indra_world

INDRA team

Feb 28, 2023

CONTENTS

CHAPTER

ONE

INDRA WORLD

INDRA World is a generalization of INDRA (originally developed for biology) for automatically collecting, assembling, and modeling the web of causal relations that drive interconnected events in regional and global systems.

INDRA World interfaces with four machine reading systems which extract concepts, events, and causal relations from text (typically reports from governmental and non-governmental organizations, news stories, and scientific publications). The extractions are processed into a standardized Statement representation and then processed (filtered, normalized, etc.).

INDRA World makes use of the general INDRA assembly logic to find relationships between statements, including matching, contradiction, and refinement (i.e., one statement is a more general or more specific version of the other). It then calculates a belief score which is based on all available evidence directly or indirectly supporting a given statement.

This repository also implements a database and service architecture to run INDRA World as a service that integrates with other systems and supports managing project-specific statement sets and incremental assembly with new reader outputs.

1.1 Installation

INDRA World can be installed directly from Github as

```
$ pip install git+https://github.com/indralab/indra_world.git
```

Additionally, INDRA World can be run via Docker with public images available through Dockerhub. For more information, see https://github.com/indralab/indra_world/tree/master/docker.

1.2 Documentation

Detailed documentation is available at: https://indra-world.readthedocs.io/en/latest/.

1.3 Command line interface

The INDRA World command line interface allows running assembly using externally supplied arguments and configurations files. This serves as an alternative to using the Python API.

```
usage: indra_world [-h]
                   (--reader-output-files READER_OUTPUT_FILES |
                    --reader-output-dart-query READER_OUTPUT_DART_QUERY |
                    --reader-output-dart-keys READER_OUTPUT_DART_KEYS)
                   [--assembly-config ASSEMBLY_CONFIG]
                   (--ontology-path ONTOLOGY_PATH |
                    --ontology-id ONTOLOGY_ID)
                    --output-folder OUTPUT_FOLDER
                   [--causemos-metadata CAUSEMOS_METADATA]

INDRA World assembly CLI

optional arguments:
  -h, --help            show this help message and exit

Input options:
  --reader-output-files READER_OUTPUT_FILES
                        Path to a JSON file whose keys are reading system
                        identifiers and whose values are lists of file paths to
                        outputs from the given system to be used in assembly.
  --reader-output-dart-query READER_OUTPUT_DART_QUERY
                        Path to a JSON file that specifies query parameters for
                        reader output records in DART.  Only applicable if DART
                        is being used.
  --reader-output-dart-keys READER_OUTPUT_DART_KEYS
                        Path to a text file where each line is a DART storage
                        key corresponding to a reader output record. Only
                        applicable if DART is being used.

Assembly options:
  --assembly-config ASSEMBLY_CONFIG
                        Path to a JSON file that specifies the INDRA assembly
                        pipeline. If not provided, the default assembly
                        pipeline will be used.
  --ontology-path ONTOLOGY_PATH
                        Path to an ontology YAML file.
  --ontology-id ONTOLOGY_ID
                        The identifier of an ontology registered in DART. Only
                        applicable if DART is being used.

Output options:
  --output-folder OUTPUT_FOLDER
                        The path to a folder to which the INDRA output will be
                        written.
  --causemos-metadata CAUSEMOS_METADATA
                        Path to a JSON file that provides metadata to be used
                        for a Causemos-compatible dump of INDRA output (which
                        consists of multiple files). THe --output-path
```

(continues on next page)

(continued from previous page)

```
                option must also be used along with this option.
```

The CLI can also be invoked through Docker. In this case, all CLI arguments that are paths, need to be made visible to Docker. To do this, the -v flag can be used to mount a host folder (in the command below, [local-path-to-mount] into the container on a given path. All CLI path arguments then need to be given with respect to the path as seen in the container. Furthermore, if any of the files referred to in CLI arguments themselves list file paths (e.g., the value of –reader-output-files), those paths need to be relative to the Docker container's mounted volume as well.

```
docker run -v [local-path-to-mount]:/data --entrypoint indra_world indralab/indra_
↪world:latest [cli-arguments]
```

CHAPTER

TWO

DOCKERIZED INDRA WORLD SERVICE

This folder contains files to run the INDRA World service through Docker containers. It also provides files to build them locally in case customizations are needed.

2.1 Running the integrated service

A docker-compose file defines how the service image and DB image need to be run. The docker-compose file refers to two images (indralab/indra_world and indralab/indra_world_db), both available publicly on Dockerhub. This means that they are automatically pulled when running `docker-compose up` unless they are already available locally.

To launch the service, run

```
docker-compose up -d
```

where the optional `-d` flag runs the containers in the background.

There are two files that need to be created containing environment variables for each container with the following names and content:

`indra_world.env`

```
INDRA_WM_SERVICE_DB=postgresql://postgres:mysecretpassword@db:5432
DART_WM_URL=<DART URL>
DART_WM_USERNAME=<DART username>
DART_WM_PASSWORD=<DART password>
AWS_ACCESS_KEY_ID=<AWS account key ID, necessary if assembled outputs need to be dumped␣
↪to S3 for CauseMos>
AWS_SECRET_ACCESS_KEY=<AWS account secret key, necessary if assembled outputs need to be␣
↪dumped to S3 for CauseMos>
AWS_REGION=us-east-1
INDRA_WORLD_ONTOLOGY_URL=<GitHub URL to ontology being used, only necessary if DART is␣
↪not used.>
LOCAL_DEPLOYMENT=1
```

Above, `LOCAL_DEPLOYMENT` should only be set if the service is intended to be run on and accessed from localhost. This enables the assembly dashboard app at `http://localhost:8001/dashboard` which can write assembled corpus output to the container's disk (this can either be mounted to correspond to a host folder or files can be copied to the host using docker cp).

`indra_world_db.env`

```
POSTGRES_PASSWORD=mysecretpassword
PGDATA=/var/lib/postgresql/pgdata
```

Note that if necessary, the default `POSTGRES_PASSWORD=mysecretpassword` setting can be changed using standard `psql` commands in the `indra_world_db` container and then committed to an image.

2.2 Building the Docker images locally

As described above, the two necessary Docker images are available on Dockerhub, therefore the following steps are only necessary if local changes to the images (beyond what can be controlled through environmental variables) are needed.

2.2.1 Building the INDRA World service image

To build the `indra_world` Docker image, run

```
docker build --tag indra_world:latest .
```

2.2.2 Initializing the INDRA World DB image

To create the `indra_world_db` Docker image from scratch, run

```
./initialize_db_image.sh
```

Note that this requires Python dependencies needed to run INDRA World to be available in the local environment.

2.3 Using the public INDRA World API

The API is deployed and documented at wm.indra.bio.

2.4 Cloud-based CauseMos integration via S3

Access to the INDRA-assembled corpora requires credentials to the shared World Modelers S3 bucket "world-modelers". Each INDRA-assembled corpus is available within this bucket, under the "indra_models" key base. Each corpus is identified by a string identifier.

2.4.1 The corpus index

The list of corpora can be obtained either using S3's list objects function or by reading the index.csv file which is maintained by INDRA. This index is a comma separated values text file which contains one row for each corpus. Each row's first element is a corpus identifier, and the second element is the UTC date-time at which the corpus was uploaded to S3. An example row in this file looks as follows

```
test1_newlines,2020-05-08-22-34-29
```

where test1_newlines is the corpus identifier and 2020-05-08-22-34-29 is the upload date-time.

