

---

# **py-processors Documentation**

***Release 3.2.1***

**Gus Hahn-Powell**

**Oct 03, 2017**



---

## Contents

---

<b>1</b>	<b>Overview</b>	<b>1</b>
<b>2</b>	<b>Annotating text</b>	<b>3</b>
<b>3</b>	<b>API Reference</b>	<b>5</b>
<b>4</b>	<b>Odin</b>	<b>7</b>
<b>5</b>	<b>API Reference</b>	<b>9</b>
<b>6</b>	<b>A walkthrough example</b>	<b>17</b>
<b>7</b>	<b>Running the NLP server</b>	<b>19</b>
<b>8</b>	<b>Jupyter notebook visualizations</b>	<b>25</b>
<b>9</b>	<b>Other ways of initializing the server</b>	<b>27</b>
<b>10</b>	<b>Keeping the server running</b>	<b>29</b>
<b>11</b>	<b>FAQ</b>	<b>31</b>
<b>12</b>	<b>Changes</b>	<b>33</b>
<b>13</b>	<b>What is it?</b>	<b>37</b>
<b>14</b>	<b>Requirements</b>	<b>39</b>
<b>15</b>	<b>Installation</b>	<b>41</b>
<b>16</b>	<b>How to use it?</b>	<b>43</b>



# CHAPTER 1

---

## Overview

---

This page provides an overview of the available text processing pipelines.



## CHAPTER 2

---

### Annotating text

---

a `.annotate` call produces a Document with the following features:

- sentence segmentation
- tokenization
- part-of-speech tagging
- lemmatization
- named entity recognition
- syntactic dependency parsing

Annotation suitable to open domain text can be performed with either a `.annotate` or a `.fastnlp.annotate` call:

```
API = ProcessorsAPI(port=8886)
# annotate the provided text using FastNLProcessor (a CoreNLP wrapper)
doc = API.fastnlp.annotate("My name is Inigo Montoya. You killed my father. Prepare_
↳to die.")
```

For annotation tuned to the biomedical domain, use `.bionlp.annotate`:

```
API = ProcessorsAPI(port=8886)
doc = API.bionlp.annotate("In contrast, the EGFR T669A mutant increased both basal_
↳EGFR and ERBB3 tyrosine phosphorylation that was not augmented by MEK inhibition.")
```





## CHAPTER 3

---

### API Reference

---

See *the API reference* for more details.



Refer to the [jupyter notebook](#) for a tutorial on using Odin from `py-processors`.

The most comprehensive reference for the Odin event extraction framework is our manual, which includes a detailed description of our rule language (`rune`):

- <http://arxiv.org/pdf/1509.07513v1.pdf>

### Planned additions to Odin

- Support in rules for comparisons using distributional semantic similarity (word vectors)
- Efficient matching gazetteers
- WordNet support in patterns
- Edit distance



This section of the documentation provides detailed information on functions, classes, and methods.

## Server Communication

Communicating with the NLP server (`processors-server`) is handled by the following classes.

### ProcessorsBaseAPI

```
class processors.api.ProcessorsBaseAPI (**kwargs)
    Bases: object
```

Manages a connection with processors-server and provides an interface to the API.

#### Parameters

- **port** (*int*) – The port the server is running on or should be started on. Default is 8886.
- **hostname** (*str*) – The host name to use for the server. Default is “localhost”.
- **log\_file** (*str*) – The path for the log file. Default is `py-processors.log` in the user’s home directory.

#### **annotate** (*text*)

Produces a Document from the provided *text* using the default processor.

#### `clu`.**annotate** (*text*)

Produces a Document from the provided *text* using CluProcessor.

#### `fastnlp`.**annotate** (*text*)

Produces a Document from the provided *text* using FastNLPPProcessor.

#### `bionlp`.**annotate** (*text*)

Produces a Document from the provided *text* using BioNLPPProcessor.

**annotate\_from\_sentences** (*sentences*)

Produces a Document from *sentences* (a list of text split into sentences). Uses the default processor.

**fastnlp.annotate\_from\_sentences** (*sentences*)

Produces a Document from *sentences* (a list of text split into sentences). Uses FastNLPProcessor.

**bionlp.annotate\_from\_sentences** (*sentences*)

Produces a Document from *sentences* (a list of text split into sentences). Uses BioNLPProcessor.

**corenlp.sentiment.score\_sentence** (*sentence*)

Produces a sentiment score for the provided *sentence* (an instance of Sentence).

**corenlp.sentiment.score\_document** (*doc*)

Produces sentiment scores for the provided *doc* (an instance of Document). One score is produced for each sentence.

**corenlp.sentiment.score\_segmented\_text** (*sentences*)

Produces sentiment scores for the provided *sentences* (a list of text segmented into sentences). One score is produced for item in *sentences*.

**odin.extract\_from\_text** (*text*, *rules*)

Produces a list of Mentions for matches of the provided *rules* on the *text*. *rules* can be a string of Odin rules, or a url ending in *.yml* or *.yaml*.

