# WNLT

Welsh Natural Language Toolkit

| | |
|---|---|
| **Organisation** | Hypermedia Research Group, University of South Wales. |
| **Project Team** | Daniel Cunliffe, Douglas Tudhope, Andreas Vlachidis, Daniel Williams |
| **Funding Body** | Welsh Government, under the Welsh-language Technology and Digital Media Grant scheme. |

This user guide provides background material and documentation for the Welsh Natural Language Toolkit.

Version 2.3 of the toolkit provides functionality which allows the toolkit to be used outside the GATE environment via a graphical user interface, command line interface or API. Version 2.3 of the toolkit also added support for Twitter.

This user guide will first discuss the CymrIE information extraction system and all of its processing resources in the chapter 1. Chapter 2 outlines the steps on how to set up WNLT's CymrIE information extraction system in the GATE Developer (GUI) application. This user guide will then describe how to use the WNLT's graphical user interface (GUI), command line interface (CLI) and application programming interface (API) in chapters 3, 4 and 5 respectively. Lastly, chapter 6 provides information about the added support for the CymrIE information extraction system for Twitter.



| **Manual version** | **Description of changes** |
|---|---|
| **2.X** | Added documentation for WNLT's GUI, CLI and API (chapters 3, 4 and 5). Add documentation for TwitterCymrIE. Created Welsh version of this User guide |
| **1.X** | Created documentation for WNLT's CymrIE information extraction system and added documentation for using the CymrIE information extraction system in GATE Developer (chapters 1 and 2). |
| **Authors** | Daniel Williams and Andreas Vlachidis |

# Table of Contents

# 1. Background

The Welsh Natural Language Toolkit (WNLT) contains a set of four core Natural Language Processing (NLP) processing resources that enable the development of **generic** computational linguistic applications and contribute to the Welsh language technology infrastructure a much needed open source NLP toolkit. The project builds on the General Architecture for Text Engineering (GATE) by adapting and expanding existing processing resources (plugins) to Welsh.

The Toolkit contains the following **four processing resources**:

- Tokenizer
- Sentence Splitter
- Part of Speech Tagger
- Morphological Analyser

The processing resources benefit from a combination of glossaries with algorithmic arrangements that address specific linguistic behaviours of the Welsh language.

## 1.1. Tokenizer

The WNLT Tokenizer extends the default GATE Tokenizer and similarly splits the text into very simple tokens such as numbers, symbols and words of different types. The Tokenizer distinguishes words in uppercase, lowercase, and between types of symbols. The processing resource uses a slightly modified version of the original GATE Tokenizer rules file and an extended JAPE post-processing transducer adapting the generic output of the Tokenizer to the requirements of the Welsh part-of-speech tagger.

### 1.1.1. Token Types

The WNLT Tokenizer delivers the same types of Tokens and Space Tokens with default ANNIE Tokenizer as listed below:

- **[Word]** including the attribute 'orth' that takes the values; upperInitial, allCaps, lowerCase, mixedCaps
- **[Number]** any combination of consecutive digits.
- **[Symbol]** any special character is a symbol
- **[Space Token]** white spaces which are divided into two types of SpaceToken - space and control

### 1.1.2. Welsh Tokenizer Modifications

The Welsh Tokenizer uses a modified version of the GATE Tokenizer file 'AlternateTokeniser.rules' which originally splits hyphenated and apostrophised cases into separate tokens. This behaviour is desirable due to the extensive and elaborate use of hyphens and apostrophe in Welsh which differs significantly from English, for example use of hyphens in adjectival compounds. A succeeding post-processing transducer joins under a

single token several types of hyphenated and apostrophised constructs. The modified version also merges punctuation and symbol under a single Token type named 'symbol'.

The modified post-processing transducer joins together in a single token **the following constructs**:

- Hyphenated place names e.g. Llanarmon-yn-Ial
- Compounds of the common prefix e.g. ad-dala, cyd-ddefnyddir, rhag-glorineiddia
- Separate constituents hyphenation for the cases d+d, d+dd, dd+d, dd+dd, ff+f, ng+g, g+g, l+l, ll+l, t+h e.g. ladd-dy, cybydd-dod, cyd-dyfu, hwynt-hwy
- Apostrophe loss of vowel initialy e.g. 'Deryn
- Apostrophe loss of vowel medially eg. i'engoed
- Apostrophe loss of final consonant e.g. cry' for cryf hapusa' for hapusaf
- Apostrophe for common contractions, cases:i,m,n,r,w,ch,th
- Ordinals e.g. 1af, 2il, 3ydd, 4ydd
- Special cases of prepositions: Ar gyfer , Er mwyn , Yn erbyn, and Oddi followed by a preposition

### 1.1.3. Init-time parameters

***encoding*** **-**The character encoding to be used for reading the input

***tokenizerRulesURL*** **-** The path to the Tokenizer rules files, the default file is located at /resources/Tokeniser/WelshTokeniser.rule

***transducerGrammarURL*** **-** The path to the post-processing transducer grammar, the default JAPE file is located at /resources/Tokeniser/postprocess.jape

### 1.1.4. Run-time parameters

***annotationSetName*** **-** The name for annotation set where the resulting Token annotations will be created. It is optional, if left blank then the 'default' annotation set is assigned.

## 1.2. Sentence Splitter

The WNLT sentence splitter segments the text into sentences using the same set of JAPE grammars used in ANNIE. Hence, it delivers annotations of type 'Sentence' and 'Split'. It also makes available an alternative ruleset (main-single-nl.jape), which considers newlines and carriage returns differently. The alternative ruleset, similarly to ANNIE, should be used when a new line on the page indicates a new sentence.

### 1.2.1. Sentence Splitter Modifications

The WNLT sentence splitter uses a list of abbreviations adapted to Welsh that help distinguish sentence marking full stops from other kinds. The abbreviations list contains 330 entries of the following categories:

- Linguistic e.g. abs (absolute), cfst (synonym)

- Narrative eg Brth (British) , e.e (for example)
- Science e.g. Seic (Psychology), Tiwt (Teutonic)
- Spatial e.g. Morg (Glamorgan)
- Temporal e.g. C.C (B.C), Mer (Wednesday)

### 1.2.2. Init-time parameters

***encoding*** **-** The character encoding to be used for reading the input

***gazetteerListsURL*** **-** The path to the gazetteer list of abbreviations, the default list is located at /resources/sentenceSplitter/gazetteer/lists.def

***transducerURL*** **-** The path to transducer grammar, the default JAPE file is located at /resources/sentenceSplitter/grammar/mainsingle-nl.jape

### 1.2.3. Run-time parameters

***inputASName*** **-** The name of the annotation set used for input. It is optional, if left blank then the 'default' annotation set is assigned.

***outputASName*** **-** The name of the output annotation set where the resulting Split and Sentence annotations will be created. It is optional, if left blank then the 'default' annotation set is assigned.

## 1.3. Part of Speech Tagger

The WNLT POS tagger is a modified version of the ANNIE's Hepple tagger. The tagger produces a part-of-speech tag as an annotation on each word or symbol. The list of tags used by the tagger is found below. The tagger uses a default lexicon which is based on the Free (GPL) Dictionary Eurfa v3.0.

### 1.3.1. List of Tags

CC - coordinating conjunction: e.g. a, ac, fel, fod
CD - cardinal number
DT - determiner: e.g. y, yr, 'r
IN - preposition: e.g. am, ap, mewn
INT - interrogative: e.g. beth, ble, sut etc.
JJ - adjective
JJR - adjective comparative
JJS - adjective superlative
NN - noun singular or mass
NNS - noun plural
NNP - proper noun singular
NNPS - proper noun plural
NNM - noun masculine
NNF - noun feminine
PDT - pre-determiner: preceding an article or possessive pronoun; e.g. ambell, prif, rhai etc.
PP - pronoun
RP - particle, such as; gor, mi, na, nac, ni, ni's

RB - adverb
UH - interjection, such as; eh, huh, nefi, sori etc
VB - verb, base form
VBD - verb past tens
VBDP - verb pluperfect
VBDI - verb imperfect
VBI - verb infinitive
VBF - verb future
PN - punctuation, such as ''[](){} 、 –…-!.?’’“”'"";\\/⁄
SC - special characters, all other cases such as £$%* etc.

### 1.3.2. Part of Speech Tagger Modifications

The WNLT POS tagger uses a lexicon of 168669 pairs of terms and tags originating from the Eurfa dictionary. A mapping exercise has mapped the original Eurfa tags (http://www.eurfa.org.uk/abbrevs.php) to ANNIE Hepple tagger like tags. Major modifications applied on the original POSTagger and Lexicon classes for classifying Welsh input. The classes were extended to recognise linguistic evidence that support word classification of unknown words beyond the limits of the Eurfa dictionary.

