# indra$_{w}m_{s}ervice$

**Aug 19, 2020**

# Contents

CHAPTER 1

# World Modelers INDRA service stack

## 1.1 Using the services

Below, SERVICE_HOST should be replaced by the address of the server on which the services are running.

### 1.1.1 Check that the service is running

This is a simple health endpoint that can be pinged to check that the service is running.

```
URL: http://SERVICE_HOST:8001/health
Method: GET
Output: {"state": "healthy", "version": "1.0.0"}
```

### 1.1.2 Read text to produce INDRA Statements

Read a given text with a reader and return INDRA Statements (below, <reader> can be eidos, sofia or cwms). Note that for *eidos* specifically, a *webservice* parameter should also be passed which points to the address on which the Eidos web service is running (see above):

```
URL: http://SERVICE_HOST:8000/<reader>/process_text
Method: POST with JSON content header
Input parameters: {"text": "rainfall causes floods"}
Output: <indra statements json>
```

### 1.1.3 Submit curations

This endpoint take a single *curations* parameter which is a list of curations according to the CauseMos JSON format in which curations are represented. This representation contains a *corpus_id* and a *project_id* as part of each curation entry, therefore these do not need to be specified separately.

```
URL: http://SERVICE_HOST:8001/submit_curations
Method: POST with JSON content header
Input parameters: {"curations": <list of curations>}
Output: {}
```

### 1.1.4 Persist curations for a given corpus on S3

The service does local caching of curations, however, it does not push curations submitted during runtime to S3 (which can be useful if someone wants to access them as a file independent of the service). This endpoint allows pushing all the curations for a given *corpus_id* to S3.

```
URL: http://SERVICE_HOST:8001/save_curations
Method: POST with JSON content header
Input parameters: {"corpus_id": "<corpus id>"}
Output: {}
```

### 1.1.5 Update beliefs

This endpoint performs a lightweight belief re-calculation based on curations obtained so far. It takes a required *corpus_id* argument and an optional *project_id* argument. If a *project_id* is provided, beliefs are calculated based on project-specific curations, otherwise, all the curations for the given corpus are taken into account.

```
URL: http://SERVICE_HOST:8001/update_beliefs
Method: POST with JSON content header
Input parameters: {"corpus_id": "<corpus id>",
                   "project_id": "<project id>"}
Output: {"38ce0c14-2c7e-4df8-bd53-3006afeaa193": 0,
 "6f2b2d69-16af-40ea-aa03-9b3a9a1d2ac3": 0.6979166666666666,
 "727adb95-4890-4bbc-a985-fd985c355215": 0.6979166666666666}
```

### 1.1.6 Re-assemble corpus

This endpoint runs a new assembly for a given *corpus_id* and *project_id* based on curations and dumps the results on S3. The project-specific statement dump appears as a sub-key under the corpus key base, as *indra-models/<corpus id>/<project id>/statements.json*.

```
URL: http://SERVICE_HOST:8001/run_assembly
Method: POST with JSON content header
Input parameters: {"corpus_id": "<corpus id>",
                   "project_id": "<project id>"}
Output: {}
```

### 1.1.7 Download curations

This endpoint allows downloading curations and the corresponding curated statements for a corpus. If a reader name is provided, the results are filtered to curations for statements that have the provided reader among its sources, otherwise all curations and their corresponding statements are returned.

```
URL: http://SERVICE_HOST:8001/download_curations
Method: POST with JSON content header
Input parameters: {"corpus_id": "<corpus id>",
                   "reader": "<reader name>"}
Output: {"curations": <list of curations>,
         "statements": {"38ce0c14-2c7e-4df8-bd53-3006afeaa193": <stmt json>}}
```

### 1.1.8 Notify INDRA of a new reader output in DART

```
URL: http://SERVICE_HOST:8001/notify
Method: POST with JSON content header
Input parameters: {"identity": "eidos",
                   "version": "3.1.4",
                   "document_id": "38ce0c14-2c7e-4df8-bd53-3006afeaa193",
                   "storage_key": "uuid.ext"}
Output: {}
```

## 1.2 INDRA assemblies on S3

Access to the INDRA-assembled corpora requires credentails 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 ("corpus_id" in the requests above).