**odin.extract\_from\_document** (*doc*, *rules*)

Produces a list of Mentions for matches of the provided *rules* on the *doc* (an instance of Document). *rules* can be a string of Odin rules, or a url ending in *.yml* or *.yaml*.

## ProcessorsAPI

**class** `processors.api.ProcessorsAPI` (*\*\*kwargs*)

Bases: `processors.api.ProcessorsBaseAPI`

Manages a connection with the processors-server jar and provides an interface to the API.

### Parameters

- **timeout** (*int*) – The number of seconds to wait for the server to initialize. Default is 120.
- **jvm\_mem** (*str*) – The maximum amount of memory to allocate to the JVM for the server. Default is “-Xmx3G”.
- **jar\_path** (*str*) – The path to the processors-server jar. Default is the jar installed with the package.
- **kee\_alive** (*bool*) – Whether or not to keep the server running when ProcessorsAPI instance goes out of scope. Default is false (server is shut down).
- **log\_file** (*str*) – The path for the log file. Default is `py-processors.log` in the user’s home directory.

**start\_server** (*jar\_path*, *\*\*kwargs*)

Starts the server using the provided *jar\_path*. Optionally takes *hostname*, *port*, *jvm\_mem*, and *timeout*.

**stop\_server** ()

Attempts to stop the server running at `self.address`.

## OdinAPI

`class processors.api.OdinAPI(address)`

Bases: object

API for performing rule-based information extraction with Odin.

**Parameters** `address` (*str*) – The base address for the API (i.e., everything preceding */api/..*)

## SentimentAnalysisAPI

`class processors.sentiment.SentimentAnalysisAPI(address)`

Bases: object

API for performing sentiment analysis

**Parameters** `address` (*str*) – The base address for the API (i.e., everything preceding */api/..*)

**corenlp**

*processors.sentiment.CoreNLPSentimentAnalyzer* – Service using [CoreNLP's tree-based system]([https://nlp.stanford.edu/~socherr/EMNLP2013\\_RNTN.pdf](https://nlp.stanford.edu/~socherr/EMNLP2013_RNTN.pdf)) for performing sentiment analysis.

## Data Structures

### Document

`class processors.ds.Document(sentences)`

Bases: object

Storage class for annotated text. Based on [*org.clulab.processors.Document*](<https://github.com/clulab/processors/blob/master/main/src/main/scala/org/clulab/processors/Document.scala>)

**Parameters** `sentences` (*[processors.ds.Sentence]*) – The sentences comprising the *Document*.

**id**

*str or None* – A unique ID for the *Document*.

**size**

*int* – The number of *sentences*.

**sentences**

*sentences* – The sentences comprising the *Document*.

**words**

*[str]* – A list of the *Document*'s tokens.

**tags**

*[str]* – A list of the *Document*'s tokens represented using part of speech (PoS) tags.

**lemmas**

*[str]* – A list of the *Document*'s tokens represented using lemmas.

**\_entities**

*[str]* – A list of the *Document*'s tokens represented using IOB-style named entity (NE) labels.

**nes**

*dict* – A dictionary of NE labels represented in the *Document* -> a list of corresponding text spans.

**bag\_of\_labeled\_deps**

*[str]* – The labeled dependencies from all sentences in the *Document*.

**bag\_of\_unlabeled\_deps**

*[str]* – The unlabeled dependencies from all sentences in the *Document*.

**text**

*str or None* – The original text of the *Document*.

**bag\_of\_labeled\_dependencies\_using** (*form*)

Produces a list of syntactic dependencies where each edge is labeled with its grammatical relation.

**bag\_of\_unlabeled\_dependencies\_using** (*form*)

Produces a list of syntactic dependencies where each edge is left unlabeled without its grammatical relation.

## Sentence

```
class processors.ds.Sentence (**kwargs)
```

Bases: object

Storage class for an annotated sentence. Based on `[org.clulab.processors.Sentence]`(<https://github.com/clulab/processors/blob/master/main/src/main/scala/org/clulab/processors/Sentence.scala>)

**Parameters**

- **text** (*str or None*) – The text of the *Sentence*.
- **words** (*[str]*) – A list of the *Sentence*'s tokens.
- **startOffsets** (*[int]*) – The character offsets starting each token (inclusive).
- **endOffsets** (*[int]*) – The character offsets marking the end of each token (exclusive).
- **tags** (*[str]*) – A list of the *Sentence*'s tokens represented using part of speech (PoS) tags.
- **lemmas** (*[str]*) – A list of the *Sentence*'s tokens represented using lemmas.
- **chunks** (*[str]*) – A list of the *Sentence*'s tokens represented using IOB-style phrase labels (ex. *B-NP*, *I-NP*, *B-VP*, etc.).
- **entities** (*[str]*) – A list of the *Sentence*'s tokens represented using IOB-style named entity (NE) labels.
- **graphs** (*dict*) – A dictionary of {graph-name -> {edges: [{source, destination, relation}], roots: [int]}}

**text**

*str* – The text of the *Sentence*.