### 1.3.3. Init-time parameters

***encoding*** **-** The character encoding to be used for reading lexicons and rules

***lexiconURL*** **-** The path to the lexicon of terms-tags pairs, the default lexicon is located at /resources/postag/lexicon

***rulesURL*** **-** The path to the ruleset file, the default ruleset file is located at /resources/postag/ruleset

### 1.3.4. Run-time parameters

***inputASName*** **-** The name of the annotation set used for input

***outputASName*** **-**The name of the annotation set used for output. This is an optional parameter. If user does not provide any value, new annotations are created under the default annotation set.

***baseTokenAnnotationType*** **-** The name of the annotation type that refers to Tokens in a document (run-time, default = Token)

***baseSentenceAnnotationType*** **-** The name of the annotation type that refers to Sentences in a document (run-time, default = Sentence).

***outputAnnotationType*** **-** POS tags are added as category features on the annotations of type *'outputAnnotationType'* (run-time, default = Token)

***posTagAllTokens*** **-** If set to false, only Tokens within each *'baseSentenceAnnotationType'* will be POS tagged (run-time, default = true).

***FailOnMissingInputAnnotations*** **-** if set to false, the PR will not fail with an ExecutionException if no input Annotations are found and instead only log a single warning message per session and a debug message per document that has no input annotations (run-time, default = true).

## 1.4. Morphological Analyser (Lemmatizer)

The Morphological Analyser takes as input a tokenized GATE document. Considering one token and its part of speech tag, one at a time, it identifies its lemma, mutation form and in some cases an affix. These values are then added as features on the Token annotation. The WNLT Morphological Analyser has significantly extended the original GATE Morphological Analyser to address the linguistic behaviour of Welsh with regards to inflection and mutation. The tool uses regular expression rules, a Lexicon of term-lemma pairs, a Gazetteer and a post-processing JAPE transducer for validating mutation propositions. The tool allows users to add new rules or modify the existing resources on their requirements.

### 1.4.1. Morphological Analyser Modifications

The rule file default.rul, which is available under the /resources/morph directory is modified for the Welsh alphabet. The file contains regular expressions that address regular and irregular forms of plural constructs. More information on how to write these rules can be found in GATE user guide at
https://gate.ac.uk/sale/tao/splitch23.html#sec:parsers:morpher:rules

The tool uses a Lexicon of 168794 term lemma pairs for providing known lemmas, a post-processing JAPE transducer for the identification of mutation forms focusing on contact mutations of Soft, Nasal and Aspirate type. The lemmatization process is as follows:

1. **Lexicon Lookup**, if is a known word provide lemma from lexicon and exit, else if unknown **proceed to 2**
2. **Regular Expressions rules**, resolve lemma using rules and in any case **proceed to 3**
3. **Post-processing Transducer**, identify cases of contact mutation based on contextual evidence. Propose new lemmas based on the contextual evidence and hard-coded Welsh language rules and **proceed to 4**
4. **Check the validity of the proposed lemmas** against a gazetteer of 168785 valid Welsh lemmas and 5885 Welsh place names. If lemmas validate set the new lemma and exit, else **proceed to 5**
5. **Revert invalid lemma** to original non-mutated lemma form.

### 1.4.2. Init-time parameters

***caseSensitive*** **-** By default, all tokens under consideration are converted into lowercase to identify their lemma and affix. If the user selects *'caseSensitive'* to be true, words are no longer converted into lowercase

***encoding*** **-** The character encoding to be used for reading lexicons and rules

***gazetteerListsURL*** **-** The path to the gazetteer list of valid lemmas, the default list is located at /resources/morph/gazetteer/lists.def

***lexiconURL*** **-** The path to the lexicon of terms-lemma pairs, the default lexicon is located at /resources/morph/lexicon

***rulesFile*** **-** The path to the file containing the regular expression patterns, the default file is located at /resources/morph/default.rul

***transducerURL*** **-** The path to post-processing transducer grammar responsible for identification and proposition of mutations, the default JAPE file is located at /resources/morph/grammar/postprocess.jape

***validationTransducerURL*** **-** The path to transducer grammar responsible for validating proposed mutations against the gazetteer of valid lemmas, the default JAPE file is located at /resources/morph/grammar/validation-main.jape

### 1.4.3. Run-time parameters

***affixFeatureName*** **-** Name of the feature that should hold the affix value.

***rootFeatureName*** **-** Name of the feature that should hold the root value.

***annotationSetName*** **-** Name of the annotation set that contains Tokens.

***considerPOSTag*** **-** Each rule in the rule file might have a separate tag, which specifies which rule to consider with what part-of-speech tag. If this option is set to false, all rules are considered and matched with all words.

***failOnMissingInputAnnotations*** **-** If set to true (the default) the PR will terminate with an Exception if none of the required input Annotations are found in a document. If set to false the PR will not terminate and instead log a single warning message per session and a debug message per document that has no input annotations.

## 1.5. CymrIE

CymrIE is an Information Extraction (Named Entity Recognition) system for Welsh. The name CymrIE is a paraphrasis of GATE's Information Extraction system ANNIE (A Nearly-New Information Extraction System). CymrIE adapts ANNIE to Welsh input using a modified version of the NE Transducer of ANNIE targeted at the requirements of the Welsh language, for example adjective – noun constructs. The system is using a wide range of Welsh

gazetteer lists to support the task of Named Entity Recognition while it maintains some of the original lists with a focus on person names and place names. CymrIE does not currently include a co-reference resolution processing resource.

The default annotation types, features and possible values produced by CymrIE the same used in ANNIE and are based on the original MUC entity types, and are as follows:

- Person
  - gender: male, female
- Location
  - locType: region, airport, city, country, county, province, other
- Organization
  - orgType: company, department, government, newspaper, team, other
- Money
- Percent
- Date
  - kind: date, time, dateTime
- Address
  - kind: email, url, phone, postcode, complete, ip, other
- Identifier
- Unknown

### 1.5.1. CymrIE Gazetteer lists

welsh_assembly_members, Major Type:person_full, Minor Type:government
welsh_charities, Major Type:organization, Minor Type:charity
welsh_coastal, Major Type:location, Minor Type:coastal
welsh_counties, Major Type:location, Minor Type:county
welsh_countries, Major Type:location, Minor Type:country
welsh_country_adj, Major Type:country_adj, Minor Type:COUNTRYADJ
welsh_country_denonyms, Major Type:country_adj, Minor Type:
welsh_currency_unit, Major Type:currency_unit, Minor Type:post_amount
welsh_date_key, Major Type:date_key, Minor Type:
welsh_date_unit, Major Type:date_unit, Minor Type:
welsh_days, Major Type:date, Minor Type:day
welsh_departments, Major Type:organization, Minor Type:government
welsh_facility, Major Type:facility, Minor Type:building
welsh_facility_key, Major Type:facility_key, Minor Type:
welsh_facility_key_ext, Major Type:facility_key_ext, Minor Type:
welsh_festival, Major Type:date, Minor Type:festival
welsh_goverment, Major Type:organization, Minor Type:government
welsh_govern_key, Major Type:govern_key, Minor Type:
welsh_greeting, Major Type:greeting, Minor Type:
welsh_hour, Major Type:time, Minor Type:hour

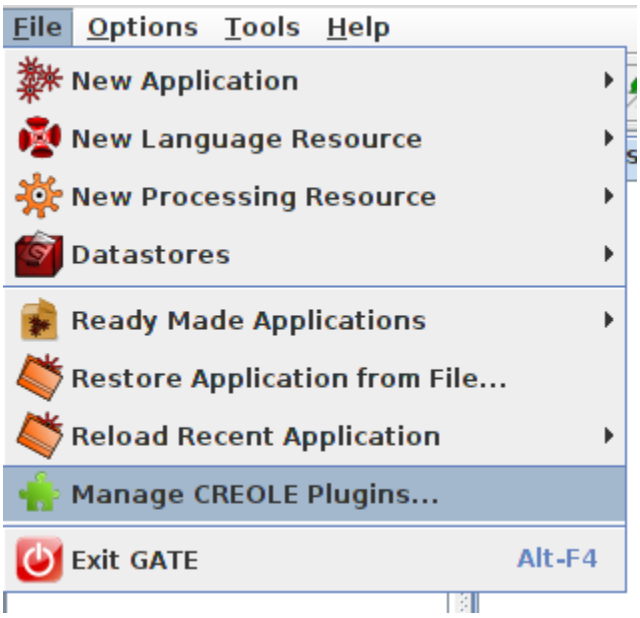
welsh_ident_prekey, Major Type:ident_key, Minor Type:pre
welsh_jobtitles_cap, Major Type:jobtitle, Minor Type:
welsh_jobtitles_lower, Major Type:jobtitle, Minor Type:
welsh_jobtitles_sen, Major Type:jobtitle, Minor Type:
welsh_lakes, Major Type:location, Minor Type:lake
welsh_loc_generalkey, Major Type:loc_general_key, Minor Type:
welsh_loc_key, Major Type:loc_key, Minor Type:post
welsh_loc_prekey, Major Type:loc_key, Minor Type:pre
welsh_ministry, Major Type:organization, Minor Type:government
welsh_months, Major Type:date, Minor Type:month
welsh_mountains, Major Type:location, Minor Type:mountain
welsh_number_fold, Major Type:number_fold, Minor Type:
welsh_numbers, Major Type:number, Minor Type:
welsh_ordinals, Major Type:date, Minor Type:ordinal
welsh_org_base, Major Type:org_base, Minor Type:
welsh_org_key, Major Type:org_key, Minor Type:
welsh_org_pre, Major Type:org_pre, Minor Type:
welsh_parishes, Major Type:location, Minor Type:parish

# 2. Using the WNLT in GATE Developer (GUI)

This chapter will take you through the basic steps of loading the CymrIE system and processing a small corpus of BBC Cymru Fyw news stories. The guide is split into three sections. The first section is about loading WNLT in GATE using the CREOLE plugin manager, the second section is about loading CymrIE and processing a corpus, and the third section is about creating, populating and processing a new corpus in CymrIE pipeline.