2.4.2 Structure of each corpus

Within the world-modelers bucket, under the indra_models key base, files for each corpus are organized under a subkey equivalent to the corpus identifier, for instance, all the files for the test1_newlines corpus are under the indra_models/test1_newlines/ key base. The list of files for each corpus are as follows

- *statements.json*: a JSON dump of assembled INDRA Statements. Each statement's JSON representation is on a separate line in this file. This is the main file that CauseMos needs to ingest for UI interaction.
- *metadata.json*: a JSON file containing key-value pairs that describe the corpus. The standard keys in this file are as follows:
  - *corpus_id*: the ID of the corpus (redundant with the corresponding entry in the index).
  - *description*: a human-readable description of how the corpus was obtained.
  - *display_name*: a human-readable display name for the corpus.
  - *readers*: a list of the names of the reading systems from which statements were obtained in the corpus.
  - *assembly*: a dictionary identifying attributes of the assembly process with the following keys:
    - *level*: the level of resolution used to assemble the corpus (e.g., "location_and_time").
    - *grounding_threshold*: the threshold (if any) which was used to filter statements by grounding score (e.g., 0.7)
  - *num_statements*: the number of assembled INDRA Statements in the corpus (i.e., statements.json).
  - *num_documents*: the number of documents that were read by readers to produce the statements that were assembled.
  - *tenant*: if DART is used, a corpus is typically associated with a tenant (i.e., a user or an institution); this field provides the tenant ID.

Note that any of these keys may be missing if unavailable, for instance, in the case of old uploads.

CHAPTER

THREE

INDRA WORLD MODULES REFERENCE

3.1 Knowledge Sources (indra_world.sources)

3.1.1 Eidos (indra_world.sources.eidos)

API (indra_world.sources.eidos.api)

indra_world.sources.eidos.api.**process_json**(*json_dict*, *grounding_ns=None*, *extract_filter=None*, *grounding_mode=None*)

Return an EidosProcessor by processing a Eidos JSON-LD dict.

Parameters

- **json_dict** (*dict*) – The JSON-LD dict to be processed.
- **grounding_ns** (*Optional[list]*) – A list of name spaces for which INDRA should represent groundings, when given. If not specified or None, all grounding name spaces are propagated. If an empty list, no groundings are propagated. Example: ['UN', 'WM'], Default: None
- **extract_filter** (*Optional[list]*) – A list of relation types to extract. Valid values in the list are 'influence', 'association', 'event'. If not given, all relation types are extracted. This argument can be used if, for instance, only Influence statements are of interest. Default: None
- **grounding_mode** (*Optional[str]*) – Selects whether 'flat' or 'compositional' groundings should be extracted. Default: 'flat'.

Returns

ep – A EidosProcessor containing the extracted INDRA Statements in its statements attribute.

Return type

EidosProcessor

indra_world.sources.eidos.api.**process_json_file**(*file_name*, *grounding_ns=None*, *extract_filter=None*, *grounding_mode='compositional'*)

Return an EidosProcessor by processing the given Eidos JSON-LD file.

This function is useful if the output from Eidos is saved as a file and needs to be processed.

Parameters

- **file_name** (*str*) – The name of the JSON-LD file to be processed.
- **grounding_ns** (*Optional[list]*) – A list of name spaces for which INDRA should represent groundings, when given. If not specified or None, all grounding name spaces are

propagated. If an empty list, no groundings are propagated. Example: ['UN', 'WM'], Default: None

- **extract_filter** (*Optional[list]*) – A list of relation types to extract. Valid values in the list are 'influence', 'association', 'event'. If not given, all relation types are extracted. This argument can be used if, for instance, only Influence statements are of interest. Default: None
- **grounding_mode** (*Optional[str]*) – Selects whether 'flat' or 'compositional' groundings should be extracted. Default: 'flat'.

Returns
ep – A EidosProcessor containing the extracted INDRA Statements in its statements attribute.

Return type
EidosProcessor

indra_world.sources.eidos.api.**process_json_str**(*json_str*, *grounding_ns=None*, *extract_filter=None*, *grounding_mode='compositional'*)

Return an EidosProcessor by processing the Eidos JSON-LD string.

Parameters

- **json_str** (*str*) – The JSON-LD string to be processed.
- **grounding_ns** (*Optional[list]*) – A list of name spaces for which INDRA should represent groundings, when given. If not specified or None, all grounding name spaces are propagated. If an empty list, no groundings are propagated. Example: ['UN', 'WM'], Default: None
- **extract_filter** (*Optional[list]*) – A list of relation types to extract. Valid values in the list are 'influence', 'association', 'event'. If not given, all relation types are extracted. This argument can be used if, for instance, only Influence statements are of interest. Default: None
- **grounding_mode** (*Optional[str]*) – Selects whether 'flat' or 'compositional' groundings should be extracted. Default: 'flat'.

Returns
ep – A EidosProcessor containing the extracted INDRA Statements in its statements attribute.

Return type
EidosProcessor

indra_world.sources.eidos.api.**process_text**(*text*, *save_json='eidos_output.json'*, *webservice=None*, *grounding_ns=None*, *extract_filter=None*, *grounding_mode='compositional'*)

Return an EidosProcessor by processing the given text.

This constructs a reader object via Java and extracts mentions from the text. It then serializes the mentions into JSON and processes the result with process_json.

Parameters

- **text** (*str*) – The text to be processed.
- **save_json** (*Optional[str]*) – The name of a file in which to dump the JSON output of Eidos.
- **webservice** (*Optional[str]*) – An Eidos reader web service URL to send the request to. If None, the reading is assumed to be done with the Eidos JAR rather than via a web service. Default: None

- **grounding_ns** (*Optional[list]*) – A list of name spaces for which INDRA should represent groundings, when given. If not specified or None, all grounding name spaces are propagated. If an empty list, no groundings are propagated. Example: ['UN', 'WM'], Default: None
- **extract_filter** (*Optional[list]*) – A list of relation types to extract. Valid values in the list are 'influence', 'association', 'event'. If not given, all relation types are extracted. This argument can be used if, for instance, only Influence statements are of interest. Default: None
- **grounding_mode** (*Optional[str]*) – Selects whether 'flat' or 'compositional' groundings should be extracted. Default: 'flat'.

Returns
ep – An EidosProcessor containing the extracted INDRA Statements in its statements attribute.

Return type
EidosProcessor

indra_world.sources.eidos.api.**reground_texts**(*texts*, *ont_yml*, *webservice=None*, *topk=10*, *filter=True*, *is_canonicalized=True*)

Return grounding for concept texts given an ontology.

Parameters

- **texts** (*list[str]*) – A list of concept texts to ground.
- **ont_yml** (*str*) – A serialized YAML string representing the ontology.
- **webservice** (*Optional[str]*) – The address where the Eidos web service is running, e.g., http://localhost:9000. If None, a local Eidos JAR is invoked via pyjnius. Default: None
- **topk** (*Optional[int]*) – The number of top scoring groundings to return. Default: 10
- **is_canonicalized** (*Optional[bool]*) – If True, the texts are assumed to be canonicalized. If False, Eidos will canonicalize the texts which yields much better groundings but is slower. Default: False
- **filter** (*Optional[bool]*) – If True, Eidos filters the ontology to remove determiners from examples and other similar operations. Should typically be set to True. Default: True

Returns
A list of the top k scored groundings for each text in the list.

Return type
list[list]

Client (indra_world.sources.eidos.client)

indra_world.sources.eidos.client.**grounding_dict_to_list**(*groundings*)

Transform the webservice response into a flat list.

indra_world.sources.eidos.client.**reground_texts**(*texts*, *ont_yml*, *webservice*, *topk=10*, *is_canonicalized=False*, *filter=True*, *cache_path=None*)

Ground concept texts given an ontology with an Eidos web service.

Parameters

- **texts** (*list[str]*) – A list of concept texts to ground.

- **ont_yml** (*str*) – A serialized YAML string representing the ontology.
- **webservice** (*str*) – The address where the Eidos web service is running, e.g., http://localhost:9000.
- **topk** (*Optional[int]*) – The number of top scoring groundings to return. Default: 10
- **is_canonicalized** (*Optional[bool]*) – If True, the texts are assumed to be canonicalized. If False, Eidos will canonicalize the texts which yields much better groundings but is slower. Default: False
- **filter** (*Optional[bool]*) – If True, Eidos filters the ontology to remove determiners from examples and other similar operations. Should typically be set to True. Default: True

Returns
A JSON dict of the results from the Eidos webservice.