**startOffsets**

*[int]* – The character offsets starting each token (inclusive).

**endOffsets**

*[int]* – The character offsets marking the end of each token (exclusive).

**length**

*int* – The number of tokens in the *Sentence*

**graphs**

*dict* – A dictionary (*str* -> *processors.ds.DirectedGraph*) mapping the graph type/name to a *processors.ds.DirectedGraph*.



**basic\_dependencies**

*processors.ds.DirectedGraph* – A *processors.ds.DirectedGraph* using basic Stanford dependencies.

**collapsed\_dependencies**

*processors.ds.DirectedGraph* – A *processors.ds.DirectedGraph* using collapsed Stanford dependencies.

**dependencies**

*processors.ds.DirectedGraph* – A pointer to the preferred syntactic dependency graph type for this *Sentence*.

**\_entities**

[*str*] – The IOB-style Named Entity (NE) labels corresponding to each token.

**\_chunks**

[*str*] – The IOB-style chunk labels corresponding to each token.

**nes**

*dict* – A dictionary of NE labels represented in the *Document* -> a list of corresponding text spans (ex. {"PERSON": [phrase 1, ..., phrase n]}). Built from *Sentence.\_entities*

**phrases**

*dict* – A dictionary of chunk labels represented in the *Document* -> a list of corresponding text spans (ex. {"NP": [phrase 1, ..., phrase n]}). Built from *Sentence.\_chunks*

**bag\_of\_labeled\_dependencies\_using** (*form*)

Produces a list of syntactic dependencies where each edge is labeled with its grammatical relation.

**bag\_of\_unlabeled\_dependencies\_using** (*form*)

Produces a list of syntactic dependencies where each edge is left unlabeled without its grammatical relation.

## DirectedGraph

**class** *processors.ds.DirectedGraph* (*kind, deps, words*)

Bases: object

Storage class for directed graphs.

**Parameters**

- **kind** (*str*) – The name of the directed graph.
- **deps** (*dict*) – A dictionary of {edges: [{source, destination, relation}], roots: [int]}
- **words** ([*str*]) – A list of the word form of the tokens from the originating *Sentence*.

**\_words**

[*str*] – A list of the word form of the tokens from the originating *Sentence*.

**roots**

[*int*] – A list of indices for the syntactic dependency graph's roots. Generally this is a single token index.

**edges**

*list[processors.ds.Edge]* – A list of *processors.ds.Edge*

**incoming**

A dictionary of {*int* -> [*int*]} encoding the incoming edges for each node in the graph.

**outgoing**

A dictionary of {*int* -> [*int*]} encoding the outgoing edges for each node in the graph.

**labeled**

*[str]* – A list of strings where each element in the list represents an edge encoded as source index, relation, and destination index (“source\_relation\_destination”).

**unlabeled**

*[str]* – A list of strings where each element in the list represents an edge encoded as source index and destination index (“source\_destination”).

**graph**

*networkx.Graph* – A *networkx.graph* representation of the *DirectedGraph*. Used by *shortest\_path*

**bag\_of\_labeled\_dependencies\_from\_tokens** (*form*)

Produces a list of syntactic dependencies where each edge is labeled with its grammatical relation.

**bag\_of\_unlabeled\_dependencies\_from\_tokens** (*form*)

Produces a list of syntactic dependencies where each edge is left unlabeled without its grammatical relation.

## Mention

```
class processors.odin.Mention(token_interval, sentence, document, foundBy, label, labels=None,
                              trigger=None, arguments=None, paths=None, keep=True,
                              doc_id=None)
```

Bases: object

A labeled span of text. Used to model textual mentions of events, relations, and entities.

**Parameters**

- **token\_interval** (*Interval*) – The span of the Mention represented as an *Interval*.
- **sentence** (*int*) – The sentence index that contains the Mention.
- **document** (*Document*) – The Document in which the Mention was found.
- **foundBy** (*str*) – The Odin IE rule that produced this Mention.
- **label** (*str*) – The label most closely associated with this span. Usually the lowest hyponym of “labels”.
- **labels** (*list*) – The list of labels associated with this span.
- **trigger** (*dict or None*) – dict of JSON for Mention’s trigger (event predicate or word(s) signaling the Mention).
- **arguments** (*dict or None*) – dict of JSON for Mention’s arguments.
- **paths** (*dict or None*) – dict of JSON encoding the syntactic paths linking a Mention’s arguments to its trigger (applies to Mentions produces from *type:”dependency”* rules).
- **doc\_id** (*str or None*) – the id of the document

**tokenInterval**

*processors.ds.Interval* – An *Interval* encoding the *start* and *end* of the *Mention*.

**start**

*int* – The token index that starts the *Mention*.

**end**

*int* – The token index that marks the end of the *Mention* (exclusive).

**sentenceObj**

*processors.ds.Sentence* – Pointer to the *Sentence* instance containing the *Mention*.

**characterStartOffset**

*int* – The index of the character that starts the *Mention*.

**characterEndOffset**

*int* – The index of the character that ends the *Mention*.