## 2.1. Loading WNLT in GATE

**Step 1:** From the File menu in GATE open the CREOLE Plug-in Manager by choosing the Manage CREOLE Plugins option.

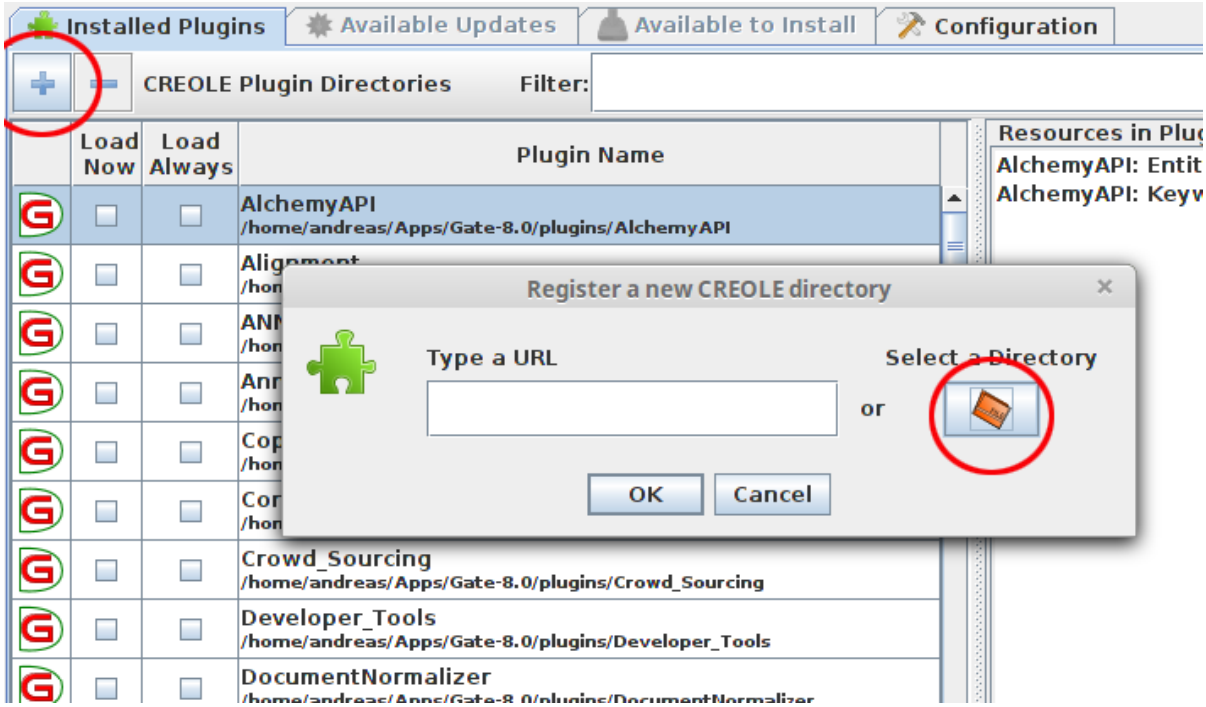



**Step 2:** From the new window (CREOLE Plug-in Manager) click on the plus sign located at the top-left corner of the window.

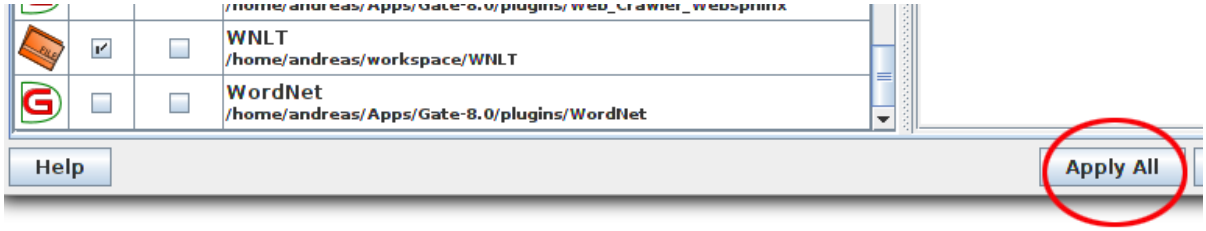
**Step 3**: A new dialog box appears, click on the Select a Directory button and select the directory WNLT which is located in your local drive to the place where you have downloaded and extracted the WNLT-CymrIE.zip.

**Step 4:** Once you have selected WNLT from your drive click Open in the dialog box

**Step 5**: The path to WNLT should now appear in the white box of the dialog window, click OK to close the window

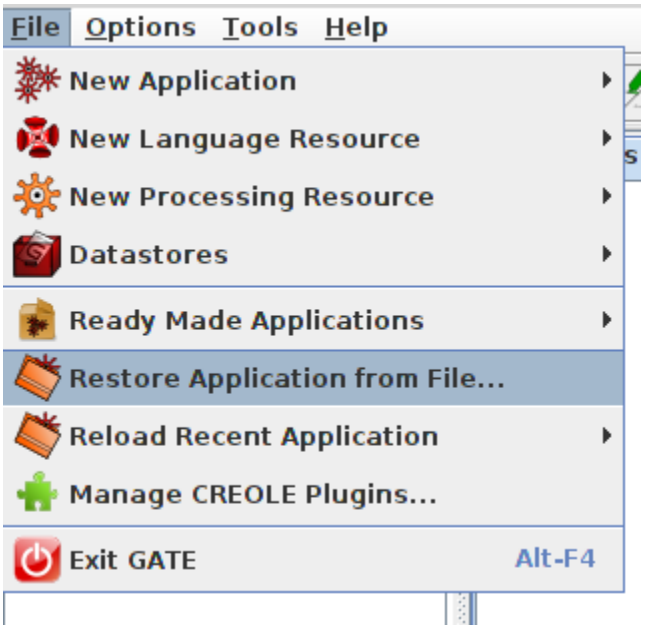



**Step 6:**The WNLT plugin should now appear in the list of plugins as seen below. Check the Load Now checkbox, click on Apply All button and Close the CREOLE Plug-in Manager window