Return type
dict

Migration Table Processor (indra_world.sources.eidos.migration_table_processor)

Processor (indra_world.sources.eidos.processor)

class indra_world.sources.eidos.processor.**EidosProcessorCompositional**(*json_dict*, *grounding_ns*)

Bases: *EidosWorldProcessor*

get_groundings(*entity*)

Return groundings as db_refs for an entity.

class indra_world.sources.eidos.processor.**EidosWorldProcessor**(*json_dict*, *grounding_ns*)

Bases: EidosProcessor

geo_context_from_ref(*ref*)

Return a ref context object given a location reference entry.

get_groundings(*entity*)

Return groundings as db_refs for an entity.

time_context_from_ref(*timex*)

Return a time context object given a timex reference entry.

indra_world.sources.eidos.processor.**ref_context_from_geoloc**(*geoloc*)

Return a RefContext object given a geoloc entry.

indra_world.sources.eidos.processor.**time_context_from_timex**(*timex*)

Return a TimeContext object given a timex entry.

3.1.2 Hume (indra_world.sources.hume)

Hume is a general purpose reading system developed by BBN.

Currently, INDRA can process JSON-LD files produced by Hume. When available, the API will be extended with access to the reader as a service.

API (indra_world.sources.hume.api)

indra_world.sources.hume.api.**process_jsonld**(*jsonld*, *extract_filter=None*, *grounding_mode=None*)

Process a JSON-LD string in the new format to extract Statements.

Parameters

- **jsonld** (*dict*) – The JSON-LD object to be processed.
- **extract_filter** (*Optional[list]*) – A list of relation types to extract. Valid values in the list are 'influence' and 'event'. If not given, all relation types are extracted. This argument can be used if, for instance, only Influence statements are of interest. Default: None
- **grounding_mode** (*Optional[str]*) – Selects whether 'flat' or 'compositional' groundings should be extracted. Default: 'flat'.

Returns

A HumeProcessor instance, which contains a list of INDRA Statements as its statements attribute.

Return type

indra_world.sources.hume.HumeProcessor

indra_world.sources.hume.api.**process_jsonld_file**(*fname*, *extract_filter=None*, *grounding_mode='compositional'*)

Process a JSON-LD file in the new format to extract Statements.

Parameters

- **fname** (*str*) – The path to the JSON-LD file to be processed.
- **extract_filter** (*Optional[list]*) – A list of relation types to extract. Valid values in the list are 'influence' and 'event'. If not given, all relation types are extracted. This argument can be used if, for instance, only Influence statements are of interest. Default: None
- **grounding_mode** (*Optional[str]*) – Selects whether 'flat' or 'compositional' groundings should be extracted. Default: 'flat'.

Returns

A HumeProcessor instance, which contains a list of INDRA Statements as its statements attribute.

Return type

indra_world.sources.hume.HumeProcessor

Processor (indra_world.sources.hume.processor)

class indra_world.sources.hume.processor.**HumeJsonLdProcessor**(*json_dict*)

This processor extracts INDRA Statements from Hume JSON-LD output.

Parameters
json_dict (*dict*) – A JSON dictionary containing the Hume extractions in JSON-LD format.

tree

The objectpath Tree object representing the extractions.

Type
objectpath.Tree

statements

A list of INDRA Statements that were extracted by the processor.

Type
list[indra.statements.Statement]

class indra_world.sources.hume.processor.**HumeJsonLdProcessorCompositional**(*json_dict*)

3.1.3 Sofia (indra_world.sources.sofia)

Sofia is a general purpose natural language processing system developed at UPitt and CMU by N. Miskov et al.

API (indra_world.sources.sofia.api)

indra_world.sources.sofia.api.**process_json**(*json_obj*, *extract_filter=None*, *grounding_mode=None*)

Return processor by processing a JSON object returned by Sofia.

Parameters

- **json_obj** (*json*) – A JSON object containing extractions from Sofia.
- **extract_filter** (*Optional[list]*) – A list of relation types to extract. Valid values in the list are 'influence' and 'event'. If not given, all relation types are extracted. This argument can be used if, for instance, only Influence statements are of interest. Default: None
- **grounding_mode** (*Optional[str]*) – Selects whether 'flat' or 'compositional' groundings should be extracted. Default: 'flat'.

Returns
sp – A SofiaProcessor object which has a list of extracted INDRA Statements as its statements attribute.

Return type
indra.sources.sofia.processor.SofiaProcessor

indra_world.sources.sofia.api.**process_json_file**(*fname*, *extract_filter=None*, *grounding_mode='compositional'*)

Return processor by processing a JSON file produced by Sofia.

Parameters

- **fname** (*str*) – The name of the JSON file to process

- **extract_filter** (*Optional[list]*) – A list of relation types to extract. Valid values in the list are 'influence' and 'event'. If not given, all relation types are extracted. This argument can be used if, for instance, only Influence statements are of interest. Default: None
- **grounding_mode** (*Optional[str]*) – Selects whether 'flat' or 'compositional' groundings should be extracted. Default: 'flat'.

Returns
A SofiaProcessor object which has a list of extracted INDRA Statements as its statements attribute.

Return type
indra.sources.sofia.processor.SofiaProcessor

indra_world.sources.sofia.api.**process_table**(*fname*, *extract_filter=None*, *grounding_mode='compositional'*)

Return processor by processing a given sheet of a spreadsheet file.

Parameters

- **fname** (*str*) – The name of the Excel file (typically .xlsx extension) to process
- **extract_filter** (*Optional[list]*) – A list of relation types to extract. Valid values in the list are 'influence' and 'event'. If not given, all relation types are extracted. This argument can be used if, for instance, only Influence statements are of interest. Default: None
- **grounding_mode** (*Optional[str]*) – Selects whether 'flat' or 'compositional' groundings should be extracted. Default: 'flat'.

Returns
sp – A SofiaProcessor object which has a list of extracted INDRA Statements as its statements attribute.

Return type
indra.sources.sofia.processor.SofiaProcessor

indra_world.sources.sofia.api.**process_text**(*text*, *out_file='sofia_output.json'*, *auth=None*, *extract_filter=None*, *grounding_mode='compositional'*)

Return processor by processing text given as a string.

Parameters

- **text** (*str*) – A string containing the text to be processed with Sofia.
- **out_file** (*Optional[str]*) – The path to a file to save the reader's output into. Default: sofia_output.json
- **auth** (*Optional[list]*) – A username/password pair for the Sofia web service. If not given, the SOFIA_USERNAME and SOFIA_PASSWORD values are loaded from either the INDRA config or the environment.
- **extract_filter** (*Optional[list]*) – A list of relation types to extract. Valid values in the list are 'influence' and 'event'. If not given, all relation types are extracted. This argument can be used if, for instance, only Influence statements are of interest. Default: None
- **grounding_mode** (*Optional[str]*) – Selects whether 'flat' or 'compositional' groundings should be extracted. Default: 'flat'.

Returns
sp – A SofiaProcessor object which has a list of extracted INDRA Statements as its statements attribute. If the API did not process the text, None is returned.