**type**

*Mention.TBM* or *Mention.EM* or *Mention.RM* – The type of the *Mention*.

**See also:**

[Odin manual](<https://arxiv.org/abs/1509.07513>)

**matches** (*label\_pattern*)

Test if the provided pattern, *label\_pattern*, matches any element in *Mention.labels*.

JSON serialization/deserialization is handled via `processors.serialization.JSONSerializer`.

## Interval

**class** `processors.ds.Interval` (*start*, *end*)

Bases: `object`

Defines a token or character span

**Parameters**

- **start** (*str*) – The token or character index where the interval begins.
- **end** (*str*) – The 1 + the index of the last token/character in the span.

## DependencyUtils

**class** `processors.paths.DependencyUtils`

Bases: `object`

A set of utilities for analyzing syntactic dependency graphs.

**build\_networkx\_graph** (*roots*, *edges*, *name*)

Constructs a `networkx.Graph`

**shortest\_path** (*g*, *start*, *end*)

Finds the shortest path in a `networkx.Graph` between any element in a list of start nodes and any element in a list of end nodes.

**retrieve\_edges** (*dep\_graph*, *path*)

Converts output of *shortest\_path* into a list of triples that include the grammatical relation (and direction) for each node-node “hop” in the syntactic dependency graph.

**simplify\_tag** (*tag*)

Maps part of speech (PoS) tag to a subset of PoS tags to better consolidate categorical labels.

**lexicalize\_path** (*sentence*, *path*, *words=False*, *lemmas=False*, *tags=False*, *simple\_tags=False*, *entities=False*)

Lexicalizes path in syntactic dependency graph using Odin-style token constraints.

**pagerank** (*networkx\_graph*, *alpha=0.85*, *personalization=None*, *max\_iter=1000*, *tol=1e-06*, *nstart=None*, *weight='weight'*, *dangling=None*)

Measures node activity in a `networkx.Graph` using a thin wrapper around `networkx` implemen-

tation of pagerank algorithm (see *networkx.algorithms.link\_analysis.pagerank*). Use with *processors.ds.DirectedGraph.graph*.

## Annotators (Processors)

Text annotation is performed by communicating with one of the following annotators (“processors”).

### CluProcessor

**class** `processors.annotators.CluProcessor` (*address*)

Bases: `processors.annotators.Processor`

Processor for text annotation based on [*org.clulab.processors.clu.CluProcessor*](<https://github.com/clulab/processors/blob/master/main/src/main/scala/org/clulab/processors/clu/CluProcessor.scala>)

Uses the Malt parser.

### FastNLPProcessor

**class** `processors.annotators.FastNLPProcessor` (*address*)

Bases: `processors.annotators.Processor`

Processor for text annotation based on [*org.clulab.processors.fastnlp.FastNLPProcessor*](<https://github.com/clulab/processors/blob/master/corenlp/src/main/scala/org/clulab/processors/fastnlp/FastNLPProcessor.scala>)

Uses the Stanford CoreNLP neural network parser.

### BioNLPProcessor

**class** `processors.annotators.BioNLPProcessor` (*address*)

Bases: `processors.annotators.Processor`

Processor for biomedical text annotation based on [*org.clulab.processors.fastnlp.FastNLPProcessor*](<https://github.com/clulab/processors/blob/master/corenlp/src/main/scala/org/clulab/processors/fastnlp/FastNLPProcessor.scala>)

CoreNLP-derived annotator.

## Sentiment Analysis

### Serialization

**class** `processors.serialization.JSONSerializer`

Bases: `object`

Utilities for serialization/deserialization of data structures.

---

### A walkthrough example

---

The following examples give an overview of how to use `py-processors`.

#### Getting started

For annotation and parsing, `py-processors` communicates with `processors-server` using a REST interface. The server can be run either via `java` directly or in a `docker container`. Let's look at how to connect to the server.



---

## Running the NLP server

---

### Option 1: processors-server.jar

This method requires `java` and a compatible `processors-server.jar` for the server. An appropriate `jar` will be downloaded automatically if one is not found.

```
from processors import *
# The constructor requires you to specify a port for running the server.
API = ProcessorsAPI(port=8886)
```

*NOTE: It may take a minute or so for the server to initialize as there are some large model files that need to be loaded.*

### Option 2: docker container

You can pull the official container from [Docker Hub](#):

```
docker pull myedibleenso/processors-server:latest
```

You can check `py-processors` for the appropriate version to retrieve:

```
import processors
# print the recommended processors-server version
print(import processors.__ps_rec__)
```

Just replace `latest` in the command above with the appropriate version (`3.1.0` onwards).

The following command will run the container in the background and expose the service on port `8886`:

```
docker run -d -p 127.0.0.1:8886:8888 --name procserv myedibleenso/processors-server
```

For a more detailed example showcasing configuration options take a look at [this docker-compose.yml file](#). You'll need to map a local port to `8888` in the container.

