Preannotator 1.0 User Manual

This is not a published version. Please don’t distribute without asking. Thanks.

Preannotator is a rule based preannotation tool which is designed to speed up preannotation process. Its main components consist of FastNER and FastContext.

# Set up

Preannotator runs in Java environment. Make sure you have Java 1.8 **JDK** (because reflection is used, which is supported by JRE) or higher version installed or set up. Unzip the files to where you prefer.

Change the permission of the shell script files (“run” and “help”) to be executable, if you use Linux system and the files are not executable yet.

# Demonstration

The following steps will briefly demonstrate a whole process of developing pre-annotations, using some dummy rules.

The example document used here is from [http://www.mtsamples.com/site/pages/sample.asp?Type=89-Discharge%20Summary&Sample=646-Cardio/Pulmo%20Discharge%20Summary](http://www.mtsamples.com/site/pages/sample.asp?Type=89-Discharge Summary&Sample=646-Cardio/Pulmo Discharge Summary). It is located in “data/input”

# Import your txt corpus to database

If you are using Linux system, go to the directory where you unzip the files, use the following command:

|  |
| --- |
| **./**run Import "data**/**input" data**/**my.sqlite corpus true txt |

The command imports all the files with extension name “txt” in the directory "**/**home**/**my**/**Documents**/**corpus", to a table named “corpus” in the SQLite database "data/my.sqlite ". If table corpus already exists, overwrite it. For more detailed information about "ImportTxtFiles", refers to the [Reference section](#_ImportTxtFiles).

If you use windows system, use the following command instead:

|  |
| --- |
| run Import "data**/**input" data**/**my.sqlite corpus true txt |

All the following windows’ commands will be similar---just remove the first two characters "**./**" and change the directory path accordingly. Thus, in the following manual, windows version will be omitted.

# Build your rules

There are some example rules in the rule files, “conf/concept\_rule.csv” and “conf/concept\_crule.csv.” The former file is to define the token-based rules, and the latter is to define character-based rules. For more information about the rule specification, please refers to [Reference section](#_Build_your_rules).

Here we only make one simple rule for each type of rules. Backup the original rule files if you want. Remove all the content in the two files. Open the files using any txt editor.

In “conf/concept\_rule.csv” we put:

|  |
| --- |
| patient Concept ACTUAL |

**Note**: The white space between “patient”and “Concept”, “Concept” and “ACTUAL” is a “TAB” character. The rule means whenever we found a word “patient”, annotate it as a “Concept”.

In “conf/concept\_rule.csv” we put:

|  |
| --- |
| 4 1 Digit  2 1 Digit |

Same here, the white spaces are “TAB” characters. The rule means whenever we found a character “4” or “2”, annotate it as “Digit”

# Test your rules

To see how your rules work, use the following command:

|  |
| --- |
| **./**run Runpipe data**/**my.sqlite corpus data**/**output.sqlite output v1 |

It will read your text corpus from table “corpus” in "data/my.sqlite" (a SQLite database file), make the annotations based on your rules, and output the annotations into the table "output" in "data/output.sqlite" marked with the annotator "v1".

To check your annotations, use DBeaver to open “data/output.sqlite”. For detailed information about how to use DBeaver, please refer to the [reference section](#_View_annotations_in). Now you will see something like:

|  |
| --- |
|  |

# Improving your rules

As you can see, “1.1 cm” is annotated as two separate digits. Now replace the rules in “conf/concept\_rule.csv” with the following two rules (this will create some false positive and false negative annotations when comparing):

|  |
| --- |
| 3 1 Digit  2 1 Digit |

Run the pipe again, but mark the annotator as “v2”:

|  |
| --- |
| **./**run Runpipe data**/**my.sqlite corpus data**/**output.sqlite output **v2** |

Now check it in DBeaver, you can see the newer version annotations:

|  |
| --- |
|  |

## Check the differences

However, you want to make sure what changes have been made in the output. So the following command will compare “v2” against “v1”:

|  |
| --- |
| **./**run Compare data**/**output.sqlite output v2 data**/**output.sqlite output v1 diff |

In the terminal, you will get comparison measurements report:

|  |
| --- |
|  |

And you can check the detailed differences in “diff” table using DBeaver:

|  |
| --- |
|  |

The note column specify whether the annotation is a false positive (annotations exist in “v2” but not in “v1”) or false negative (annotations exist in “v1” but not in “v2”).