Return type
indra.sources.sofia.processor.SofiaProcessor

Processor (indra_world.sources.sofia.processor)

class indra_world.sources.sofia.processor.**SofiaExcelProcessor**(*relation_rows*, *event_rows*, *entity_rows*, ***kwargs*)

Bases: *SofiaProcessor*

An Excel processor extracting statements from reading done by Sofia

extract_events(*event_rows*, *relation_rows*)

Extract Event statements of a Sofia document in Excel format

Parameters

- **event_rows** (Iterator[Tuple[Cell, ...]]) – The extracted event data from an Excel document
- **relation_rows** (Iterator[Tuple[Cell, ...]]) – The extracted relation data from an Excel document

Return type
None

extract_relations(*relation_rows*)

Extract Influence statements from relation events

Parameters
relation_rows (Iterator[Tuple[Cell, ...]]) – The extracted relation data from an Excel document

Return type
None

process_events(*event_rows*)

Process the events of Sofia document extractions in Excel format

Parameters
event_rows (Iterator[Tuple[Cell, ...]]) – The extracted event data from an Excel document

Returns
A dict of event keyed by their event index

Return type
processed_event_dict

class indra_world.sources.sofia.processor.**SofiaJsonProcessor**(*jd*, ***kwargs*)

Bases: *SofiaProcessor*

A JSON processor extracting statements from reading done by Sofia

extract_events(*jd*)

Extract Event statements from a Sofia document extraction

Parameters
jd (Dict[str, str]) – A dictionary with document extractions

Return type
None

extract_relations(*jd*)

Extract Influence statements from a Sofia document extraction

Parameters
jd (Dict[str, Any]) – A dictionary with document extractions

Return type
None

process_entities(*jd*)

Process the entities of a Sofia document extraction

Parameters
jd (Dict[str, Any]) – The extracted data from a document

Returns
A dictionary of processed entities keyed by their entity index

Return type
ent_dict

process_events(*jd*)

Process the event of a Sofia document extraction

Parameters
jd (Dict[str, Any]) – The extracted data from a document

Returns
A dictionary of processed events keyed by their event index

Return type
processed_event_dict

class indra_world.sources.sofia.processor.**SofiaProcessor**(*score_cutoff=None*, *grounding_mode='compositional'*)

Bases: object

A processor extracting statements from reading done by Sofia

get_compositional_grounding(*event_entry*)

Get the compositional grounding for an event

Parameters
event_entry (Dict[str, str]) – The event to get the compositional grounding for

Returns
The name of the grounding and a tuple of representing the compositional grounding

Return type
grounding

get_event(*event_entry*)

Get an Event with the pre-set grounding mode

The grounding mode is set at initialization of the class and is stored in the attribute *grounding_mode*.

Parameters
event_entry (Dict[str, str]) – The event to process

Returns
An Event statement

Return type
event

`get_event_compositional`(*event_entry*)

Get an Event with compositional grounding

Parameters
event_entry (Dict[str, str]) – The event to process

Returns
An Event statement

Return type
event

`get_event_flat`(*event_entry*)

Get an Event with flattened grounding

Parameters
event_entry (Dict[str, str]) – The event to process

Returns
An Event statement

Return type
event

`get_meaningful_events`(*raw_event_dict*)

Process events by extracting polarity

Parameters
raw_event_dict (Dict[str, Any]) – A dict of events to process

Returns
A dict of event data

Return type
processed_event_dict

`get_relation_events`(*rel_dict*)

Get a list of the event indices associated with a causal entry

Parameters
rel_dict (Dict[str, str]) – A causal entry to extract event indices from

Returns
A list of event indices

Return type
relation_events

3.1.4 CWMS (indra_world.sources.cwms)

CWMS is a variant of the TRIPS system. It is a general purpose natural language understanding system with applications in world modeling. For more information, see: http://trips.ihmc.us/parser/cgi/cwmsreader

API (indra_world.sources.cwms.api)

indra_world.sources.cwms.api.**process_ekb**(*ekb_str*, *extract_filter=None*, *grounding_mode='flat'*)

Processes an EKB string produced by CWMS.

Parameters

- **ekb_str** (*str*) – EKB string to process
- **extract_filter** (*Optional[list]*) – A list of relation types to extract. Valid values in the list are 'influence', 'association', 'event' and 'migration'. If not given, only Influences are extracted since processing other relation types can be time consuming. This argument can be used if the extraction of other relation types such as Events are also of interest.
- **grounding_mode** (*Optional[str]*) – Selects whether 'flat' or 'compositional' groundings should be extracted. Default: 'flat'.

Returns
cp – A CWMSProcessor, which contains a list of INDRA statements in its statements attribute.

Return type
indra.sources.cwms.CWMSProcessor

indra_world.sources.cwms.api.**process_ekb_file**(*fname*, *extract_filter=None*, *grounding_mode='flat'*)

Processes an EKB file produced by CWMS.

Parameters

- **fname** (*str*) – Path to the EKB file to process.
- **extract_filter** (*Optional[list]*) – A list of relation types to extract. Valid values in the list are 'influence', 'association', 'event' and 'migration'. If not given, only Influences are extracted since processing other relation types can be time consuming. This argument can be used if the extraction of other relation types such as Events are also of interest.
- **grounding_mode** (*Optional[str]*) – Selects whether 'flat' or 'compositional' groundings should be extracted. Default: 'flat'.

Returns
cp – A CWMSProcessor, which contains a list of INDRA statements in its statements attribute.

Return type
indra.sources.cwms.CWMSProcessor

indra_world.sources.cwms.api.**process_text**(*text*, *save_xml='cwms_output.xml'*, *extract_filter=None*, *grounding_mode='flat'*)

Processes text using the CWMS web service.

Parameters

- **text** (*str*) – Text to process
- **save_xml** (*Optional[str]*) – A file name in which to dump the output from CWMS. Default: cwms_output.xml

- **extract_filter** (*Optional[list]*) – A list of relation types to extract. Valid values in the list are 'influence', 'association', 'event' and 'migration'. If not given, only Influences are extracted since processing other relation types can be time consuming. This argument can be used if the extraction of other relation types such as Events are also of interest.
- **grounding_mode** (*Optional[str]*) – Selects whether 'flat' or 'compositional' groundings should be extracted. Default: 'flat'.

Returns
cp – A CWMSProcessor, which contains a list of INDRA statements in its statements attribute.

Return type
indra.sources.cwms.CWMSProcessor

Processor (indra_world.sources.cwms.processor)

exception indra_world.sources.cwms.processor.**CWMSError**

Bases: Exception

class indra_world.sources.cwms.processor.**CWMSProcessor**(*xml_string*)

Bases: object

The CWMSProcessor currently extracts causal relationships between terms (nouns) in EKB. In the future, this processor can be extended to extract other types of relations, or to extract relations involving events.

For more details on the TRIPS EKB XML format, see http://trips.ihmc.us/parser/cgi/drum

Parameters
xml_string (*str*) – A TRIPS extraction knowledge base (EKB) in XML format as a string.

tree

An ElementTree object representation of the TRIPS EKB XML.

Type
xml.etree.ElementTree.Element

doc_id

Document ID

Type
str

statements

A list of INDRA Statements that were extracted from the EKB.

Type
list[indra.statements.Statement]

sentences

The list of all sentences in the EKB with their IDs

Type
dict[str: str]

paragraphs

The list of all paragraphs in the EKB with their IDs

Type
dict[str: str]

par_to_sec

A map from paragraph IDs to their associated section types

Type
dict[str: str]

event_from_event(*event_term*)

Return an Event from an EVENT element in the EKB.

extract_causal_relations()

Extract Influence Statements from the EKB.

extract_events()

Extract standalone Events from the EKB.

influence_from_event(*event*)

Return an Influence from an EVENT element in the EKB.

influence_from_relation(*relation*)

Return an Influence from a CC element in the EKB.

migration_from_event(*event_term*)

Return a Migration event from an EVENT element in the EKB.

class indra_world.sources.cwms.processor.**CWMSProcessorCompositional**(*xml_string*)

Bases: *CWMSProcessor*

3.1.5 DART (indra_world.sources.dart)

API (indra_world.sources.dart.api)

indra_world.sources.dart.api.**get_record_key**(*rec*)

Return a key for a DART record for purposes of deduplication.

indra_world.sources.dart.api.**get_unique_records**(*recs*)

Deduplicate DART records based on an identifier key.

indra_world.sources.dart.api.**print_record_stats**(*recs*)

Print statistics for a list of DART records.

Client (indra_world.sources.dart.client)

A client for accessing reader output from the DART system.

class indra_world.sources.dart.client.**DartClient**(*storage_mode='web'*, *dart_url=None*, *dart_uname=None*, *dart_pwd=None*, *local_storage=None*)

A client for the DART web service with optional local storage.

