# NLIMED

Natural Language Interface for Model Entity Discovery (NLIMED) is an interface to search model entity (i.e. flux of sodium across basolateral plasma membrane, concentration of potassium in the portion of tissue fluid) from the collection of biosimulation models in repository. The interface utilise the RDF inside biosimulation models and metadata from OBO Library. Currently, the interface only retrieves model entities from the Physiome Repository Model (PMR), but in the future it will also show model entities from other repositories.

In general, NLIMED works by converting natural language query into SPARQL, so it may help researcher to avoid the rigid syntax of SPARQL, query path consisting multiple predicates, and detail description of class ontologies.

## General instruction

1. Download or clone all files in a new folder
2. This interface implements three parset types, Stanford Parser, NLTK Parser, and BIO Portal parser.

- If you want to utilise BIO Portal parser, you should get an apikey through https://bioportal.bioontology.org/help#Getting_an_API_key and setup the apikey in Settings.py file

```
apikey = 'put your apikey here'
```

- If you want to utilise Stanford Parser, you should download jar files of stanford-corenlp and stanford-english-corenlp-models. These files can be downloaded from https://stanfordnlp.github.io/CoreNLP/download.html, as a full zip package, then extract the zip file to your new folder. Based on stanford-corenlp version downloaded, you will find a different folder and files' name. For example, in this works, we use stanford-corenlp-full-2018-10-05.zip, so it will create stanford-corenlp-full-2018-10-05 folder. The corresponded stanford-corenlp file is stanford-corenlp-3.9.2.jar and stanford-english-corenlp-models file is stanford-english-corenlp-2018-10-05-models.jar. Then, modify QueryAnnotator.py file at line 182, 184, and 185, set these to your your downloaded file names. Here are the lines you should modify:

```
STANFORD = os.path.join(".", "stanford-corenlp-full-2018-10-05") #folder name
    server = CoreNLPServer(
       os.path.join(STANFORD, "stanford-corenlp-3.9.2.jar"), #corenlp file name
       os.path.join(STANFORD, "stanford-english-corenlp-2018-10-05-models.jar"),
#model file name
    )
```

## Running NLIMED (query to get model entities)

In order to get model entities from the PMR you can run the NLIMED.py from command prompt or terminal. For example, for query 'flux of sodium', the command should be:

- with Stanford parser:

  ```
  python NLIMED.py –stanford 'flux of sodium'
  ```

  Note: running with Stanford parser will cause delay local server setup for the first run. However, for the next run

the delay will be disappeared.

- with NLTK parser:

```
python NLIMED.py –nltk 'flux of sodium'
```

- with OBO Library parser:

```
python NLIMED.py –obo 'flux of sodium'
```

Note: running with OBO Library parser is slower than other parsers, because it is using a webservice depended on the Internet connection.

# Utilising code in NLIMED for your works

## Annotate Query

- with Stanford parser:

```
# StanforAnnotator(topConsider, settings)
# param:
#   - topConsider (mandatory) = minimal value is 1, indicating the number of
considered ontology class for each phrase
#   - settings (optional) = is a **kvargs, setting for multiplier coefficient of
metadata to measure the weight of candidate ontology classes
#        - m_prefDef    = multiplier value of Preferred Definition metadata
#        - m_synonym    = multiplier value of Synonym metadata
#        - m_definition = multiplier value of Definition metadata
#        - m_mention    = multiplier value of local metadata
query = 'flux of sodium through basolateral plasma membrane'
annotator = StanfordAnnotator(1)
annResult = annotator.annotate(query)

# another example
query = 'flux of sodium through basolateral plasma membrane'
annotator = StanfordAnnotator(1, m_prefDef = 2, m_synonym = 1, m_definition = 1,
m_mention = 1)
annResult = annotator.annotate(query)
```

- with NLTK parser:

```
# NLTKAnnotator(topConsider, settings)
# param:
#   - topConsider (mandatory) = minimal value is 1, indicating the number of
considered ontology class for each phrase
#   - settings (optional) = is a **kvargs, setting for multiplier coefficient of
metadata to measure the weight of candidate ontology classes
#        - m_prefDef    = multiplier value of Preferred Definition metadata
#        - m_synonym    = multiplier value of Synonym metadata
#        - m_definition = multiplier value of Definition metadata
```

```
#        - m_mention    = multiplier value of local metadata
query = 'concentration of potassium in the portion of tissue fluid'
annotator = NLTKAnnotator(1)
annResult = annotator.annotate(query)

# another example
query = 'concentration of potassium in the portion of tissue fluid'
annotator = NLTKAnnotator(1, m_prefDef = 2, m_synonym = 1, m_definition = 1,
m_mention = 1)
annResult = annotator.annotate(query)
```

- with OBO Library parser:

```
# OBOLIBAnnotator(topConsider, settings)
# param:
#   - topConsider (mandatory) = minimal value is 1, indicating the number of
considered ontology class for each phrase
query = 'concentration of potassium in the portion of tissue fluid'
annotator = OBOLIBAnnotator(1)
annResult = annotator.annotate(query)
```

The structure of annResult is same for all annotator. Here is the result of StanfordAnnotator:

```
Type:        dict
String form: {'phrases': ['concentration', 'potassium', 'portion tissue
fluid'], 'result': [[['http://identifiers.org/opb/OPB_00340',
'http://purl.obolibrary.org/obo/CHEBI_29103',
'http://purl.obolibrary.org/obo/FMA_9673'], 5.0555421807834975]]}
Length:      2
Docstring:
dict() -> new empty dictionary
dict(mapping) -> new dictionary initialized from a mapping object's
    (key, value) pairs
dict(iterable) -> new dictionary initialized as if via:
    d = {}
    for k, v in iterable:
        d[k] = v
dict(**kwargs) -> new dictionary initialized with the name=value pairs
    in the keyword argument list.  For example:  dict(one=1, two=2)
```

## Construct SPARQL and Execute SPARQL

```
sg = SPARQLGenerator()
if len(annResult) > 0:
    annTerm = annResult['result'][0]
    sparqlSet = sg.constructSPARQL(*annTerm[0])
    # the constructed SPARQL can be 0 or more
    for sparql in sparqlSet:
        # parse each SPARQL to the Physiome Model Repository
        results = sg.runSparQL(sparql)
        sg.print_results(results,'flat')
```

# Recreate Indexes (RDF Graph Index and Text Feature Index)

## Build SPARQL index from the beginning, started by collecting data from the PMR

```
"""COLLECT CELLML AND CREATE INDEX FOR SPARQL"""
import IndexSPARQL
idxSparql = IndexSPARQL()
idxSparql.buildIndex('-build')
```

If rdf from the PMR is already collected,

```
"""COLLECT CELLML AND CREATE INDEX FOR SPARQL"""
import IndexSPARQL
idxSparql = IndexSPARQL()
idxSparql.buildIndex('-skip','getcellmlLinkContent')
```

## Build annotator index

```
"""COLLECT OBOLIBRARY DATA AND CREATE INDEX FOR ANNOTATION"""
import IndexAnnotation
idxAnno = IndexAnnotation()
idxAnno.collectOboAttributes()
idxAnno.developInvertedIndex()
```