Once the container is running, you can connect to it via `py-processors`:

```
from processors import *
# provide the local port that you mapped to 8888 on the running container
API = ProcessorsBaseAPI(port=8886)
```

## Annotating text

Text can be annotated automatically with these linguistic attributes.

```
# try annotating some text using FastNLPProcessor (a CoreNLP wrapper)
doc = API.fastnlp.annotate("My name is Inigo Montoya. You killed my father. Prepare_
↳to die.")

# you can also annotate text already segmented into sentences
doc = API.fastnlp.annotate_from_sentences(["My name is Inigo Montoya.", "You killed_
↳my father.", "Prepare to die."])

# There should be 3 Sentence objects in this Document
doc.size

# A Document contains the words, pos tags, lemmas, named entities, and syntactic_
↳dependencies of its component Sentences
doc.bag_of_labeled_deps

# We can access the named entities for the Document as a dictionary mapping an NE_
↳label -> list of named entities
doc.nes

# A Sentence contains words, pos tags, lemmas, named entities, and syntactic_
↳dependencies
doc.sentences[0].lemmas

# get the first sentence
s = doc.sentences[0]

# the number of tokens in this sentence
s.length

# the named entities contained in this sentence
s.nes

# generate labeled dependencies using "words", "tags", "lemmas", "entities", or token_
↳index ("index")
s.bag_of_labeled_dependencies_using("tags")

# generate unlabeled dependencies using "words", "tags", "lemmas", "entities", or_
↳token index ("index")
s.bag_of_unlabeled_dependencies_using("lemmas")

# play around with the dependencies directly
deps = s.dependencies

# see what dependencies lead directly to the first token (i.e. token 0 is the_
↳dependent of what?)
deps.incoming[0]
```



```

# see what dependencies are originating from the first token (i.e. token 0 is the
↳head of what?)
deps.outgoing[0]

# find all shortest paths between "name" and either "Inigo" or "Montoya".
deps.shortest_paths(start=1, end=[3,4])

# find the shortest path between "name" and either "Inigo" or "Montoya". Prefer a
↳path that involves a "nsubj" relation.
sp = deps.shortest_path(start=1, end=[3,4],
scoring_func=lambda path: 9000 if any(seg[1] == "nsubj" for seg in path) else 0)

# generate an Odin-like pattern with partial lexicalization
DependencyUtils.lexicalize_path(sentence=s, path=sp, lemmas=True, tags=True)

# limit lexicalization to tokens 1 and 4 (if present)
DependencyUtils.lexicalize_path(sentence=s, path=sp, lemmas=True, tags=True, limit_
↳to=[1,4])

# run PageRank on the dependency graph to find nodes with the most activity.
# SPOILER: When using reverse=True, the nodes with the highest weight are usually the
↳sentential predicate and its args
deps.pagerank(reverse=True)

# find out which nodes are most central to the dependency graph
deps.degree centrality()

# retrieve the likely semantic head for a sentence.
from processors.paths import HeadFinder
doc2 = API.annotate("acute renal failure")
sentence = doc2.sentences[0]
# select the graph to examine (default is "stanford-collapsed") and
# optionally limit to a set of PoS tags (regex or str)
head_idx = sentence.semantic_head(graph_name="stanford-collapsed", valid_tags=None)
head_word = sentence.words[head_idx] if head_idx else None

# try using BioNLPPProcessor
biodoc = api.bionlp.annotate("We next considered the effect of Ras monoubiquitination
↳on GAP-mediated hydrolysis")

# check out the bio-specific entities
biodoc.nes

```

## Serializing to/from json

Once you've annotated text, you can serialize it to json for later loading.

```

# serialize to/from JSON!
json_file = "serialized_doc_example.json"
ross_doc = api.fastnlp.annotate("We don't make mistakes, just happy little accidents.
↳")

# serialize to JSON
with open(json_file, "w") as out:

```

```
out.write(ross_doc.to_JSON())

# load from JSON
with open(json_file, "r") as jf:
    d = Document.load_from_JSON(json.load(jf))
```

## Perform sentiment analysis

You can perform sentiment analysis using CoreNLP's tree-based system.

```
# get sentiment analysis scores
review = "The humans are dead."
doc = API.fastnlp.annotate(review)

# try Stanford's tree-based sentiment analysis
# you'll get a score for each Sentence
# scores are between 1 (very negative) - 5 (very positive)
scores = API.sentiment.corenlp.score_document(doc)

# you can pass text directly
scores = API.sentiment.corenlp.score_text(review)

# ... or a single sentence
score = API.sentiment.corenlp.score_sentence(doc.sentences[0])

# ... or from text already segmented into sentences
lyrics = ["My sugar lumps are two of a kind", "Sweet and white and highly refined",
↪ "Honeys try all kinds of tomfoolery", "to steal a feel of my family jewelry"]
scores = API.sentiment.corenlp.score_segmented_text(lyrics)
```