Parameters

- **storage_mode** (*Optional[str]*) – If *web*, the configured DART URL and credentials are used to communicate with the DART web service. If *local*, a local storage is used to access and store reader outputs.
- **dart_url** (*Optional[str]*) – The DART service URL. If given, it overrides the DART_WM_URL configuration value.

- **dart_uname** (*Optional[str]*) – The DART service user name. If given, it overrides the DART_WM_USERNAME configuration value.
- **dart_pwd** (*Optional[str]*) – The DART service password. If given, it overrides the DART_WM_PASSWORD configuration value.
- **local_storage** (*Optional[str]*) – A path that points to a folder for local storage. If the storage_mode is *web*, this local_storage is used as a local cache. If the storage_mode is *local*, it is used as the primary location to access reader outputs. If given, it overrides the INDRA_WM_CACHE configuration value.

cache_record(*record*, *overwrite=False*)

Download and cache a given record in local storage.

Parameters
record (*dict*) – A DART record.

cache_records(*records*, *overwrite=False*)

Download and cache a list of records in local storage.

Parameters
records (*list[dict]*) – A list of DART records.

download_output(*storage_key*)

Return content from the DART web service based on its storage key.

Parameters
storage_key (*str*) – A DART storage key.

Returns
The content corresponding to the storage key.

Return type
str

get_local_storage_path(*record*)

Return the local storage path for a DART record.

get_ontology(*ontology_id*)

Return the DART ontology record JSON for the given ontology ID.

get_ontology_graph(*ontology_id*)

Return the ontology graph for the given ontology ID.

get_output_from_record(*record*)

Return reader output corresponding to a single record.

Parameters
record (*dict*) – A single DART record.

Returns
The reader output corresponding to the given record.

Return type
str

get_outputs_from_records(*records*)

Return reader outputs corresponding to a list of records.

Parameters
records (*list of dict*) – A list of records returned from the reader output query.

Returns
A two-level dict of reader output keyed by reader and then document id.

Return type
dict(str, dict)

`get_reader_output_records`(*readers=None*, *versions=None*, *document_ids=None*, *timestamp=None*, *tenant=None*, *ontology_id=None*, *unique=False*)

Return reader output metadata records by querying the DART API

Query json structure:
{"readers": ["MyAwesomeTool", "SomeOtherAwesomeTool"], "versions": ["3.1.4", "1.3.3.7"], "document_ids": ["qwerty1234", "poiuyt0987"], "timestamp": {"before": "yyyy-mm-ddThh:mm:ss", "after": "yyyy-mm-ddThh:mm:ss"}}

Parameters

- **readers** (*list*) – A list of reader names
- **versions** (*list*) – A list of versions to match with the reader name(s)
- **document_ids** (*list*) – A list of document identifiers
- **timestamp** (*dict("before"|"after",str)*) – The timestamp string must be formatted "yyyy-mm-ddThh:mm:ss".
- **tenant** (*Optional[str]*) – Return only records for the given tenant.
- **ontology_id** (*Optional[str]*) – Return only records for the given ontology ID.
- **unique** (*Optional[bool]*) – If true, records that are duplicates are collapsed. Default: False.

Returns
The JSON payload of the response from the DART API

Return type
dict

`get_reader_versions`(*reader*)

Return the available versions for a given reader.

`get_tenant_ontology`(*tenant_id*, *version=None*)

Return the DART ontology record JSON for the given tenant ID and optional version.

`get_tenant_ontology_graph`(*tenant_id*, *version=None*)

Return the ontology graph for the given tenant ID and optional version.

`indra_world.sources.dart.client.prioritize_records`(*records*, *priorities=None*)

Return unique records per reader and document prioritizing by version.

Parameters

- **records** (*list of dict*) – A list of records returned from the reader output query.
- **priorities** (*dict of list*) – A dict keyed by reader names (e.g., cwms, eidos) with values representing reader versions in decreasing order of priority.

Returns
records – A list of records that are unique per reader and document, picked by version priority when multiple records exist for the same reader and document.

Return type
list of dict

3.2 Knowledge assembly modules (indra_world.assembly)

3.2.1 Statement preprocessing (indra_world.assembly.preprocess)

indra_world.assembly.preprocess.**preprocess_statements**(*raw_statements*, *steps*)

Run a preprocessing pipeline on raw statements.

Parameters

- **raw_statements** (List[Statement]) – A list of INDRA Statements to preprocess.
- **steps** (List[Dict[str, Any]]) – A list of AssemblyPipeline steps that define the steps of preprocessing.

Returns
A list of preprocessed INDRA Statements.

Return type
preprocessed_statements

3.2.2 Assembly operations (indra_world.assembly.operations)

class indra_world.assembly.operations.**CompositionalRefinementFilter**(*ontology*, *nproc=None*)

extend(*stmts_by_hash*)

Extend the initial data structures with a set of new statements.

Parameters
stmts_by_hash (*dict[int, indra.statements.Statement]*) – A dict of statements keyed by their hashes.

get_related(*stmt*, *possibly_related=None*, *direction='less_specific'*)

Return a set of statement hashes that a given statement is potentially related to.

Parameters

- **stmt** (*indra.statements.Statement*) – The INDRA statement whose potential relations we want to filter.
- **possibly_related** (*set or None*) – A set of statement hashes that this statement is potentially related to, as determined by some other filter. If this parameter is a set (including an empty set), this function should return a subset of it (intuitively, this filter can only further eliminate some of the potentially related hashes that were previously determined to be potential relations). If this argument is None, the function must assume that no previous filter was run before, and should therefore return all the possible relations that it determines.
- **direction** (*str*) – One of 'less_specific' or 'more_specific. Since refinements are directed relations, this function can operate in two different directions: it can either find less specific potentially related stateemnts, or it can find more specific potentially related statements, as determined by this argument.

Returns
A set of INDRA Statement hashes that are potentially related to the given statement.

Return type
set of int

initialize(*stmts_by_hash*)

Initialize the filter class with a set of statements.

The filter can build up some useful data structures in this function before being applied to any specific statements.

Parameters
stmts_by_hash (*dict[int, indra.statements.Statement]*) – A dict of statements keyed by their hashes.

indra_world.assembly.operations.**get_expanded_events_influences**(*stmts*)

Return a list of all standalone events from a list of statements.

indra_world.assembly.operations.**location_matches_compositional**(*stmt*)

Return a matches_key which takes geo-location into account.

indra_world.assembly.operations.**location_refinement_compositional**(*st1*, *st2*, *ontology*, *entities_refined=True*)

Return True if there is a location-aware refinement between stmts.

indra_world.assembly.operations.**make_display_name**(*comp_grounding*)

Return display name from a compositional grounding with 'of' linkers.

indra_world.assembly.operations.**make_display_name_linear**(*comp_grounding*)

Return display name from compositional grounding with linear joining.

indra_world.assembly.operations.**merge_deltas**(*stmts_in*)

Gather and merge original Influence delta information from evidence.

This function is only applicable to Influence Statements that have subj and obj deltas. All other statement types are passed through unchanged. Polarities and adjectives for subjects and objects respectivey are collected and merged by travesrsing all evidences of a Statement.

Parameters
stmts_in (*list[indra.statements.Statement]*) – A list of INDRA Statements whose influence deltas should be merged. These Statements are meant to have been preassembled and potentially have multiple pieces of evidence.

Returns
stmts_out – The list of Statements now with deltas merged at the Statement level.

Return type
list[indra.statements.Statement]

indra_world.assembly.operations.**remove_namespaces**(*stmts*, *namespaces*)

Remove unnecessary namespaces from Concept grounding.

indra_world.assembly.operations.**remove_raw_grounding**(*stmts*)

Remove the raw_grounding annotation to decrease output size.