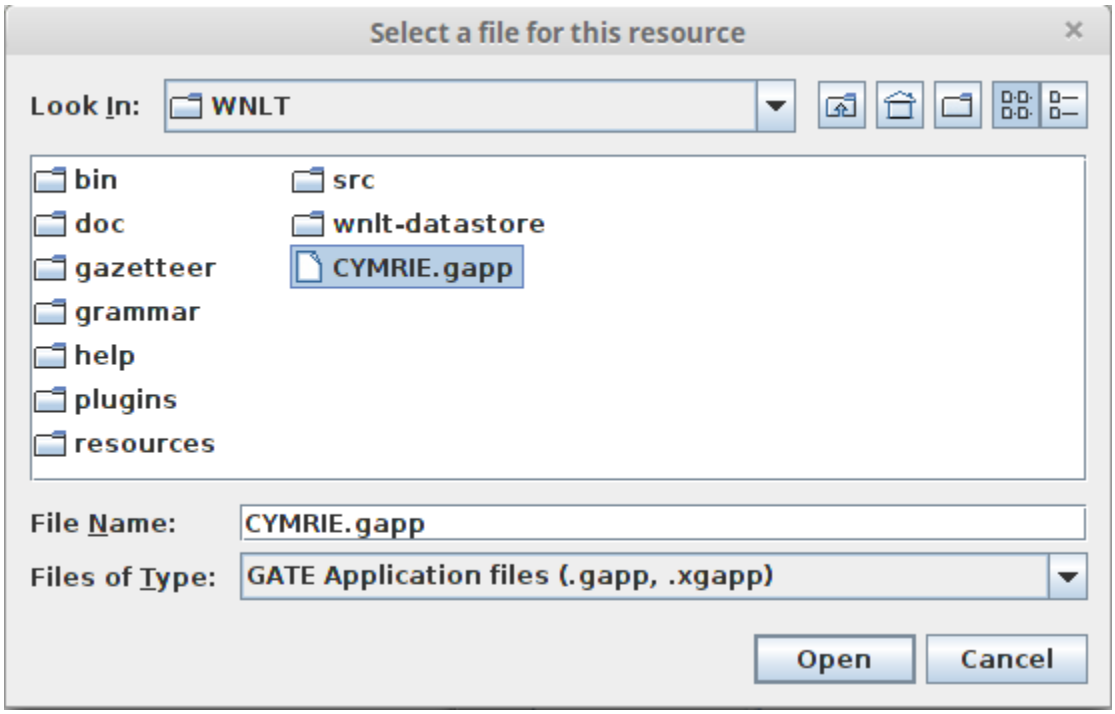



## 2.2. Loading the CymrIE system in GATE

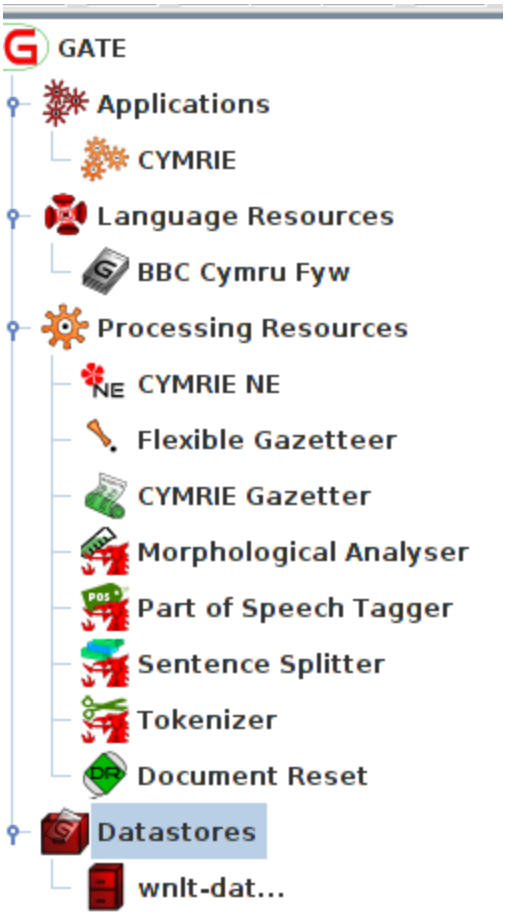
**Step 1:** From the File menu in GATE load CymrIE by choosing the Restore Application from File option.




**Step 2:** Select the file CYMRIE.gapp located in the WNLT folder and click Open



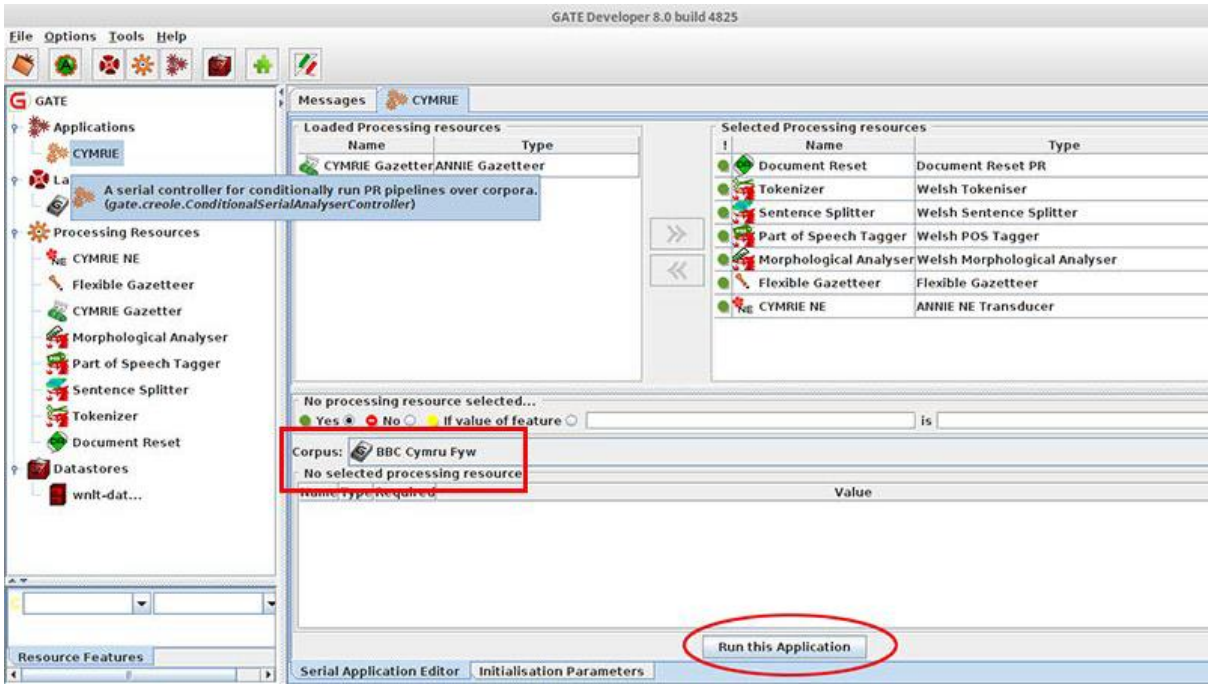

**Step3 :** Inspect the loaded Processing Resources at the right hand side of GATE . You should see list of processing resources like the one below. Also the corpus BBC Cymru Fyw is loaded under the Language Resources and the wnlt-datastore is loaded under Datastores.




**Step 4:** Double click on CymrIE application and the pipeline will appearing on screen as seen below

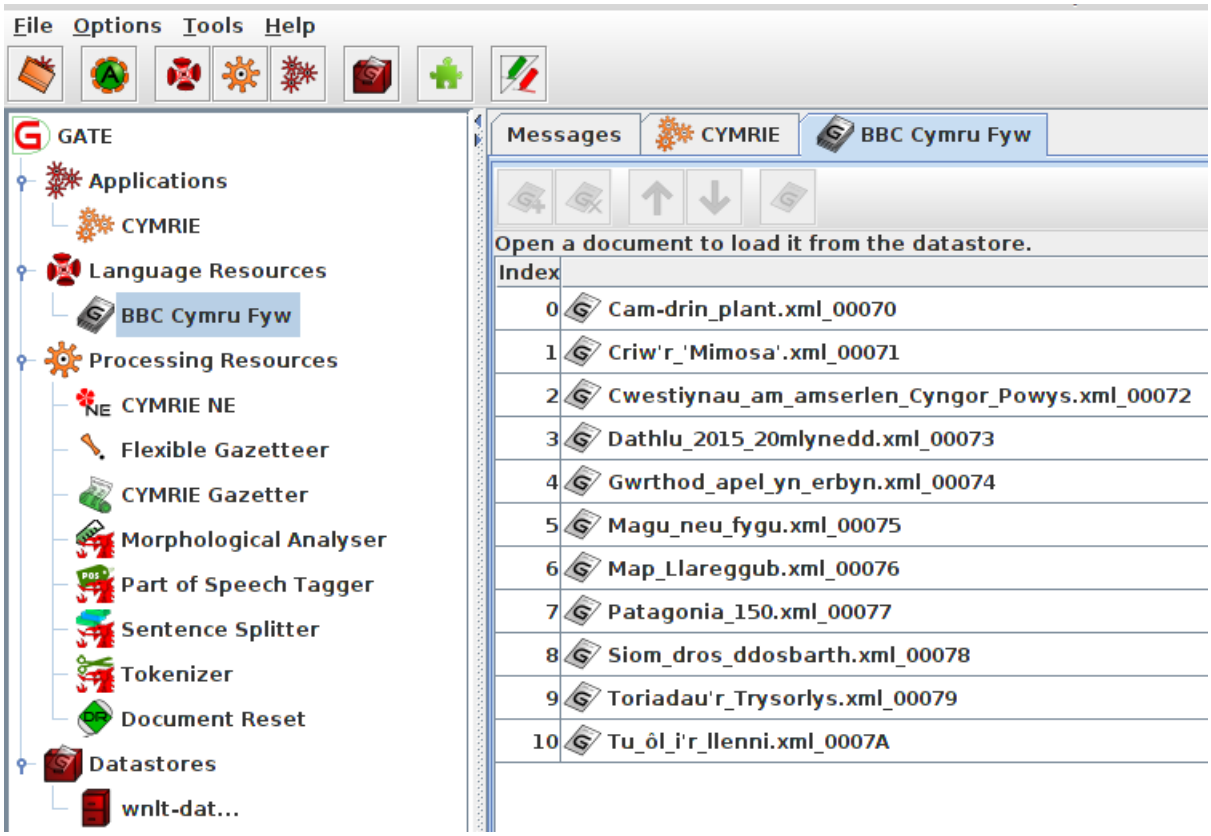
**Step 5:** The BBC Cymru Fyw corpus should be already selected as the corpus for processing as see below if not selected it from the dropdown box