## Rule-based information extraction (IE) with Odin

If you're unfamiliar with writing Odin rules, see our manual for a primer on the language: <http://arxiv.org/pdf/1509.07513v1.pdf>

```
# Do rule-based IE with Odin!
# see http://arxiv.org/pdf/1509.07513v1.pdf for details
example_rule = """
- name: "ner-person"
  label: [Person, PossiblePerson, Entity]
  priority: 1
  type: token
  pattern: |
    [entity="PERSON"]+
    |
    [tag=/^N/]* [tag=/^N/ & outgoing="cop"] [tag=/^N/]*
  """

example_text = """
Barack Hussein Obama II is the 44th and current President of the United States and
↪ the first African-American to hold the office.
He is a Democrat.
Obama won the 2008 United States presidential election, on November 4, 2008.
```

```

He was inaugurated on January 20, 2009.
"""

# take a look at the .label, .labels, and .text attributes of each mention
mentions = API.odin.extract_from_text(example_text, example_rule)
# visualize the structure of a mention as colored output in the terminal
for m in mentions: print(m)

# Alternatively, you can provide a rule URL. The URL should end with .yaml or .yml.
rules_url = "https://raw.githubusercontent.com/clulab/reach/
↳508697db2217ba14cd1fa0a99174816cc3383317/src/main/resources/edu/arizona/sista/demo/
↳open/grammars/rules.yml"

mentions = API.odin.extract_from_text(example_text, rules_url)

# You can also perform IE with Odin on a Document.
barack_doc = API.annotate(example_text)
mentions = API.odin.extract_from_document(barack_doc, rules_url)

# mentions can be serialized as well
mentions_json_file = "mentions.json"

with open(mentions_json_file, "w") as out:
    out.write(JSONSerializer.mentions_to_JSON(mentions))

# loading from a file is also handled via JSONSerializer
with open(mentions_json_file, "r") as jf:
    mentions = JSONSerializer.mentions_from_JSON(json.load(jf))

```



---

## Jupyter notebook visualizations

---

py-processors supports some custom notebook-based visualizations, but you'll need to install the extra [jupyter] module in order to use them:

```
pip install "py-processors[jupyter]"
```

These visualizations make use of our fork of [displaCy](#). You can now visualize a Sentence graph as an SVG image using `visualization.JupyterVisualizer.display_graph()`:

```
from processors.visualization import JupyterVisualizer as viz
# run this snippet within a jupyter notebook
text = "To be loved by unicorns is the greatest gift of all."
doc = API.annotate(text)
viz.display_graph(doc.sentences[0], graph_name="stanford-collapsed")
```

Mentions can also be visualized in a notebook:

```
# run this snippet within a jupyter notebook
rules = """
rules:
- name: "ner-location"
  label: [Location, PossibleLocation, Entity]
  priority: 1
  type: token
  pattern: |
    [entity="LOCATION"]+ | Twin Peaks

- name: "ner-person"
  label: [Person, PossiblePerson, Entity]
  priority: 1
  type: token
  pattern: |
    [entity="PERSON"]+

- name: "ner-org"
  label: [Organization, Entity]
```

```
priority: 1
type: token
pattern: |
    [entity="ORGANIZATION"]+

- name: "ner-date"
  label: [Date]
  priority: 1
  type: token
  pattern: |
    [entity="DATE"]+

- name: "missing"
  label: Missing
  pattern: |
    trigger = [lemma=go] missing
    theme: Person = <xcomp nsubj
    date: Date? = prep_on
"""
mentions = API.odin.extract_from_text("FBI Special Agent Dale Cooper went missing on_
↪June 10, 1991. He was last seen in the woods of Twin Peaks. ", rules=rules)

for m in mentions: viz.display_mention(m)
```

---

## Other ways of initializing the server

---

### Using a custom processors-server

When initializing the API, you can specify a path to a custom `processors-server.jar` using the `jar_path` parameter:

```
from processors import *  
  
API = ProcessorsAPI(port=8886, jar_path="path/to/processors-server.jar")
```

Alternatively, you can set an environment variable, `PROCESSORS_SERVER`, with the path to the `jar` you wish to use. In your `.bashrc` (or equivalent), add this line with the path to the `jar` you wish to use with `py-processors`:

```
export PROCESSORS_SERVER="path/to/processors-server.jar"
```

Remember to source your profile:

```
source path/to/your/.profile
```

`py-processors` will now prefer this `jar` whenever a new API is initialized.

*NOTE: If you decide that you no longer want to use this environment variable, remember to both remove it from your profile and run `unset PROCESSORS_SERVER` from the shell.*

### Allocating memory

By default, the server will be run with 3GB of RAM. You might be able to get by with a little less, though. You can start the server with a different amount of memory with the `jvm_mem` parameter:

```
from processors import *  
# run the sever with 2GB of memory  
API = ProcessorsAPI(port=8886, jvm_mem="-Xmx2G")
```

*NOTE: This won't have any effect if the server is already running on the given port.*



## CHAPTER 10

---

### Keeping the server running

---

If you've launched the server via `java, py-processors` will by default attempt to shut down the server whenever an API instance goes out of scope (ex. your script finishes or you exit the interpreter).