3.2.3 Matches functions (indra_world.assembly.matches)

indra_world.assembly.matches.**event_location_time_matches**(*event*)

Return Event matches key which takes location and time into account.

indra_world.assembly.matches.**get_location**(*stmt*)

Return the grounded geo-location context associated with a Statement.

indra_world.assembly.matches.**get_location_from_object**(*loc_obj*)

Return geo-location from a RefContext location object.

indra_world.assembly.matches.**get_time**(*stmt*)

Return the time context associated with a Statement.

indra_world.assembly.matches.**has_location**(*stmt*)

Return True if a Statement has grounded geo-location context.

indra_world.assembly.matches.**has_time**(*stmt*)

Return True if a Statement has time context.

indra_world.assembly.matches.**location_matches**(*stmt*)

Return a matches_key which takes geo-location into account.

indra_world.assembly.matches.**location_matches_compositional**(*stmt*)

Return a matches_key which takes geo-location into account.

3.2.4 Refinement functions (indra_world.assembly.refinement)

class indra_world.assembly.refinement.**CompositionalRefinementFilter**(*ontology*, *nproc=None*)

extend(*stmts_by_hash*)

Extend the initial data structures with a set of new statements.

Parameters

stmts_by_hash (*dict[int, indra.statements.Statement]*) – A dict of statements keyed by their hashes.

get_related(*stmt*, *possibly_related=None*, *direction='less_specific'*)

Return a set of statement hashes that a given statement is potentially related to.

Parameters

- **stmt** (*indra.statements.Statement*) – The INDRA statement whose potential relations we want to filter.
- **possibly_related** (*set or None*) – A set of statement hashes that this statement is potentially related to, as determined by some other filter. If this parameter is a set (including an empty set), this function should return a subset of it (intuitively, this filter can only further eliminate some of the potentially related hashes that were previously determined to be potential relations). If this argument is None, the function must assume that no previous filter was run before, and should therefore return all the possible relations that it determines.
- **direction** (*str*) – One of 'less_specific' or 'more_specific. Since refinements are directed relations, this function can operate in two different directions: it can either find less specific potentially related stateemnts, or it can find more specific potentially related statements, as determined by this argument.

Returns
A set of INDRA Statement hashes that are potentially related to the given statement.

Return type
set of int

initialize(*stmts_by_hash*)

Initialize the filter class with a set of statements.

The filter can build up some useful data structures in this function before being applied to any specific statements.

Parameters
stmts_by_hash (*dict[int, indra.statements.Statement]*) – A dict of statements keyed by their hashes.

indra_world.assembly.refinement.**event_location_refinement**(*st1*, *st2*, *ontology*, *entities_refined*, *ignore_polarity=False*)

Return True if there is a location-aware refinement between Events.

indra_world.assembly.refinement.**event_location_time_refinement**(*st1*, *st2*, *ontology*, *entities_refined*)

Return True if there is a location/time refinement between Events.

indra_world.assembly.refinement.**get_agent_key**(*agent*, *comp_idx*)

Return a key for an Agent for use in refinement finding.

Parameters
agent (*indra.statements.Agent or None*) – An INDRA Agent whose key should be returned.

Returns
The key that maps the given agent to the ontology, with special handling for ungrounded and None Agents.

Return type
tuple or None

indra_world.assembly.refinement.**location_refinement**(*st1*, *st2*, *ontology*, *entities_refined*)

Return True if there is a location-aware refinement between stmts.

indra_world.assembly.refinement.**location_refinement_compositional**(*st1*, *st2*, *ontology*, *entities_refined=True*)

Return True if there is a location-aware refinement between stmts.

indra_world.assembly.refinement.**location_time_refinement**(*st1*, *st2*, *ontology*, *entities_refined*)

Return True if there is a location/time refinement between stmts.

3.2.5 Incremental Assembler (indra_world.assembly.incremental_assembler)

class indra_world.assembly.incremental_assembler.**AssemblyDelta**(*new_stmts*, *new_evidences*, *new_refinements*, *beliefs*, *matches_fun=None*)

Represents changes to the assembly structure as a result of new statements added to a set of existing statements.

new_stmts

A dict of new statement keyed by hash.

Type
dict[str, indra.statements.Statement]

new_evidences

A dict of new evidences for existing or new statements keyed by statement hash.

Type
dict[str, indra.statements.Evidence]

new_refinements

A list of statement hash pairs representing new refinement links.

Type
list[tuple]

beliefs

A dict of belief scores keyed by all statement hashes (both old and new).

Type
dict[str, float]

matches_fun

An optional custom matches function. When using a custom matches function for assembly, providing it here is necessary to get correct JSON serialization.

Type
Optional[Callable[[Statement], str]]

to_json()

Return a JSON representation of the assembly delta.

class indra_world.assembly.incremental_assembler.**IncrementalAssembler**(*prepared_stmts*, *refinement_filters=None*, *matches_fun=<function location_matches_compositional>*, *curations=None*, *post_processing_steps=None*, *ontology=None*)

Assemble a set of prepared statements and allow incremental extensions.

Parameters

- **prepared_stmts** (*list[indra.statements.Statement]*) – A list of prepared INDRA Statements.
- **refinement_filters** (*Optional[list[indra.preassembler.refinement.RefinementFilter]]*) – A list of refinement filter classes to be used for refinement finding. Default: the standard set of compositional refinement filters.
- **matches_fun** (*Optional[function]*) – A custom matches function for determining matching statements and calculating hashes. Default: matches function that takes compositional grounding and location into account.
- **curations** (*dict[dict]*) – A dict of user curations to be integrated into the assembly results, keyed by statement hash.

- **post_processing_steps** (*list[dict]*) – Steps that can be used in an INDRA AssemblyPipeline to do post-processing on statements.

refinement_edges

A set of tuples of statement hashes representing refinement links between statements.

Type
set

add_statements(*stmts*)

Add new statements for incremental assembly.

Parameters
stmts (*list[indra.statements.Statement]*) – A list of new prepared statements to be incrementally assembled into the set of existing statements.

Returns
An AssemblyDelta object representing the changes to the assembly as a result of the new added statements.

Return type
AssemblyDelta

static annotate_evidences(*stmt*)

Add annotations to evidences of a given statement.

apply_curations()

Apply the set of curations to the de-duplicated statements.

static build_refinements_graph(*stmts_by_hash*, *refinement_edges*)

Return a refinements graph based on statements and refinement edges.

deduplicate()

Build hash-based statement and evidence data structures to deduplicate.

get_all_supporting_evidence(*sh*)

Return direct and indirect evidence for a statement hash.

get_beliefs()

Calculate and return beliefs for all statements.

get_curation_effect(*old_hash*, *curation*)

Return changed matches hash as a result of curation.

get_refinements()

Calculate refinement relationships between de-duplicated statements.

get_statements()

Return a flat list of statements with their evidences.

indra_world.assembly.incremental_assembler.**parse_factor_grounding_curation**(*cur*)

Parse details from a curation that changes a concept's grounding.

indra_world.assembly.incremental_assembler.**parse_factor_polarity_curation**(*cur*)

Parse details from a curation that changes an event's polarity.

3.2.6 Statistics (indra_world.assembly.stats)

3.3 Ontology Module (indra_world.ontology)

Module containing the implementation of an IndraOntology for the World Modelers use case.

3.3.1 World Ontology (indra_world.ontology.ontology)

class indra_world.ontology.ontology.**WorldOntology**(*url*, *yml=None*)

Represents the ontology used for World Modelers applications.

Parameters
url (*str*) – The URL or file path pointing to a World Modelers ontology YAML.

url

The URL or file path pointing to a World Modelers ontology YAML.

Type
str

yml

The ontology YAML as loaded by the yaml package from the URL.

Type
list

add_entry(*entry*, *examples=None*, *neg_examples=None*)

Add a new ontology entry with examples.

This works by adding the entry to the yml attribute first and then reloading the entire yaml to build a new graph.

Parameters

- **entry** (*str*) – The new entry.
- **examples** (*Optional[list of str]*) – Examples for the new entry.
- **neg_examples** (*Optional[list of str]*) – Negative examples for the new entry.

build_relations(*node*, *tree*, *prefix*)

Build relations for the classic ontology format <= v3.0

build_relations_new_format(*node*, *prefix*)

Build relations for the new ontology format > v3.0

dump_yml_str()

Return a string-serialized form of the loaded YAML

Returns
The YAML string of the ontology.