**Step 6**: Click on Run this Application button



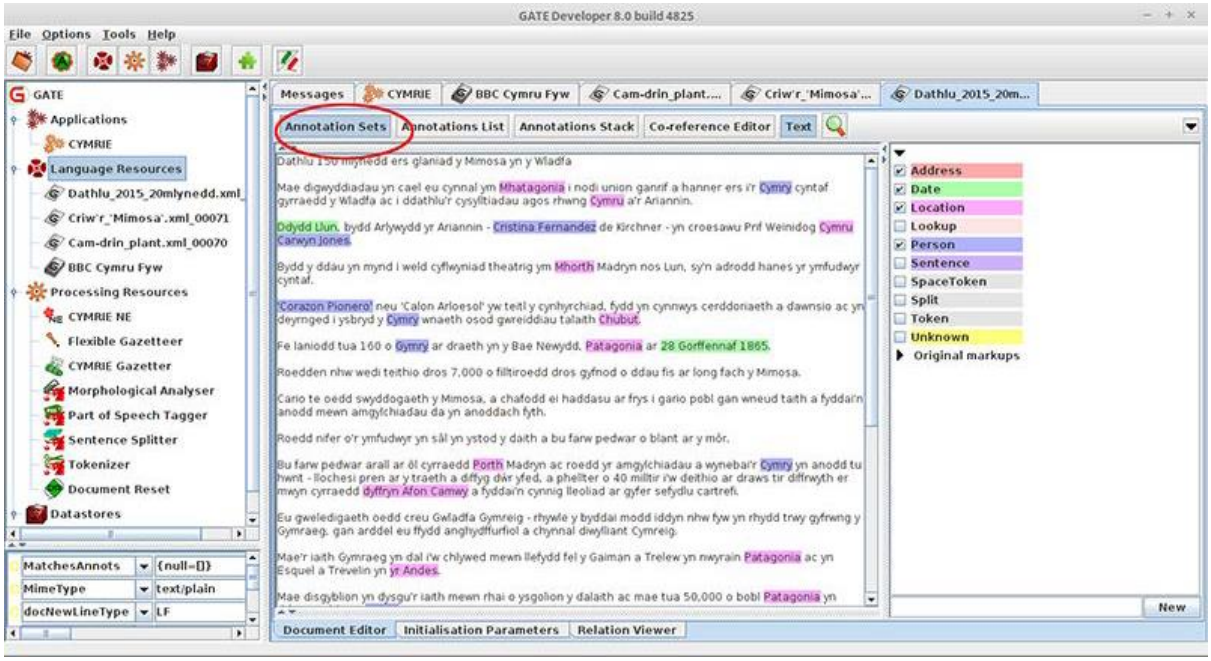

**Step 7**: Double click (or right click and open) on BBC Cymru Fyw corpus located in Language Resources to view the list of documents contained in the corpus.

**Step 8:** Double click (or right click and open) on any of the documents in the corpus to open it



**Step 9:** Under the Language Resources double click (or right click and show) a document from the list to view its contents

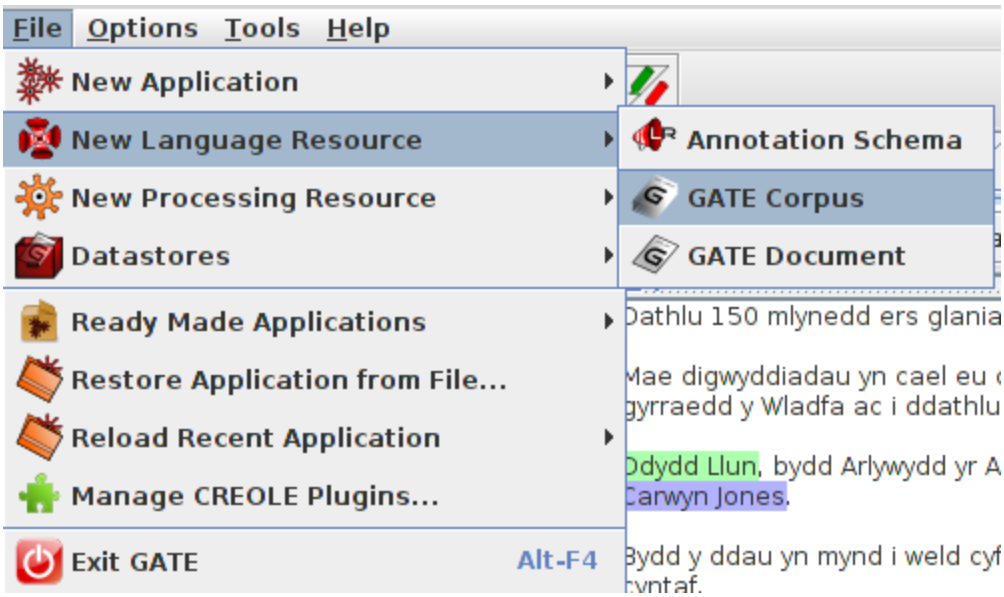
**Step 10:** Click on Annotation Sets to view the annotations produced by the pipeline.




## 2.3. Adding a New Corpus in GATE

**Step 1:** From the File menu create a GATE Corpus by choosing New Language Resource > GATE Corpus

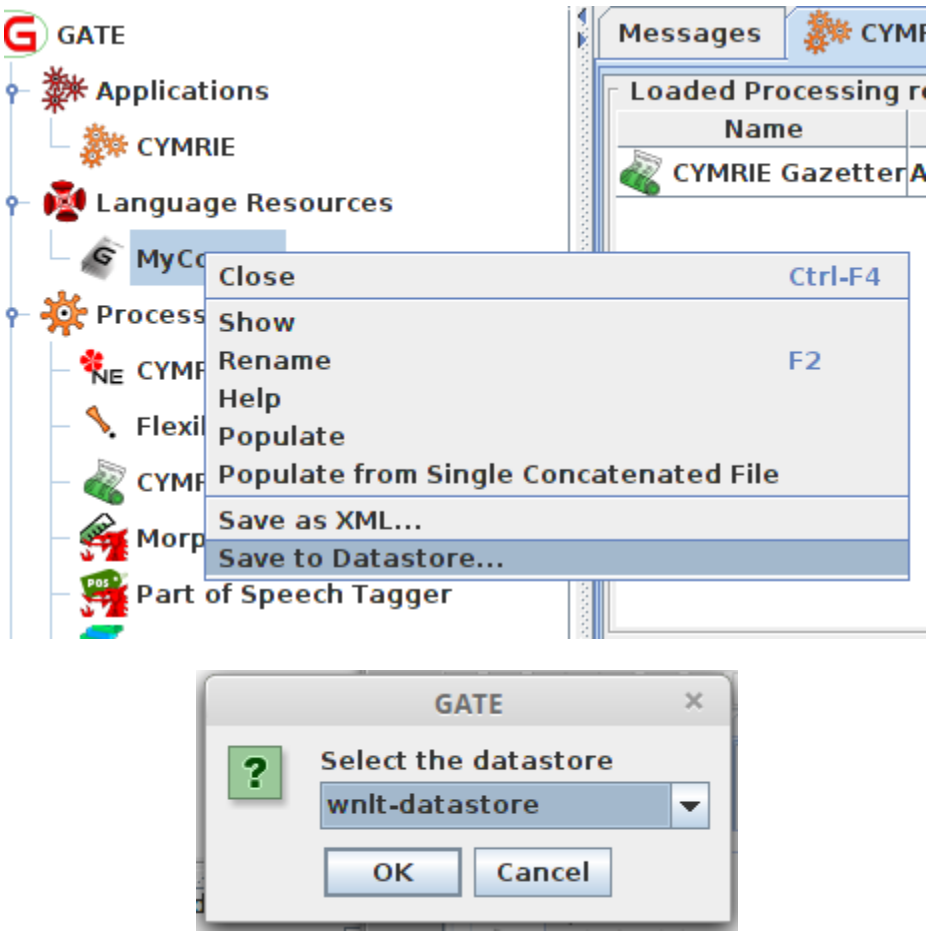
**Step 2:** In the pop-up dialog box name the Corpus e.g. MyCorpus and click the OK button