### 1.2.1 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.

### 1.2.2 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. As of May 2020, each statement's JSON representation is on a separate line in this file. Any corpus uploaded before that has a standard JSON structure. This is the main file that CauseMos needs to ingest for UI interaction.
- *raw_statements.json*: a JSON dump of raw INDRA Statements. This file is typically not needed in downstream usage, however, the INDRA curation service needs to have access to it for internal assembly tasks.
- *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.

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

- *curations.json*: a JSON file which persists curations as collected by INDRA. This is the basis of surfacing reader-specific curations in the download_curation endpoint (see above).

## 1.3 Setting up the services locally

These instructions describe setting up and using the INDRA service stack for World Modelers applications, in particular, as a back-end for the CauseMos UI.

The instructions below run each Docker container with the `-d` option which will run containers in the background. You can list running containers with their ids using `docker ps` and stop a container with `docker stop <container id>`.

### 1.3.1 Setting up the Eidos service

Clone the Eidos repo and cd to the Docker folder

```
git clone https://github.com/clulab/eidos.git
cd eidos/Docker
```

Build the Eidos docker image

```
docker build -f DockerfileRunProd . -t eidos-webservice
```

Run the Eidos web service and expose it on port 9000

```
docker run -id -p 9000:9000 eidos-webservice
```

### 1.3.2 Setting up the general INDRA service

Pull the INDRA docker image from DockerHub

```
docker pull labsyspharm/indra
```

Run the INDRA web service and expose it on port 8000

```
docker run -id -p 8000:8080 --entrypoint gunicorn labsyspharm/indra:latest \
    -w 1 -b :8000 -t 600 rest_api.api:app
```

Note that the `-w 1` parameter specifies one service worker which can be set to a higher number if needed.

### 1.3.3 Setting up the INDRA live curation service

Assuming you already have the INDRA docker image, run the INDRA live feedback service with the following parameters:

```
docker run -id -p 8001:8001 --env-file docker_variables --entrypoint \
python labsyspharm/indra /sw/indra/indra/tools/live_curation/live_curation.py
```

Here we use the tag `--env-file` to provide a file containing environment variables to the docker. In this case, we need to provide `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` to allow the curation service to access World Modelers corpora on S3. The file content should look like this:

```
AWS_ACCESS_KEY_ID=<aws_access_key_id>
AWS_SECRET_ACCESS_KEY=<aws_secret_access_key>
```

Replace `<aws_access_key_id>` and `<aws_secret_access_key>` with your aws access and secret keys.

# CHAPTER 2

## INDRA WM service modules

**exception** indra_wm_service.**InvalidCorpusError**

**class** indra_wm_service.curator.**LiveCurator**(*scorer=None*, *corpora=None*, *eidos_url=None*, *ont_manager=None*, *cache=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/indra-wm-service/checkouts/latest/indra_wm_service/_local_cache')*)

Class coordinating the real-time curation of a corpus of Statements.

**Parameters**

- **scorer** (*indra.belief.BeliefScorer*) – A scorer object to use for the curation
- **corpora** (*dict[str, Corpus]*) – A dictionary mapping corpus IDs to Corpus objects.

**get_corpus**(*corpus_id*, *check_s3=True*, *use_cache=True*)

Return a corpus given an ID.

If the corpus ID cannot be found, an InvalidCorpusError is raised.

**Parameters**

- **corpus_id** (*str*) – The ID of the corpus to return.
- **check_s3** (*bool*) – If True, look on S3 for the corpus if it's not currently loaded. Default: True
- **use_cache** (*bool*) – If True, look in local cache before trying to find corpus on s3. If True while check_s3 if False, this option will be ignored. Default: False.

**Returns** The corpus with the given ID.

**Return type** *Corpus*

**get_curations**(*corpus_id*, *reader=None*)

Download curations for corpus id filtered to reader

**Parameters**

- **corpus_id** (*str*) – The ID of the corpus to download curations from

- **reader** (*str*) – The name of the reader to filter to. Has to be among valid reader names of 'all'.

**Returns** A dict containing the requested curations

**Return type** dict

**reset_scorer**()
Reset the scorer used for curation.

**save_curations**(*corpus_id*, *save_to_cache=True*)
Save the current state of curations for a corpus given its ID

If the corpus ID cannot be found, an InvalidCorpusError is raised.