You can filter to see false positive annotations only by type “note=’fp’” in the search input field, or false negative only using “note=’fn’”.

Then, you can modify your rule accordingly, based on the compared results.

# View your annotations in highlighted text

There are three ways to view the annotations within original document context using highlighted text. For detailed information, please refer to the [reference section](#_View_annotations). Here we will try one of them, using UIMA annotation viewer. Type following command:

|  |
| --- |
| **./**run Runpipe data**/**my.sqlite corpus data**/**output xmi |

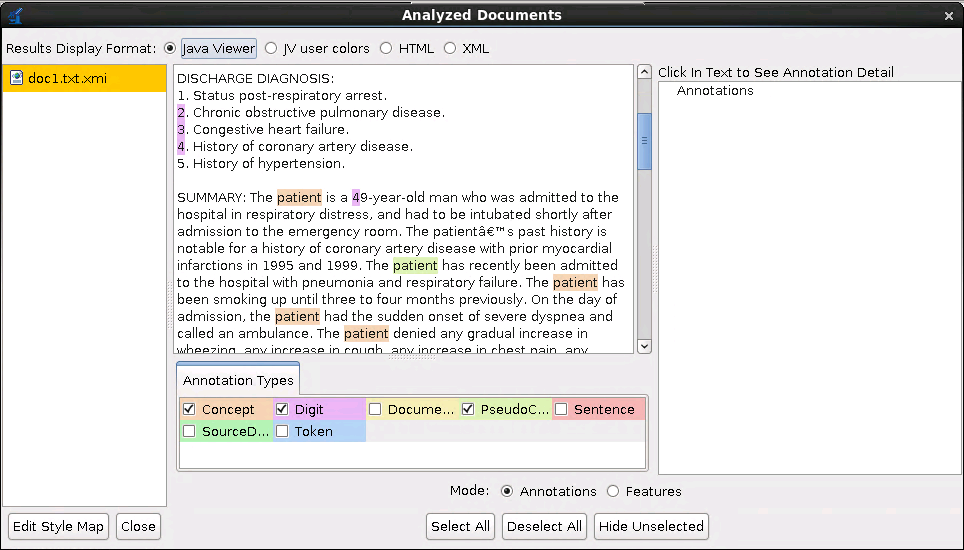
This will output the annotations into directory “data/ouput” using UIMA XMI format.

|  |
| --- |
| **./**run Viewer |

It pop up a setting window:

|  |
| --- |
|  |

For the input Directory, Browse to and select “data/output”. For the ”TypeSystem or AE Descriptor File”, Browse to and select “desc/type/customized.xml”. Click the “ View” button, the annotations will show up:



# Reference:

There two scripts are runnable: “help” and “run” (in Windows system, the script files are “run.bat” and “help.bat”).

To run the script, in Linux systems, use “./help” or “./run”. In Windows systems, use “help” and “run”.

## help

Type “./help” in terminal, will print out all the classes available to use. Type “./help {classname}” will print out detailed information about a given class. For instance, “./help ImportTxtFiles” will print out the following information in terminal:

|  |
| --- |
|  |

## run

Use “./run classname parameters” to execute classes. There are classes in preannotator1.0:

ImportTxtFiles, Runpipe, Viewer, RunCPE, CompareWithGold, XMISQLSimpleComparator.

### ImportTxtFiles

This program import txt files into database (SQLite or MySQL). It takes at lease 3 parameters:

1. Input directory where the txt files are stored.

2. Database file. SQLite: uses the path of SQLite file (must use extension “sqlite”). MySQL: uses the path of MySQL configuration file(must use extension “mysql”), e.g. conf/sf.mysql

3. The table to save the imported data.

4. (optional) Whether overwrite previous table if the table exists

5. (optional) File types (extension names separated by whitespace) that should be included.