**Step 3:** The new corpus should now appear on the left side panel under Language Resources

**Step 4**: Save the corpus to the wnlt-datastore by right-clicking on the corpus icon and selecting Save to Datastore

**Step 5**: From the pop-up dialog box select wnlt-datastore and click the OK button




**Step 6:** From the File menu create a GATE Document by choosing New Language Resource > GATE Document

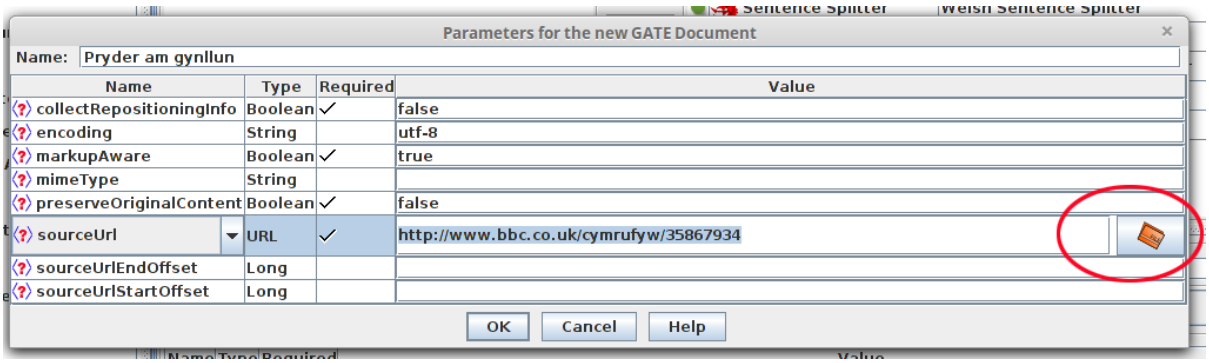
**Step 7:** In the new dialog box

**a)** Give a Name to the document, in this example 'Pryder am gynllun'

**b)** In the encoding field type utf-8

**c)** Select the document by typing a web address (URL) in this example 'http://www.bbc.co.uk/cymrufyw/35867934'

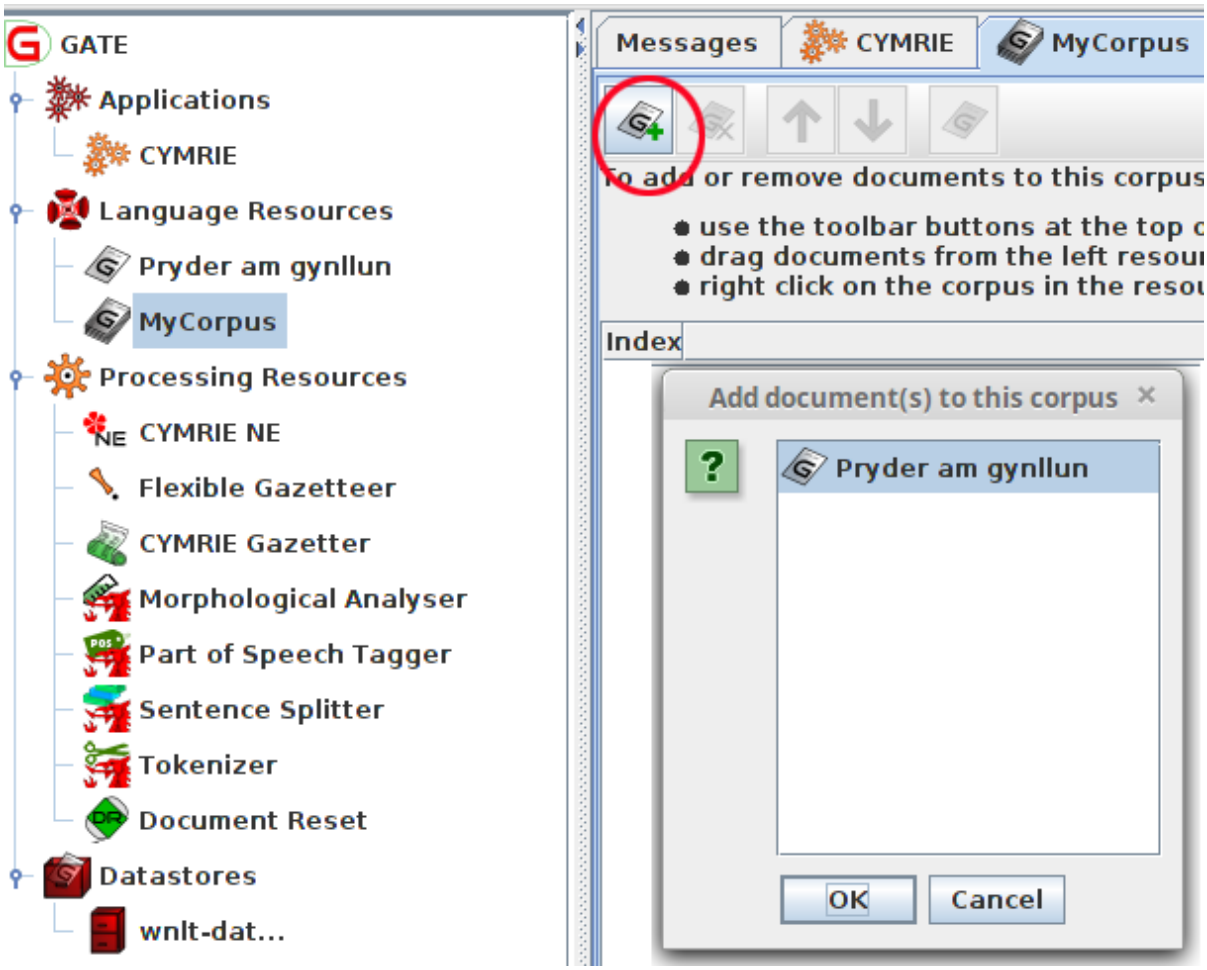
**Alternative c)** Instead of typing a web address you can Open a document from your local drive by clicking the Folder button on the right hand side and selecting a local document of your preference (list of supported document formats at https://gate.ac.uk/sale/tao/splitch5.html#x8-940005.5)




**Step 8:** Double click on the Corpus (My Corpus) icon located under the Language Resources to open the Corpus Editor

**Step 9:** Click on the Green Cross button (circled in red below) to open the Add document(s) to this corpus dialog box

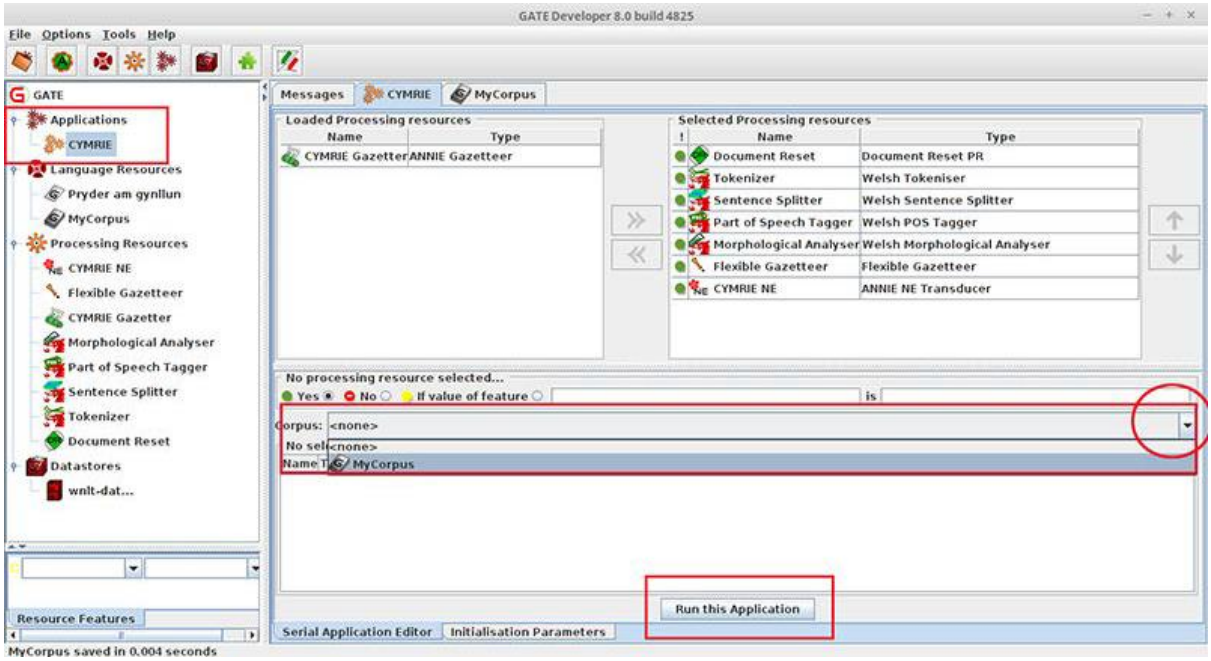
**Step 10:** Select the document (in this case Pryder am gynllun) and click the OK button




