



CLAMP

A Guide to the Extraction of
Clinical Concepts

Reference
Manual

Table of Contents

1. Introduction	3
2. System Requirements	4
3. Installation	6
4. How to run CLAMP	6
5. Package Description	7
6. Import existing projects into the new version	9
7. Contact us	9
8. NLP Components	10
8.1 NLP Components	10
8.2 Sentence Detector	11
8.3 Tokenizer	15
8.4 Pos Tagger	18
8.5 Chunker	19
8.6 Named Entity Recognizer	20
8.7 Assertion Identifier	25
8.8 Ruta_Rule_Engine	27
8.9 Section Identifier	28
8.10 UMLS Encoder	30
8.11 User_Defined_Components	31
9. Machine Learning components	33
9.1 NER Feature Extractor	33
9.1.1 DF_Brown_clustering_feature	34
9.1.2 DF_Dictionary_lookup_feature	35
9.1.3 DF_Ngram_feature	37
9.1.4 DF_prefix_suffix_feature	37
9.1.5 DF_Random_indexing_feature	37
9.1.6 DF_Section_feature	38dicted

9.1.7 DF_Sentence_pattern_feature	38
9.1.8 DF_Word_embedding_feature	38
9.1.9 DF_Word_shape_feature	39
9.1.10 DF_Words_regular_expression_feature	39
10. Build a Pipeline	41
10.1 Create and Run a Pipeline	41
10.2 Configure the pipeline	43
10.3 Component dependency & Auto fix	44
10.4 Import input files	46
11. Run the pipeline	50
12. Output visualization	51
13. Built-in pipelines	53
14. Export pipeline as a jar file	57
15. Annotation	58
15.1 Annotate corpus	58
15.1.1 Create a project	58
15.1.2 Import annotation files	60
15.1.3 Define entity & relation types	63
15.1.4 Start Annotation	67
15.1.5 Visualization of entity & relation	68
15.1.6 Pre-Annotation of entity and relation	69
16. Machine learning model development	71
16.1 Building machine learning models (NER model)	71
16.2 Check output models & logs	74
16.3 Use your own model in pipeline	75
16.4 Visualization for error analysis	77

Clinical Language Annotation Modeling and Processing Toolkit System

Version 1.1.7

1. Introduction

The CLAMP System is a comprehensive clinical Natural Language Processing software that enables recognition and automatic encoding of clinical information in narrative patient reports. In addition to running a clinical concept extraction pipeline as well as an annotation pipeline, the individual components of the system can also be used as independent modules. The system lends itself for diverse applications in a broad range of clinical domains. The high performance language processing framework in CLAMP consists of the following key building blocks:

NLP Pipelines

CLAMP components builds on a set of high performance NLP components that were proven in several clinical NLP challenges such as i2b2 , ShARe/CLEF , and SemEVAL. A pipeline can be created and customized by a simple drag and drop on the individual CLAMP components in the order that is desired. Upon creation of the pipeline, CLAMP checks for errors in sequence and directs the user to the appropriate logical order with insertion of the required components for a working pipeline. The CLAMP components are supported by knowledge resources consisting of medical abbreviations, dictionaries, section headers, and a corpus of 400 annotated clinical notes derived from MTsamples, a freely available resource of clinical narrative text. CLAMP also provides built-in pipelines ready for use out of the box for a series of common clinical applications.

Machine learning and hybrid approaches

The CLAMP framework provides alternative components for some tasks, utilizing rule based methods and/or machine learning methods such as support vector machines, and conditional random fields. These components can be customized by re-training on an annotated corpus, or editing the rule sets within the CLAMP GUI to achieve a custom NLP task. The CLAMP GUI version also provides built-in functionality to test the model, using the annotated corpora or n-fold cross validation.

Corpus management and annotation tool:

The user interface also provides required tools to maintain and annotate text corpora. It hosts an improved version of the brat annotation tool (reference?) for textual annotations.

2. System Requirements

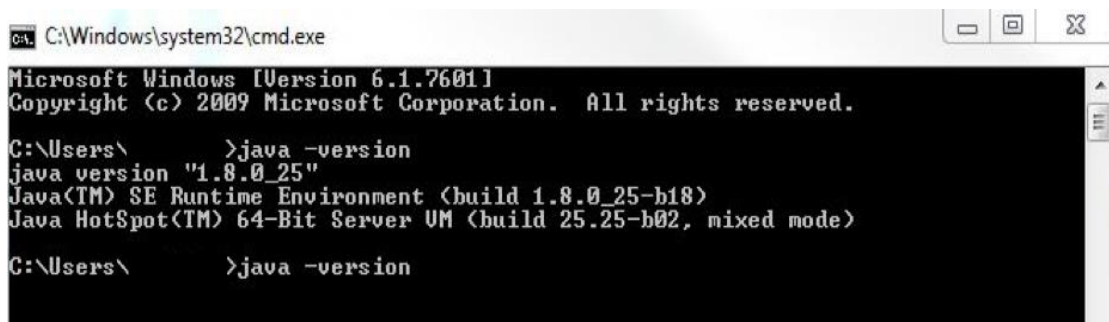
CLAMP is a stand-alone Java application based on the Eclipse platform technologies. CLAMP uses the Apache UIMA (Unstructured Information Management Architecture) framework. The annotation module of CLAMP incorporates and enhances the brat rapid annotation tool . For the other individual constituents, Apache OpenNLP toolkit, Liblinear and CRF Suite are utilized in addition to in-house rule-based components. CLAMP also use the UIMA Ruta (Rule based Text Annotation) as a rule engine to help users specify rules.

CLAMP is distributed as a ready-to-use binary package that can either be executed at the command line or carries the associated Graphic User Interface (GUI). Our distribution package includes components for jar files, CRFSuite, and a Lucene index of all levels of UMLS data.

The only prerequisite necessary to compile CLAMP is JRE 1.8 (Java Runtime Environment). Please ensure that you have Java 8 or higher installed in your system. Run the following command in both Mac and Windows to check your version:

```
java -version
```

Here is an example of what you will see when running the command in Windows:



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\ >java -version
java version "1.8.0_25"
Java(TM) SE Runtime Environment (build 1.8.0_25-b18)
Java HotSpot(TM) 64-Bit Server VM (build 25.25-b02, mixed mode)