Return type
str

initialize()

Load the World Modelers ontology from the web and build the graph.

indra_world.ontology.ontology.**load_world_ontology**(*url=None*, *default_type='compositional'*)

Load the world ontology from a given URL or file path.

indra_world.ontology.ontology.**load_yaml_from_path**(*path*)

Return a YAML object loaded from a YAML file URL.

3.4 Belief Engine (indra_world.belief)

indra_world.belief.**get_eidos_bayesian_scorer**(*prior_counts=None*)

Return a BayesianScorer based on Eidos curation counts.

Returns
A BayesianScorer belief scorer instance.

Return type
scorer

indra_world.belief.**get_eidos_scorer**()

Return a SimpleScorer based on Eidos curated precision estimates.

Returns
A SimpleScorer instance loaded with default prior probabilities as well as prior probabilities derived from curation-based counts.

Return type
scorer

indra_world.belief.**load_eidos_curation_table**()

Return a pandas table of Eidos curation data.

Returns
A pandas dataframe of the curation data.

Return type
table

3.5 Output assemblers (indra_world.assemblers)

As opposed to INDRA, the importance of output/model assemblers in INDRA World is minor, since other systems such as Delphi (https://github.com/ml4ai/delphi) and DySE (https://dl.acm.org/doi/10.1145/3359115.3359123) take on the role of converting assembled INDRA Statements into probabilistic and logical dynamical models, respectively.

3.5.1 CAG Assembler (indra_world.assemblers.cag)

Assemble simple graphs of assembled INDRA Statements that can be embedded into websites or notebooks.

Assembler (indra_world.assemblers.cag.assembler)

class indra_world.assemblers.cag.assembler.**CAGAssembler**(*stmts=None*)

Assembles a causal analysis graph from INDRA Statements.

Parameters

stmts (*Optional[list[indra.statement.Statements]]*) – A list of INDRA Statements to be assembled. Currently supports Influence Statements.

statements

A list of INDRA Statements to be assembled.

Type

list[indra.statements.Statement]

CAG

A networkx MultiDiGraph object representing the causal analysis graph.

Type

nx.MultiDiGraph

add_statements(*stmts*)

Add a list of Statements to the assembler.

export_to_cytoscapejs()

Return CAG in format readable by CytoscapeJS.

Returns

A JSON-like dict representing the graph for use with CytoscapeJS.

Return type

dict

generate_jupyter_js(*cyjs_style=None*, *cyjs_layout=None*)

Generate Javascript from a template to run in Jupyter notebooks.

Parameters

- **cyjs_style** (*Optional[dict]*) – A dict that sets CytoscapeJS style as specified in https://github.com/cytoscape/cytoscape.js/blob/master/documentation/md/style.md.
- **cyjs_layout** (*Optional[dict]*) – A dict that sets CytoscapeJS layout parameters.

Returns

A Javascript string to be rendered in a Jupyter notebook cell.

Return type

str

make_model(*grounding_ontology='UN'*, *grounding_threshold=None*)

Return a networkx MultiDiGraph representing a causal analysis graph.

Parameters

- **grounding_ontology** (*Optional[str]*) – The ontology from which the grounding should be taken (e.g. UN, FAO)
- **grounding_threshold** (*Optional[float]*) – Minimum threshold score for Eidos grounding.

Returns

The assembled CAG.

Return type
nx.MultiDiGraph

3.5.2 Figaro Assembler (indra_world.assemblers.figaro)

A proof-of-concept assembler for INDRA Statements into probabilistic programs in the Figaro (https://github.com/p2t2/figaro) framework.

Assembler (indra_world.assemblers.figaro.assembler)

3.5.3 TSV Assembler (indra_world.assemblers.tsv)

Assemble tab separated spreadsheets of assembled INDRA Statements for curation purposes.

3.6 Indra World Service (indra_world.service)

Note: This is the documentation of the codebase used in the INDRA World service. Documentation of the service API can be found here.

3.6.1 INDRA World Database (indra_world.service.db)

Database Manager (indra_world.service.db.manager)

class indra_world.service.db.manager.**DbManager**(*url*)

Manages transactions with the assembly database and exposes an API for various operations.

add_curation_for_project(*project_id*, *stmt_hash*, *curation*)

Add curations for a given project.

add_dart_record(*reader*, *reader_version*, *document_id*, *storage_key*, *date*, *output_version=None*, *labels=None*, *tenants=None*)

Insert a DART record into the database.

add_project(*project_id*, *name*, *corpus_id=None*)

Add a new project.

Parameters

- **project_id** (*str*) – The project ID.
- **name** (*str*) – The project name
- **corpus_id** (*Optional[str]*) – The corpus ID from which the project was derived, if available.

add_records_for_project(*project_id*, *record_keys*)

Add document IDs for a project with the given ID.

add_statements_for_record(*record_key*, *stmts*, *indra_version*)

Add a set of prepared statements for a given document.

create_all()

Create all the database tables in the schema.

execute(*operation*)

Execute an insert operation on the current session and return results.

get_corpus_for_project(*project_id*)

Return the corpus ID that a project was derived from, if available.

get_curations_for_project(*project_id*)

Return curations for a given project

get_dart_records(*reader=None*, *document_id=None*, *reader_version=None*, *output_version=None*, *labels=None*, *tenants=None*)

Return storage keys for DART records given constraints.

get_full_dart_records(*reader=None*, *document_id=None*, *reader_version=None*, *output_version=None*, *labels=None*, *tenants=None*)

Return full DART records given constraints.

get_projects()

Retyurn a list of all projects.

get_session()

Return the current active session or create one if not available.

get_statements()

Return all prepared statements in the DB.

get_statements_for_document(*document_id*, *reader=None*, *reader_version=None*, *indra_version=None*)

Return prepared statements for a given document.

get_statements_for_record(*record_key*)

Return prepared statements for given record key.

get_statements_for_records(*record_keys*, *batch_size=1000*)

Return prepared statements for given list of record keys.

get_tenant_for_corpus(*corpus_id*)

Return the tenant for a given corpus, if available.

query(**query_args*)

Run and return results of a generic query.

sql_query(*query_str*)

Run and return results of a generic SQL query.

Database Schema (indra_world.service.db.Schema)

class indra_world.service.db.schema.**Corpora**(***kwargs*)

class indra_world.service.db.schema.**CorpusRecords**(***kwargs*)

class indra_world.service.db.schema.**Curations**(***kwargs*)

class indra_world.service.db.schema.**DartRecords**(***kwargs*)

class indra_world.service.db.schema.**Ontologies**(***kwargs*)

class indra_world.service.db.schema.**PreparedStatements**(***kwargs*)

class indra_world.service.db.schema.**ProjectRecords**(***kwargs*)

class indra_world.service.db.schema.**Projects**(***kwargs*)

3.6.2 Service controller (indra_world.service.controller)

3.6.3 REST API (indra_world.service.app)

class indra_world.service.app.**AddProjectRecords**(*api=None*, **args*, ***kwargs*)

methods: Optional[List[str]] = {'OPTIONS', 'POST'}

A list of methods this view can handle.

post()

Add project records and assemble them.

Parameters

- **project_id** (*str*) – ID of a project to add records.
- **records** (*list[dict]*) – A list of records to add, each should have a 'storage_key'.

Returns
delta_json – A JSON representation of AssemblyDelta.

Return type
json

class indra_world.service.app.**CwmsProcessText**(*api=None*, **args*, ***kwargs*)

methods: Optional[List[str]] = {'OPTIONS', 'POST'}

A list of methods this view can handle.

post()

Process text with CWMS and return INDRA Statements.

Parameters
text (*str*) – Text to process

Returns
statements – A list of extracted INDRA Statements.

Return type
list[indra.statements.Statement.to_json()]

class indra_world.service.app.**EidosProcessJsonld**(*api=None*, **args*, ***kwargs*)

methods: Optional[List[str]] = {'OPTIONS', 'POST'}

A list of methods this view can handle.

post()

Process an EIDOS JSON-LD and return INDRA Statements.

