterate over the relations to upload.

exception StatementJSONDecodeError

Bases: Exception

get_ag_ns_id(*ag*)

Return a namespace, identifier tuple for a given agent.

Parameters
ag (Agent) – The agent to get the namespace and identifier for.

Return type
Tuple[str, str]

Returns
A namespace, identifier tuple.

2.5.10 INDRA Ontology Processor (`indra_cogex.sources.indra_ontology`)

Processor for the INDRA ontology.

class OntologyProcessor(*ontology=None*)

Bases: *Processor*

Processor for the INDRA ontology.

Initialize the INDRA ontology processor.

Parameters
ontology (Optional[IndraOntology]) – An instance of an INDRA ontology. If none, loads the INDRA bio_ontology.

get_nodes()

Iterate over the nodes to upload.

get_relations()

Iterate over the relations to upload.

2.5.11 InterPro Processor (`indra_cogex.sources.interpro`)

Processor for the InterPro database.

This was added in https://github.com/bgyori/indra_cogex/pull/125.

See also:

https://ftp.ebi.ac.uk/pub/databases/interpro/current_release/

class InterproProcessor(*force=False*)

Processor for Interpro.

Initialize the InterPro processor.

get_nodes()

Iterate over the nodes to upload.

get_relations()

Iterate over the relations to upload.

2.5.12 Odinson Processor (`indra_cogex.sources.odinson`)

The Odinson Processor

Odinson Client (indra_cogex.sources.odinson.client)

The Odinson client

Odinson Document (indra_cogex.sources.odinson.document)

The Odinson document API

Odinson Grammar (indra_cogex.sources.odinson.grammars)

The Odinson grammar API

2.5.13 Pathways Processor (indra_cogex.sources.pathways)

2.5.14 PubMed Processor (indra_cogex.sources.pubmed)

2.5.15 Sider Processor (indra_cogex.sources.sider)

2.6 INDRA CoGEx Representation (indra_cogex.representation)

This documentation goes over helper functions and the python objects that represent Neo4j Nodes and Relations.

Representations for nodes and relations to upload to Neo4j.

class Node(*db_ns*, *db_id*, *labels*, *data=None*)

Representation for a node.

Initialize the node.

Parameters

- **db_ns** (str) – The namespace associated with the node. Uses the INDRA standard.
- **db_id** (str) – The identifier within the namespace associated with the node. Uses the INDRA standard.
- **labels** (Collection[str]) – A collection of labels for the node.
- **data** (Optional[Mapping[str, Any]]) – An optional data dictionary associated with the node.

grounding()

Get the grounded namespace and identifier for this node as a tuple

Return type
Tuple[str, str]

Returns
A tuple of the namespace and identifier for the node.

classmethod standardized(*, *db_ns*, *db_id*, *name=None*, *labels*)

Initialize the node, but first standardize the prefix/identifier/name.

Parameters

- **db_ns** (str) – The namespace associated with the node.

- **db_id** (str) – The identifier within the namespace associated with the node.
- **name** (Optional[str]) – An optional name for the node.
- **labels** (Collection[str]) – A collection of labels for the node.

Return type
Node

Returns
A node with standardized prefix/identifier/name.

`to_json()`

Serialize the node to JSON.

Return type
Dict[str, Union[Collection[str], Dict[str, Any]]]

Returns
A JSON representation of the node.

`class Relation(source_ns, source_id, target_ns, target_id, rel_type, data=None)`

Representation for a relation.

Initialize the relation.

Parameters

- **source_ns** (str) – The namespace associated with the source node.
- **source_id** (str) – The identifier within the namespace associated with the source node.
- **target_ns** (str) – The namespace associated with the target node.
- **target_id** (str) – The identifier within the namespace associated with the target node.
- **rel_type** (str) – The type of relation.
- **data** (Optional[Mapping[str, Any]]) – An optional data dictionary associated with the relation.

`to_json()`

Serialize the relation to JSON format.

Return type
Dict[str, Union[Mapping[str, Any], Dict]]

Returns
A JSON representation of the relation.

`indra_stmts_from_relations(rels)`

Convert a list of relations to INDRA Statements.

Any relations that aren't representing an INDRA Statement are skipped.

Parameters
rels (Iterable[*Relation*]) – A list of Relations.

Return type
List[Statement]

Returns
A list of INDRA Statements.

norm_id(*db_ns*, *db_id*)

Normalize an identifier.

Parameters

- **db_ns** – The namespace of the identifier.
- **db_id** – The identifier.