C:\Users\ >java -version
```

If your java version is not 1.8, it is available for download from the Oracle website at <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>. An UMLS account is required in order to use Level 2 and higher data in the UMLS encoding component of the system. The account can be created at <https://uts.nlm.nih.gov/home.html>. You will have to enter your UMLS username and password when prompted by CLAMP in order to utilise the UMLS encoding component.

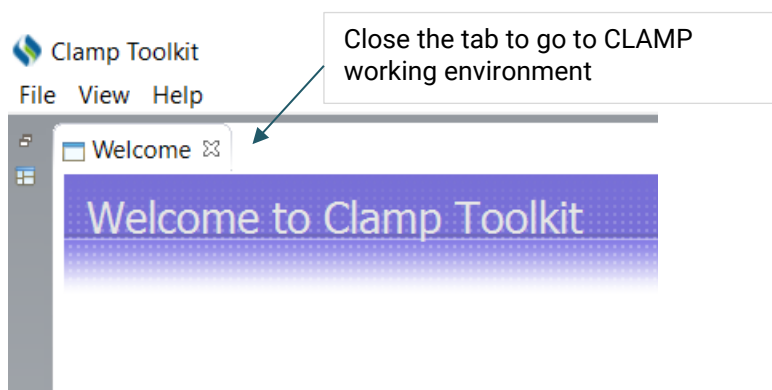
CLAMP also uses the computer's default browsers to visualize the clinical documents. Since all browsers do not completely support all the aspects of the technologies used to implement the visualization, limitations exist in term of running the CLAMP annotation module in the browsers. On the Windows OS, the Internet Explorer should be higher than IE9; On Macintosh computers, Safari (all versions) works well with CLAMP.

3. Installation

The CLAMP System is provided as a .zip file. After downloading the compressed file, unzip the package in the directory of choice and voila!! the system is ready for use. Installation instructions are the same for both Windows and Macintosh computers. For the CLAMP command line version please refer to the readme file. For further information and troubleshooting, please refer CLAMP website at <http://clamp.uth.edu>.

4. How to run CLAMP

You can run the GUI version of CLAMP by double clicking on the startCLAMP icon. Once the software is completely loaded, you will notice a welcome tab. Close the tab to go to CLAMP working environment.



5. Package Description

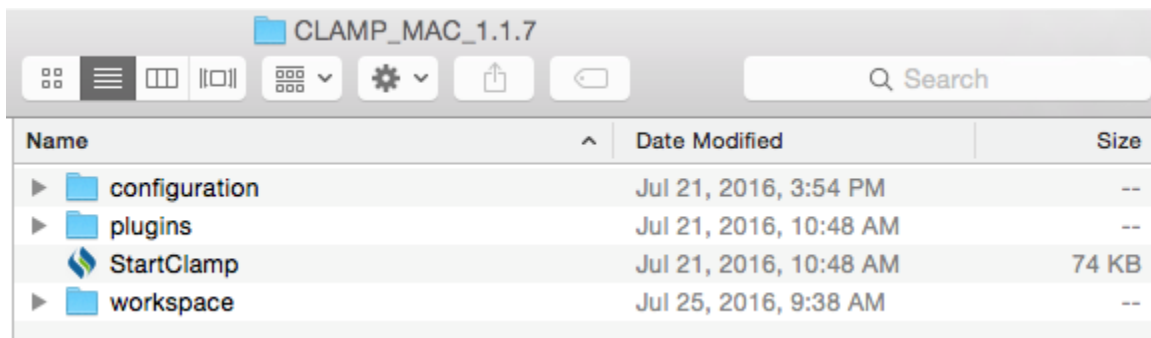
Since CLAMP is a stand-alone eclipse plugin, its folder structure is similar to other eclipse plugins.

Configuration Folder:

This folder contains CLAMP configuration files.

StartCLAMP:

This is the launching point for the CLAMP GUI. In Windows, this is an executable file while in Mac, this is an application.



Workspace Folder:

This folder contains seven sub-folders:

1. **ComponentLibrary**: contains the components used in machine learning feature extraction and NLP functions.
2. **MyCorpus**: contains the customized corpus built by the users.
3. **MyPipeline**: contains the customized pipeline created by users for clinical notes processing.
4. **PipelineLibrary**: contains the built-in pipelines ready to use for a series of common clinical applications.
5. **Log**: Includes CLAMP run-time log files
6. **Metadata**: The metadata used by CLAMP are included in this folder.
7. **Resources**: This folder includes third-party libraries. Currently it has two items:
 - 7.1 CRFSuite: the CRF implementation for Name Entity Recognition tasks

7.2 Umls_index: the Lucene index built for CLAMP based on the UMLS thesaurus. If you want to use UMLS terminologies, then you will need to create an UMLS account. Please follow the following link to create an UMLS account if you do not have any.

7.3 <https://uts.nlm.nih.gov//license.html>

The following table lists libraries included in CLAMP.

groupid	artifactId	version
org.cleartk	cleartk-ml-liblinear	2.0.0
org.ini4j	ini4j	0.5.2
org.apache.uima	uimafit-core	2.1.0
com.google.code.gson	gson	2.3
org.apache.uima	<u>uimaj</u> -core	2.6.0
org.apache.uima	uimaj-cpe	2.6.0
org.apache.uima	<u>uimaj</u> -document-annotation	2.6.0
de.bwaldvogel	liblinear	1.94
org.apache.lucene	lucene-core	5.2.1
org.apache.lucene	<u>lucene</u> -analyzers-common	5.2.1
org.apache.lucene	lucene-queryparser	5.2.1
org.apache.opennlp	opennlp-tools	1.5.1-incubating
org.apache.ctakes	<u>ctakes</u> -type-system	3.2.0
org.cleartk	<u>cleartk</u> -named-entity	0.6.6
com.googlecode.clearnlp	clearnlp	1.3.1
commons- <u>codec</u>	commons- <u>codec</u>	20041127.091804
dom4j	dom4j	1.6.1
org.apache.uima	<u>ruta</u> -ep-engine	2.3.0
javax.servlet	servlet-api	3.0-alpha-1
com.sun.jersey	jersey-client	1.19
junit	junit	4.12
commons- <u>cli</u>	commons- <u>cli</u>	1.3
net.jodah	concurrentunit	0.4.2
org.javatuples	javatuples	1.2

6. Import existing projects into the new version

On Windows, simply copy contents of your previous work folder (*i.e. from older `Clamp_x.xx.xx_win\workspace\MyPipeline\` contents to new `Clamp_x.xx.xx_win\workspace\MyPipeline\` contents*) using **Windows Explorer** and **restart CLAMP** if it's already running. On startup, CLAMP will recognize these projects and import them into your workspace.

On MacOSX, similarly copy contents of your previous work folder (*i.e. from older `Clamp_x.xx.xx_win/workspace/MyPipeline/` contents to new `Clamp_x.xx.xx_win/workspace/MyPipeline/` contents*) using **Finder** and **restart CLAMP** if it's already running. On startup, CLAMP will recognize these projects and import them into your new workspace.

7. Contact us

The CLAMP System was developed by Dr. Hua Xu's team from the School of Biomedical Informatics at the University of Texas Health Science Center in Houston.

For technical issues, please contact: Jingqi.Wang@uth.tmc.edu

For any other issues, please contact: Anupama.E.Gururaj@uth.tmc.edu

8. NLP Components

8.1 NLP Components

NLP components are used for processing text. CLAMP offers multiple NLP components that are the building blocks for performing any of the NLP tasks. The individual components are provided as pre-built modules that are to be used in building pipelines for automatic text processing, as well as training customized machine learning models. Figure 8.1 displays the CLAMP NLP components, as well as its associated tools, algorithms and resources. In this section, we will provide details about each NLP component including their function as well as ways to customize them using your own dictionary/model. In [“Build a Pipeline”](#) section, we have tutorials that touch on use cases wherein the components are utilized in various applications.

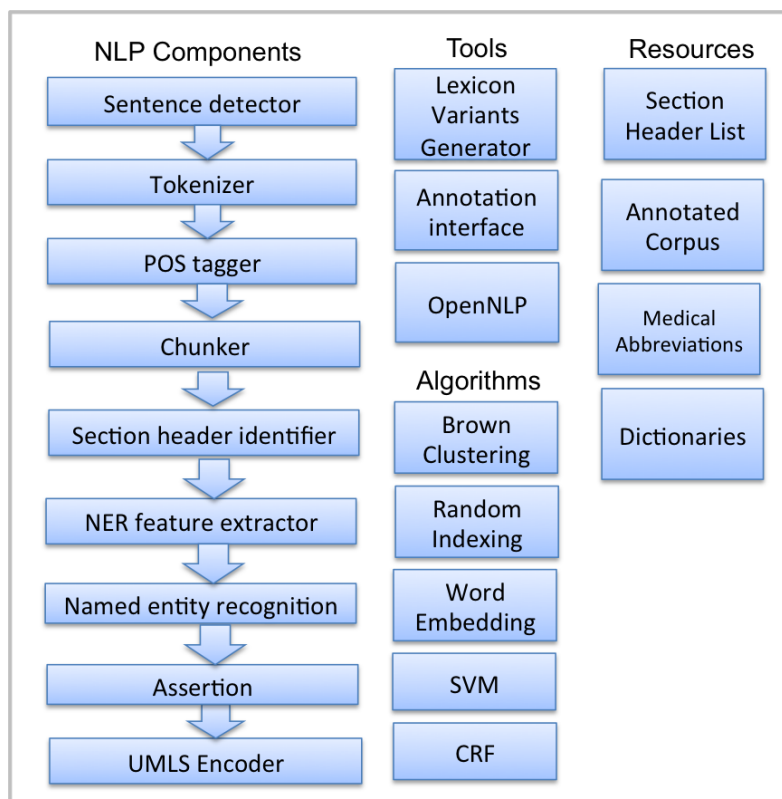


Figure 8.1 Schema of NLP Components

8.2 Sentence Detector

A sentence is defined as the longest whitespace trimmed character sequence between two punctuation marks. A Sentence Detector utilizes different methods to detect a sentence. As shown in Figure 8.2, CLAMP provides three different models to detect a sentence:

1. **DF_CLAMP_Sentence_Detector**
2. **DF_CLAMP_Sentence_by_newline**, and
3. **DF_CLAMP_OpenNLP_sentence_detector**

Each model is described in detail in the following sections.

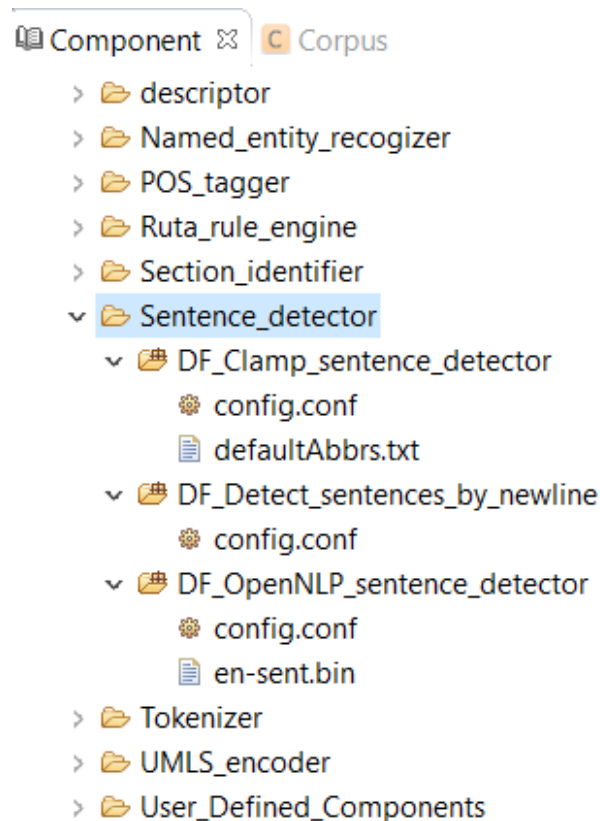


Figure 8.2 Three sentence detectors and their configuration files

1. DF_CLAMP_Sentence_Detector:

DF_CLAMP_Sentence_Detector is the default sentence detector in CLAMP. It is designed specifically for clinical notes and takes into account the distinctive characteristics observed in sentences found in clinical texts.

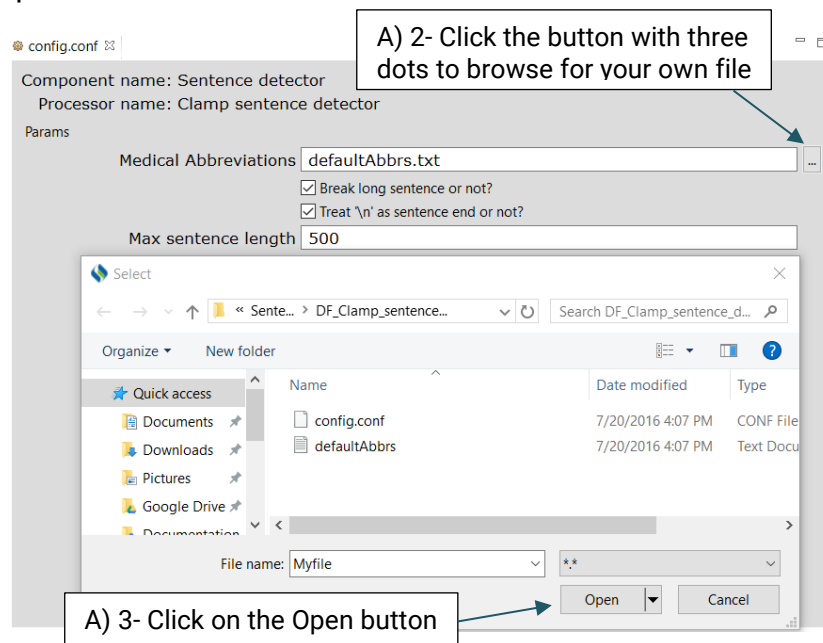
To configure the DF_CLAMP_Sentence_Detector, please click on the config file. A pop-up window opens where you can modify two parameters: **Medical Abbreviation**, and **Max Sentence Length**.

Medical Abbreviation:

There are some medical abbreviations that have punctuation marks at their beginning (".NO2) while some of them have it at the end (spec.). Providing a list of such abbreviations would help the detector to identify sentences more accurately. By default, CLAMP has provided a comprehensive list of medical abbreviation which can be found in this file: *defaultAbbrs.txt*

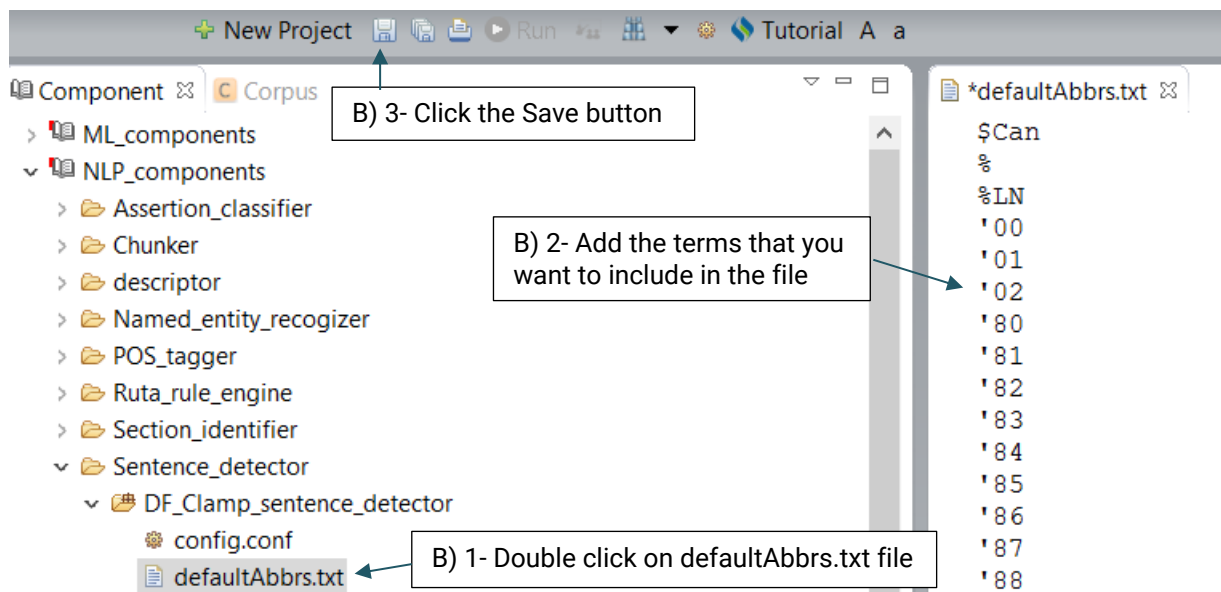
A) To replace the abbreviation file:

1. Double click on config.conf file to open it
2. Click on the button with three dots to browse for your own file
3. Click on the open button



B) To edit the current file:

1. Double click on the defaultAbbrs.txt file to open it
2. Add the terms that you want to include in the abbreviation file
3. Click on the Save button on the toolbar



Max Sentence Length

Checking the checkbox for “Break long sentences or not?” allows users to break long sentences into the number of words that they have specified in the input textbox. Please refer to Figure 8.3 for more information.

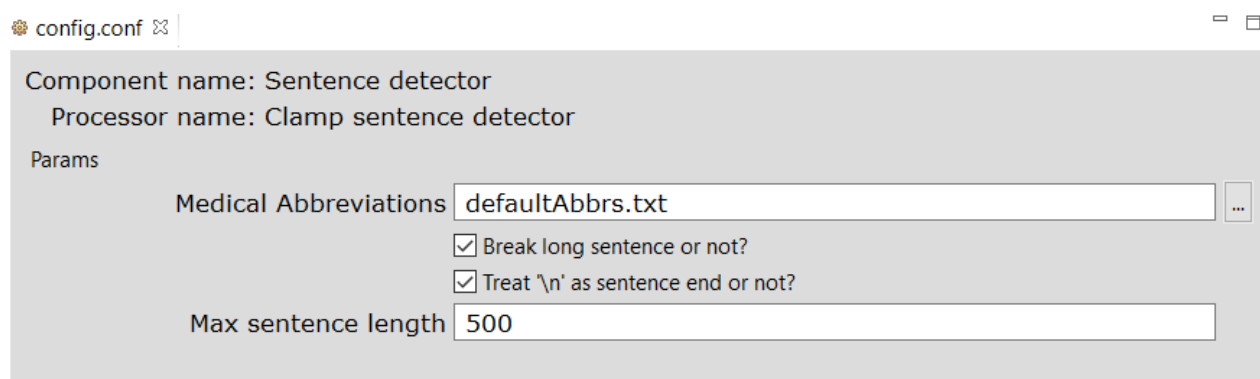