Parameters

- **jsonld** (*str*) – The JSON-LD string to be processed.

- **grounding_ns** (*Optional[list]*) – A list of name spaces for which INDRA should represent groundings, when given. If not specified or None, all grounding name spaces are propagated. If an empty list, no groundings are propagated. Example: ['UN', 'WM'], Default: None
- **extract_filter** (*Optional[list]*) – A list of relation types to extract. Valid values in the list are 'influence', 'association', 'event'. If not given, all relation types are extracted. This argument can be used if, for instance, only Influence statements are of interest. Default: None
- **grounding_mode** (*Optional[str]*) – Selects whether 'flat' or 'compositional' groundings should be extracted. Default: 'flat'.

Returns
statements – A list of extracted INDRA Statements.

Return type
list[indra.statements.Statement.to_json()]

class indra_world.service.app.**EidosProcessText**(*api=None*, **args*, ***kwargs*)

methods: Optional[List[str]] = {'OPTIONS', 'POST'}
A list of methods this view can handle.

post()
Process text with EIDOS and return INDRA Statements.

Parameters
- **text** (*str*) – The text to be processed.
- **webservice** (*Optional[str]*) – An Eidos reader web service URL to send the request to. If None, the reading is assumed to be done with the Eidos JAR rather than via a web service. Default: None
- **grounding_ns** (*Optional[list]*) – A list of name spaces for which INDRA should represent groundings, when given. If not specified or None, all grounding name spaces are propagated. If an empty list, no groundings are propagated. Example: ['UN', 'WM'], Default: None
- **extract_filter** (*Optional[list]*) – A list of relation types to extract. Valid values in the list are 'influence', 'association', 'event'. If not given, all relation types are extracted. This argument can be used if, for instance, only Influence statements are of interest. Default: None
- **grounding_mode** (*Optional[str]*) – Selects whether 'flat' or 'compositional' groundings should be extracted. Default: 'flat'.

Returns
statements – A list of extracted INDRA Statements.

Return type
list[indra.statements.Statement.to_json()]

class indra_world.service.app.**GetAllRecords**(*api=None*, **args*, ***kwargs*)

get()
Get all DART records captured by the service.

Returns
records – A list of records.

Return type
list[dict]

methods: Optional[List[str]] = {'GET', 'OPTIONS'}
A list of methods this view can handle.

class indra_world.service.app.**GetProjectCurations**(*api=None*, **args*, ***kwargs*)

get()
Get project curations.

Parameters
project_id (*str*) – ID of a project.

Returns
curations – A list of curations for the project.

Return type
list[dict]

methods: Optional[List[str]] = {'GET', 'OPTIONS'}
A list of methods this view can handle.

class indra_world.service.app.**GetProjectRecords**(*api=None*, **args*, ***kwargs*)

get()
Get records for a project.

Parameters
project_id (*str*) – ID of a project.

Returns
records – A list of records for the project.

Return type
list[dict]

methods: Optional[List[str]] = {'GET'}
A list of methods this view can handle.

class indra_world.service.app.**GetProjects**(*api=None*, **args*, ***kwargs*)

get()
Get a list of all projects.

Returns
records – A list of projects.

Return type
list[dict]

methods: Optional[List[str]] = {'GET', 'OPTIONS'}
A list of methods this view can handle.

class indra_world.service.app.**Health**(*api=None*, **args*, ***kwargs*)

methods: Optional[List[str]] = {'GET', 'OPTIONS'}
A list of methods this view can handle.

class indra_world.service.app.**HumeProcessJsonld**(*api=None*, **args*, ***kwargs*)

methods: Optional[List[str]] = {'OPTIONS', 'POST'}

A list of methods this view can handle.

post()

Process Hume JSON-LD and return INDRA Statements.

Parameters
jsonld (*str*) – The JSON-LD string to be processed.

Returns
statements – A list of extracted INDRA Statements.

Return type
list[indra.statements.Statement.to_json()]

class indra_world.service.app.**NewProject**(*api=None*, **args*, ***kwargs*)

methods: Optional[List[str]] = {'OPTIONS', 'POST'}

A list of methods this view can handle.

post()

Create new project.

Parameters

- **project_id** (*str*) – ID of a new project.
- **project_name** (*str*) – Name of a new project.
- **corpus_id** (*str*) – ID of a corpus.

class indra_world.service.app.**Notify**(*api=None*, **args*, ***kwargs*)

methods: Optional[List[str]] = {'OPTIONS', 'POST'}

A list of methods this view can handle.

post()

Add and process DART record.

Parameters

- **identity** (*str*) – Name of the reader.
- **version** (*str*) – Reader version.
- **document_id** (*str*) – ID of a document to process.
- **storage_key** (*str*) – Key to store the record with.
- **output_version** (*str*) – The output version (typically ontology version).
- **labels** (*list of str*) – A list of labels for the output.
- **tenants** (*list of str*) – A list of tenants for the output.

class indra_world.service.app.**SofiaProcessJson**(*api=None*, **args*, ***kwargs*)

methods: Optional[List[str]] = {'OPTIONS', 'POST'}

A list of methods this view can handle.

post()

Process a Sofia JSON and return INDRA Statements.

Parameters

- **json** (*str*) – The JSON string to be processed.
- **extract_filter** (*Optional[list]*) – A list of relation types to extract. Valid values in the list are 'influence', 'association', 'event'. If not given, all relation types are extracted. This argument can be used if, for instance, only Influence statements are of interest. Default: None
- **grounding_mode** (*Optional[str]*) – Selects whether 'flat' or 'compositional' groundings should be extracted. Default: 'flat'.

Returns
statements – A list of extracted INDRA Statements.

Return type
list[indra.statements.Statement.to_json()]

class indra_world.service.app.**SubmitCurations**(*api=None*, **args*, ***kwargs*)

methods: Optional[List[str]] = {'OPTIONS', 'POST'}
A list of methods this view can handle.

post()
Submit curations.

Parameters
- **project_id** (*str*) – ID of a project.
- **curations** (*list[dict]*) – A list of curations to submit.
- **calculate_mappings** (*bool*) – Whether to calculate and return a mappings data structure representing the changes made by the curations to the overall project corpus. This is False by default since for large projects this calculation can take a long time.

Returns
mappings – For any statement matches hashes that have changed due to the curations submitted here, the new hash (after applying the curation) is given. Statements whose hash didn't change, or if a curation for some reason couldn't be applied, the given statement is not added to the return value.

Return type
dict

3.6.4 Corpus manager (indra_world.service.corpus_manager)

This module allows running one-off assembly on a set of DART records (i.e., reader outputs) into a 'seed corpus' that can be dumped on S3 for loading into CauseMos.

class indra_world.service.corpus_manager.**CorpusManager**(*db_url*, *dart_records*, *corpus_id*, *metadata*, *dart_client=None*, *tenant=None*, *ontology=None*)

Corpus manager class allowing running assembly on a set of DART records.

assemble()
Run assembly on the prepared statements.

This function loads all the prepared statements associated with the corpus and then runs assembly on them.

dump_local(*base_folder*, *causemos_compatible=True*)
Dump assembled corpus into local files.

dump_s3()

Dump assembled corpus onto S3.

prepare(*records_exist=False*)

Run the preprocessing pipeline on statements.

This function adds the new corpus to the DB, adds records to the new corpus, then processes the reader outputs for those records into statements, preprocesses the statements, and then stores these prepared statements in the DB.

indra_world.service.corpus_manager.**download_corpus**(*corpus_id*, *fname*)

Download a given corpus of assembled statements from S3.

Parameters

- **corpus_id** (str) – The ID of the corpus.
- **fname** (str) – The file in which the downloaded corpus should be written.

Return type
None

indra_world.service.corpus_manager.**get_corpus_index**()

Return the corpus index as a list of tuples with corpus IDs and dates.

3.7 INDRA World Dashboard (indra_world.dashboard)

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

i

INDEX

E

G

H

I

L

M

N

T

U

W

Y