#### For example:

|  |
| --- |
| **./**run Import "**/**home**/**my**/**Documents**/**training" data**/**my.mysql corpus true txt |

The command above will read all the “txt” or “text” files under “/home/my/Documents/training” to MySQL database, create a table “corpus”. The MySQL connect setting is defined in “conf/my.mysql”.

|  |
| --- |
| **./**run Import "**/**home**/**my**/**Documents**/**training" data**/**my.sqlite corpus true txt |

The command above will read the same files to a SQLite database, create a table “corpus”. The SQLite database file is “data/my.sqlite”.

### Runpipe

This program takes 3~5 parameters, depends on which format of intput and output data.

1: The is the sql file or mysql configuration file.

2: The table where text corpus is saved.

The rest parameters will be taken for output format:

If the 3rd parameter is a directory:

(1) If 4th parameter is “xmi” the runner will output UIMA format (.xmi) files into that directory.

(2) If 4th parameter is “brat” the runner will output Brat format (.anno for annotations, .txt for source text) files into that directory.

(3) If 4th parameter is “ehost” the runner will output eHOST format (.knowtator.xml under “saved” directory, .txt under “corpus” directory) files into that directory.

If the 3rd parameter is a sql file or mysql configuration file, the runner will take the 4th and 5th parameters to output the data into database table.

The 4th parameter is the table to output, the 5th parameter is add the annotator's name (to differentiate different versions of output)

#### For example:

|  |
| --- |
| **./**run Runpipe conf**/**my.mysql corpus conf**/**my.mysql output v1 true txt |

The command above will read all the text records in “corpus” table from MySQL database using “conf/my.mysql” configuration, and output the preannotations to “output” table using “conf/my.mysql” configuration, and mark all the annotations with “v1” as the annotator.

|  |
| --- |
| **./**run Runpipe conf**/**my. sqlite corpus conf**/**output.sqlite output v1 true txt |

The command above will read all the text records in “corpus” table in “*data/my.sqlite*” file, and output the preannotations to “output” table in “*data/output.sqlite”*, and mark all the annotations with “v1” as the annotator.

**NOTE**: if both input and output are SQLite, they **must be different files**, because sqlite does not support multi-thread access.

|  |
| --- |
| **./**run Runpipe conf**/**my.mysql corpus data**/**output xmi |

The command above will read all the text records in “corpus” table from MySQL database using “conf/my.mysql” configuration, and output the pre-annotations to “data/output” directory in “xmi” format—can be viewed in annotation viewer.

|  |
| --- |
| **./**run Runpipe conf**/**my.mysql corpus data**/**output brat |

The command above will read all the text records in “corpus” table using “conf/my.mysql” configuration, and output the preannotations to “data/output” directory in “Brat” annotation format—can be viewed in Brat.

|  |
| --- |
| **./**run Runpipe conf**/**my.mysql corpus data**/**output ehost v1 |

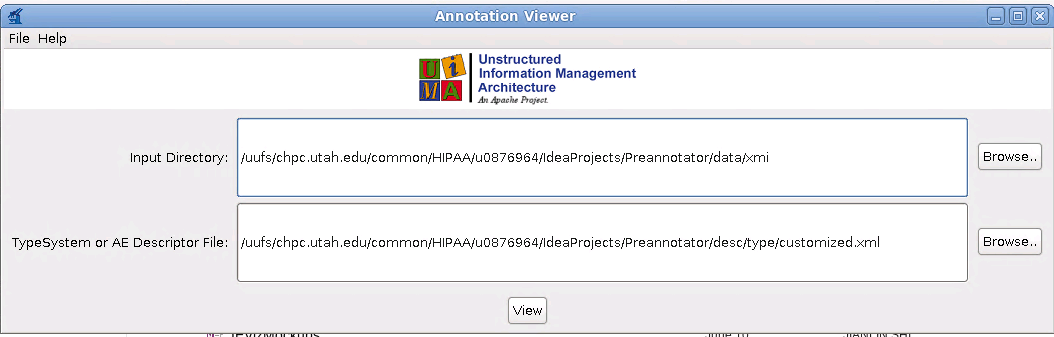
The command above will read all the text records in “corpus” table using “conf/my.mysql” configuration, and output the preannotations to “data/output” directory in “eHOST” annotation format—can be viewed in eHOST. The annotator will be named “v1”.