Figure 8.3 Interface for config.conf of the DF_CLAMP_Sentence_Detector

2. DF_CLAMP_Sentence_by_newline

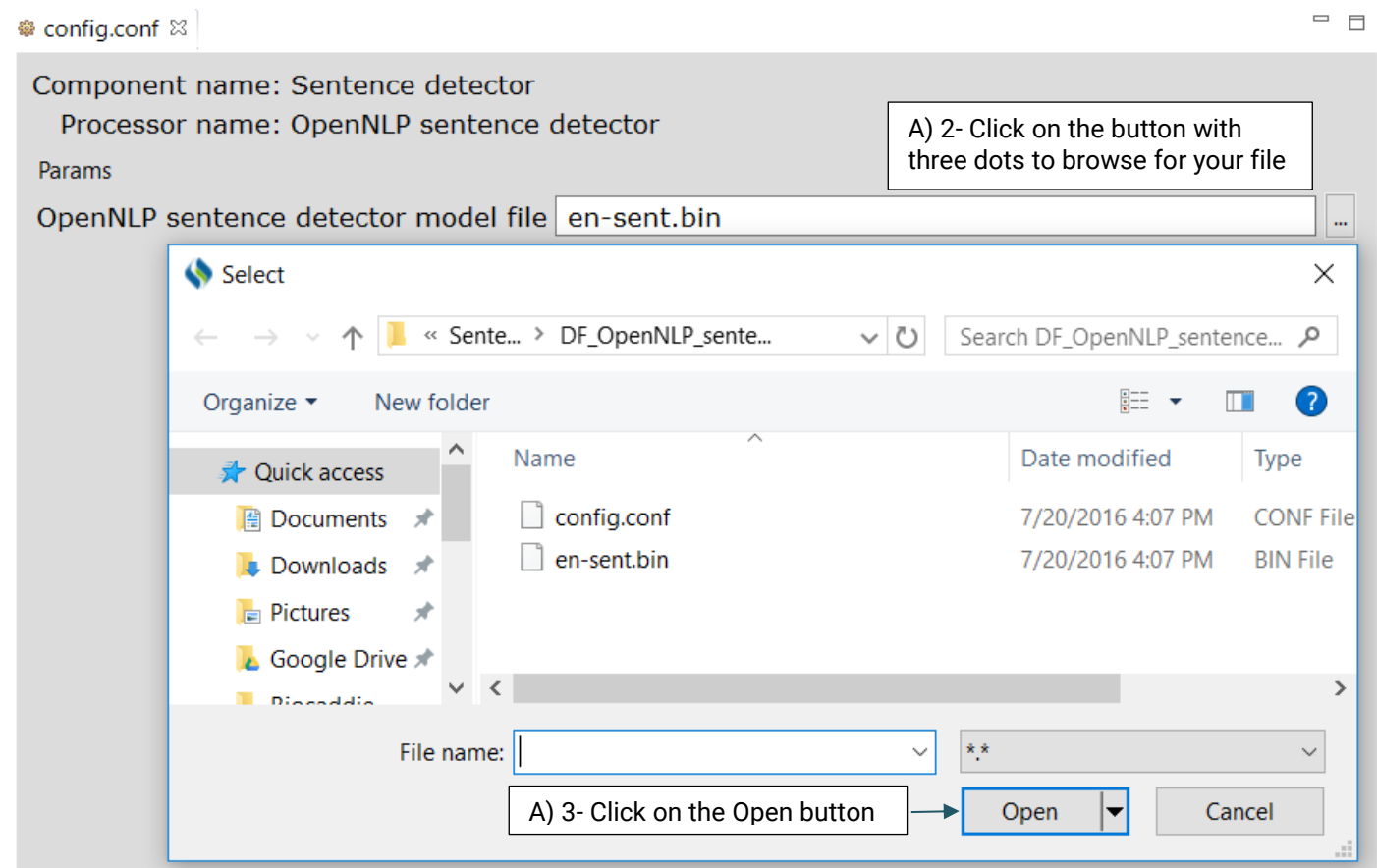
This detector will identify new sentences using the line breaks in the file, i.e., each line in the file is treated as a single sentence.

3. DF_CLAMP_OpenNLP_sentence_detector.

This is an OpenNLP sentence detector which advanced users can use its config.conf file to change its default model.

A) To replace the default model:

1. Double click on config.conf file to open it
2. Click on the button with three dots to browse for your own file
3. Click on the open button



8.3 Tokenizer

A Tokenizer segments the text into a sequence of tokens. As shown in Figure 8.4, CLAMP provides three different models of tokenizer:

1. DF_CLAMP_Tokenizer

2. DF_OpenNLP_Tokenizer

3. DF_Tokenize_by_spaces

Each model will be described in more details.

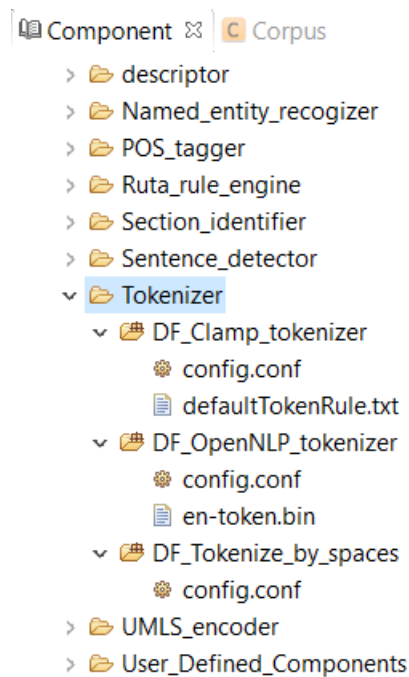


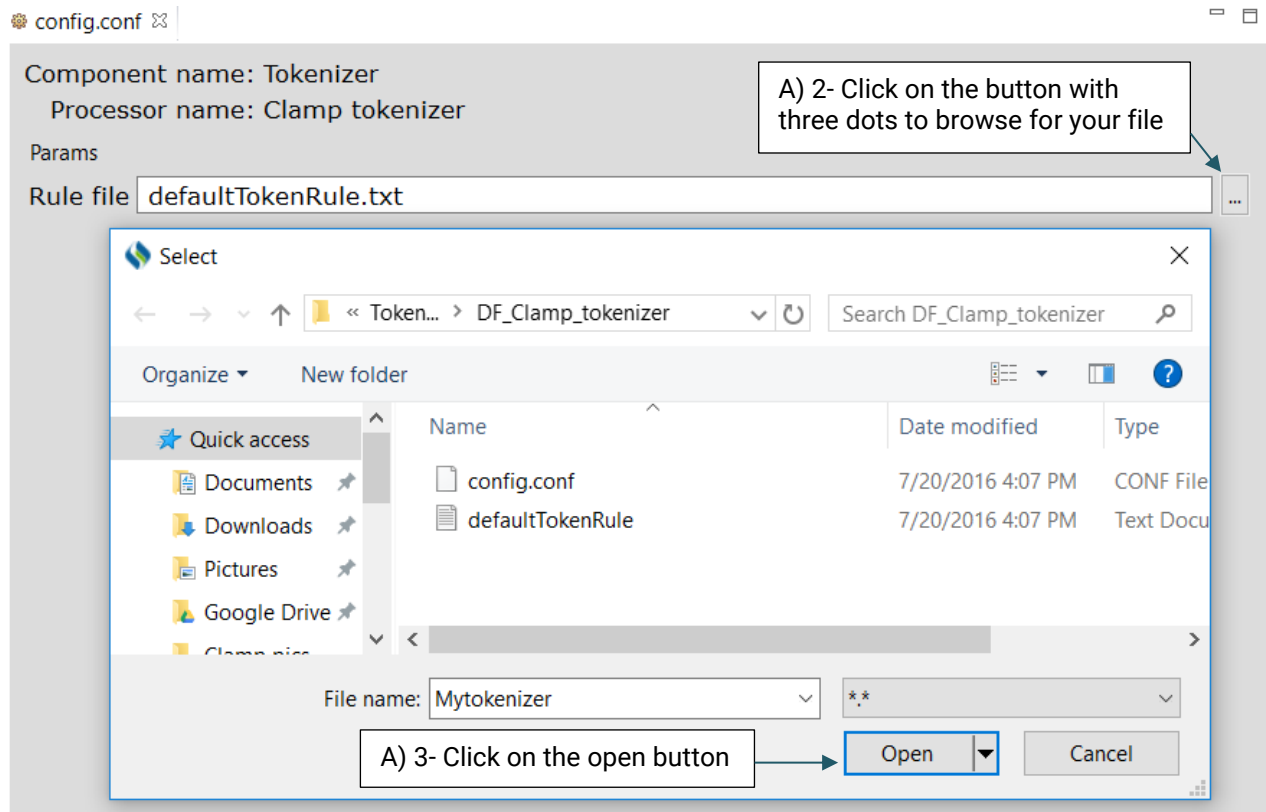
Figure 8.4 Three tokenizers and their configuration files

1) DF_CLAMP_Tokenizer

DF_CLAMP_Tokenizer is the default tokenizer designed specifically for clinical notes. Advanced users can use the config.conf file to change the default tokenization.

A) To replace the default file:

1. Double click on config.conf file to open it
2. Click on the button with three dots to browse for your own file
3. Click on the open button

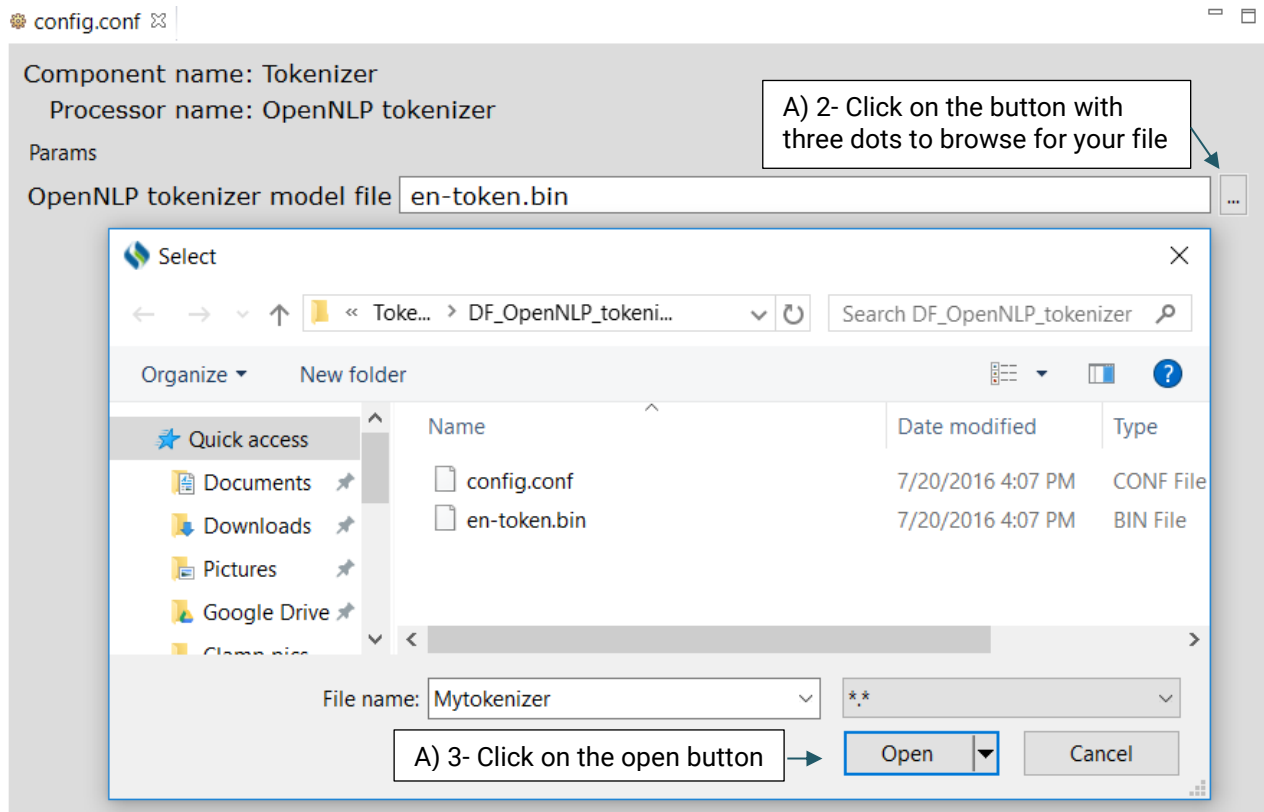


2) DF_OpenNLP_Tokenizer

This is an OpenNLP tokenizer. Advanced users can use its config.conf file to change its default model, en-token.bin.

A) To replace the default file:

1. Double click on config.conf file to open it
2. Click on the button with three dots to browse for your own file
3. Click on the open button



3) DF_Tokenize_by_spaces.

This tokenizer uses the spaces in a sentence to separate the tokens.

8.4 Pos Tagger

A Pos tagger allows users to assign parts of speech to each token. As shown in Figure 8.5, CLAMP currently provides only one pos tagger, **DF_OpenNLP_pos_tagger**, designed specifically for clinical text. This tagger is built from re-training the OpenNLP pos tagger on a dataset of clinical notes, namely, the MiPACQ corpus. (<http://clear.colorado.edu/compsem/index.php?page=endendsystems&sub=mipacq>).

Advanced users can use the config.conf file to change the default pos tagger model-mipacq_pos.bin.

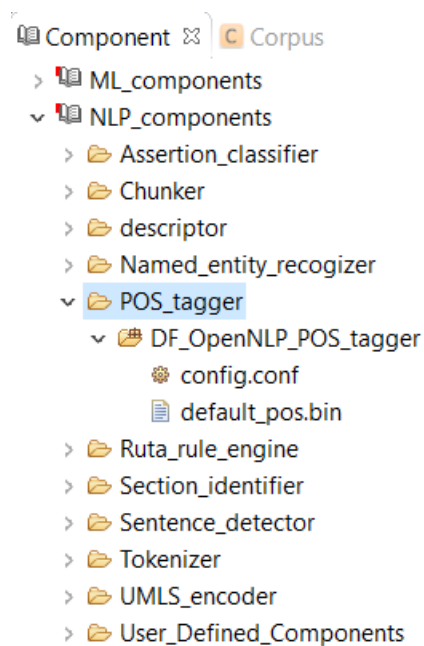


Figure 8.5 DF_OpenNLP_pos_tagger and its configuration files

8.5 Chunker

A chunker does a shallow parsing of a sentence and identifies the syntactic constituents such as noun phrases, verb phrases, and etc. As shown in Figure 8.6, CLAMP currently provides only one single chunker, DF_OpenNLP_chunker, which is a wrapper of the chunker in OpenNLP. Advanced users can use the config.conf file to change the default chunker model- en-chunker.bin.

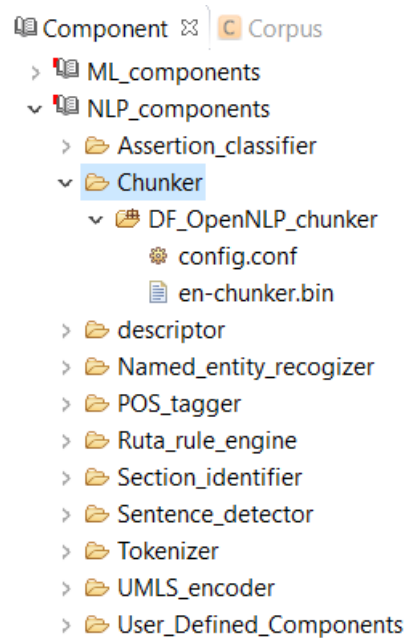


Figure 8.6 DF_OpenNLP_chunker and its configuration files

8.6 Named Entity Recognizer

A named entity recognizer identifies named entities and their semantic types in text. Typically, named entities refer to clinical concepts in CLAMP. As shown in Figure 8.7, CLAMP provides two different models for named entity recognition:

1. DF_CRF_based_named_entity_recognizer ,and

2. DF_Dictionary_lookup

3. DF_Regular_expression_NER

Each model will be described in more details.

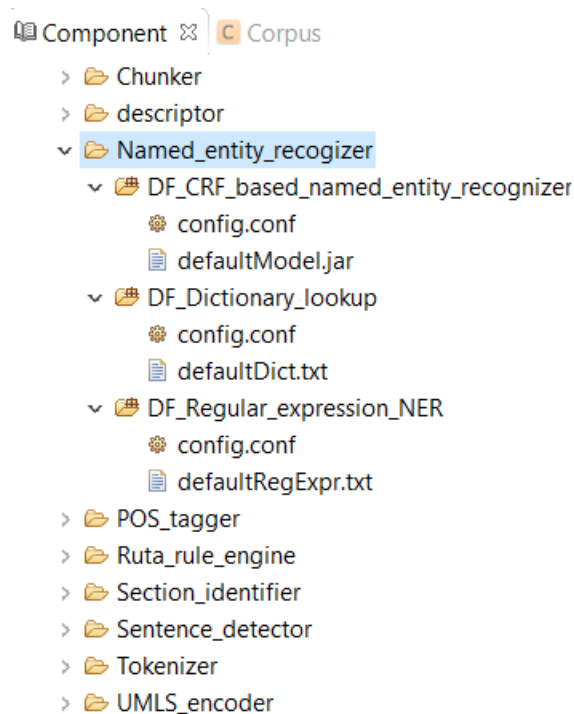


Figure 8.7 Three named entity recognizers and their configuration files

1. DF_CRF_based_named_entity_recognizer

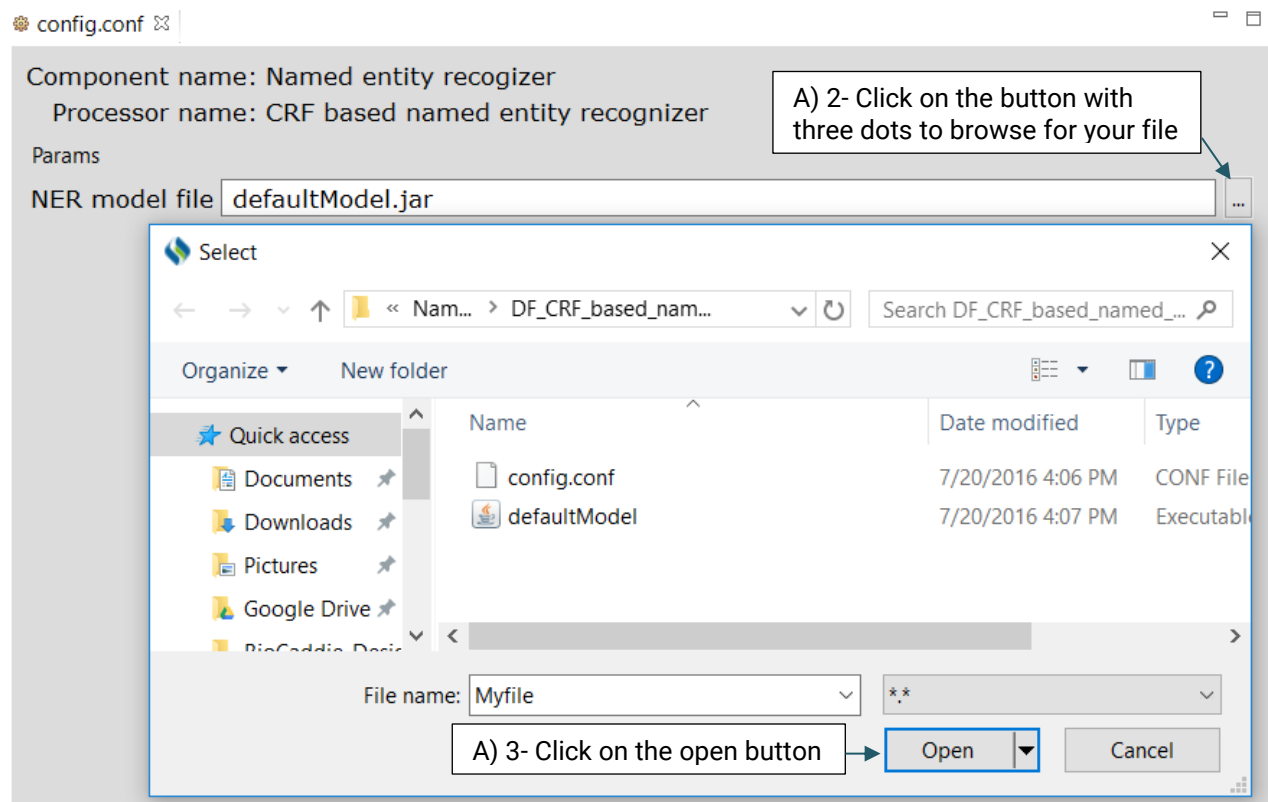
DF_CRF_based_named_entity_recognizer is the default named entity recognizer used in CLAMP. The recognizer identifies three types of clinical concepts:

Problems, treatments, and tests.

It is built from training the CRF model on a dataset of clinical notes, namely, the i2b2 2010 challenge corpus (<https://www.i2b2.org/NLP/Relations/>). Advanced users can use the config.conf file to change the default recognizer model as in the file defaultModel.jar.

A) To replace the default file:

1. Double click on the config.conf file to open it
2. Click on the button with three dots to browse for your own file
3. Click on the open button



2. DF_Dictionary_lookup

DF_Dictionary_lookup uses terms in the dictionary to match them directly with the identified named entities. Currently the defaultDic.txt used in CLAMP consists of terms and their semantic types from UMLS (<https://www.nlm.nih.gov/research/umls/>). The semantic type of the matched term in UMLS is assigned to the recognized named entity.

To configure DF_Dictionary_lookup:

First, click on the config file under the DF_Dictionary_matcher folder. This will open up a new window that takes the following three parameters: **Case sensitive**, **Stemming** and **Dictionaries**. (Figure 6.8)

Case sensitive

If you check the checkbox for “Case sensitive”, the matcher will differentiate between capital and lowercase letters when searching for a term in the dictionary. For example, “Breast Cancer” will not be matched with “breast cancer”.

Stemming

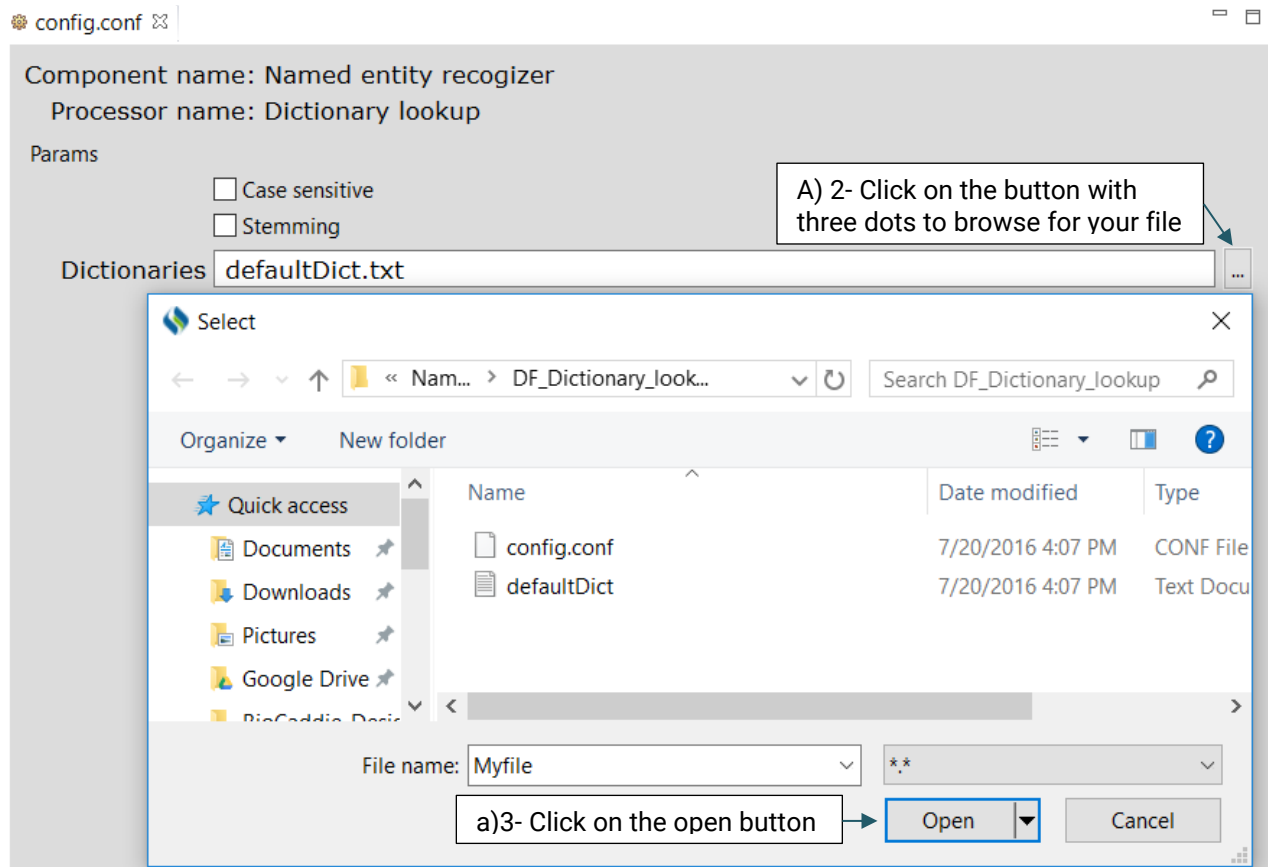
If you check the checkbox for “Stemming”, the matcher will match the stemmed form of a candidate named entity with the terms in the dictionary. For example, “breast cancers” will be matched to “breast cancer”.

Dictionaries

You can also replace or edit the dictionary file suggested for this function.

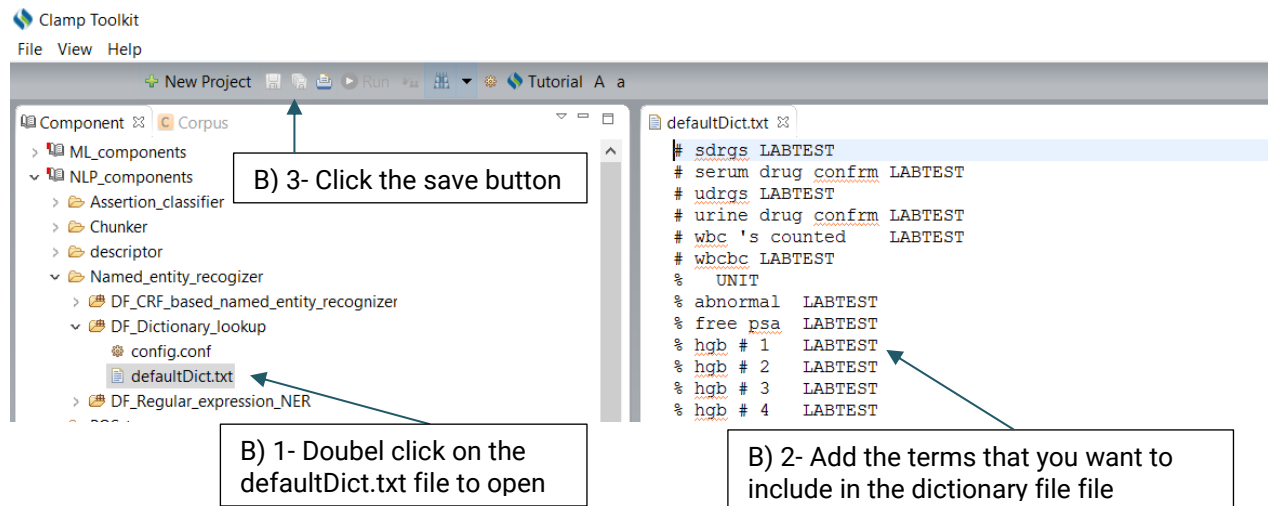
A) To replace the default dictionary file:

1. Double click on config.conf file to open it
2. Click on the button with three dots to browse for your own file
3. Click on the open button



B) To edit the current dictionary file:

1. Double click on the defaultDict.txt file to open it
2. Add the terms that you want to include in the dictionary file
3. Click the Save button at the top of the page



3. DF_Regular_expression_NER

Using the defaultRegExpr.txt file, this module can identify named entities. defaultRegExpr.txt file can contain several regular expression. If a phrase matches a regular expression, it is recognized as a named entity. You can add your own regular expression to the existing file by double clicking the file and add the items that you want to include.

8.7 Assertion Identifier

An Assertion identifier checks whether there is a negation related to a specific clinical concepts in the text. A negation means the absence or opposite of something positive. CLAMP Assertion Identifier provides a mechanism to examine the real-world implications of annotations in a given clinical text. The defaultNegexDict.txt file which contains common negation patterns is used by CLAMP to check for negation in a clinical text. You can either replace or edit this file by following the steps below (Figure 8.8).

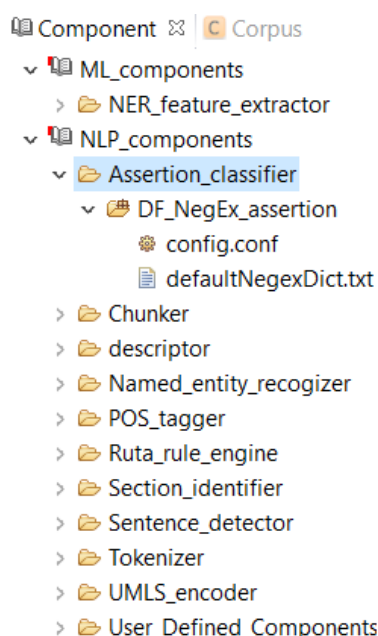
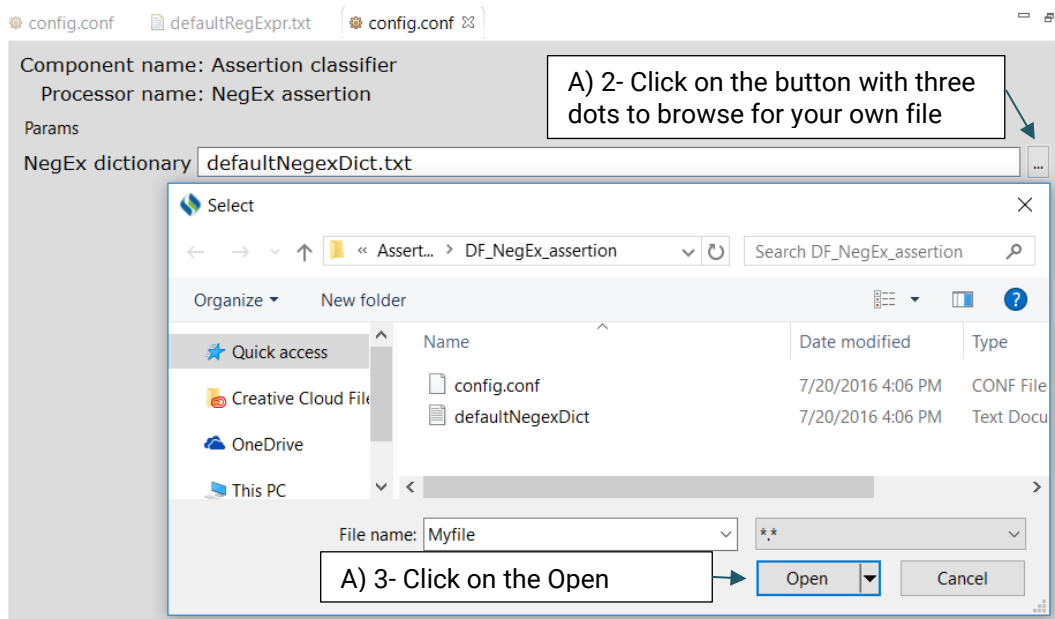


Figure 8.8 Assertion identifier and its configuration file

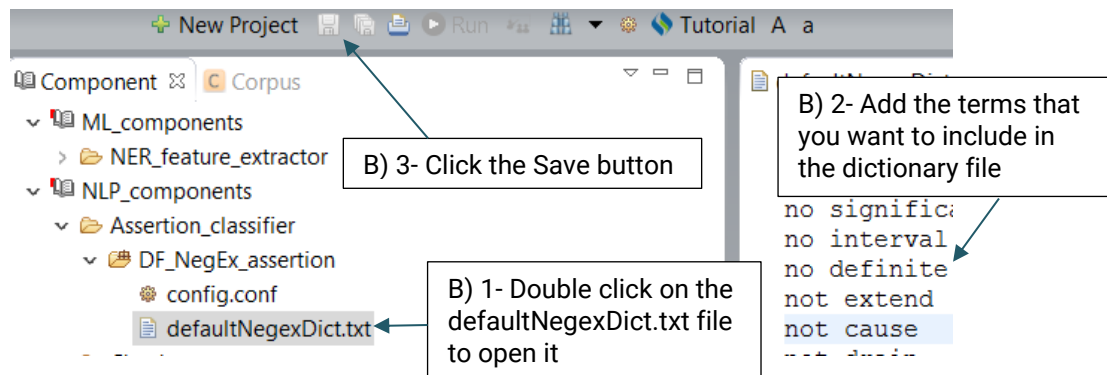
A) To replace the Negation list file:

1. Double click on config.conf file to open it
2. Click on the button with three dots to browse for your own file
3. Click on the open button



B) To edit the current dictionary file:

1. Double click on the defaultNegExDict.txt file to open it
2. Add the terms that you want to include in the dictionary file
3. Click the Save button at the top of the page



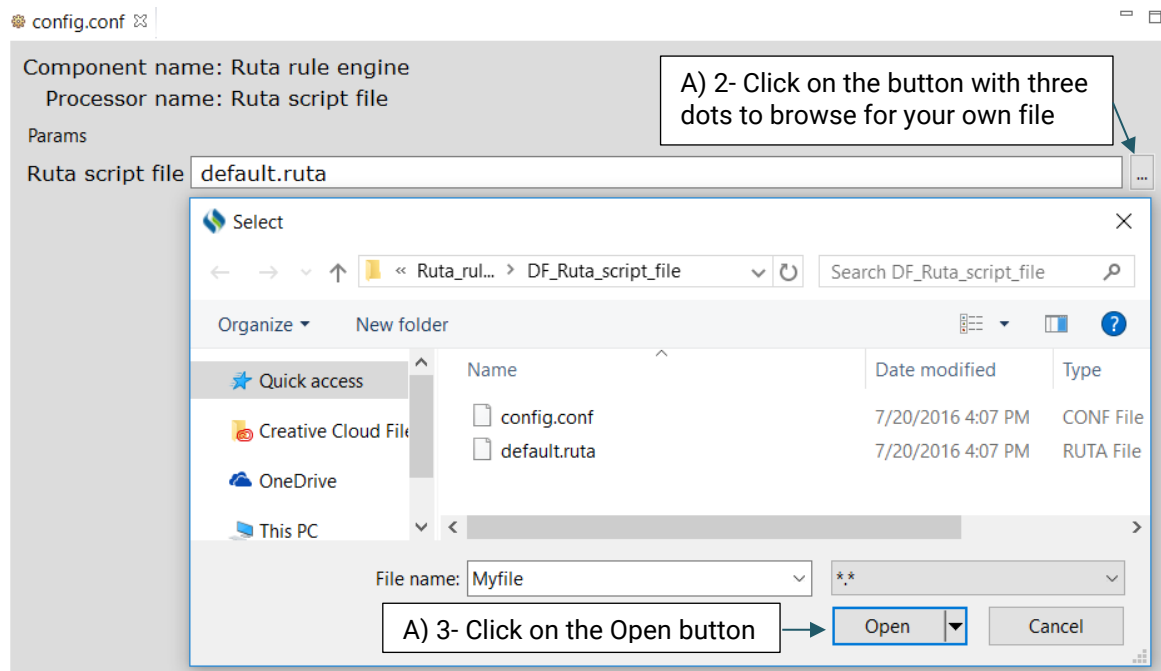
8.8 Ruta_Rule_Engine

UIMA Ruta rules can be used to create or modify annotations as well as create features for annotations. Ruta rules in general can consist of a sequence of rule elements. A simple rule elements consist of four parts: A matching condition, an optional quantifier, an optional list of conditions and an optional list of actions. For more information please visit:

<https://uima.apache.org>

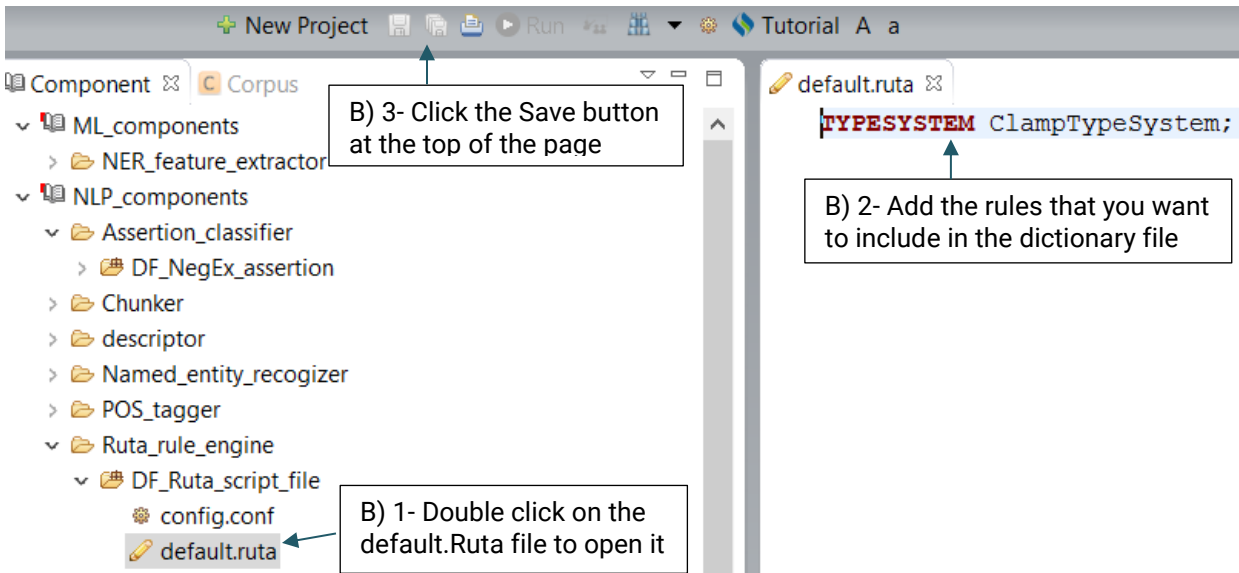
A) To replace the default file:

1. Double click on config.conf file to open it
2. Click on the button with three dots to browse for your own file
3. Click on the open button



B) To edit the current dictionary file:

1. Double click on the default.Ruta file to open it
2. Add the terms that you want to include in the dictionary file
3. Click the Save button at the top of the page



8.9 Section Identifier

The section header identifier component identifies the section headers in a clinical note based on a predefined dictionary and categorizes them into general categories (Figure 8.9). E.g. the section header “ICD 10 code” will be assigned to the “icd_code” category.

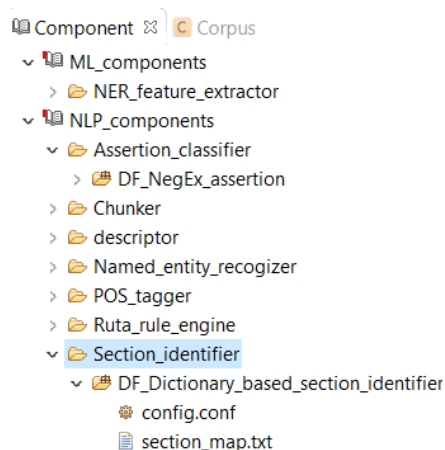
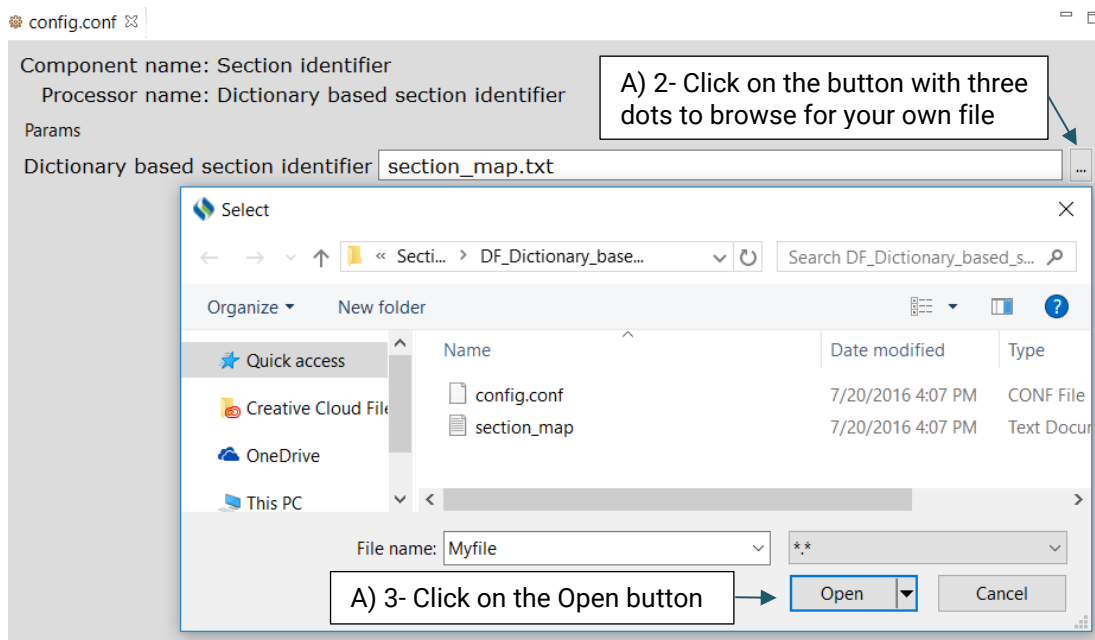


Figure 8.9 Section header identifier and its configuration file

You can replace or edit the default dictionary, section_map.txt, following the steps below:

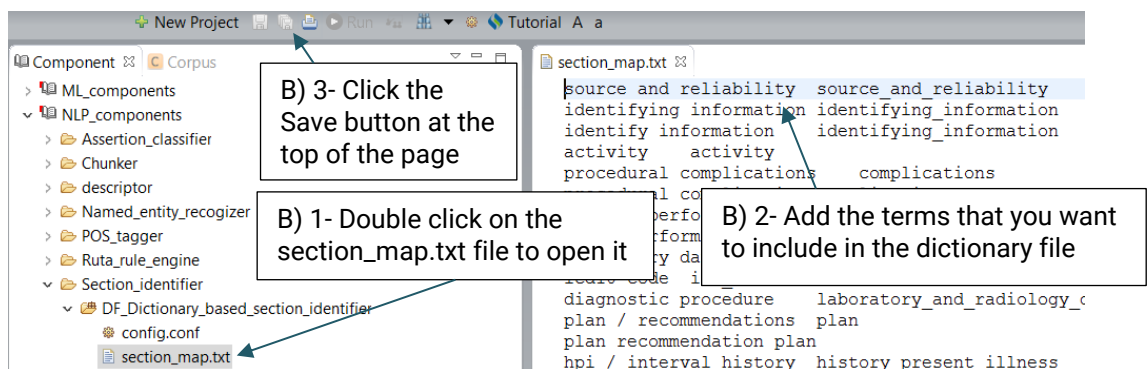
A) To replace the default file:

1. Double click on config.conf file to open it
2. Click on the button with three dots to browse for your own file
3. Click on the open button



B) To add additional section headers to the current file:

1. Double click on the section_map.txt file to open it
2. Add the terms that you want to include in the file
3. Click the Save button at the top of the page



8.10 UMLS Encoder

A UMLS Encoder matches the terms of clinical concepts to its corresponding CUIs in UMLS. For example, the term “breast cancer” will be encoded into the CUI of “C6006142” in UMLS. Currently CLAMP provides a default dictionary based on the UMLS encoder as shown in Figure 8.10.

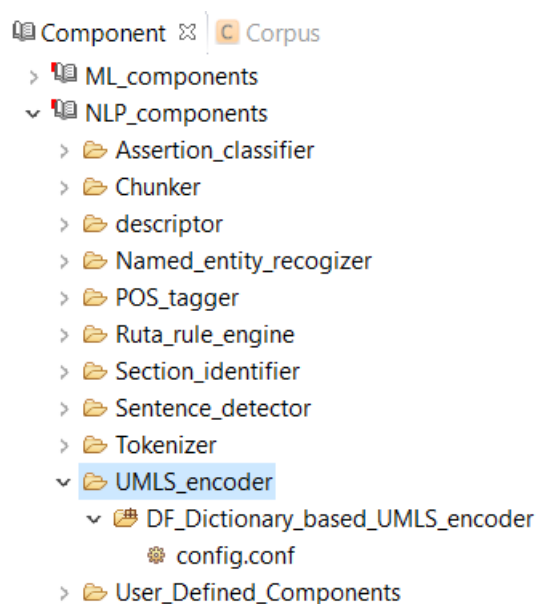


Figure 8.10 A dictionary based UMLS encoder

8.11 User_Defined_Components

DF_Drug_Attribute_Connector:

This is a context free grammar parser which is extracted from Medex. It is used to connect medication to its possible attributes such as dose.

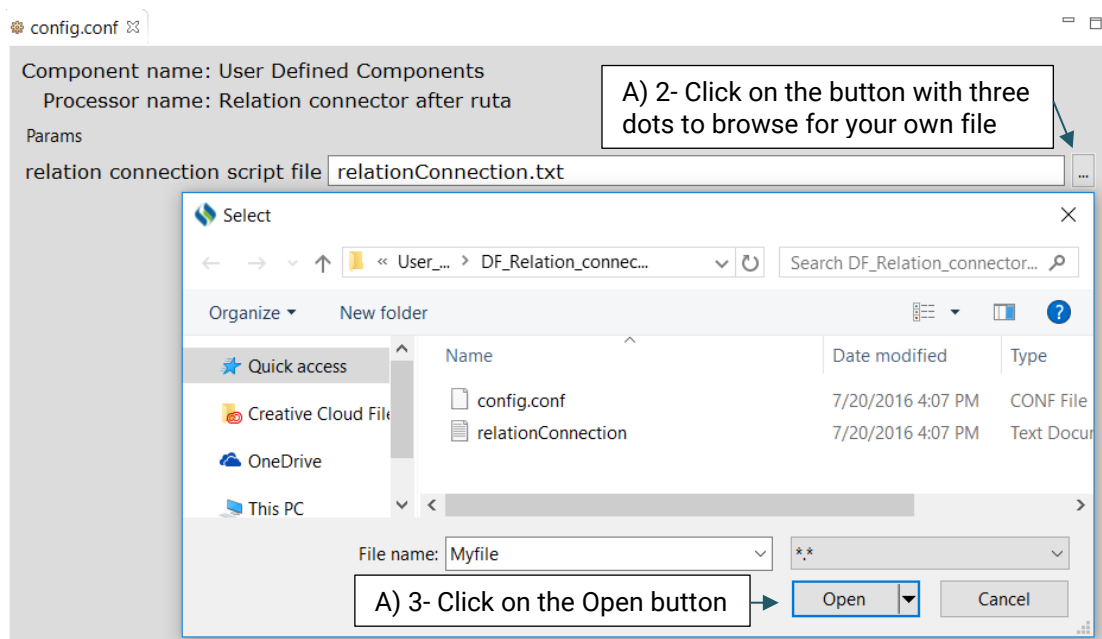
DF_Relation_connector_after_ruta:

While connecting two named entities using Ruta is relatively easy, it can not be used to provide a name for that relationship.

Advanced users can generate their own file and replace it with the system's default file or edit the default file.

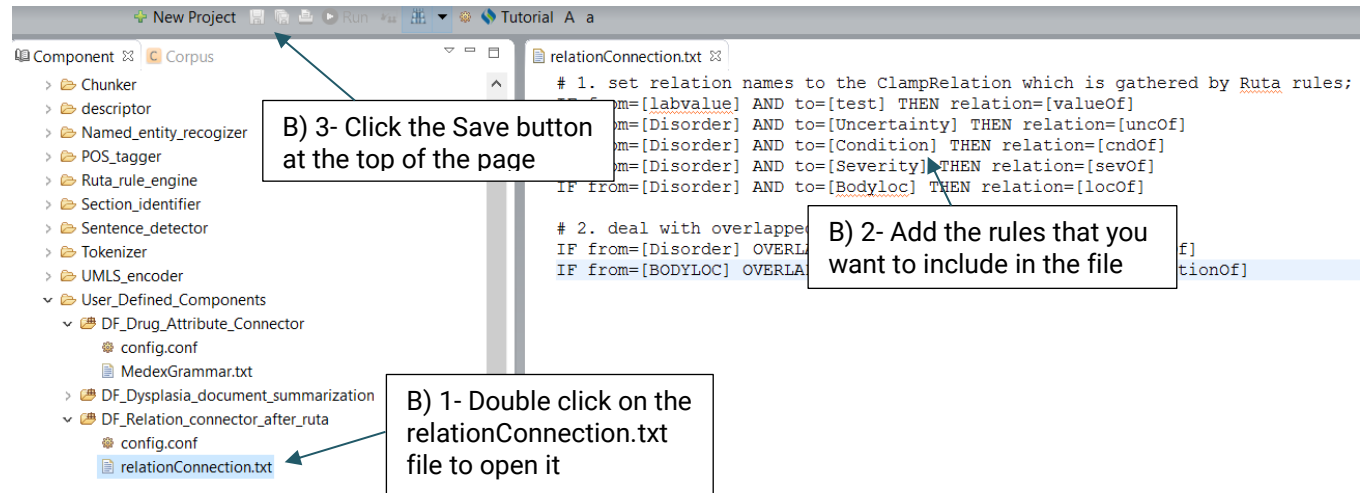
A) To replace the default file:

1. Double click on config.conf file to open it
2. Click on the button with three dots to browse for your own file
3. Click on the open button



B) To edit the current dictionary file:

1. Double click on the relationConnection.txt file to open it
2. Add the terms that you want to include in the file
3. Click the Save button at the top of the page



9. Machine Learning components

9.1 NER Feature Extractor

This component consists of different feature extractors (Figure 9.1), which are used for extracting different types of features for named entity recognition, CLAMP users will use this component to build their own named entity recognizer in a corpus annotation project (Refer to Section 4.2) . Similar to the previous components, we can customize these features by changing or replacing their default config files. Explanation of each extractor is as follows:

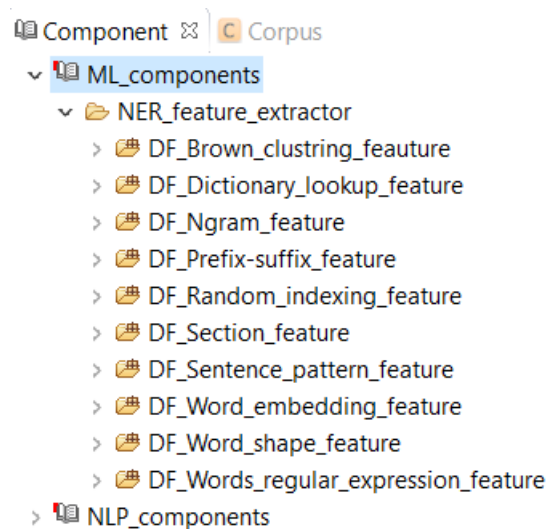


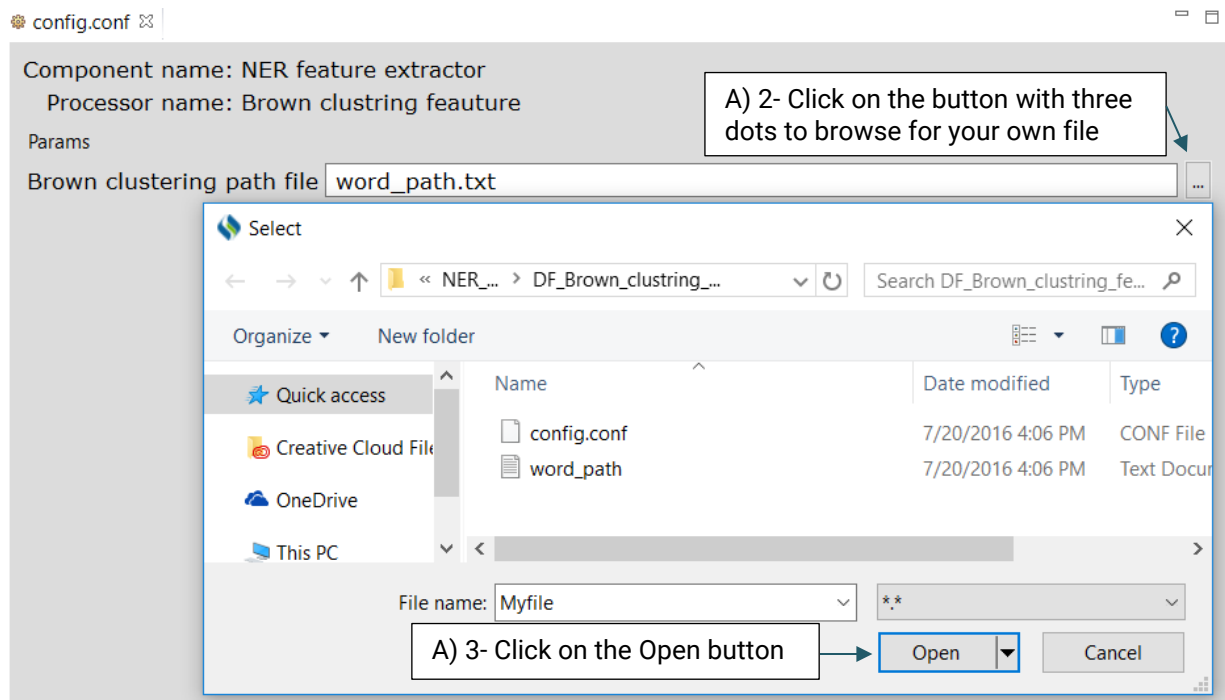
Figure 9.1 List of NER feature extractors

9.1.1 DF_Brown_clustering_feature

It is a type of word representation feature generated on the unlabeled data which is provided by the SemEval 2014 Challenge. Advanced users can replace their own Brown clustering file with the system's default file.

A) To replace the default file:

1. Double click on config.conf file to open it
2. Click on the button with three dots to browse for your own file
3. Click on the open button



For more information on how to create your own Brown Clustering file visit:

<https://github.com/percyliang/brown-cluster>

9.1.2 DF_Dictionary_lookup_feature

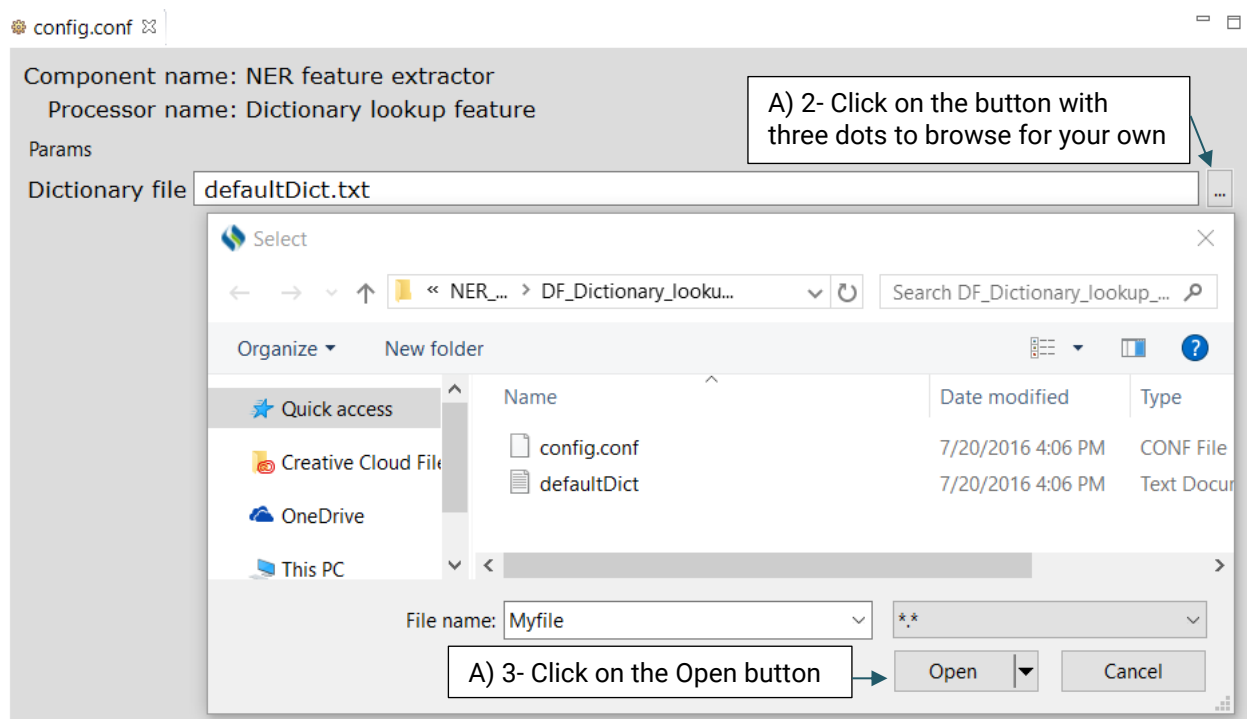
This extractor uses a dictionary consisting of terms and their semantic types from UMLS to extract potential features.

Advanced users can replace or edit the default file following the steps below:

Note: The format of the content should be as the same as the default file: (phrase then tab then semantic type)

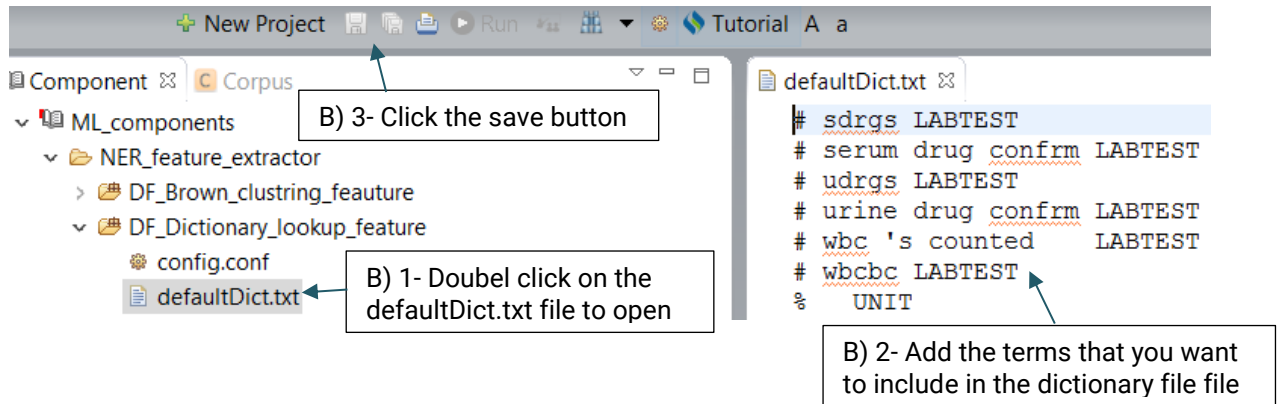
A) To replace the default file:

1. Double click on config.conf file to open it
2. Click on the button with three dots to browse for your own file
3. Click on the open button



B) To edit the default file:

1. Double click on the word_path.txt file to open it
2. Add the terms that you want to include in the file
3. Click the Save button at the top of the page



9.1.3 DF_Ngram_feature

This module uses the words along with their part-of-speech (pos) tagging as NER features.

9.1.4 DF_prefix_suffix_feature

This function extracts the prefix and suffix of words that may be a representative of a specific type of named entities.

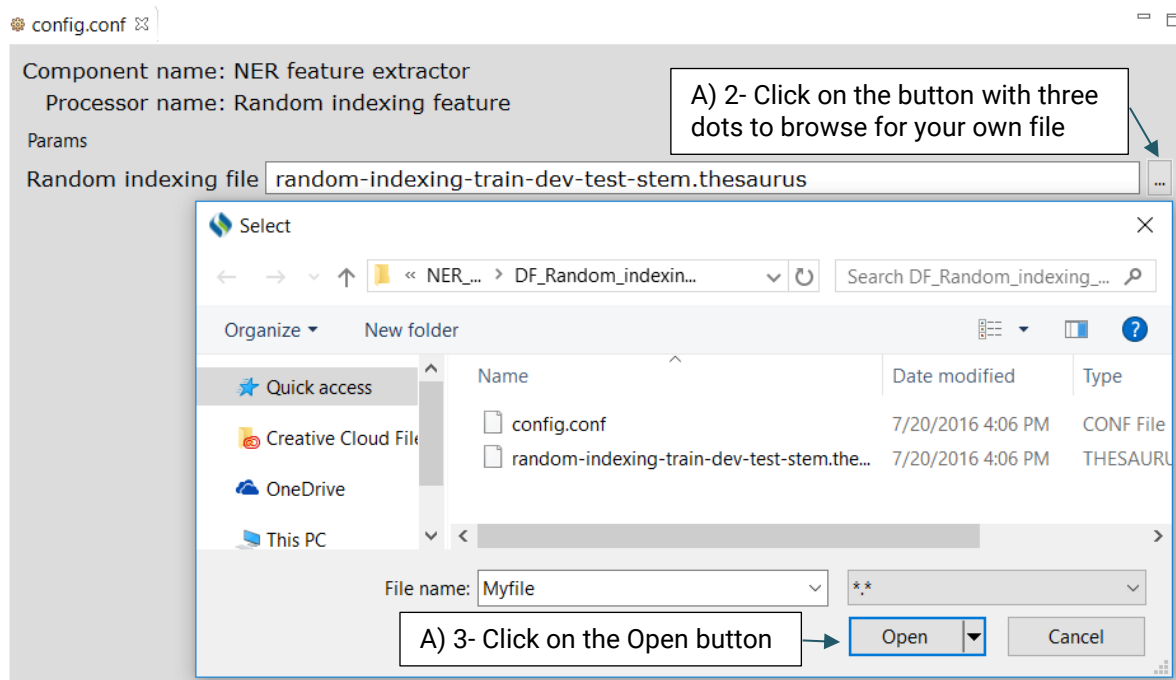
9.1.5 DF_Random_indexing_feature

Similar to the brown clustering, it is a type of word representation feature generated on unlabeled data using a 3rd party package. For more information visit:

<https://jcheminf.springeropen.com>

A) To replace the default file:

1. Double click on config.conf file to open it
2. Click on the button with three dots to browse for your own file
3. Click on the open button



9.1.6 DF_Section_feature

This function extracts the section in which a candidate named entity presents.

9.1.7 DF_Sentence_pattern_feature

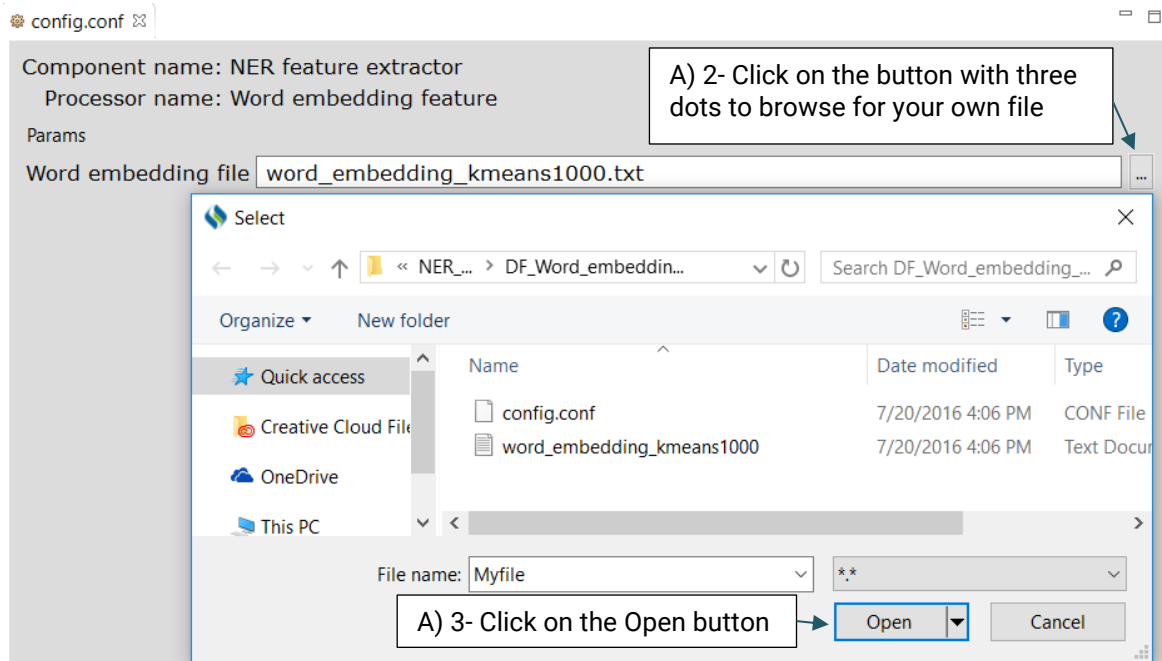
This function distinguishes the pattern of a sentence by CLAMP built in rules.

9.1.8 DF_Word_embedding_feature

Similar to the brown clustering and random indexing, it is a type of distributed word representation feature generated on the unlabeled data (MIMIC II) provided by the SemEval 2014 Challenge using a neural network. Advanced users can replace the default file with their own file.

A) To replace the default file:

1. Double click on config.conf file to open it
2. Click on the button with three dots to browse for your own file
3. Click on the open button



9.1.9 DF_Word_shape_feature

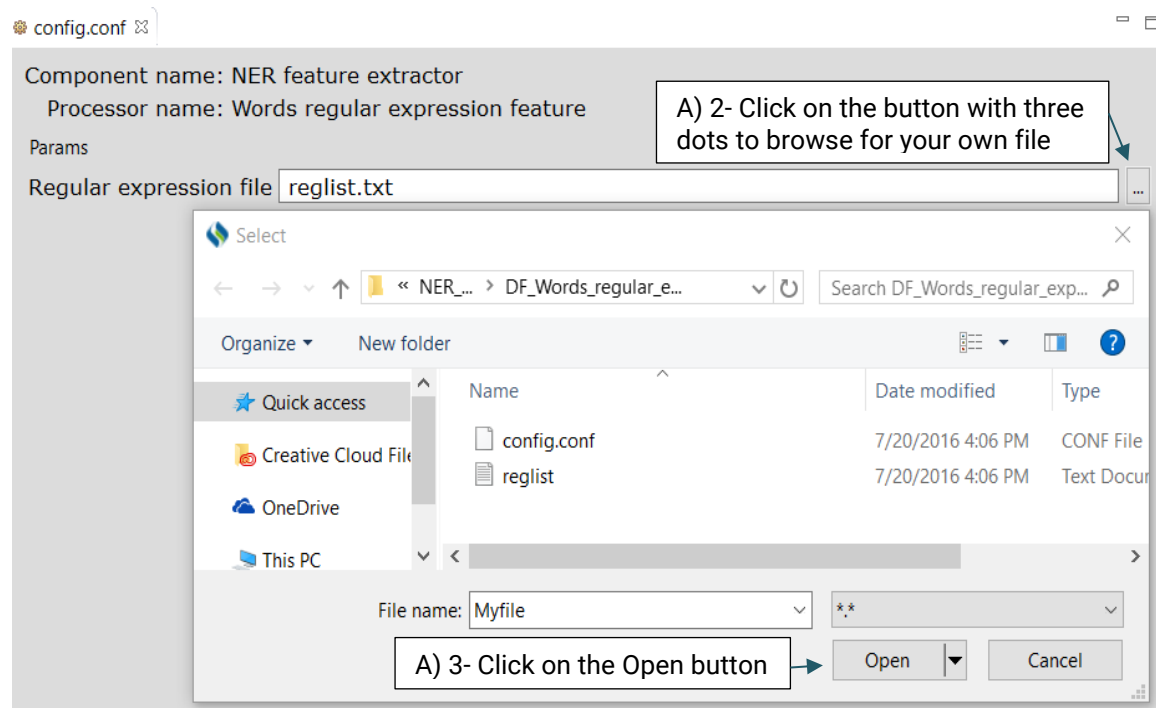
This function extracts the type of a word; it identifies whether or not it begins with an english letter, number, and etc.

9.1.10 DF_Words_regular_expression_feature

This function extracts the regular expression patterns of words that may indicate a specific type of named entity. Advanced users can create their own regular expressions or edit the default file.

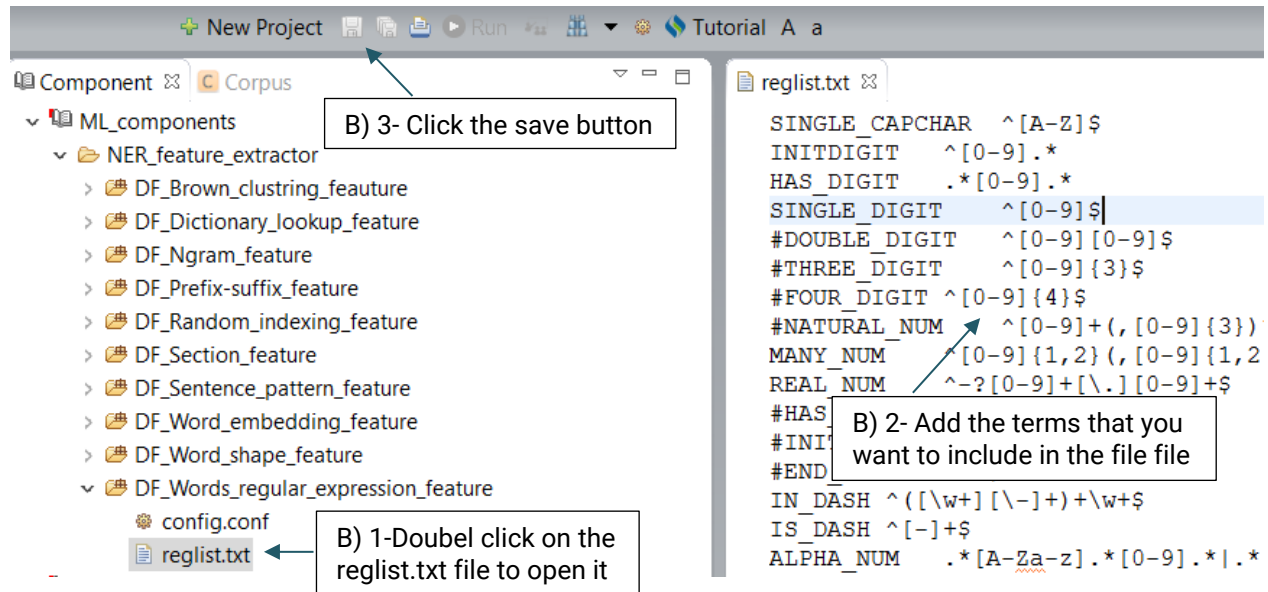
A) To replace the default file:

1. Double click on config.conf file to open it
2. Click on the button with three dots to browse for your own file
3. Click on the open button



B) To edit the default file:

1. Double click on the reglist.txt file to open it
2. Add the terms that you want to include in the file
3. Click the Save button at the top of the page



10. Build a Pipeline

10.1 Create and Run a Pipeline

Running a pipeline refers to the use of a set of NLP components to identify the specified information , including sentence segmentation, tokenization, part of speech tagging, abbreviations, etc. The NLP components are executed in a sequence based on the functional dependency amongst them.

In order to recognize clinical concepts within clinical text:

1. You need to create a project
2. You need to configure the pipeline
3. You need to import the files that you want to be analyzed
4. You need to process the imported files by running them through the pipeline.

Follow the steps below to build a pipeline:

A) Create a new project

1. Click on the plus (+) sign at the top left corner of the screen as shown in Figure 10.1.

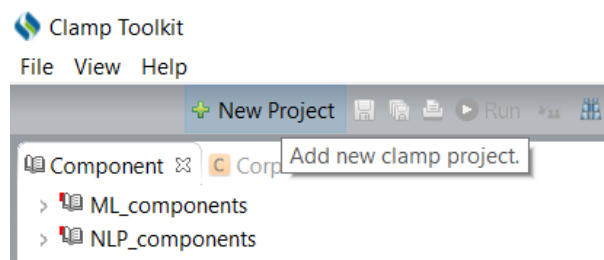


Figure 10.1 Create a new project

2. On the pop-up window (Figure 10.2), enter a name for your project, for example: "Clinical_concept_recognition".
3. Select NLP Pipeline as the project type.
4. Click the Finish button.

A new project with the specified name is created and is placed under Mypipeline folder on the Pipeline panel at the lower left of the screen (Figure 10.3)

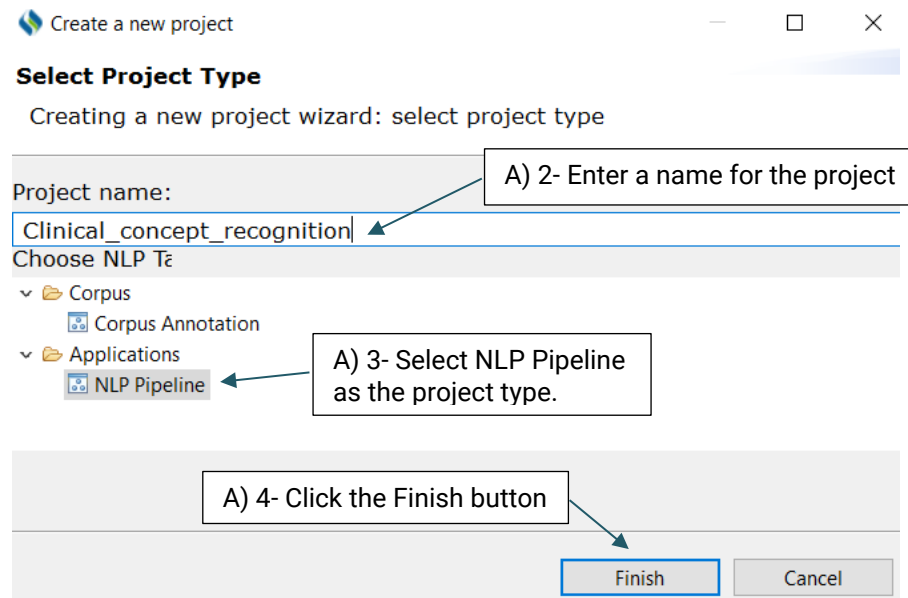


Figure 10.2 Creating a new NLP pipeline project

