# Function of different classes in the project

# serviceStart/ServiceStarter.java

This class starts the process of converting Twitter data to Linked Data. It reads the config.properties file which has the various properties required for this conversion process. Following are the details of the properties in config.properties file:

**ConsumerKey=**fafa

**ConsumerSecret=**adafa

**access\_token=**asxad

**access\_token\_secret=**xyuz

These 4 properties are used when the tweets are collected from Twitter Streaming API. To get the values of these properties we should first create an app at <https://dev.twitter.com/> . On filling out the information for creating an app all these 4 values will be generated.

**className=**com.mysql.jdbc.Driver

**DB\_URL=**

**DB\_USER=**

**DB\_PASSWD=**

These 4 properties are used when we want to store the rdfised tweets in an SDB triple store.To create the necessary tables and indexes required for a full SDB store ***storing/TableCreator.java*** class should be run first.

**processType=**text

This property is used to specify from where the tweets are collected for processing. The 3 possible values for this property are: text, csv and api.

**readFile=**C:/Users/abc/tweetFile.txt

In case the processType value is text or csv, the value of this property will be the location of the text or csv file.

**fileType=**TextFile

The value of this property provides information whether to store the rdfised tweet in a SDB triple store or a RDF file. Possible values of this property are TextFile and SDB

**storeFile=**C:/Users/abc/rdfisedFile.rdf

In case the fileType value is TextFile, the value of this property will be the location where we want to store the rdfised file.

**nerClassifier=**C:/Users/abc/trainedNERClassifier-ner-model.ser.gz

The value of this property mentions the location of the trained NER Classifier.

**performInterlinking=**yes

To specify if we want to perform interlinking of the entities present in the local dataset to the similar real world entities present in DBPedia. Possible values are yes and no.

**silklslPerFile=**C:/Users/abc/silk-lslPer.txt

**silklslOrgFile=**C:/Users/abc/silk-lslOrg.txt

**silklslLocFile=**C:/Users/abc/silk-lslLoc.txt

The values of these 3 properties mention the location of SILK-LSL files.

**silkGeneratedPerFile=**C:/Users/abc/accepted\_linksPer.xml

**silkGeneratedOrgFile=**C:/Users/abc/accepted\_linksOrg.xml

**silkGeneratedLocFile=**C:/Users/abc/accepted\_linksLoc.xml

The value of these 3 properties mentions the location of files generated by SILK.

**storeFirstLinkFile=**C:/Users/abc/firstLink24.rdf

To specify the location of file generated after the Step 1 (Page 71, section 5.5.1 in thesis) of interlinking (i.e. links generated using SILK)

**performRedirect=**yes

The value of this property is used to specify whether to perform redirect interlinking (Page 72, Point 5 in thesis) using SILK or custom code written for this architecture. Using SILK, the redirect interlinking can give better results as there are various comparison metrics which we can specify in SILK-LSL file, however, using custom code for interlinking the process of interlinking becomes fast as compared to using SILK.

Possible values are yes and no; yes for using custom code and no for using SILK.

**storeRedirectFile=**C:/Users/abc/testingrdfisedRE.rdf

This property mentions the location to store the file generated after performing redirect interlinking. The value of this property has to be provided independent of whether the value of performRedirect is yes or no.

**performSecondInterlinking=**yes

The value of this property mentions whether to perform step 2 (Page 73, section 5.5.2 in thesis) of interlinking. Possible values are yes and no.

**storeSecondLinkFile=**C:/Users/abc/secondLink2.rdf

To specify the location of file generated after step 2 of interlinking.

**domainEntity=**http://dbpedia.org/resource/United\_Kingdom\_general\_election,\_2010

The domain entity, for example: in the thesis it was UK General Elections 2010.

# serviceStart/ServiceVariables.java

This class contains the getter and setter methods for using the properties mentioned in config.properties file.

# serviceStart/Authorization.java

This class is used to gain authorized access to Twitter Streaming API.

# collection/CSVFileReader.java , collection/StreamingAPIReader.java and collection/TextFileReader.java

These classes read the tweets from different input sources (csv file, text file or Twitter Streaming API).

# extraction/AnnotatedTweet.java

This class contains the getter and setter methods for using the different metadata like author name, creation time, hashatgs present in the tweet.

# extraction/HashtagExtractor.java

In case the tweets are read from the csv file or text file, this class is used to extract the hashtags from tweet text.

# extraction/NamedEntityExtractor.java

This class used Stanford NER Classifier to extract entities present in the text of tweets. It gives the output in XML format.

# extraction/XMLParser.java

This class parses the XML output produced using the previous class. It returns a hashmap containing person, location and organization as keys and an arraylist of the corresponding entities as the value.

# rdfising/Triple.java

This class contains the getter and setter methods for the triples extracted from the tweet.

# rdfising/TweetRDFModel.java

This class creates a RDF model using the triples generated from the previous classes.

# storing/FileStorage.java

This class stores the rdfised tweets into a RDF file.

# storing/SDBStorage.java

This class stores the rdfised stored in a SDB Triple store. The configuration parameters for this SDB store are described in sdb.ttl file.

# storing/TableCreator.java

To create the necessary tables and indexes required for a full SDB store, this class should be run before beginning the process of converting Twitter Data to Linked Data.

# firstInterlinking/SILKAPICaller.java

This class calls the SILK Single Machine which starts the process of interlinking the entities present in local dataset to the similar entities present in DBPedia.

# firstInterlinking/LinksExtractor.java

This class parses the file generated after using SILK. It extracts the similar entities in DBPedia which is found after using SILK.

# firstInterlinking/redirectEntityLinker.java

In case we want to perform redirect interlinking using the custom code, this class comes into use. This class performs the redirect interlinking.

# secondInterlinking/DomainEntityQuerier.java, secondInterlinking/ResourceQuerier.java

The first class begins the process of 2nd step of interlinking as it queries to extract those property values of domain entity which are literals (string). The second class queries those property values of domain entity which are resources. This class extracts the literal (string) property values of these resources.

# secondInterlinking/disambiguation.java

This class extracts the disambiguation values of those entities that are not previously linked using 1st step of interlinking (SILK). These entities are usually entities that are present with half name error like Clegg, Cameroon, etc.

# secondInterlinking/BagofWords.java

This class queries the different disambiguation entities and extracts their property values that are literals.

# secondInterlinking/CosineSimilarity.java

This class computes the cosine similarity between the local entity and the possible similar entity in DBPedia.

# insertingLinks/FileLinkInserter.java, insertingLinks/FileRedirectLinkInserter.java, insertingLinks/FileSecondLinkInserter.java and insertingLinks/SDBLinkUpdater.java

These classes updates the rdfised file and SDB triple store with the new links found using interlinking classes.