**Step 11:** Double click on CYMRIE application (located in Applications) to view the pipeline as seen below.

**Step 12:** Select MyCorpus from the Corpus drop-down box

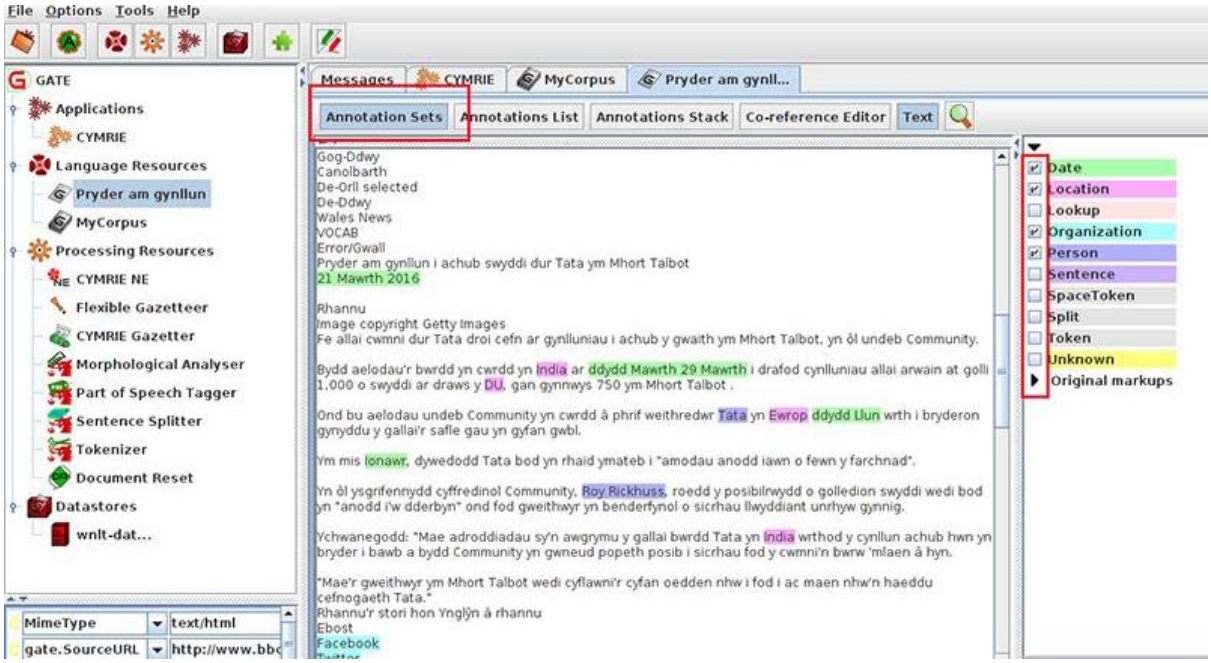
**Step 13:** Click the Run this Application button to execute the pipeline.




**Step 14:** Double click the 'Pryder am gynllun' document icon from the Language Resources to view the document

**Step 15:** Click on Annotation Sets button to reveal the produced annotations

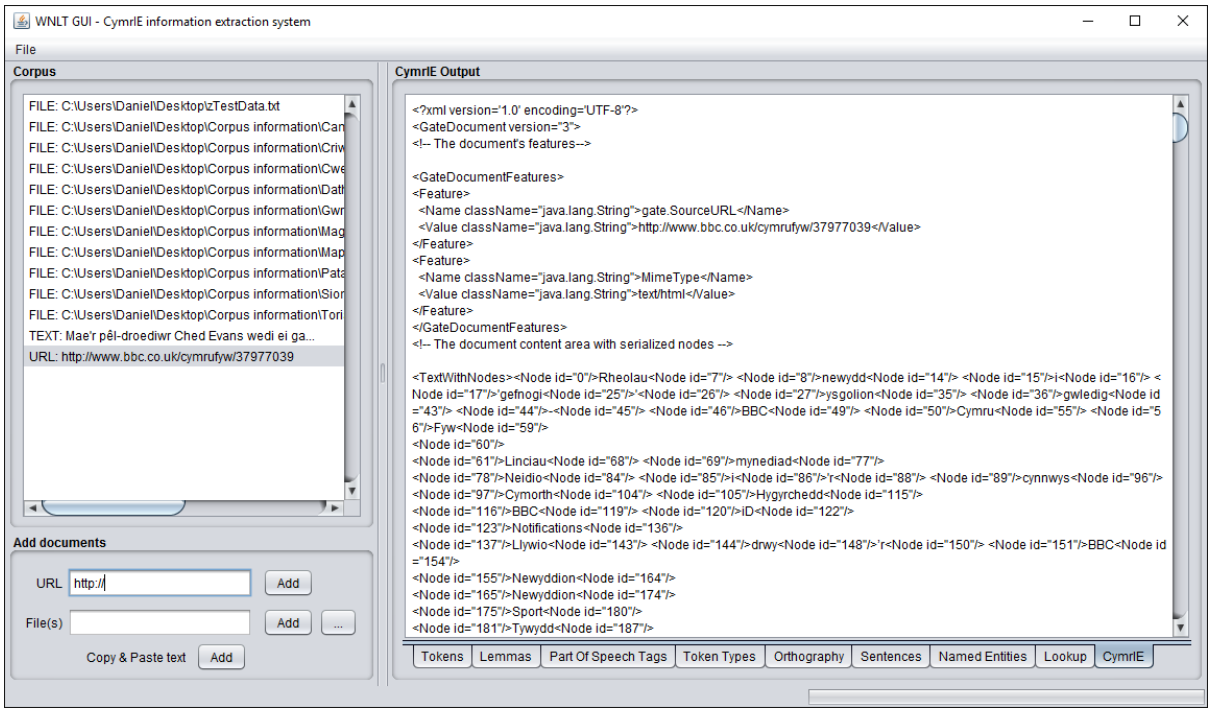
**Step 16**: Toggle the Annotation Types ON and OFF from the left hand side panel by checking - unchecking the relevant boxes.



# 3. WNLT's graphical user interface (GUI)

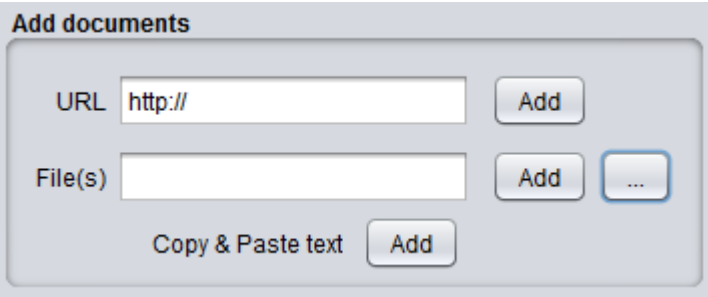
The WNLT's graphical user interface (GUI) can be launched by double clicking the ***wnlt.jar*** file in the root directory of WNLT's folder. After the application is launched, the GATE API and the CymrIE information extraction system will be initialised. The initialisation process uses multiple threads and unavoidably utilises a lot of the CPU.

This application allows the user to use the CYRMIE information extraction system to *tokenize*, *lemmatize*, *find part of speech tags*, *get token types*, *get orthographies*, *get named entities* and *get lookups performed by the CymrIE system*.

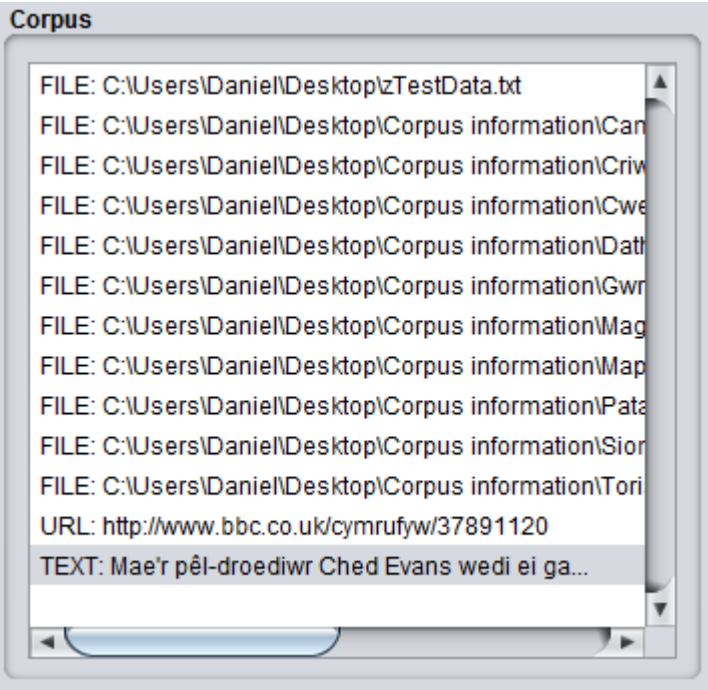



## 3.1. Functionality

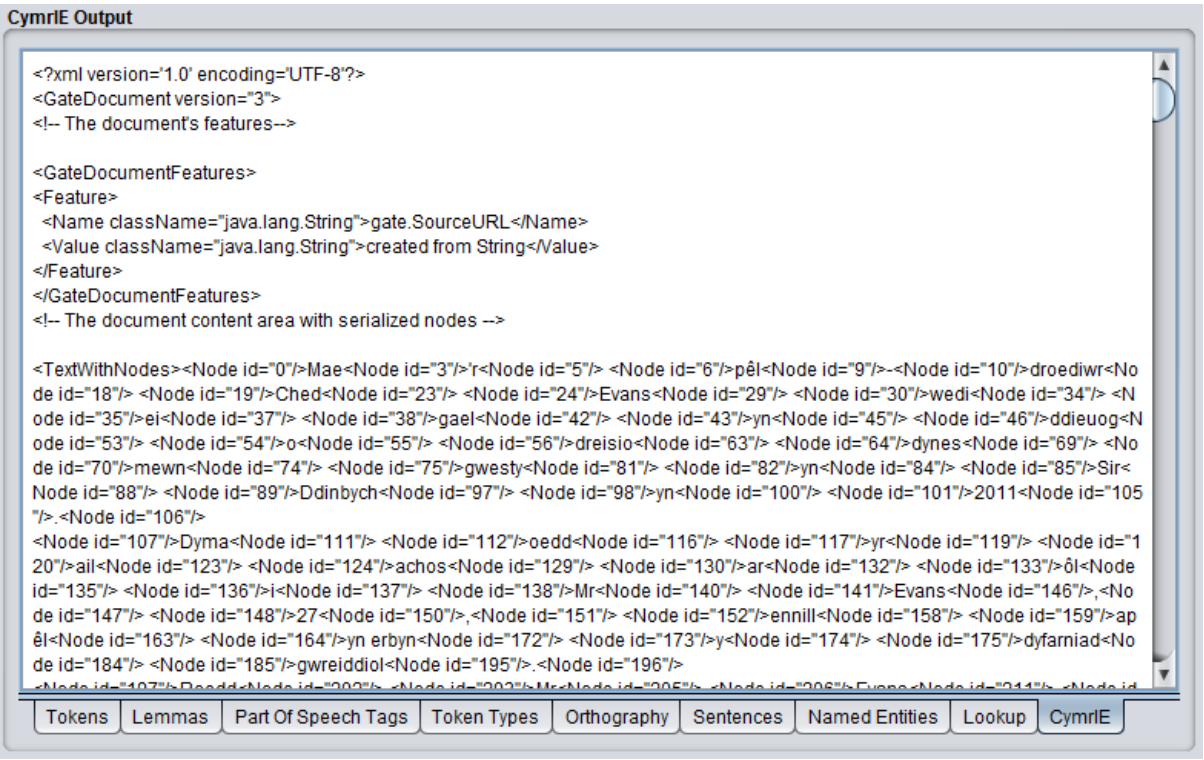
The user can add URLs, Files and Welsh text to the application by using the 'Add documents' part of the application.