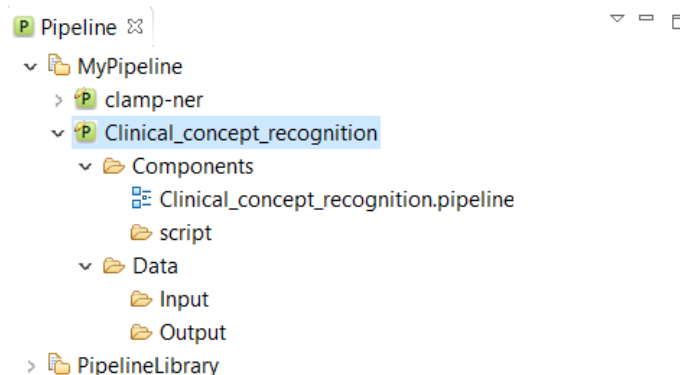


Figure 10.3 A project with the specified name is created and is placed under Mypipeline folder

Double click the pipeline name to view its content. As you can see, it contains two folders “Components”, and “Data”. The Components folder contains the pipeline configuration file. The Data folder includes two folders: Input, and Output. The Input folder holds the files that are processed by the pipeline. The results obtained by running the pipeline are saved in the output folder.

10.2 Configure the pipeline

To configure a pipeline double click on the .pipeline file from the newly created pipeline project to open it in the middle window on the screen. (Figure 10.4).

Here you can drag and drop the NLP components from the Component panel. Since we want to recognize clinical concepts using NLP components, we drag the `DF_CRF_based_name_entity_recognizer` from the `NLP_components` to the pipeline.

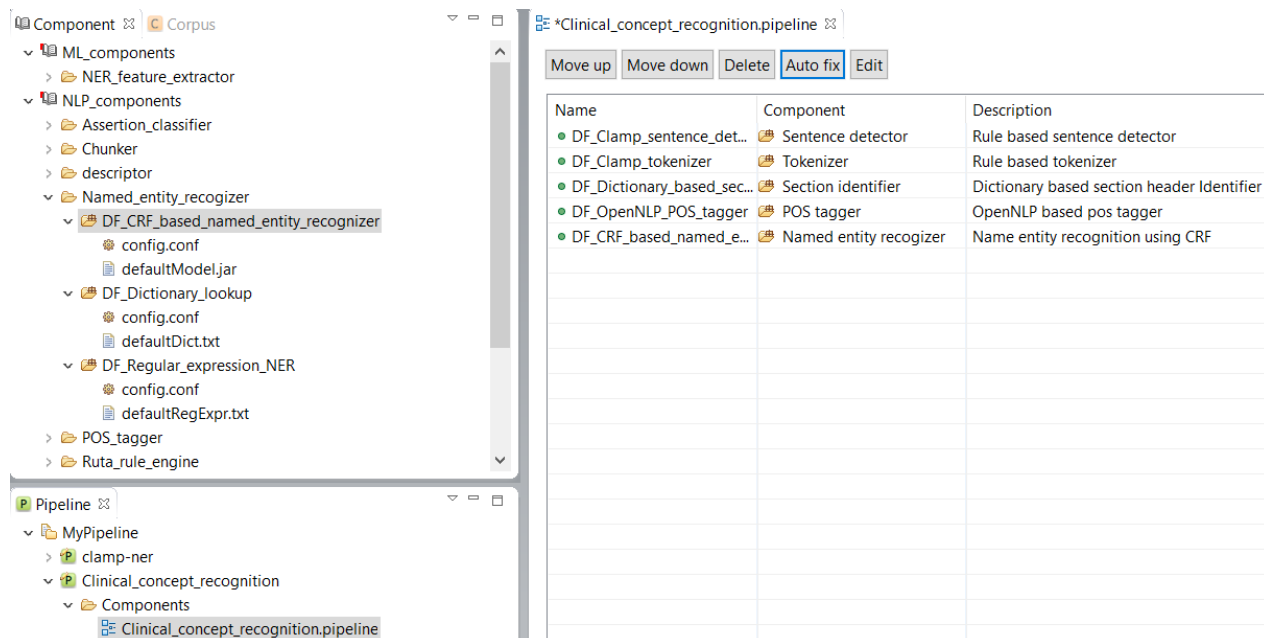


Figure 10.4 Pipeline configuration window

10.3 Component dependency & Auto fix

As shown in Figure 10.5, there is a red X sign in front of the newly added component, “CRF based named entity recognizer”. This sign indicates that the named entity recognizer component is dependant on other NLP components that are missing from the current pipeline. In our example, the clinical notes first need to be processed by the sentence detector, tokenizer, section identifier, and POS tagger components before processing by the named entity recognizer.

To fix this issue, simply click on the Auto fix button at the top of the panel. This automatically adds the required components to the pipeline. The sequence of the individual components from top to bottom reflect the order in which they will run to process your input data.

After the required components are added (Figure 10.6), the **red X** sign changes to the **green circle** sign indicating the accuracy of the order of the components. Once you see the green sign for each of the displayed NLP components, click on the Save button at the top of the screen to save your changes.

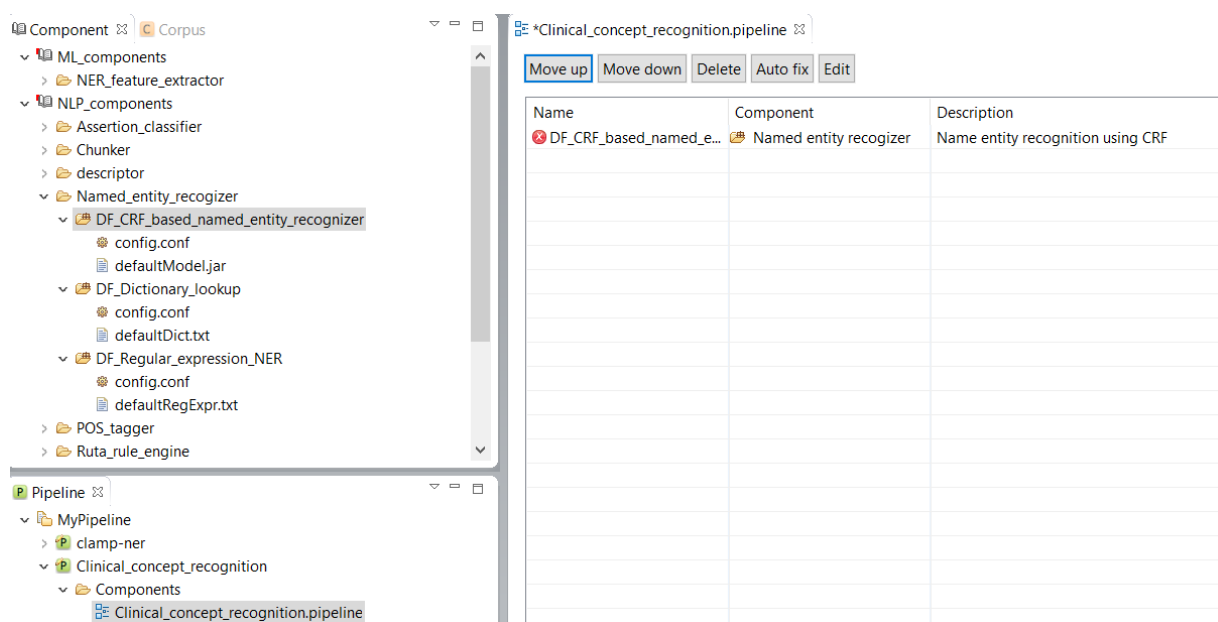


Figure 10.5 A wrong pipeline for clinical concept recognition needs to be fixed with dependent NLP models

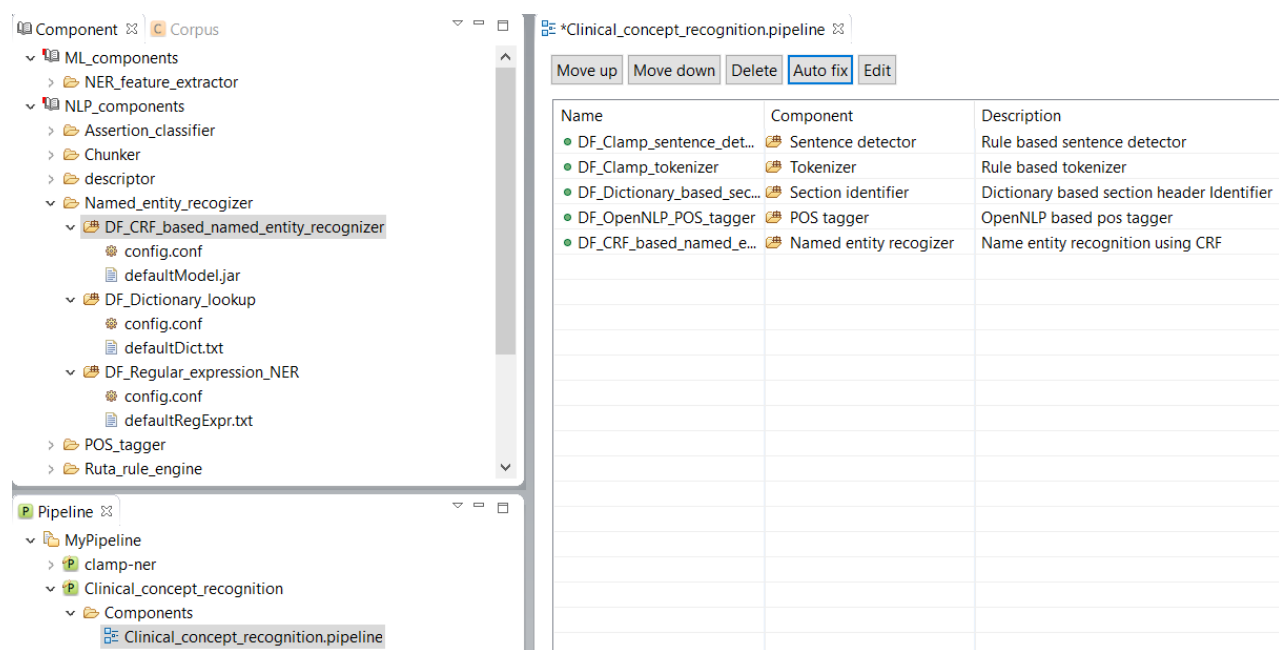


Figure 10.6 - A correct pipeline for clinical concept recognition with all necessary NLP models.

10.4 Import input files

Once the pipeline is configured, you will need to import your desired files to the Input folder using the the following steps:

1. In the PipelineView, right click on the Input folder under the Data folder, then select the import (Figure 10.7). A pop-up menu appears which lets you select the files that you want to import.
2. Click on the small arrow next to the General folder to expand it, then select File System as the import source.
3. Click on the Next button (Figure 10.8)

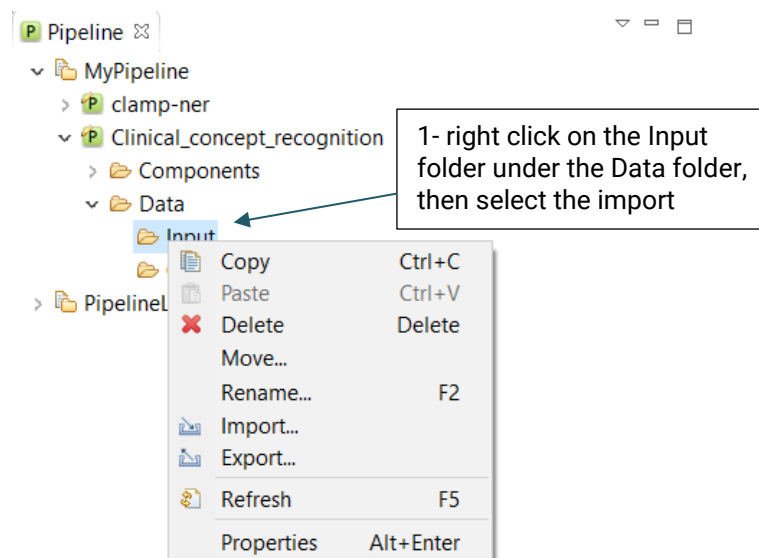


Figure 10.7 Drop-downContext menu for importing the input files

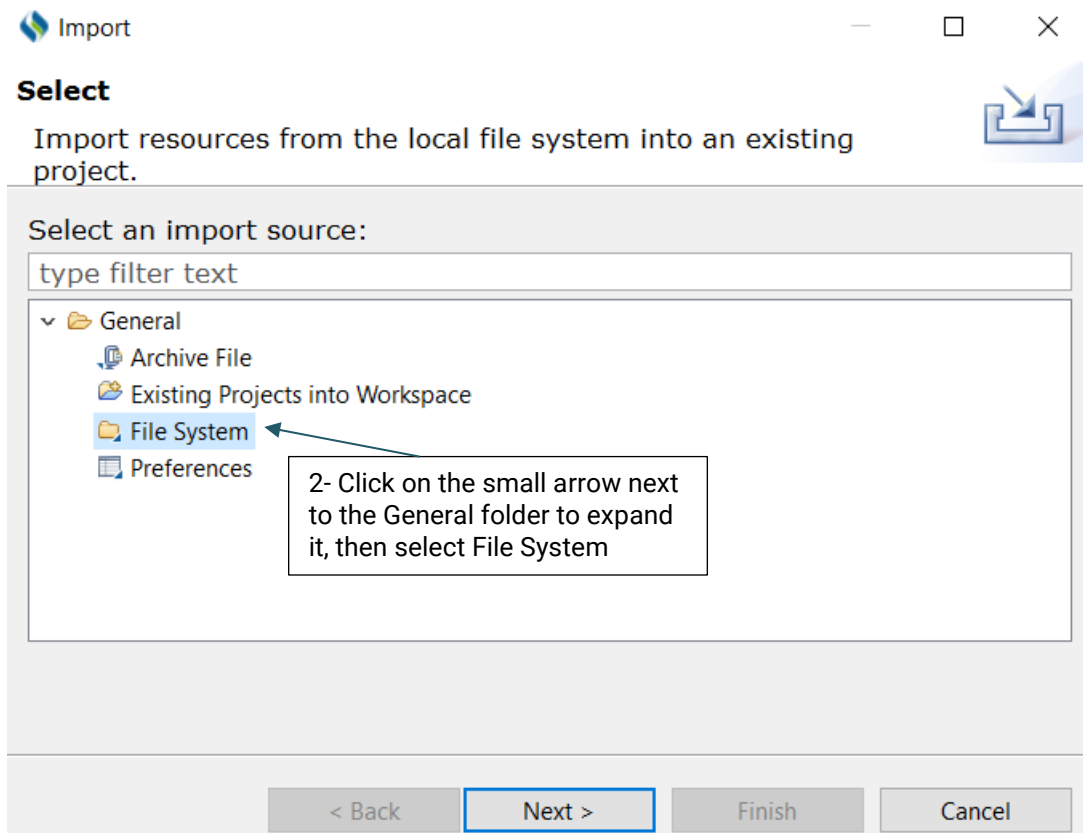


Figure 10.8 Import resources from the local file system into an existing project

- Next, as shown in Figure 10.9, click on the Browse button on the top of the window to choose the folder of your choice. The selected folder will be displayed on the left side of the window, and the files inside the folder will be displayed on the right side.
- Click the checkbox for the files that you want to run the pipeline on; currently the CLAMP pipeline can only process files with the **.txt extension**. Also, CLAMP assists you in selecting your desired files in three different ways: *Filter Types*, *Select All* and *Deselect All*.

Filter Types: Allows you to define the type of files that will be imported. For example, you may only want to import files with the .txt extension

Select All: Allows you to choose all displayed files

Deselect All: Allow you to deselect the files that have already been selected

6. Click on the Browse button next to the “Into folder” field to choose the folder that you want to import your files to. Here, we keep the default directory
7. Click on the Finish button.

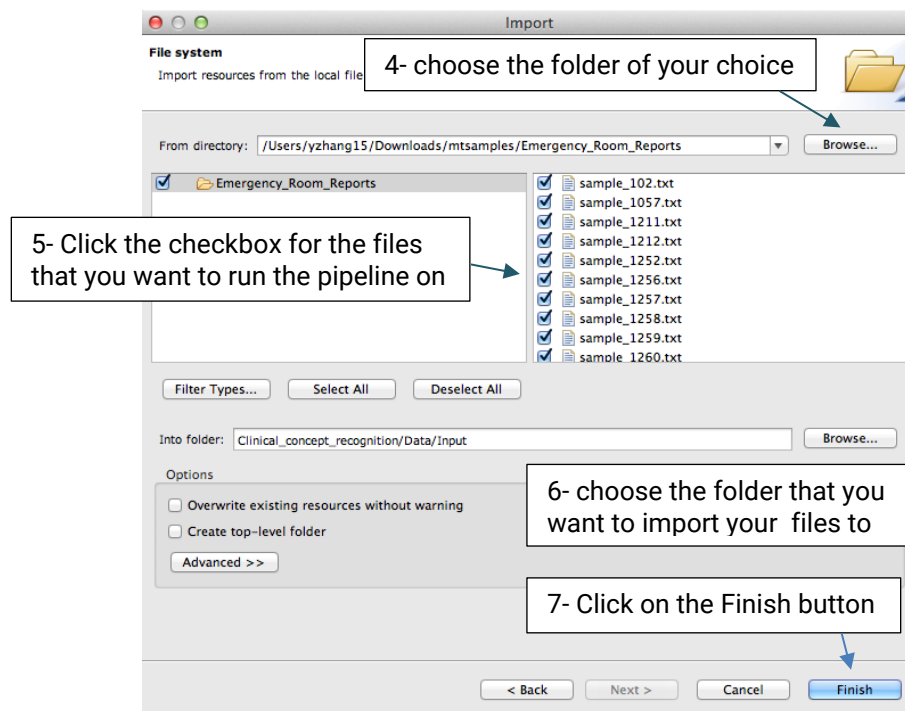


Figure 10.9 Window of input files selection

Now, you can double click on the Input folder to see the imported files (Figure 10.10). Similarly, you can also double click on each file to view its content (Figure 10.11).

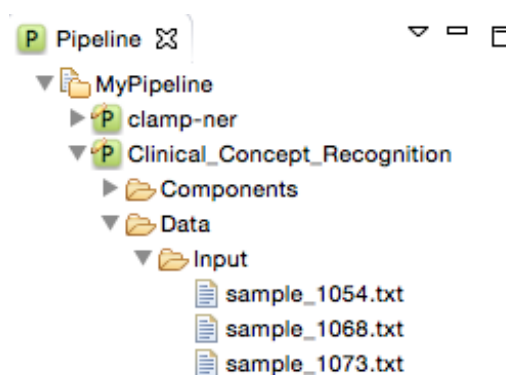


Figure 10.10 Imported files under the Input folder

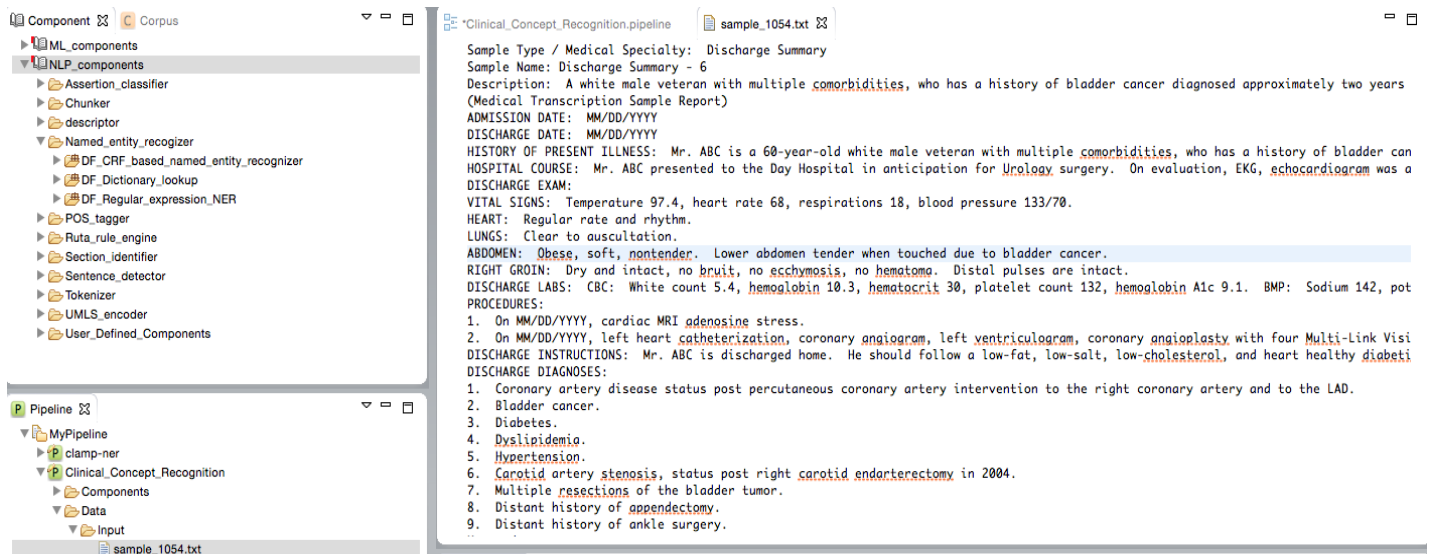


Figure 10.11 View the content of input file sample_1054.txt directly in the interface

11.Run the pipeline

After you have configured the pipeline and imported the input files, you can start running the pipeline. To run a pipeline, simply click on the run icon at the top of the screen as shown in Figure 11.1.

Once the pipeline starts running, you can check the progress of the input file processing from the Console window and the progress bar at the bottom of the screen (Figure 11.2). You can always stop the processing at anytime by clicking on the red stop button next to the progress bar.

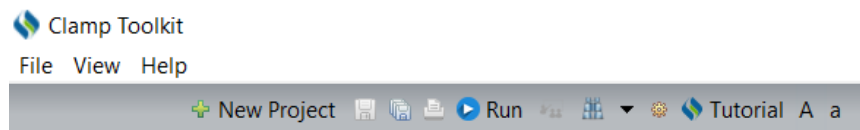


Figure 11.1 Running the pipeline

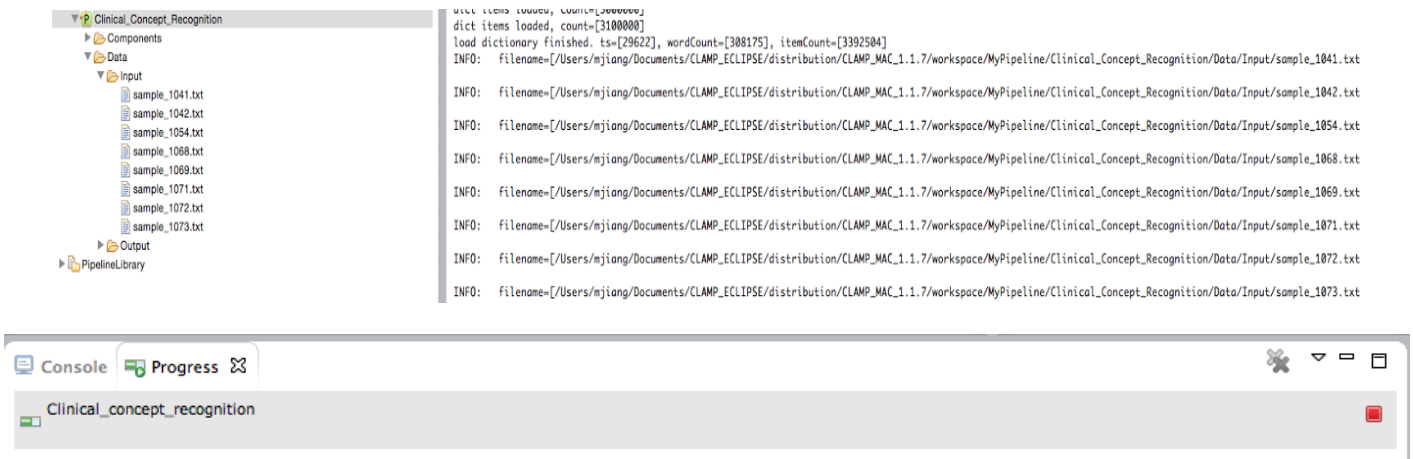


Figure 11.2 check the progress of the input file processing from the Console window

12. Output visualization

Once running the pipeline is completed, the generated files are displayed in the Output folder. These files can be viewed in two different formats (.xmi , .txt):

Clicking on a file with the .xmi extension allows you to view its original content annotated with recognized clinical concepts. Different types of clinical concepts will be highlighted with different colors. (Figure 12.1)

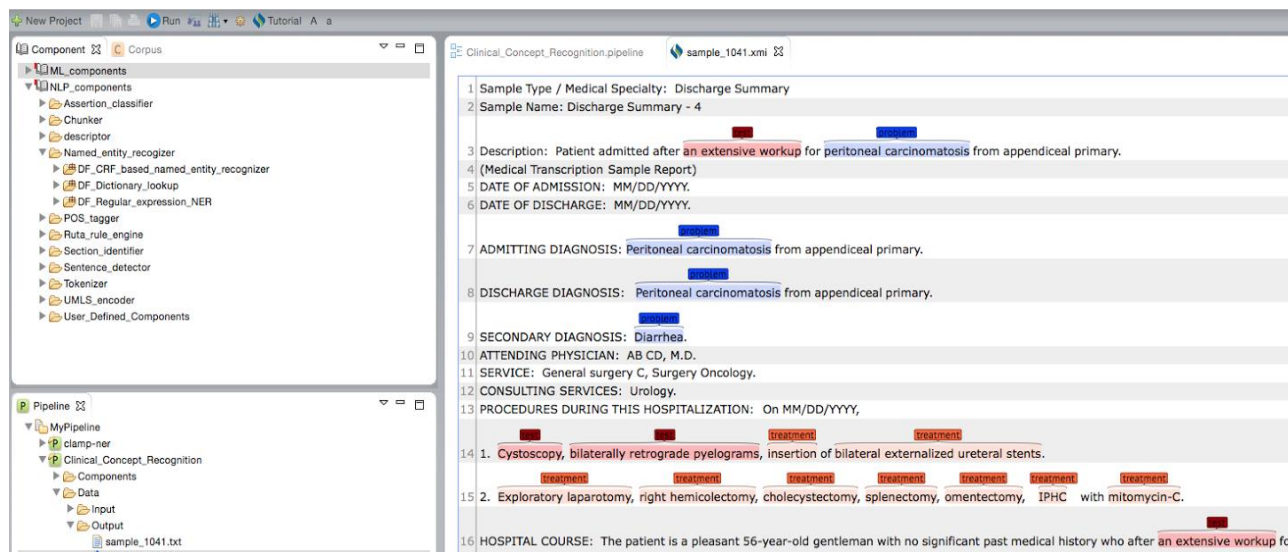


Figure 12.1 View of text annotated with recognized clinical concepts

Clicking on a file with the .txt extension will display a view of tab delimited, detailed output information in a new window. As shown in Figure 12.2, each line in the file illustrates the detailed information of one recognized clinical concept. The following information is included in a tab delimited output:

1. **Start Index:** Starting position of the recognized concept.
2. **End Index:** Ending position of the recognized concept.
3. **Semantic Type:** Semantic type of the recognized concept.
4. **CUI:** The **C**oncept **U**nique **I**dentifier of the concept in Unified Medical Language System (UMLS). If the pipeline does not include the model of UMLS encoder, the value of this column will be "null".

5. **Assertion:** If the pipeline does not include the model of Assertion identifier, the value of this column will be “null”.
6. **Concept Mention:** Referring to a concept, i.e., named entity in the text.

Start Index	End Index	Semantic Type	CUI	Assertion	Concept Mention
124	143	test	null	null	an extensive workup
148	173	problem	null	null	peritoneal carcinomatosis
323	348	problem	null	null	Peritoneal carcinomatosis
398	423	problem	null	null	Peritoneal carcinomatosis
472	480	problem	null	null	Diarrhea
554	664	test	null	null	Cyst
666	699	test	null	null	biopsy
701	710	treatment	null	null	insertion
714	752	treatment	null	null	bilateral externalized ureteral stents
803	810	treatment	null	null	Exploratory laparotomy
820	831	treatment	null	null	right hemicolectomy
833	844	treatment	null	null	cholecystectomy
846	850	treatment	null	null	splenectomy
856	867	treatment	null	null	omentectomy
98	100	treatment	null	null	IPHC
101	102	treatment	null	null	mitomycin-C
1159	1172	test	null	null	an extensive workup
1183	1196	test	null	null	peritoneal carcinomatosis
1198	1208	treatment	null	null	a routine preoperative evaluation
1230	1254	treatment	null	null	baseline labs
1350	1362	test	null	null	bowel prep
1368	1402	treatment	null	null	ureteral stent placement
1423	1448	treatment	null	null	a cystoscopy
1450	1469	treatment	null	null	bilateral ureteral stent placement
1471	1486	treatment	null	null	an exploratory laparotomy
1488	1499	treatment	null	null	right hemicolectomy