If you'd prefer to keep the server alive, you'll need to initialize the API with `keep_alive=True`:

```
from processors import *  
  
API = ProcessorsAPI(port=8886, keep_alive=True)
```

This is useful if you're sharing access to the server on a network, or if you have a bunch of independent tasks and would prefer to avoid waiting for the server to initialize again and again.



### I want the latest `processors-server.jar`

In that case, take a look over [here](#).

### Something is already running on port `xxxx`, but I don't know what. Help!

Try running the following command:

```
lsof -i :<portnumber>
```

You can then kill the responsible process using the reported PID

### Does `py-processors` produce a (server) log?

Yep! By default, it will write to `~/py-processors.log`. You can specify a different location when initializing the API:

```
from processors import *  
  
API = ProcessorsAPI(port=8886, log_file="my/desired/log/file.log")
```



# CHAPTER 12

---

## Changes

---

- v3.2.0:
  - ProcessorsAPI now inherits from ProcessorsBaseAPI
  - ProcessorsBaseAPI can be used with a `docker backend`
  - Updated documentation
  - Updated requirements for building documentation
  - More tests covering syntactic dependencies
- v3.1.0:
  - Upgraded processors-server version to v3.1.0
  - Added support for CluProcessor
  - `odin` variables can now be used in imports
  - Log file no longer prefixed with `.`
  - `jupyter notebook` visualizations now treated as an extra module that can be installed via `pip install "py-processors[jupyter]"`
- v3.0.3:
  - `jupyter notebook` visualizations for Sentence graphs (dependency parses) and Mention structure
    - \* `visualization.JupyterVisualizer.display_graph()`
    - \* `visualization.Jupyter.Visualizer.display_mention()`
  - Display errors for broken Odin rules
  - Upgraded processors-server version to v3.0.2
  - Added `HeadFinder` and `HeadFinder.semantic_head()`
  - `pagerank` bug fix related to outdated parameter
  - Make `Sentence`, `DirectedGraph`, and `Mention` hashable

- v3.0.2:
  - Added support to reverse directed graph for pagerank, resulting in predicate and argument nodes having higher weights
  - `DependencyUtils.directed_relation()`
  - Added `DirectedGraph.degree centrality()`, `DirectedGraph.in_degree centrality()`, and `DirectedGraph.out_degree centrality()`
  - pagerank now returns a `collections.Counter`
  - Optionally limit `DependencyUtils.lexicalize_path()` to specific token indices
- v3.0.1:
  - Fixed an off-by-one error for character offsets when creating mentions
- v3.0.0:
  - PageRank for dependency graphs via `networkx`
  - Find shortest paths in dependency graphs via `networkx`
  - Updated api to match v.3.0 of `processors-server`
  - Support for Odin `@Mention. argname`
  - json compatible with `org.clulab.processors`
- v2.9.7:
  - .nes support for IOB format
- v2.9.6:
  - Simplified path checks for jar
- v2.9.5:
  - Docstring for `ProcessorsAPI`
  - Updated Odin rules url for tests
  - Check version of jar against recommended version
  - Cleanup (kwargs usage, path resolution, etc.)
- v2.9.4:
  - jar is now downloaded when first used
- v2.9.3:
  - Pass hostname to command for starting server
  - Fixed an Odin test and added needed resources
- v2.9.2:
  - Added `.stop_server()` test
- v2.9.1:
  - Create `Mention` from `Mention.trigger` when `trigger` is not `None`
- v2.9:
  - `Mention.arguments` bug fix related to creating `Mentions` for each arg corresponding to each role
  - `Dependencies.incoming` and `Dependencies.outgoing` bug fixes

- Implemented custom **eq** and **ne** for core data structures
- v2.8:
  - Dependencies bug fix related to initializing from json
  - Added `keep_alive` boolean parameter to `ProcessorsAPI` constructor to provide a way to keep the server running when instance goes out of scope
  - Compatibility fixes for 2.x
  - Updated api to match v.2.7 of `processors-server`
    - \* handle pre-segmented text (preserve provided sentence segmentation in `.annotate` and `.sentiment.corenlp.score_segmented_text` calls)
- v2.7:
  - Added `Mention` class and support for rule-based information extraction with Odin
  - Updated api to match v.2.5 of `processors-server`
- v2.6:
  - Added interface to CoreNLP's tree-based sentiment analysis
  - Rewrote json serialization and loading to mirror changes in `processors-server` v2.2
- v2.4:
  - Added support for json serialization
- v2.1:
  - Added interface to `BioNLPPProcessor`
  - Download latest `processors-server.jar` as part of installation
- v1.0:
  - Basic functionality (interface to `FastNLPPProcessor`)





## CHAPTER 13

---

### What is it?

---

`py-processors` is a Python wrapper for the CLU Lab's `processors` NLP library. `py-processors` relies on `processors-server`.

Though `compatible*` with Python 2.x, this library was developed with 3.x in mind.



The server component can be run either via `docker` or directly with `java`.