Return type

str

Returns

The normalized identifier.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

indra_cogex.apps.gla, 3
indra_cogex.assembly, 3

c

indra_cogex.client.enrichment.discrete, 8
indra_cogex.client.enrichment.signed, 10
indra_cogex.client.enrichment.utils, 11
indra_cogex.client.neo4j_client, 13
indra_cogex.client.queries, 21
indra_cogex.client.subnetwork, 31

i

indra_cogex.indexing, 33

r

indra_cogex.representation, 40

s

indra_cogex.sources.bgee, 35
indra_cogex.sources.cbioportal, 35
indra_cogex.sources.cellmarker, 36
indra_cogex.sources.chembl, 36
indra_cogex.sources.clinicaltrials, 37
indra_cogex.sources.goa, 37
indra_cogex.sources.indra_db, 38
indra_cogex.sources.indra_ontology, 39
indra_cogex.sources.interpro, 39
indra_cogex.sources.processor, 34

INDEX

Symbols

--all
    indra_cogex-indexing command line option, 34
--exist-ok
    indra_cogex-indexing command line option, 34
--index-evidence-nodes
    indra_cogex-indexing command line option, 34
--index-indra-relations
    indra_cogex-indexing command line option, 34
--index-nodes
    indra_cogex-indexing command line option, 34

A

add_node() (*Neo4jClient method*), 13
add_nodes() (*Neo4jClient method*), 13
add_nodes() (*NodeAssembler method*), 3
add_relations() (*Neo4jClient method*), 13
assemble_nodes() (*NodeAssembler method*), 3
autoclient() (*in module indra_cogex.client.neo4j_client*), 20

B

BgeeProcessor (*class in indra_cogex.sources.bgee*), 35

C

CcleCnaProcessor (*class in indra_cogex.sources.cbioportal*), 35
CcleDrugResponseProcessor (*class in indra_cogex.sources.cbioportal*), 35
CcleMutationsProcessor (*class in indra_cogex.sources.cbioportal*), 36
CellMarkerProcessor (*class in indra_cogex.sources.cellmarker*), 36
ChemblIndicationsProcessor (*class in indra_cogex.sources.chembl*), 36
cli() (*Processor class method*), 34
ClinicaltrialsProcessor (*class in indra_cogex.sources.clinicaltrials*), 37
close_session() (*Neo4jClient method*), 13
collect_gene_sets() (*in module indra_cogex.client.enrichment.utils*), 11
create_nodes() (*Neo4jClient method*), 14
create_single_property_node_index() (*Neo4jClient method*), 14
create_single_property_relationship_index() (*Neo4jClient method*), 14
create_tx() (*Neo4jClient method*), 14

D

DbProcessor (*class in indra_cogex.sources.indra_db*), 38
delete_all() (*Neo4jClient method*), 14
dump() (*Processor method*), 34

E

EvidenceProcessor (*class in indra_cogex.sources.indra_db*), 38

G

get_ag_ns_id() (*in module indra_cogex.sources.indra_db*), 38
get_aggregate_node() (*NodeAssembler method*), 3
get_all_relations() (*Neo4jClient method*), 14
get_cli() (*Processor class method*), 34
get_common_sources() (*Neo4jClient method*), 15
get_common_targets() (*Neo4jClient method*), 15
get_diseases_for_trial() (*in module indra_cogex.client.queries*), 21
get_drugs_for_side_effect() (*in module indra_cogex.client.queries*), 21
get_drugs_for_target() (*in module indra_cogex.client.queries*), 22
get_drugs_for_targets() (*in module indra_cogex.client.queries*), 22
get_drugs_for_trial() (*in module indra_cogex.client.queries*), 22
get_edge_counter() (*in module indra_cogex.client.queries*), 22