### Viewer

This program doesn't take any parameters. It will open an annotation viewer for xmi files.

|  |
| --- |
| **./**run Viewer |

It will open a window for you to set up:



The inputDirectory need to be set to the directory where the xmi files are stored.

The TypeSystem or AE Descriptor file need to be set to “desc/type/All\_Types.xml”. If “desc/type/customized.xml” exists, you should set to “desc/type/customized.xml” instead, which means you rule files have used some customized concepts.

### RunCPE

This program will run CPE through GUI or command line. It is only useful when you want to customize the components of your pipeline. It takes 0~2 parameters:

1. If there is no parameters, GUI will be initiated.

2. If there is one parameter, it should be the path of CPE descriptor.

3. If there is a second parameter, it should be the logFile, which will save the performance report.

### Compare

This program compare the difference of two annotators (two different versions of preannotators). It can log the difference into database for later review, and output F scores.

You need at lease 6 parameters.

1: the SQL file of annotations(point to the sql file if using SQLite, or the mysql configuration file if using MySQL)

2: the table of annotator1's annotations

3: the name of annotator1

4: the SQL file of annotations(point to the sql file if using SQLite, or the mysql configuration file if using MySQL)

5: the table of annotator2's annotations

6: the name of annotator2

7(optional): output table name to save the difference. If don't provide, only scores will be reported.

8(optional): The concept type to compare (optional). If not defined, will compare all the concept types

9(optional): The category (subType) of concepts to compare(optional). If not defined, will compare all the categories

#### For example:

|  |
| --- |
| **./**run Compare data**/**my.mysql output v2 data**/**my.mysql output v1 diff |

The command above will compare the annotations of annotator “v2” against “v1”, and output the difference in “diff” table.

The “note” column will specify the disagreement, whether is false positive (“fp”: annotations in “v2” but not in “v1”) or false negative (“fn”: annotations in “v1” but not in “v2”)

### CompareWithGold

Compare with previous program, this program will filter out the difference of two annotators using gold standard.

It only log the false positive annotations of annotator1, if the annotator1's annotations don't in annotator2's and gold standard.

It only log the false negative annotations of annotator1, if the annotator1 does't have the annotations but annotator2 and gold standard have.

You need at lease 9 parameters.

1: the SQL file of annotations(point to the sql file if using SQLite, or the mysql configuration file if using MySQL)

2: the table of annotator1's annotations

3: the name of annotator1

4: the SQL file of annotations(point to the sql file if using SQLite, or the mysql configuration file if using MySQL)

5: the table of annotator2's annotations

6: the name of annotator2

7: the SQL file of gold annotations(point to the sql file if using SQLite, or the mysql configuration file if using MySQL)

8: the table of gold annotations

9: the name of gold annotator

10(optional): output table name to save the difference. If don't provide, only scores will be reported.

11(optional): The concept type to compare (optional). If not defined, will compare all the concept types

12(optional): The category (subType) of concepts to compare(optional). If not defined, will compare all the categories

#### For example:

|  |
| --- |
| **./**run CompareWithGold data**/**my.mysql output v2 data**/**my.mysql output v1 data**/**my.mysql output gold diff |

The command above will compare the annotations of annotator “v2” against “v1”, and output the difference in “diff” table.

The difference is that CompareWithGold will ignore the “false positive” annotations if the annotation exist in gold standard, and ignore the “false negative” annotations if the annotations do not exist in gold standard. In short, you will have a purified difference ouput.

## View annotations

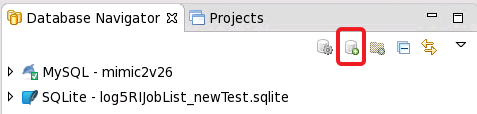
### View annotations in database table

You can use any database management software to view the preannotator’s data. However, DBeaver is recommended, because it provide more convenient view for long text as showing below.

You can download it from <http://dbeaver.jkiss.org/download/>.