### Option 1

- `docker` and the `myedibleenso/processors-server` container

### Option 2

- Java 8
- `processor-sever` (v3.1.0)
  - this dependency will be retrieved automatically during installation
- At least 2GB of RAM free for the server (I recommend 3GB+)



# CHAPTER 15

---

## Installation

---

`py-processors` can be installed via `pip`. The library also has a `jupyter extras` module which adds widgets/visualizations to `jupyter` notebooks.

### basic installation

```
pip install py-processors
```

### basic + jupyter notebook widgets

```
pip install py-processors[jupyter]
```

### bleeding edge

```
pip install git+https://github.com/myedibleenso/py-processors.git
```



## CHAPTER 16

---

How to use it?

---

See *the walkthrough example*





## Symbols

`_chunks` (Sentence attribute), 13  
`_entities` (Document attribute), 11  
`_entities` (Sentence attribute), 13  
`_words` (DirectedGraph attribute), 13

## A

`annotate()` (ProcessorsBaseAPI method), 9  
`annotate()` (ProcessorsBaseAPI.bionlp method), 9  
`annotate()` (ProcessorsBaseAPI.clu method), 9  
`annotate()` (ProcessorsBaseAPI.fastnlp method), 9  
`annotate_from_sentences()` (ProcessorsBaseAPI method), 9  
`annotate_from_sentences()` (ProcessorsBaseAPI.bionlp method), 10  
`annotate_from_sentences()` (ProcessorsBaseAPI.fastnlp method), 10

## B

`bag_of_labeled_dependencies_from_tokens()` (DirectedGraph method), 14  
`bag_of_labeled_dependencies_using()` (Document method), 12  
`bag_of_labeled_dependencies_using()` (Sentence method), 13  
`bag_of_labeled_deps` (Document attribute), 11  
`bag_of_unlabeled_dependencies_from_tokens()` (DirectedGraph method), 14  
`bag_of_unlabeled_dependencies_using()` (Document method), 12  
`bag_of_unlabeled_dependencies_using()` (Sentence method), 13  
`bag_of_unlabeled_deps` (Document attribute), 12  
`basic_dependencies` (Sentence attribute), 12  
`BioNLPPProcessor` (class in processors.annotators), 16  
`build_networkx_graph()` (DependencyUtils method), 15

## C

`characterEndOffset` (Mention attribute), 15

`characterStartOffset` (Mention attribute), 14  
`CluProcessor` (class in processors.annotators), 16  
`collapsed_dependencies` (Sentence attribute), 13  
`corenlp` (SentimentAnalysisAPI attribute), 11

## D

`dependencies` (Sentence attribute), 13  
`DependencyUtils` (class in processors.paths), 15  
`DirectedGraph` (class in processors.ds), 13  
`Document` (class in processors.ds), 11

## E

`edges` (DirectedGraph attribute), 13  
`end` (Mention attribute), 14  
`endOffsets` (Sentence attribute), 12  
`extract_from_document()` (ProcessorsBaseAPI.odin method), 10  
`extract_from_text()` (ProcessorsBaseAPI.odin method), 10

## F

`FastNLPPProcessor` (class in processors.annotators), 16

## G

`graph` (DirectedGraph attribute), 14  
`graphs` (Sentence attribute), 12

## I

`id` (Document attribute), 11  
`incoming` (DirectedGraph attribute), 13  
`Interval` (class in processors.ds), 15

## J

`JSONSerializer` (class in processors.serialization), 16

## L

`labeled` (DirectedGraph attribute), 13  
`lemmas` (Document attribute), 11  
`length` (Sentence attribute), 12

lexicalize\_path() (DependencyUtils method), 15

## M

matches() (Mention method), 15

Mention (class in processors.odin), 14

## N

nes (Document attribute), 11

nes (Sentence attribute), 13

## O

OdinAPI (class in processors.api), 11

outgoing (DirectedGraph attribute), 13

## P

pagerank() (DependencyUtils method), 15

phrases (Sentence attribute), 13

ProcessorsAPI (class in processors.api), 10

ProcessorsBaseAPI (class in processors.api), 9

## R

retrieve\_edges() (DependencyUtils method), 15

roots (DirectedGraph attribute), 13

## S

score\_document() (ProcessorsBaseAPI.corenlp.sentiment method), 10

score\_segmented\_text() (ProcessorsBaseAPI.corenlp.sentiment method), 10

score\_sentence() (ProcessorsBaseAPI.corenlp.sentiment method), 10

Sentence (class in processors.ds), 12

sentenceObj (Mention attribute), 14

sentences (Document attribute), 11

SentimentAnalysisAPI (class in processors.sentiment), 11

shortest\_path() (DependencyUtils method), 15

simplify\_tag() (DependencyUtils method), 15

size (Document attribute), 11

start (Mention attribute), 14

start\_server() (ProcessorsAPI method), 10

startOffsets (Sentence attribute), 12

stop\_server() (ProcessorsAPI method), 10

## T

tags (Document attribute), 11

text (Document attribute), 12

text (Sentence attribute), 12

tokenInterval (Mention attribute), 14

type (Mention attribute), 15

## U

unlabeled (DirectedGraph attribute), 14

## W

words (Document attribute), 11