get_entity_to_regulators() (*in module indra_cogex.client.enrichment.utils*), 11
get_entity_to_targets() (*in module indra_cogex.client.enrichment.utils*), 12
get_evidences_for_mesh() (*in module indra_cogex.client.queries*), 22
get_evidences_for_stmt_hash() (*in module indra_cogex.client.queries*), 23
get_evidences_for_stmt_hashes() (*in module indra_cogex.client.queries*), 23
get_genes_for_go_term() (*in module indra_cogex.client.queries*), 23
get_genes_for_pathway() (*in module indra_cogex.client.queries*), 24
get_genes_in_tissue() (*in module indra_cogex.client.queries*), 24
get_go() (*in module indra_cogex.client.enrichment.utils*), 12
get_go_terms_for_gene() (*in module indra_cogex.client.queries*), 24
get_mesh_ids_for_pmid() (*in module indra_cogex.client.queries*), 24
get_mutated_genes() (*in module indra_cogex.client.queries*), 25
get_node_counter() (*in module indra_cogex.client.queries*), 25
get_nodes() (*BgeeProcessor method*), 35
get_nodes() (*CcleCnaProcessor method*), 35
get_nodes() (*CcleDrugResponseProcessor method*), 35
get_nodes() (*CcleMutationsProcessor method*), 36
get_nodes() (*CellMarkerProcessor method*), 36
get_nodes() (*ChemblIndicationsProcessor method*), 36
get_nodes() (*ClinicaltrialsProcessor method*), 37
get_nodes() (*DbProcessor method*), 38
get_nodes() (*EvidenceProcessor method*), 38
get_nodes() (*GoaProcessor method*), 37
get_nodes() (*InterproProcessor method*), 39
get_nodes() (*OntologyProcessor method*), 39
get_nodes() (*Processor method*), 34
get_ontology_child_terms() (*in module indra_cogex.client.queries*), 25
get_ontology_parent_terms() (*in module indra_cogex.client.queries*), 25
get_pathways_for_gene() (*in module indra_cogex.client.queries*), 26
get_phenotype_gene_sets() (*in module indra_cogex.client.enrichment.utils*), 12
get_pmids_for_mesh() (*in module indra_cogex.client.queries*), 26
get_predecessors() (*Neo4jClient method*), 15
get_property_from_relations() (*Neo4jClient static method*), 15
get_reactome() (*in module indra_cogex.client.enrichment.utils*), 12
get_relations() (*BgeeProcessor method*), 35
get_relations() (*CcleCnaProcessor method*), 35
get_relations() (*CcleDrugResponseProcessor method*), 35
get_relations() (*CcleMutationsProcessor method*), 36
get_relations() (*CellMarkerProcessor method*), 36
get_relations() (*ChemblIndicationsProcessor method*), 36
get_relations() (*ClinicaltrialsProcessor method*), 37
get_relations() (*DbProcessor method*), 38
get_relations() (*EvidenceProcessor method*), 38
get_relations() (*GoaProcessor method*), 37
get_relations() (*InterproProcessor method*), 39
get_relations() (*Neo4jClient method*), 16
get_relations() (*OntologyProcessor method*), 39
get_relations() (*Processor method*), 35
get_schema_graph() (*in module indra_cogex.client.queries*), 26
get_session() (*Neo4jClient method*), 16
get_shared_pathways_for_genes() (*in module indra_cogex.client.queries*), 26
get_side_effects_for_drug() (*in module indra_cogex.client.queries*), 27
get_source_agents() (*Neo4jClient method*), 16
get_source_relations() (*Neo4jClient method*), 17
get_sources() (*Neo4jClient method*), 17
get_stmts_for_mesh() (*in module indra_cogex.client.queries*), 27
get_stmts_for_paper() (*in module indra_cogex.client.queries*), 27
get_stmts_for_pubmeds() (*in module indra_cogex.client.queries*), 27
get_stmts_for_stmt_hashes() (*in module indra_cogex.client.queries*), 28
get_stmts_meta_for_stmt_hashes() (*in module indra_cogex.client.queries*), 28
get_successors() (*Neo4jClient method*), 17
get_target_agents() (*Neo4jClient method*), 17
get_target_relations() (*Neo4jClient method*), 18
get_targets() (*Neo4jClient method*), 18
get_targets_for_drug() (*in module indra_cogex.client.queries*), 28
get_targets_for_drugs() (*in module indra_cogex.client.queries*), 28
get_tissues_for_gene() (*in module indra_cogex.client.queries*), 29
get_trials_for_disease() (*in module indra_cogex.client.queries*), 29
get_trials_for_drug() (*in module indra_cogex.client.queries*), 29
get_wikipathways() (*in module indra_cogex.client.enrichment.utils*), 13

go_ora() (*in module indra_cogex.client.enrichment.discrete*), 8
GoaProcessor (*class in indra_cogex.sources.goa*), 37
grounding() (*Node method*), 40