1501	1512	treatment	null	null	cholecystectomy
1518	1522	treatment	null	null	splenectomy
1528	1539	treatment	null	null	omentectomy
1568	1581	problem	null	null	IPHC
1660	1682	problem	null	null	mitomycin-C
1689	1699	treatment	null	null	complications
1963	1971	problem	null	null	persistent tachycardia
2040	2050	treatment	null	null	extubation
					His pain
					a PCA pump

Figure 12.2 Tab delimited format of output files

13. Built-in pipelines

In order to facilitate a convenient utility of CLAMP, a series of pipelines that could be directly adopted in common clinical applications are pre_built and displayed in PipelineLibrary (Figure 13.1). Users can directly drag one of them (e.g., smoking_status, Figure 13.2) from the PipelineLibrary and drop it under My Pipeline. The required NLP components of these pipelines are already configured, as illustrated in Figure 13.3. CLAMP allows you to customize each of these components to fit your needs. Now, you need to import your files; for more information go to [“Import input files”](#) section.

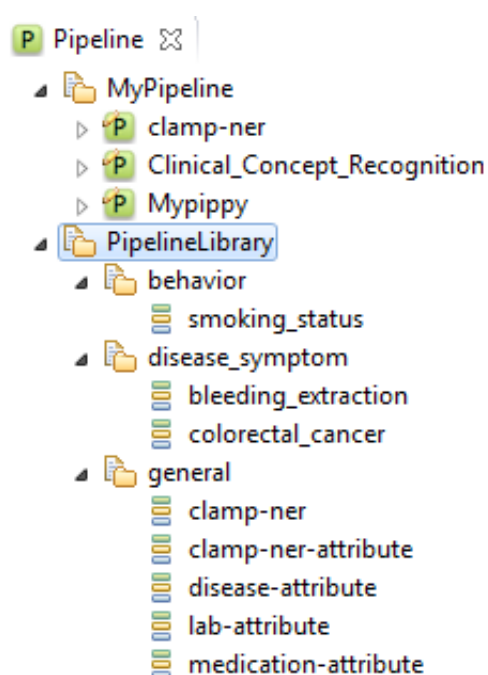


Figure 13.1 Built-in pipeline library in CLAMP

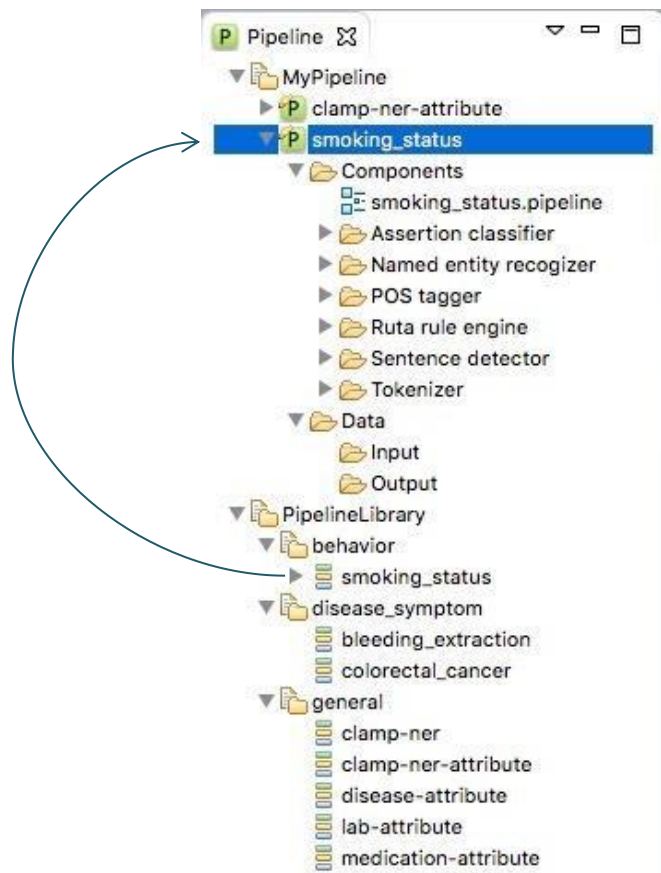


Figure 13.2 Dragging smoking_status and drop it under MyPipeline

<div> <div>Move up</div> <div>Move down</div> <div>Delete</div> <div>Auto fix</div> <div>Edit</div> </div>		
Name	Component	Description
● DF_Clamp_sentence_detector	📁 Sentence detector	Rule based sentence detector
● DF_Clamp_tokenizer	📁 Tokenizer	Rule based tokenizer
● DF_OpenNLP_POS_tagger	📁 POS tagger	OpenNLP based pos tagger
● new_Dictionary_lookup	📁 Named entity recognizer	dictionary lookup algorithm
● DF_NegEx_assertion	📁 Assertion classifier	Assertion info detection using NegEx
● DF_Ruta_script_file	📁 Ruta rule engine	Ruta script

Figure 13.3 Built-in pipeline library in CLAMP

Depending on what your use case is, the current built-in pipelines are divided into the following categories:

1. General: automatically annotates concepts and their attribute for general use, including:

CLAMP-ner: annotates the disease, procedure and medication concepts

CLAMP-ner-attribute: annotates the attributes of disease (e.g., body location of a disease), lab procedure (e.g., value of a lab test) and medication (e.g., dosage of a medication) concepts

Disease-attribute: annotate the attributes of diseases, including body locations (e.g., left atrium), severity degrees (e.g., mild, severe) and uncertainty (e.g., probably).

Lab-attribute: annotates the attributes of lab procedures

Medication-attribute: annotates the attributes of medications

2. Disease_symptom: automatically annotates symptoms of diseases, including:

Bleeding_extraction: annotates bleeding symptoms

Colorectal_cancer: annotates symptoms of colorectal cancer

3. Behavior: automatically annotates behaviors of patients , including:

Smoking_status: annotates whether or not the patient is in a smoking status, and whether the patient has a smoking history.

Figure 13.4 illustrates an example of using the disease-attribute pipeline in our pipeline library to annotate attributes and their relations with diseases.

Limited study. No color doppler was performed. The left atrium is normal in size. Left ventricular wall thicknesses are normal. The left ventricular cavity size is normal. There is severe regional left ventricular systolic dysfunction with mid and distal LV akinesis and relative sparing of the base of the LV. Overall left ventricular systolic function is severely depressed. Right ventricular chamber size and free wall motion are normal. Right ventricular chamber size is normal. Right ventricular systolic function is hard to assess but is probably normal. The aortic valve is not well seen. Mitral valve leaflets are mildly thickened. There is no pericardial effusion.

14. Export pipeline as a jar file

In order to export a pipeline as a jar file and use it in the command line version, please follow the steps below (Figure 14.1):

1. Go to your desired pipeline folder
2. Click on the small arrow next to it to expand it
3. Right click on the Components folder and select “Export as jar”

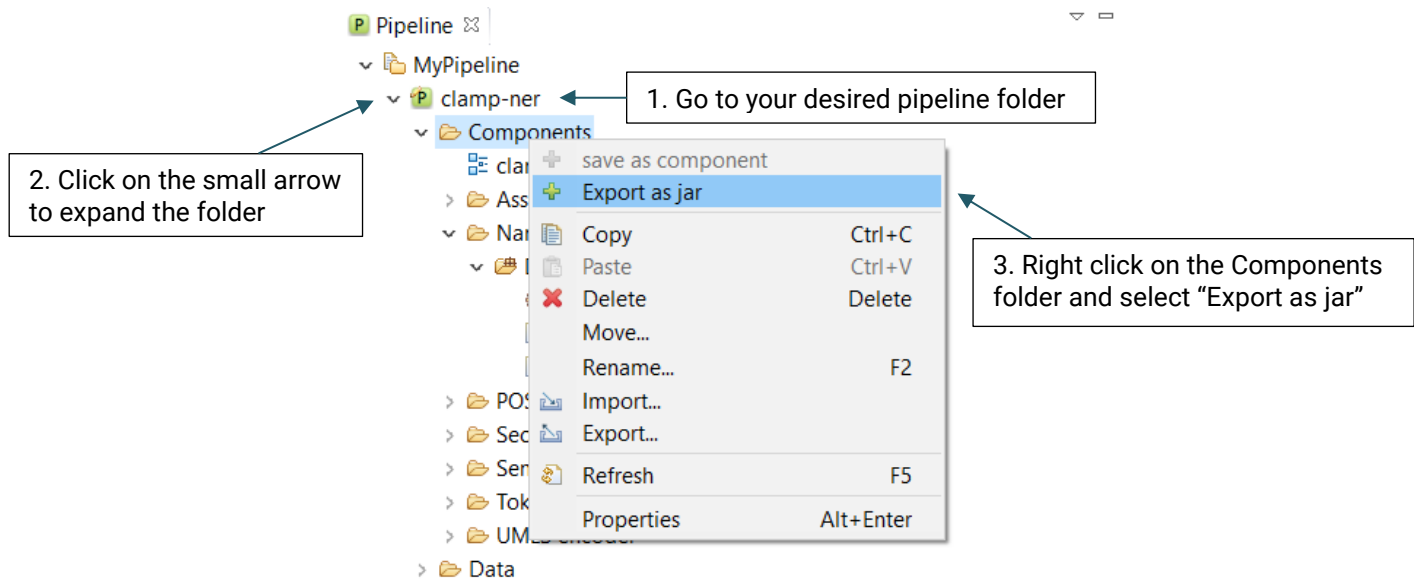


Figure 14.1 Export a pipeline as a jar

15.Annotation

15.1 Annotate corpus

The CLAMP annotation module enables you to annotate customized entities and specify relations between them in your desired corpus . These annotations enable you to assign additional clinical information to a selected text and develop an annotated corpus that's more suitable to the specific task that you have. Task-specific models can be developed and used in the machine-learning modules of CLAMP or any other system of your choice.

Before using this function, you need to:

1. Create a project
2. Import the files that you want to annotate

After completing these steps, you will be able to annotate the imported files based on some predefined structure. The following steps will guide you on how to perform the steps mentioned above.

15.1.1 Create a project

A) To create a project:

1. Click on the plus (+) sign at the top left corner of the screen as shown in Figure15.1.

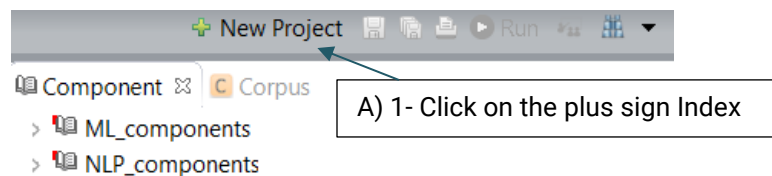


Figure 15.1 Step 1 to create a new project

2. On the pop-up window, enter a name for your project, e.g., Drug_name_annotation, (Figure 15.2).
3. Select Corpus Annotation as the project type.
4. Click the Finish button.

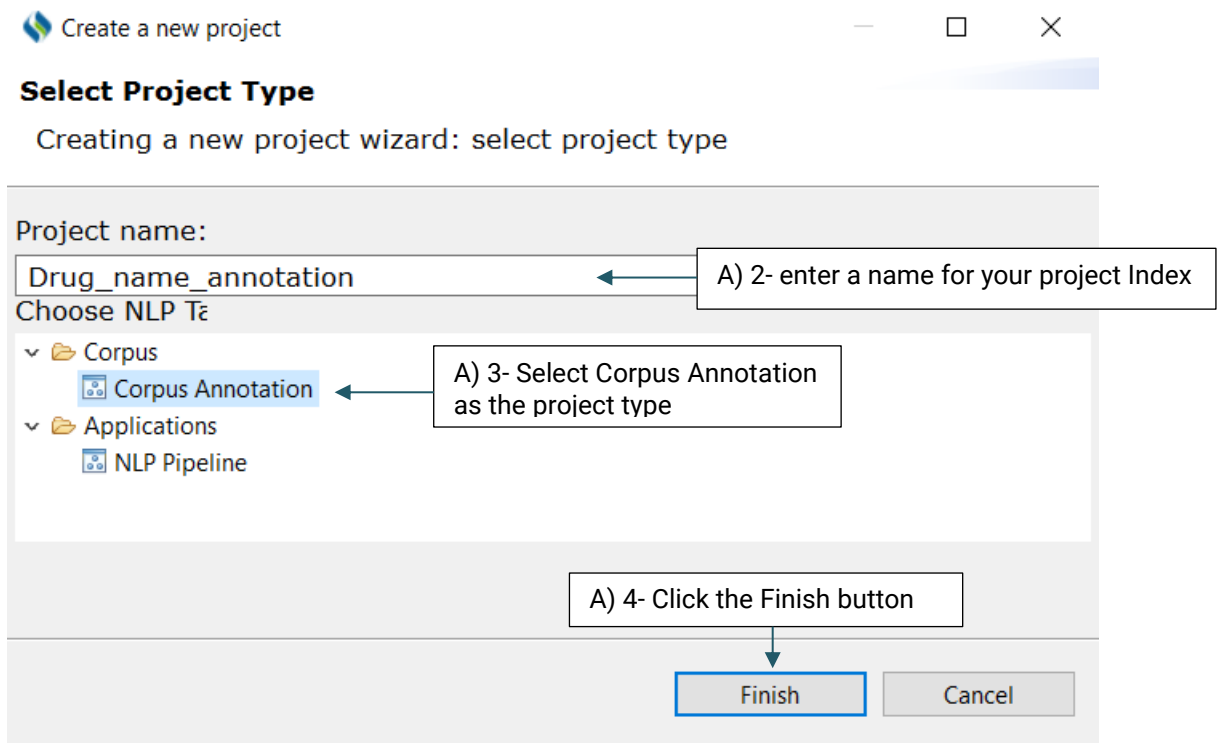


Figure 15.2 Creating a new Corpus Annotation project

A new project with the name that you have specified is created and placed in the Corpus panel. (Figure 15.3)

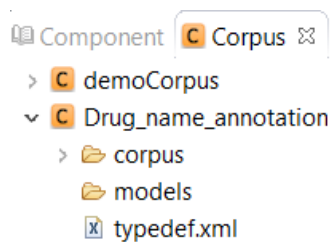


Figure 15.3 Creating a new Corpus Annotation project

Double click the project name to view its content. The created project contains two main folders:

Corpus: Contains the files that will be annotated

Models: Contains the machine learning models generated from the annotated files. In addition, the prediction results generated from the n-fold cross-validation process and gold standard annotations are included in this folder.

15.1.2 Import annotation files

After you have created a project, follow the steps defined below to import the files that you want to annotate:

(Please note that you can import the files to either train or test folders.)

A) To import the files that you want to annotate:

1. Right click on the train folder under the corpus folder in the CorpusView panel
2. Select the import function from the context menu (Figure 15.4). A pop-up window will appear.

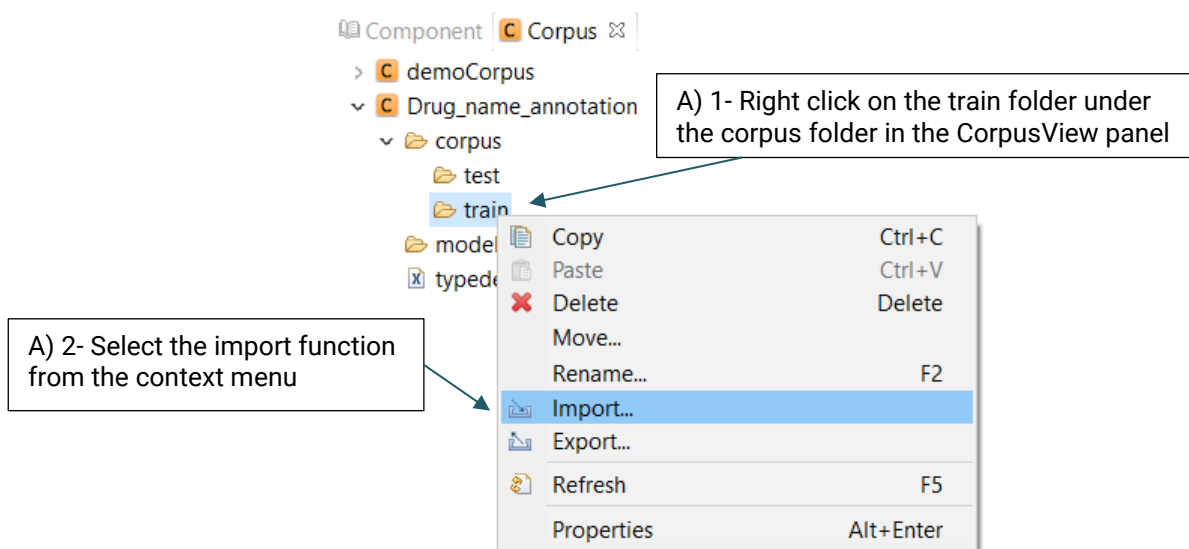
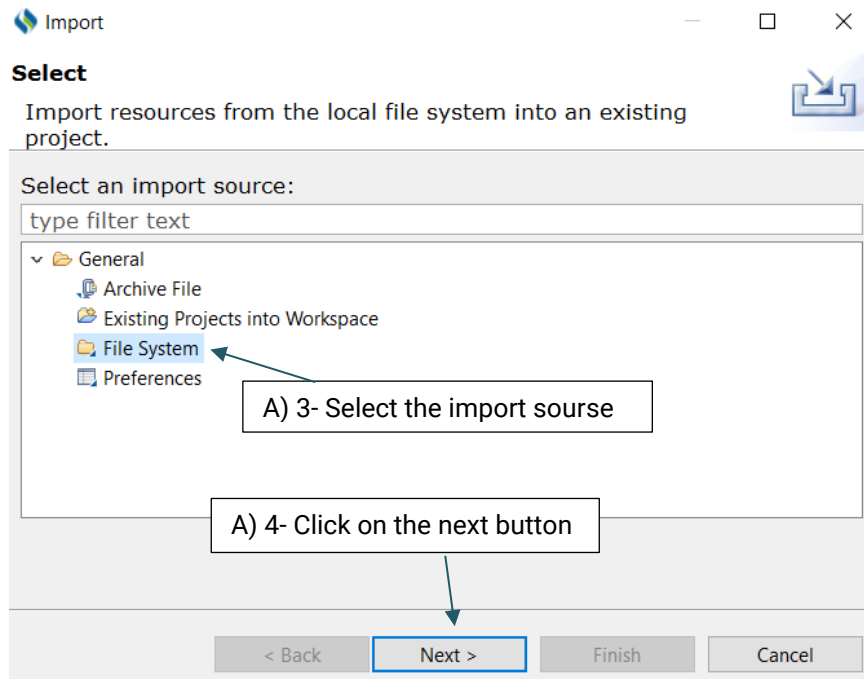


Figure 15.4 Context menu for importing the input files

3. On the pop-up window, select the import source. Here, you need to select “File System” which is already selected by default.
4. Click on the Next button



5. Click on the Browse button on the top of the window to choose the folder of your choice. The selected folder will be displayed on the left side of the window, and the content of the folder will be displayed on the right side. To import you desired files, check the checkboxes next to the files of your choice.

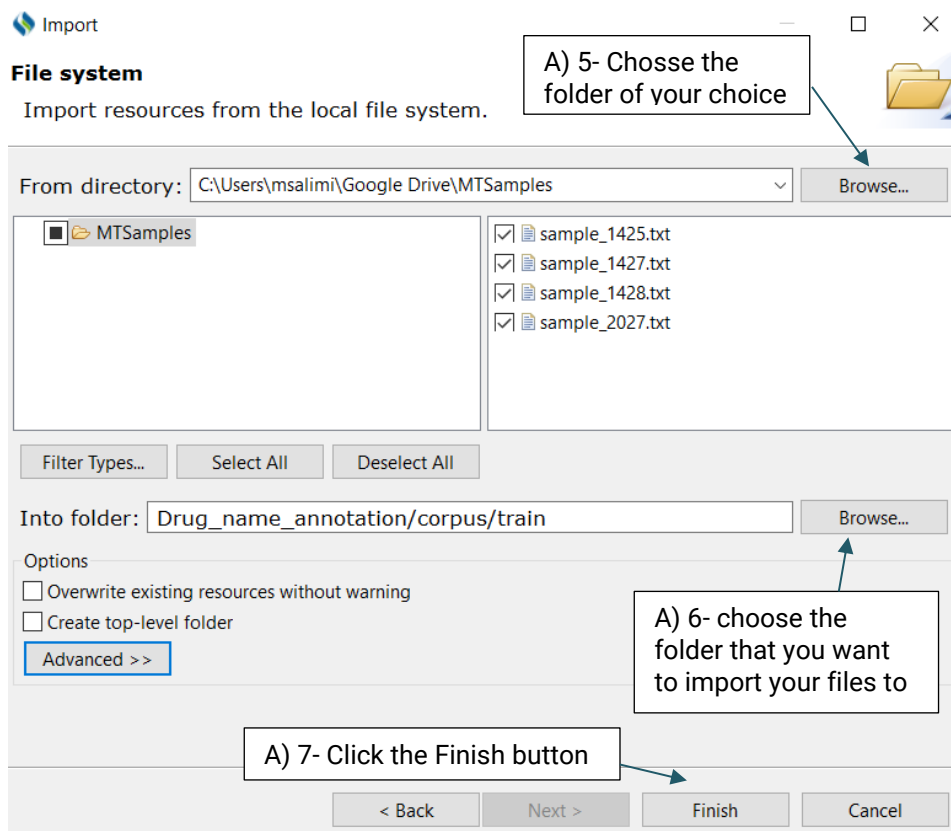
You also have three options to choose from: *Filter Types/ Select All/ Deselect All*

Filter Types: Allows you to define the type of files that will be imported. The only extensions that you will work with in CLAMP are txt, and .xml. For example, you may only want to import files with the .txt extension

Select All: Allows you to choose all displayed files

Deselect All: Allow you to deselect the files that have already been selected

6. Click on the Browse button next to the "Into folder" field to choose the folder that you want to import your files to. Here, we keep the default directory
7. Click on the Finish button.



Now that the selected files have been imported to your desired folder, you can start annotating them. Double click on the files to open them in the middle window and annotate them. Upon double clicking each file, you will notice that another file with the same name but a different extension (.xmi) has been added to your folder and displayed on the screen. This is the file type used by CLAMP for display and interaction purposes (Figure 15.5).