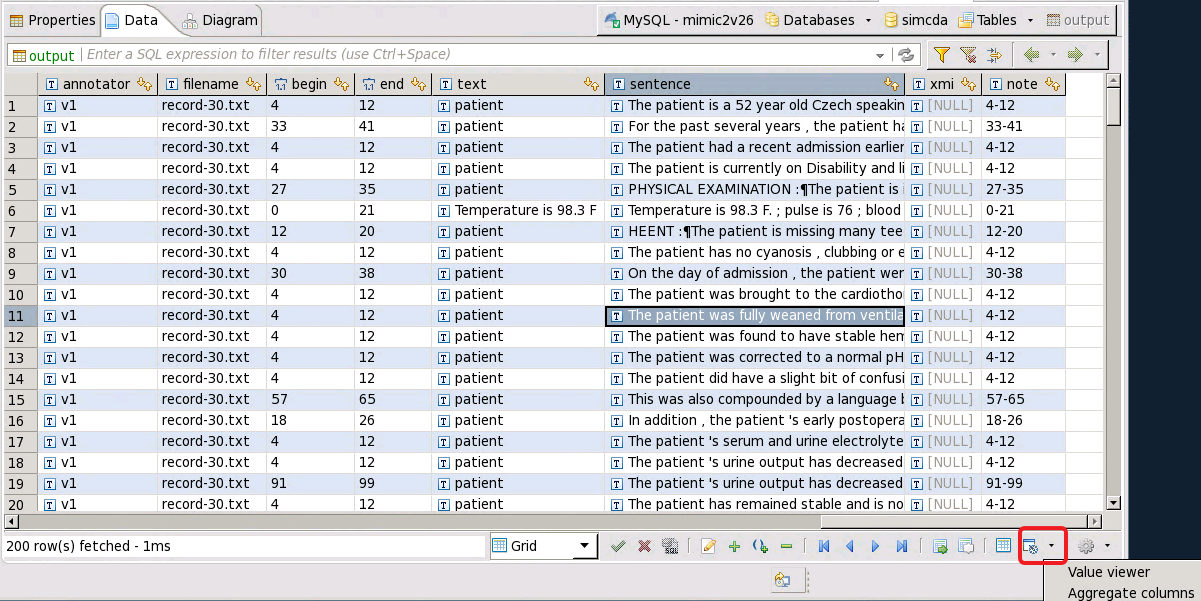
#### Set up database connections

Click the button below, follow the instructions to set up MySQL or SQLite connections.

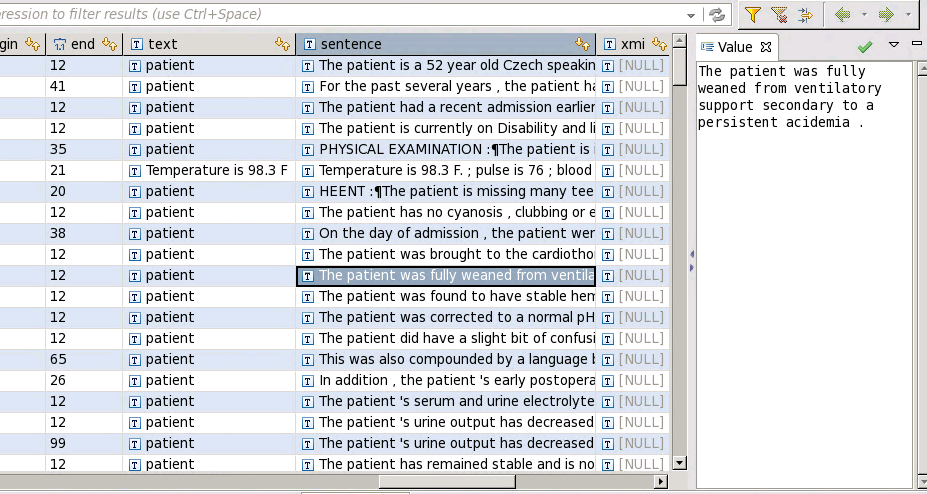


### View annotations:

Enable value viewer:



After enabled:

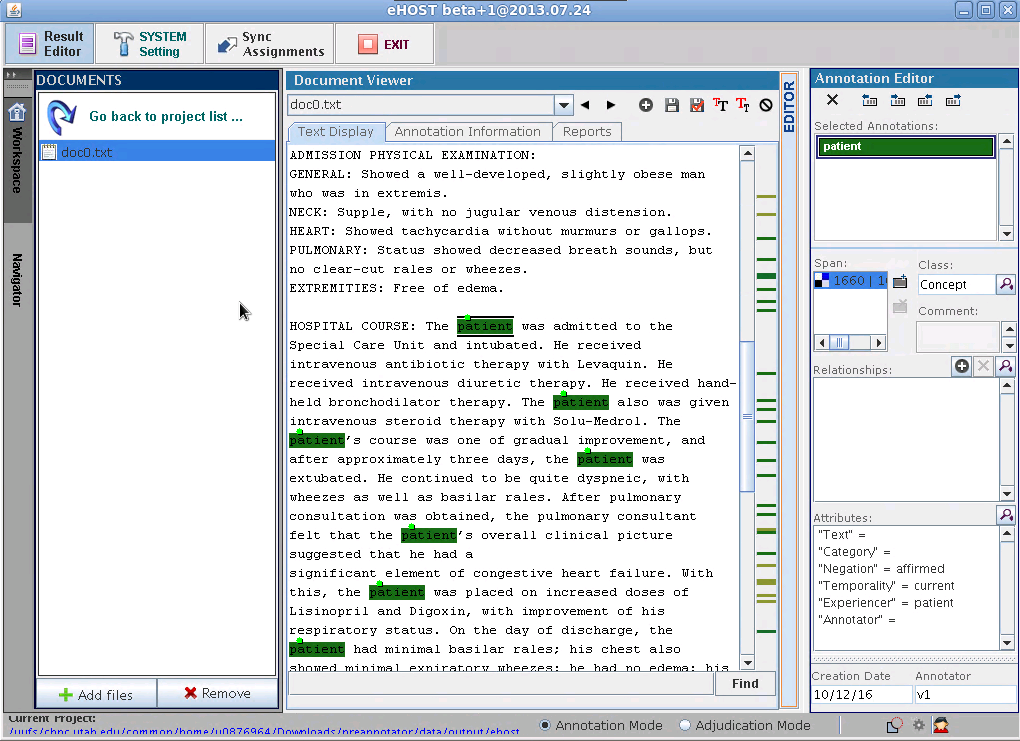


### View annotations in eHOST

Run the Preannotator using command similar to:

*./run Runpipe conf/my.mysql corpus data/output ehost v1*

It will output eHOST format to the specified directory. Open eHOST software, change the project directory to the output directory (“data/ouput” in the example above), you will be able to view the annotations in eHOST.