H

has_relation() (*Neo4jClient method*), 18

I

index_evidence_on_stmt_hash() (*in module indra_cogex.indexing*), 33
index_indra_rel_on_stmt_hash() (*in module indra_cogex.indexing*), 33
index_nodes_on_id() (*in module indra_cogex.indexing*), 33
indra_cogex.apps.gla
    module, 3
indra_cogex.assembly
    module, 3
indra_cogex.client.enrichment.discrete
    module, 8
indra_cogex.client.enrichment.signed
    module, 10
indra_cogex.client.enrichment.utils
    module, 11
indra_cogex.client.neo4j_client
    module, 13
indra_cogex.client.queries
    module, 21
indra_cogex.client.subnetwork
    module, 31
indra_cogex.indexing
    module, 33
indra_cogex.representation
    module, 40
indra_cogex.sources.bgee
    module, 35
indra_cogex.sources.cbioportal
    module, 35
indra_cogex.sources.cellmarker
    module, 36
indra_cogex.sources.chembl
    module, 36
indra_cogex.sources.clinicaltrials
    module, 37
indra_cogex.sources.goa
    module, 37
indra_cogex.sources.indra_db
    module, 38
indra_cogex.sources.indra_ontology
    module, 39
indra_cogex.sources.interpro
    module, 39
indra_cogex.sources.processor
    module, 34
indra_cogex-indexing command line option
    --all, 34
    --exist-ok, 34
    --index-evidence-nodes, 34
    --index-indra-relations, 34
    --index-nodes, 34
indra_downstream_ora() (*in module indra_cogex.client.enrichment.discrete*), 8
indra_mediated_subnetwork() (*in module indra_cogex.client.subnetwork*), 31
indra_stmts_from_relations() (*in module indra_cogex.representation*), 41
indra_subnetwork() (*in module indra_cogex.client.subnetwork*), 32
indra_subnetwork_go() (*in module indra_cogex.client.subnetwork*), 32
indra_subnetwork_relations() (*in module indra_cogex.client.subnetwork*), 32
indra_subnetwork_tissue() (*in module indra_cogex.client.subnetwork*), 32
indra_upstream_ora() (*in module indra_cogex.client.enrichment.discrete*), 9
InterproProcessor (*class in indra_cogex.sources.interpro*), 39
is_drug_target() (*in module indra_cogex.client.queries*), 29
is_gene_in_pathway() (*in module indra_cogex.client.queries*), 30
is_gene_in_tissue() (*in module indra_cogex.client.queries*), 30
is_gene_mutated() (*in module indra_cogex.client.queries*), 30
is_go_term_for_gene() (*in module indra_cogex.client.queries*), 30
is_side_effect_for_drug() (*in module indra_cogex.client.queries*), 31
isa_or_partof() (*in module indra_cogex.client.queries*), 31

L

load_goa() (*in module indra_cogex.sources.goa*), 37

M

main() (*in module indra_cogex.client.enrichment.signed*), 10
module
    indra_cogex.apps.gla, 3
    indra_cogex.assembly, 3
    indra_cogex.client.enrichment.discrete, 8
    indra_cogex.client.enrichment.signed, 10
    indra_cogex.client.enrichment.utils, 11
    indra_cogex.client.neo4j_client, 13
    indra_cogex.client.queries, 21

indra_cogex.client.subnetwork, 31
indra_cogex.indexing, 33
indra_cogex.representation, 40
indra_cogex.sources.bgee, 35
indra_cogex.sources.cbioportal, 35
indra_cogex.sources.cellmarker, 36
indra_cogex.sources.chembl, 36
indra_cogex.sources.clinicaltrials, 37
indra_cogex.sources.goa, 37
indra_cogex.sources.indra_db, 38
indra_cogex.sources.indra_ontology, 39
indra_cogex.sources.interpro, 39
indra_cogex.sources.processor, 34
MOLECULE_SQL (*in module indra_cogex.sources.chembl*), 36

N

neo4j_to_node() (*Neo4jClient static method*), 19
neo4j_to_relation() (*Neo4jClient class method*), 19
neo4j_to_relations() (*Neo4jClient static method*), 19
Neo4jClient (*class in indra_cogex.client.neo4j_client*), 13
Node (*class in indra_cogex.representation*), 40
node_to_agent() (*Neo4jClient static method*), 19
NodeAssembler (*class in indra_cogex.assembly*), 3
norm_id() (*in module indra_cogex.representation*), 41

O

OntologyProcessor (*class in indra_cogex.sources.indra_ontology*), 39

P

phenotype_ora() (*in module indra_cogex.client.enrichment.discrete*), 9
Processor (*class in indra_cogex.sources.processor*), 34

Q

query_dict() (*Neo4jClient method*), 19
query_dict_value_json() (*Neo4jClient method*), 20
query_nodes() (*Neo4jClient method*), 20
query_relations() (*Neo4jClient method*), 20
query_tx() (*Neo4jClient method*), 20

R

reactome_ora() (*in module indra_cogex.client.enrichment.discrete*), 9
Relation (*class in indra_cogex.representation*), 41
reverse_causal_reasoning() (*in module indra_cogex.client.enrichment.signed*), 10

S

session (*Neo4jClient attribute*), 20
SQL (*in module indra_cogex.sources.chembl*), 36
standardized() (*Node class method*), 40
StatementJSONDecodeError, 38

T

to_json() (*Node method*), 41
to_json() (*Relation method*), 41

W

wikipathways_ora() (*in module indra_cogex.client.enrichment.discrete*), 10