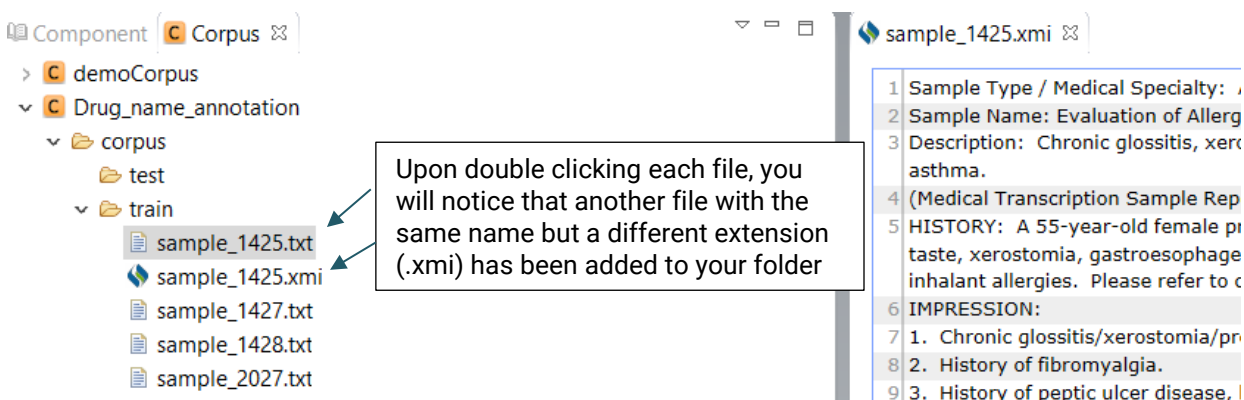


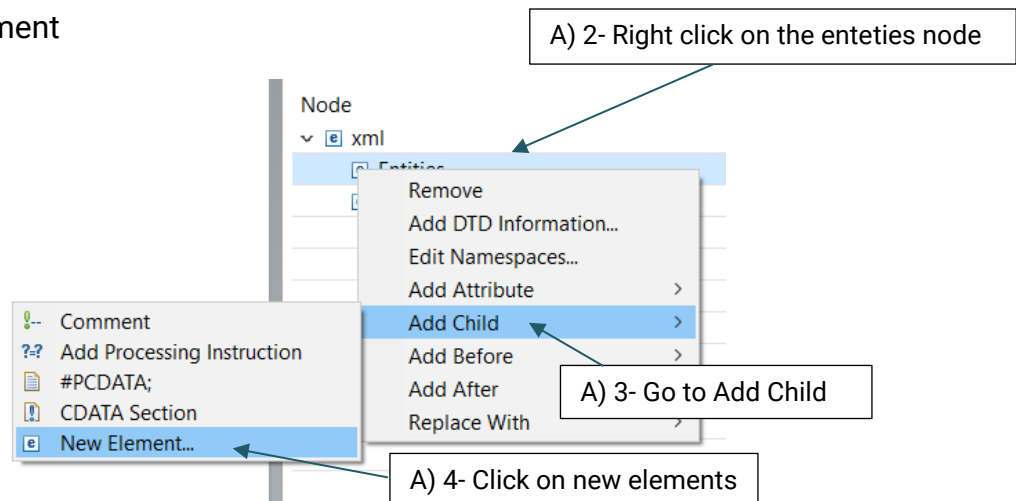
Figure 15.5 The content of an annotation file

15.1.3 Define entity & relation types

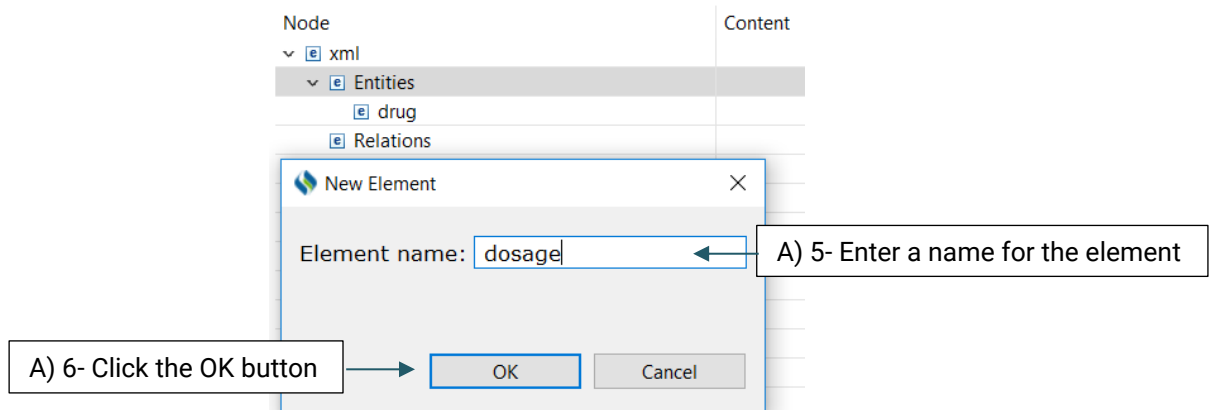
Before starting annotation, you need to define the semantic types that you will use for this purpose. Semantic types in CLAMP refer to entities(e.g, 'problem/treatment/test') and the relations between them.

A) To define a new entity type:

1. Double click on the typedef.xmi file under the models folder to open it. Using this file, you will be able to define a schema for entities and the relation types among them:
2. Right click on the Entities node
3. Go to "Add Child"
4. Click on New Element



5. In the pop up window, enter a name for the element
6. Click the OK button



The created element will be added to the Entity node (Figure 15.6)

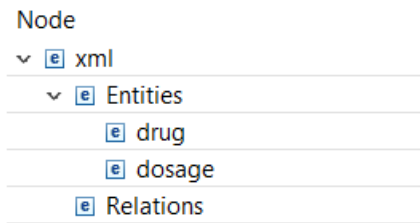
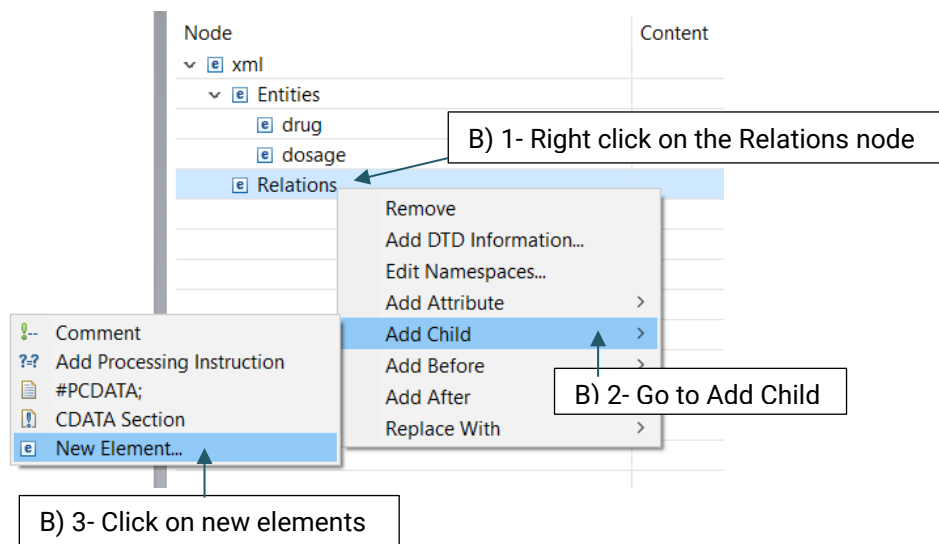


Figure 15.6 The created element will be added to the Entity node

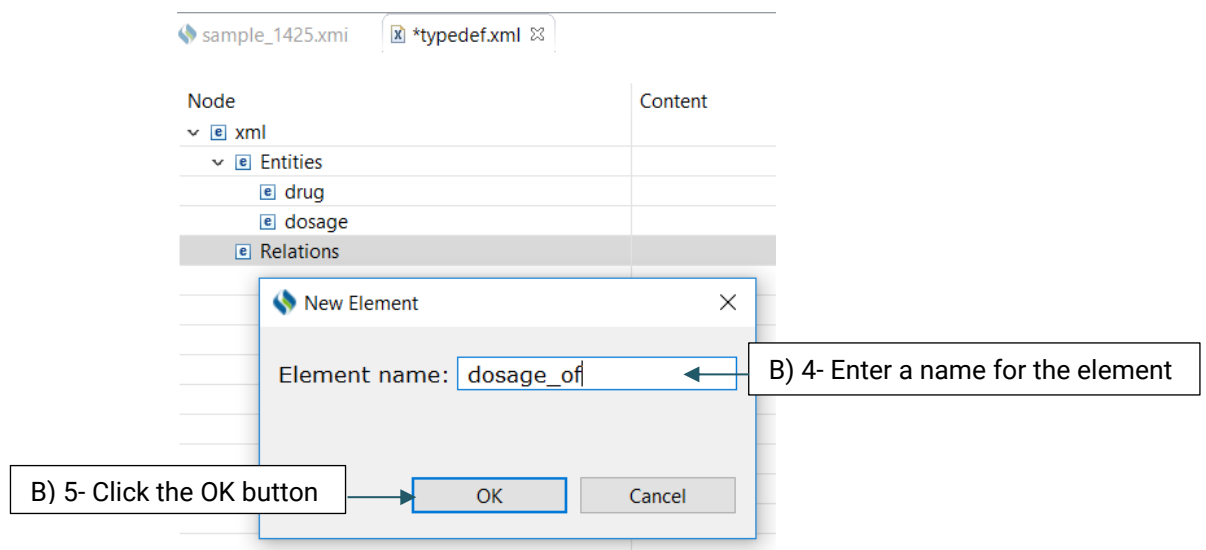
The above steps should be repeated for every element that you want to add to the Entity node.

B) To define a new relation type:

1. Right click on the Relations node
2. Go to "Add Child"
3. Click on New Element



4. In the pop up window, enter a name for the relation
5. Click the OK button

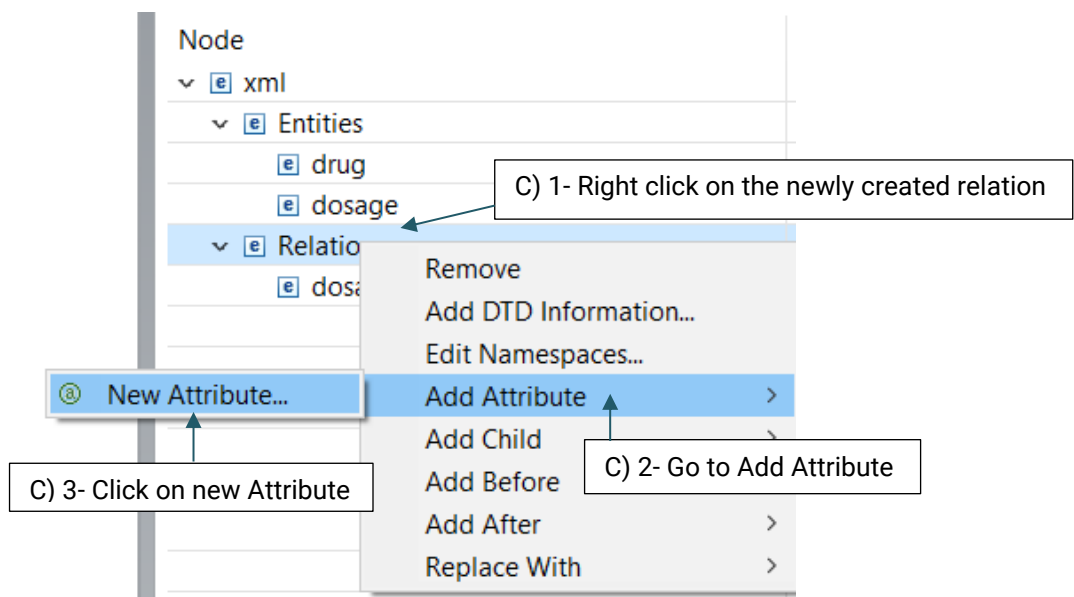


Then, you need to decide which entities are involved in this relation. There are two roles of arguments an entity can hold in a relation: **From**, and **To**.

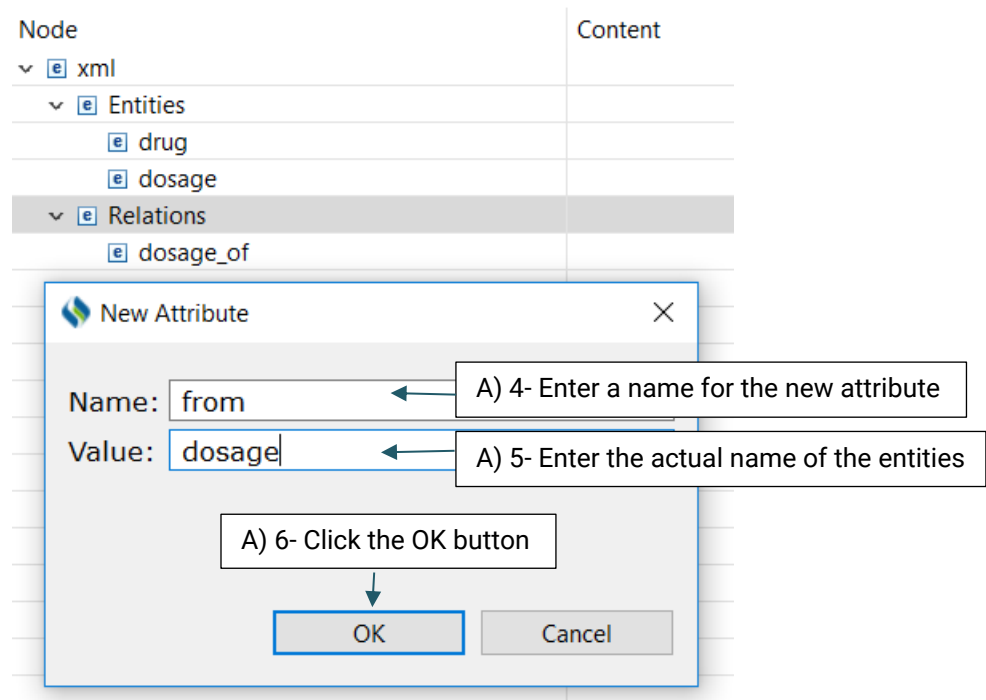
“From” refers to an independent entity while “To” indicates the dependent entity.

C) To select the entities that are involved in a relation:

1. Right click on the newly created relation
2. Go to “Add Attribute”
3. Click on “New Attribute”



4. In the pop up window, enter a name for the new attribute (you will use “from” for the independent entity, and “to” for the dependent entity)
5. Enter the actual name of the entities for the Value field
6. Click the OK button
7. Click on Save at the top of the window



15.1.4 Start Annotation

Now that you have set your desired schema, you are ready to start annotating your corpus. First, open your desired .xmi file, then:

To assign entity: Place your mouse over a word/phrase to highlight it and assign an appropriate entity to the selected text. (Figure 15.6)

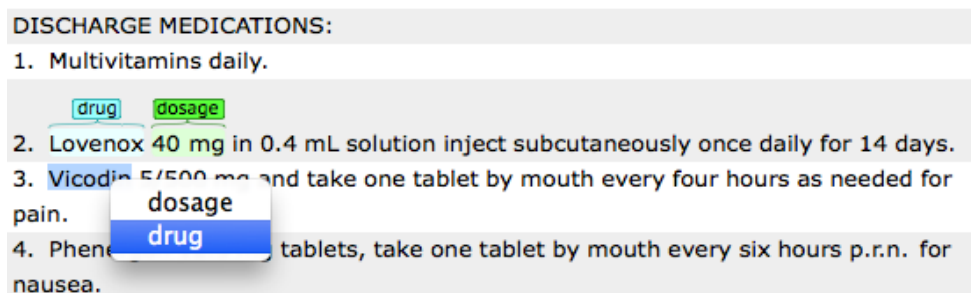


Figure 15.6 Named entity annotation

To assign relation: By dragging your mouse from an independent entity and dropping it to a dependent entity, the names of possible relations will occur. Choose the appropriate relation name by clicking on one of the displayed names. (Figure 15.7)

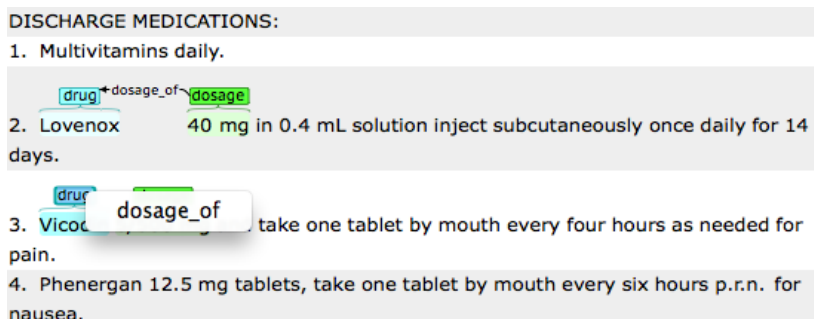


Figure 15.7 Relation annotation

Please remember that you can only assign a relation to the entities that have already been defined in that relation.

Once you have completed annotating the corpus, save your changes by clicking on the save button at the top of the screen.

15.1.5 Visualization of entity & relation

Although different colors are automatically assigned to the different items in the “Display Option”, (Figure 15.8), you are able to change them at any time.

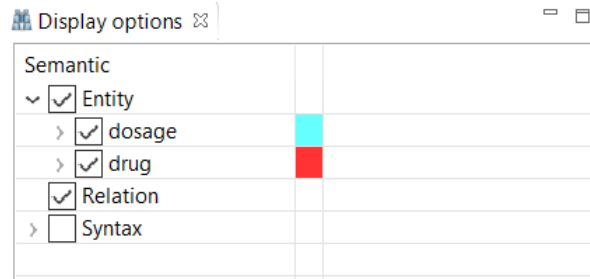
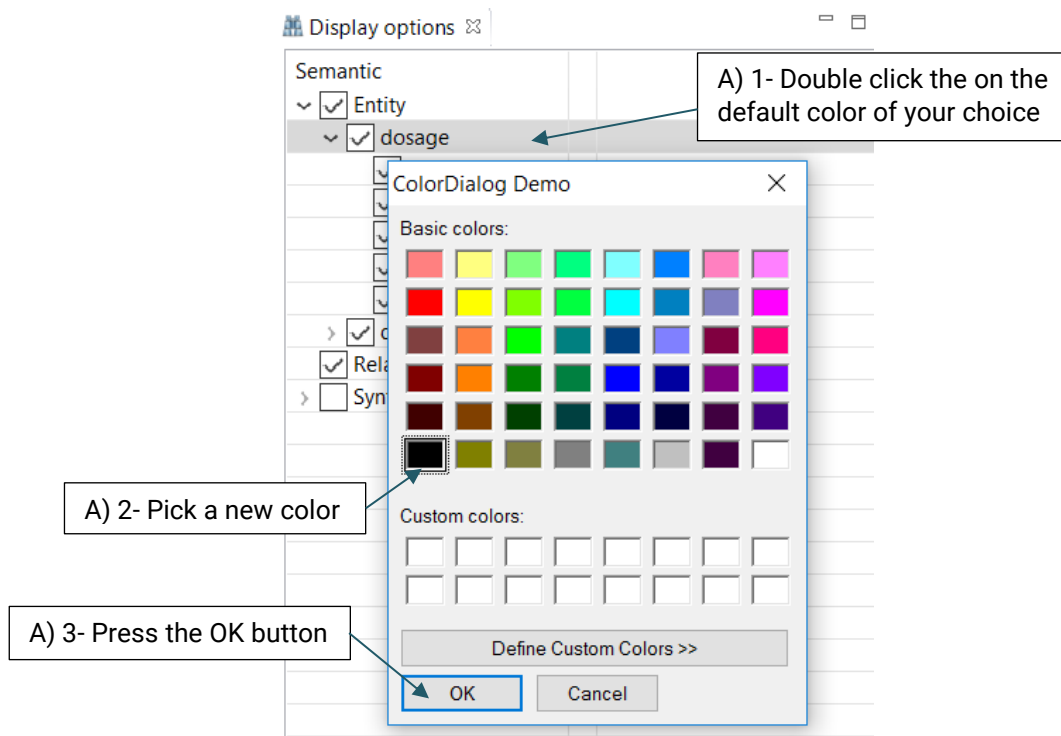


Figure 15.8 Change visualization of the annotated entity/relation

A) To change the default colors for semantic types:

1. Double click on the default color for the entity of your choice
2. Pick a new color from the color picker window
3. Press the OK button



15.1.6 Pre-Annotation of entity and relation

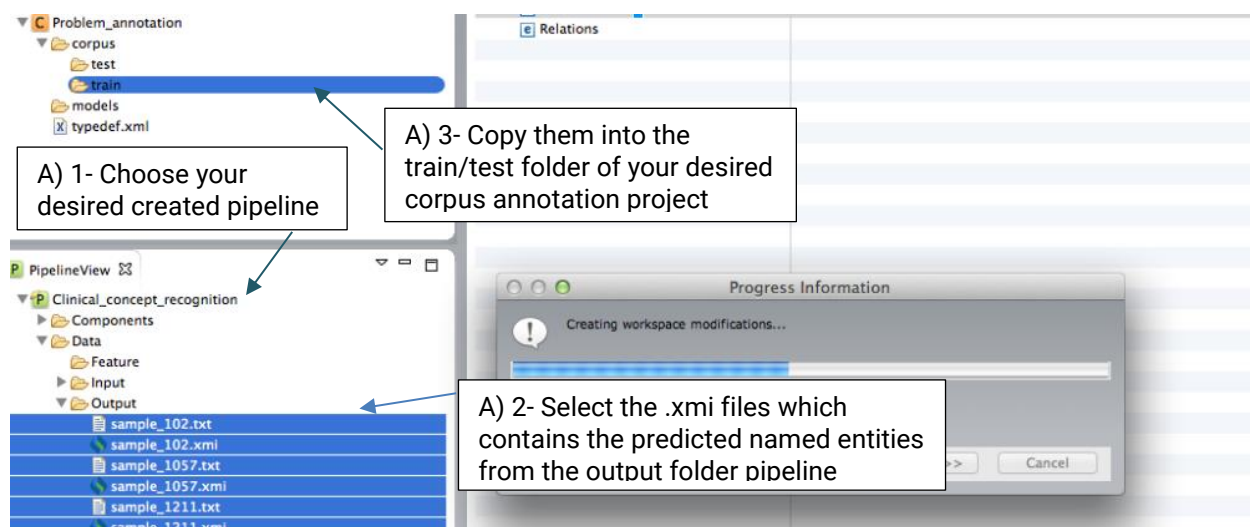
As annotating a corpus from scratch may be a time-consuming and costly process, CLAMP offers an advanced feature called “pre-annotation” function which facilitates this process. The “pre-annotation” function relies on the existing models in CLAMP and is highly customizable.

A) Using pre-annotation function

1. Choose your desired pipeline to annotate your files in a corpus project.

For more information on how to run a pipeline, go to [“Run the pipeline”](#) section.

2. Select the .xmi files which contains the predicted named entities from the output folder.
3. Copy them into the train/test folder of your desired corpus annotation project. To copy, right click on the selected files and choose copy.



Double click on the files to view their contents in a new window. As you can see in Figure 15.9, the identified named entities in the file are already highlighted. Now you can start your own annotation.

1	Sample Type / Medical Specialty: Emergency Room Reports
2	Sample Name: Consult - ICU Management
3	Description: Consultation for ICU management for a patient with possible problem portal vein and problem superior mesenteric vein thrombus leading to mesenteric ischemia.
4	(Medical Transcription Sample Report)
5	REASON FOR CONSULTATION: ICU management.
6	HISTORY OF PRESENT ILLNESS: The patient is a 43-year-old gentleman who presented from an outside hospital with complaints of problem right upper quadrant pain in the abdomen, which revealed problem possible portal vein and problem superior mesenteric vein thrombus leading to mesenteric ischemia. The patient was transferred to the ABCD Hospital where he had a weeklong course with progressive improvement in his status after aggressive care including intubation, fluid resuscitation, and watchful waiting. The patient clinically improved; however, his white count remained

Figure 15.9 A file with pre-annotated named entities

16. Machine learning model development

Clamp enables you to build your own machine learning model based on a corpus that you have already annotated or a pre annotated one that you have imported into a corpus annotation project. The model can be used for predictions on new files. In the current version of Clamp, CRF (Conditional Random Field) is used to build machine-learning model for named entity recognition (NER).

The first step to build a Machine Learning model is to configure its schema. After configuring the schema, you will be able to start running the training model and evaluation processes. Once these processes are completed, you can view the generated model, its associated log files, and named entities predicted by the model in the output folder. The following steps will guide you on how to perform the steps mentioned above.

16.1 Building machine learning models (NER model)

1. Select your desired train folder on the Corpus panel
2. Click on the “Train Model” button at the top of the window as shown in Figure 16.1

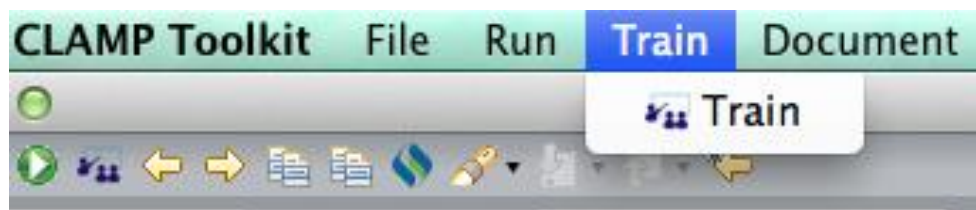


Figure 16.1

3. On the pop up window as shown in Figure 16.2, enter a name for the model that you are building
4. Click the checkbox for the features that you want to include in your model

Note: During the model building process, the training files can not be annotated. Clicking on the text of the training files pops up an alert window indicating that the user operation is waiting for a function to complete, (Figure 16.3).

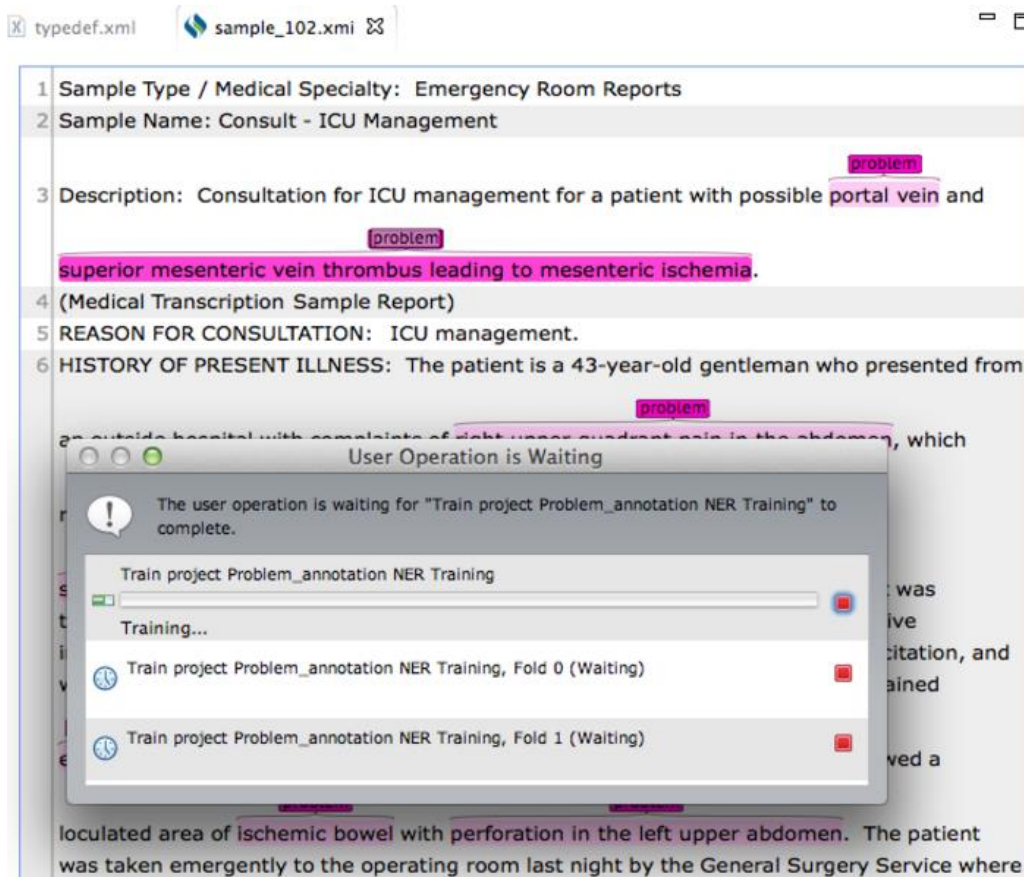


Figure 16.3 Annotations on the training file will be paused during the model building process

16.2 Check output models & logs

By default the built models, their associated logs, and the named entities predicted by each model (in the output sub-folder) are stored in the models folder. As shown in Figure 16.4, both the model built during n-fold cross validation and the model trained on the whole training set are also stored in the directory. The content of the log files includes the output information of the training process and the evaluation performance of each specific folder for cross validation. (Figure 16.5)

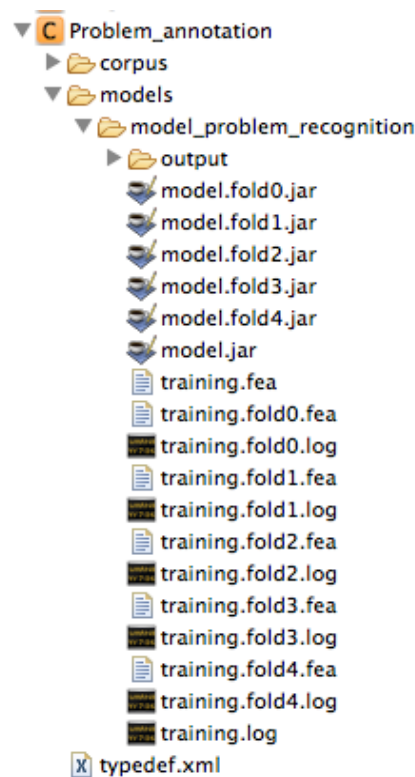


Figure 16.4 Annotations on the training file will be paused during the model building process