**Parameters**

- **corpus_id** (*str*) – the ID of the corpus to save the
- **save_to_cache** (*bool*) – If True, also save the current curation to the local cache. Default: True.

**submit_curations**(*curations*, *save=True*)
Submit correct/incorrect curations fo a given corpus.

**Parameters**

- **curations** (*list of dict*) – A list of curationss.
- **save** (*bool*) – If True, save the updated curations to the local cache. Default: True

**update_beliefs**(*corpus_id*, *project_id=None*)
Return updated belief scores for a given corpus.

**Parameters** **corpus_id** (*str*) – The ID of the corpus for which beliefs are to be updated.

**Returns** A dictionary of belief scores with keys corresponding to Statement UUIDs and values to new belief scores.

**Return type** dict

**update_metadata**(*corpus_id*, *meta_data*, *save_to_cache=True*)
Update the meta data for a given corpus

**Parameters**

- **corpus_id** (*str*) – The ID of the corpus to update the meta data for
- **meta_data** (*dict*) – A json compatible dict containing the meta data
- **save_to_cache** (*bool*) – If True, also update the local cache of the meta data dict. Default: True.

**class** indra_wm_service.corpus.**Corpus**(*corpus_id*, *statements=None*, *raw_statements=None*, *meta_data=None*, *aws_name='wm'*, *cache=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/indra-wm-service/checkouts/latest/indra_wm_service/_local_cache')*)
Represent a corpus of statements with curation.

**Parameters**

- **corpus_id** (*str*) – The key by which the corpus is identified.
- **statements** (*list[indra.statement.Statement]*) – A list of INDRA Statements to embed in the corpus.

- **raw_statements** (*list[indra.statement.Statement]*) – A List of raw statements forming the basis of the statements in 'statements'.
- **meta_data** (*dict*) – A dict with meta data associated with the corpus
- **aws_name** (*str*) – The name of the profile in the AWS credential file to use. 'default' is used by default.
- **cache** (*Pathlib.path*) – A Pathlib.path object representing the path to a cache folder.

**statements**
A dict of INDRA Statements keyed by UUID.

**Type** dict

**raw_statements**
A list of the raw statements

**Type** list

**curations**
A list keeping track of the curations submitted so far for Statements in the corpus.

**Type** list

**get_curations**(*look_in_cache=False*)
Get curations for the corpus

**Parameters** **look_in_cache** (*bool*) – If True, look in local cache if there are no curations loaded

**Returns** The curations for this corpus, if any

**Return type** dict

**s3_get**(*bucket='world-modelers'*, *cache=True*, *raise_exc=False*)
Fetch a corpus object from S3 in the form of three json files

The json files representing the object have S3 keys of the format <s3key>/statements.json and <s3key>/raw_statements.json.

**Parameters**

- **bucket** (*str*) – The S3 bucket to fetch the Corpus from. Default: 'world-modelers'.
- **cache** (*bool*) – If True, look for corpus in local cache instead of loading it from s3. Default: True.
- **raise_exc** (*bool*) – If True, raise InvalidCorpusError when corpus failed to load

**s3_put**(*bucket='world-modelers'*, *cache=True*)
Push a corpus object to S3 in the form of three json files

The json files representing the object have S3 keys of the format <key_base_name>/<name>/<file>.json

**Parameters**

- **bucket** (*str*) – The S3 bucket to upload the Corpus to. Default: 'world-modelers'.
- **cache** (*bool*) – If True, also create a local cache of the corpus. Default: True.

**Returns** **keys** – A tuple of three strings giving the S3 key to the pushed objects

**Return type** tuple(str)

**save_curations_to_cache**()
Save current curations to cache

**upload_curations**(*look_in_cache=False*, *save_to_cache=False*, *bucket='world-modelers'*)
Upload the current state of curations for the corpus

**Parameters**

- **look_in_cache** (*bool*) – If True, when no curations are available check if there are curations cached locally. Default: False
- **save_to_cache** (*bool*) – If True, also save current curation state to cache. If look_in_cache is True, this option will have no effect. Default: False.
- **bucket** (*str*) – The bucket to upload to. Default: 'world-modelers'.

CHAPTER 3

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## i

# Index