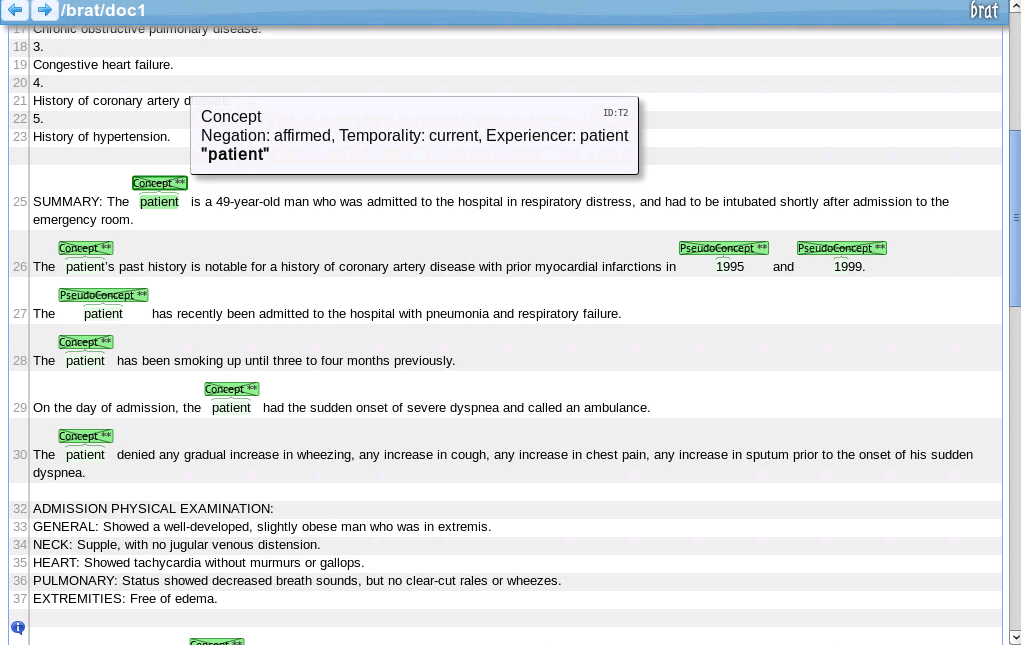


### View annotations in Brat

Run the Preannotator using command similar to:

*./run Runpipe conf/my.mysql corpus data/output brat*

It will output Brat annotation format to the specified directory (“data/output/brat” in the example above). Copy the directory to the “data” directory under the Brat installed directory. You will be able to view the anntoations in Brat.



### View annotations in UIMA Annotation Viewer

Please refers to [Viewer](#_Viewer).

## Build your rules

The Preannotator contains several NLP components: character-based rule processor (CP), token-based rule processor (TP), context processor (process context information) and concept consolidator (consolidate the concepts, pseudo-concepts, negated concepts, historical concepts and non-patient concepts).

The last two components don’t need modification. You may need to modify the rules for one or both of the first two components.

TP usually runs faster than CP. So always consider TP first.

### Build token-based rules:

Token-based rules are processed by TP. A TP consists of three components separated by a “tab” character:

1. Rule string
2. Mapped Concept Name
3. **ACTUAL** or **PSEUDO** (if the concept is a real concept or a pseudo concept)

The rule definition is simple, use the exact words (separated by a whitespace) to specify a rule. In addition, you can use two wildcards:

**“\w+”** represents any word.

**“\d+”** represent any digit.

For example:

“fever of \d+” represent a string starting with “fever of “and followed by any digit.

TP also support capturing one group with parentheses, with allow you to specify a subset of tokens in a rule to be annotated concept. For example:

“fever of (\d+)” or “fever of (\d+” will only annotate the digits when a match is found. Because the group is end the rule, the close parentheses can be omitted.

**Note**: current TP doesn’t support capturing multiple groups. If you do need to do so, use multiple rules instead.

For the mapped concept name, if you omit the namespace, the namespace: “edu.utah.bmi.type.system” will automatically added to your concept.

### Build character-based rules:

Character-based rules are processed by CP. Compared with TP rules, the CP rule is more close to regular expressions but with some difference and simplifications.

Wildcard definition:

“(“ Beginning of capturing a group

“)“ End of capturing a group

“\p“ A punctuation

“\“ plus following characters

“+“ An addition symbol (to distinguish the “+” after a wildcard)

“(“ A left parentheses symbol

“)“ A right parentheses symbol

“[“ A left square bracket symbol

“]“ A right square bracket symbol

“d“ A digit

“C“ A capital letter

“c“ A lowercase letter

“s“ A whitespace

“a“ A Non-whitespace character

“u“ A unusual character: not a letter, not a number, not a punctuation, not a whitespace

“n“ A return

“[“ and “]” with “|” can be used to define logic “or” rules:

For example: “[ab|\d]” can represent a two-character string “ab” or a digit.

**Note:** current CP doesn’t support nested square brackets. For instance, “[ab|bc[d|e]]” is not supported

The wildcard plus “+”: to match 1 or more characters that meet the corresponding wildcard definition

**Note**: Because CP use loops to support “+”, it doesn't check any other possible rules that might match the replicates at the same time.

The rule like “\c+t” won't work as expected: the rule will be matched all the way to “t” when checking “\c+”.

For example:

“Vital[|s][|\s+][|:][|\s+](9[3|4|5|6|7][|.\d][|F|\s+F|\s+degree|\s+degrees|\s+degree\s+F|\s+degrees\s+F])“ can match “Vital: 93 degree” or “Vitals : 97.9 F degrees”.