```
#####
2015-10-07 14:53:23      Evaluation result:
2015-10-07 14:53:23      correct      predict      gold      P      R      F1      semantic
2015-10-07 14:53:23      82      103      159      0.796      0.516      0.626      treatment
2015-10-07 14:53:23      660      742      777      0.889      0.849      0.869      problem
2015-10-07 14:53:23      163      178      196      0.916      0.832      0.872      test
```

Figure 16.5 Cross-validation performance in training.fold0.log

16.3 Use your own model in pipeline

The steps below show how you should use your own model to recognize named entities:

1. Make sure that you have selected your desired project folder on the Corpus panel
2. Go to models> model_xxx> output, then, right click on the file labeled model.jar and select copy (Figure 16.6)
3. Go to the pipeline panel and select the pipeline of your choice
4. Click on the small arrow next to it to expand it
5. Go to Components -> Named Entity Recognizer -> *DF_CRF_based_named_entity_recognize*
6. Paste the copied file into “CRF based name entity recognizer” folder by right clicking on the folder and choosing past
7. Double click on the Config.conf file in the “CRF based named entity recognizer” folder to open it
8. Click on the three dots button to replace the default model for “CRF based named entity recognizer” with your own model (Figure 16.7)
9. Click on the Open button
10. Click the save button at the top of the page to save the changes (Figure 16.8)

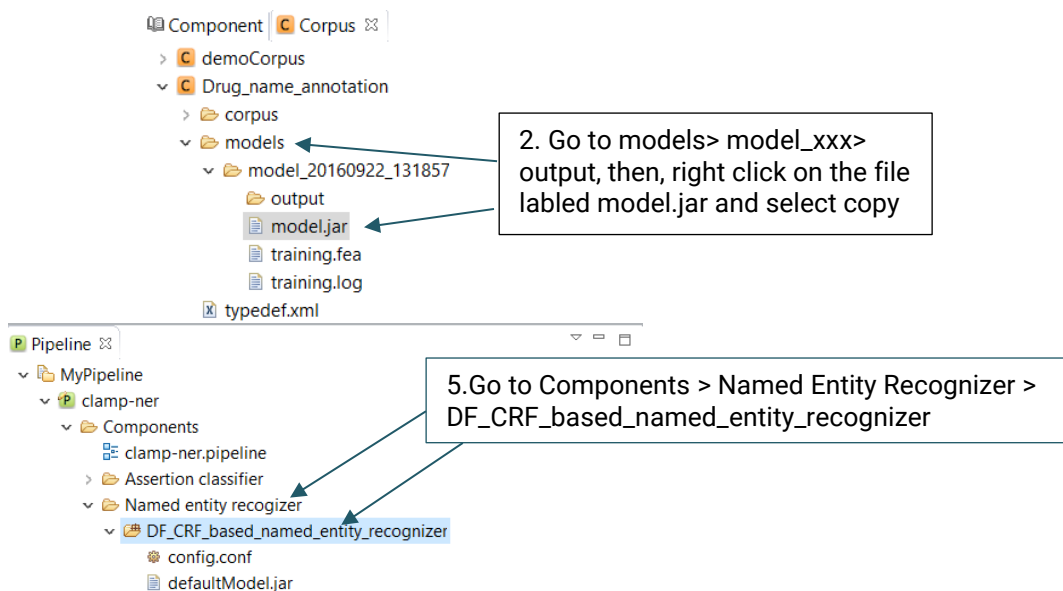


Figure 16.6 How to use your own model to recognize name entities

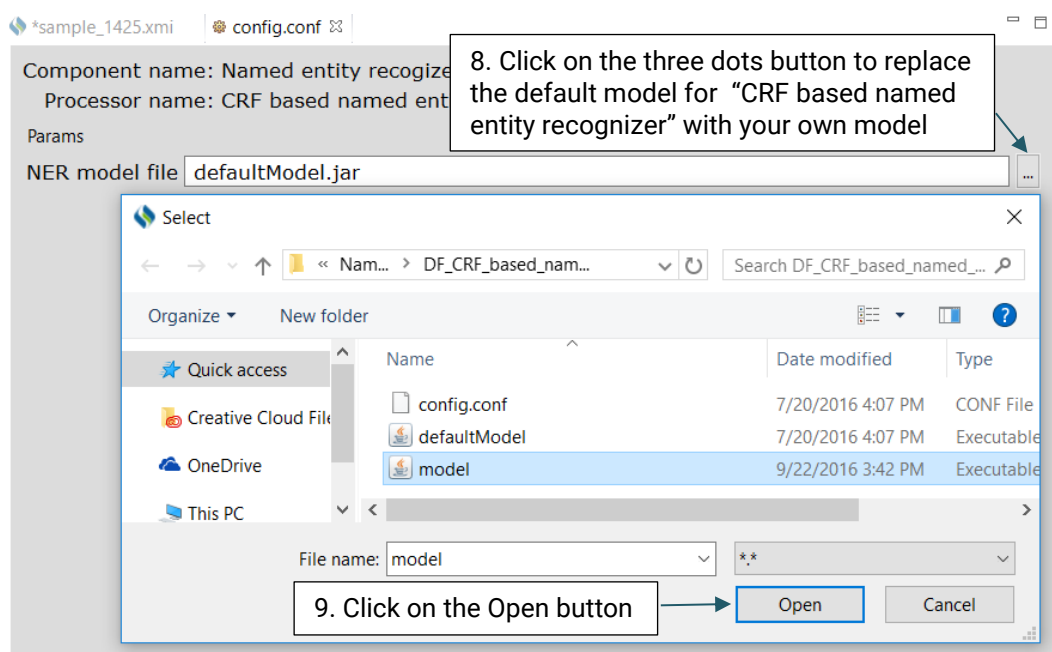


Figure 16.7 Replace the default model with your own model

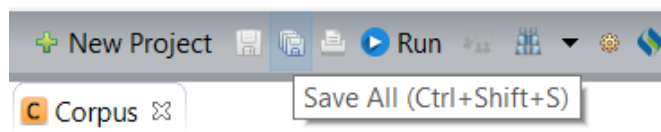


Figure 16.8 Click the Save button at the top of the page

16.4 Visualization for error analysis

You will be able to evaluate the performance of the NER model only if you have already checked the checkbox(es) for “Use test set” or/and “CV, fold” when creating the model. For more information on creating a new NER model, go to [Building machine learning models](#) section. Once the model is built, you can conduct an error analysis to compare the gold-standard annotations with the predicted ones (the annotations that are built based on the model that you have specified).

To perform error analysis:

Double click on one of the .xml files listed in the output folder of your choice on the corpus panel. This will open a new window where you can see the original text along with both gold-standard and predicted annotations (Figure 16.9).

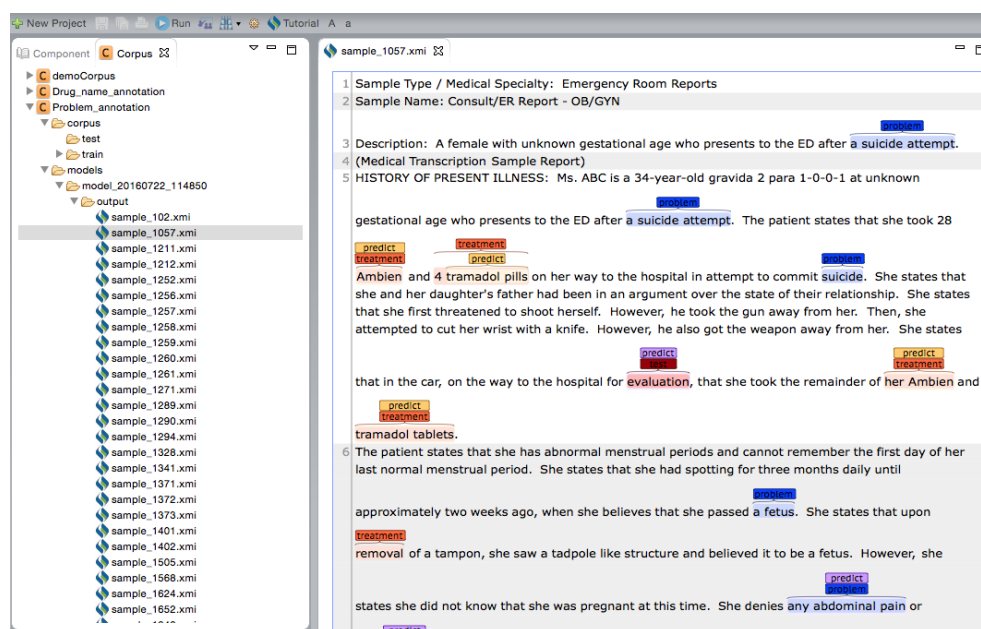


Figure 16.9 - The .xml file content which includes the original text along with both gold-standard and predicted annotations

Please note that all named entities in both gold-standard and predicted annotations are listed on the "Display Options" panel. You can choose which named entities to be highlighted in the text file and assign different colors to them as described in [“Visualization of entity and relation types”](#) section.