After documents (Files, URLs and/or Welsh text) are added, the user can select particular documents to display more information, documents are selected from the 'Corpus' part of the application.




The CymrIE information extraction system runs on the user selected document and the system's output is displayed in the 'CymrIE Output' part of the application.




The 'CymrIE Output' part of the application shows the output of the CymrIE information extraction system which is organised into different tabs. The 'Tokens' tab outputs each token on a separate line.

The ‘Lemmas’, ‘Part Of Speech’, ‘Token Types’ and ‘Orthography’ tabs output their results corresponding to each token in the ‘Tokens’ tab.

The ‘Sentences’ tab outputs all of the sentences found by the CymrIE information extraction system on each line. As each sentence can span multiple tokens and words the following information is given in order:

- The start character offset in the URL, File or String for the sentence (inclusive)
- The end character offset in the URL, File or String for the sentence (exclusive)
- The corresponding start token (see Tokens tab) of the sentence (inclusive)
- The corresponding end token (see Tokens tab) of the sentence (inclusive)
- The sentence as a String

An example:

*4166 4325 780 825 Pan o'n nhw'n danglo carot o flaen ein trwyn ni a cynnig ysgol £5m i ni, a oedd yn deg i ni i droi'r ysgol yna nawr, a'r plant yn diodde, falle, o achos hynny?*

The ‘Named Entities’ tab outputs all of the named entities found by the CymrIE information extraction system on each line. As each named entity can span multiple tokens and words the following information is given in order:

- The start character offset in the URL, File or String for the named entity (inclusive)
- The end character offset in the URL, File or String for the named entity (exclusive)
- The corresponding start token (see Tokens tab) of the named entity (inclusive)
- The corresponding end token (see Tokens tab) of the named entity (inclusive)
- The type of named entity
- An array of features about the named entity
- The named entity as a String

An example:

*662 678 100 102 Date {rule=DateName, ruleFinal=DateOnlyFinal, kind=date} 15 Tachwedd 2016*

The ‘Lookup’ tab outputs the gazetteer lookups performed by the CymrIE information extraction system. As each lookup can span multiple tokens and words the following information is given in order:

- The start character offset in the URL, File or String for the lookup (inclusive)
- The end character offset in the URL, File or String for the lookup (exclusive)
- The corresponding start token (see Tokens tab) of the lookup (inclusive)
- The corresponding end token (see Tokens tab) of the lookup (inclusive)
- The lookup’s ‘*majorType*’ value
- The lookup’s ‘*minorType*’ value
- The lookup as a String

An example:

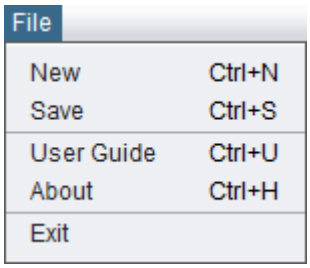
*407 410 66 66 organization company BBC*

For more information on the '*majorType*' and '*minorType*' values see the section 1.5.1.

The 'CymrIE' tab shows the CymrIE information extraction system's output in GATE's xml format. The output of this tab can be seen in the image above.

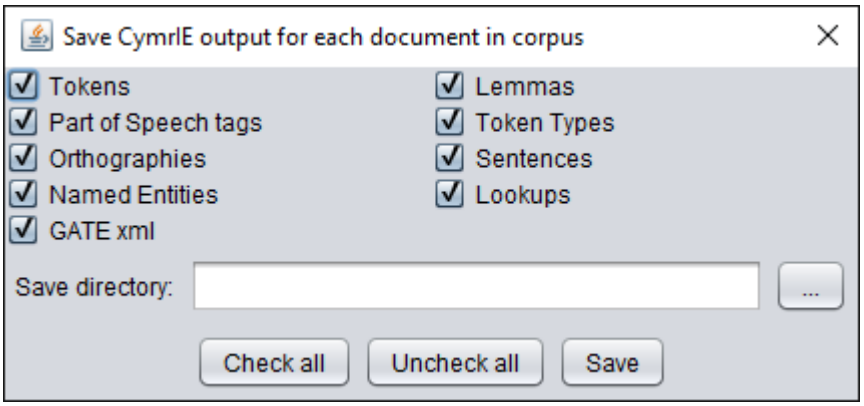
## 3.2. Menu

The File menu provides the user with additional functionality, as can be seen by the following image.




The 'New' menu item removes all of the documents in the application. The 'User Guide' menu item shows this user guide. The 'About' menu item displays some information about the WNLT project.

The 'Save' menu item opens a dialog which allows the user to choose what CymrIE output to save and the directory to the save this information. This dialog saves the selected CymrIE outputs of each URL, File and String of Welsh text in the corpus.

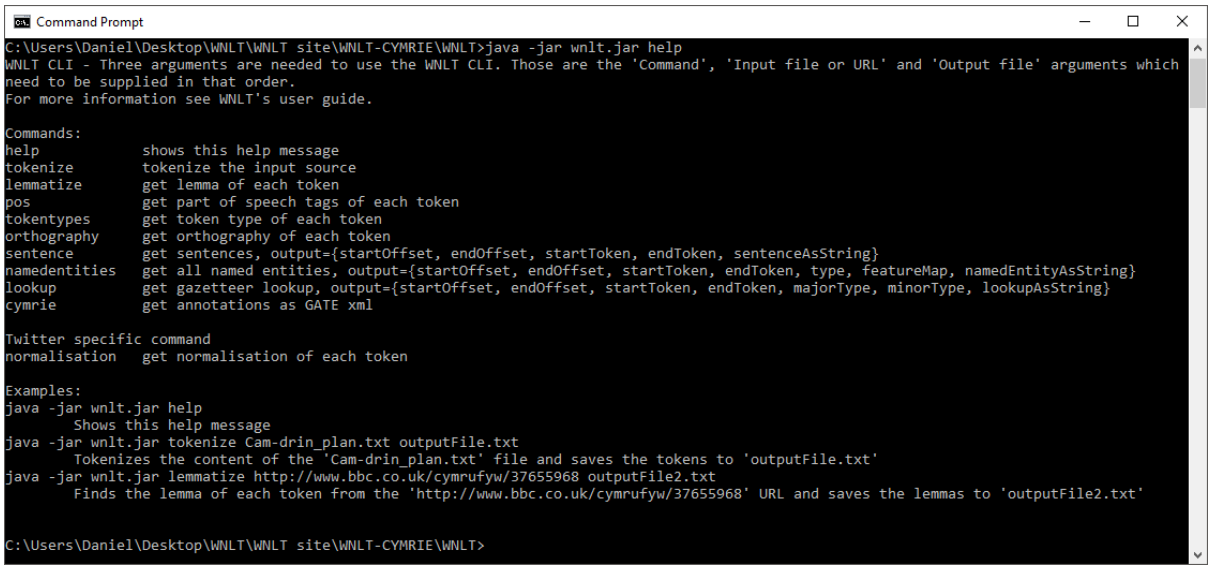


# 4. WNLT's command line interface (CLI)

Users can use WNLT via the command line interface (CLI) by running the *wnlt.jar* file from a command line interface. To display the help message of the WNLT CLI, use the following command in the WNLT project's root directory:

***java -jar wnlt.jar help***

This will output the following message:




The command line interface of the WNLT needs three arguments in order from the user:

1. Command – The data that the user wishes to save after the executing the CymrIE information extraction system. The commands are listed in the image.
2. Input file – The input file or URL which contains Welsh text
3. Output file – The file to output the results of the CymrIE information extraction system.

The user can use the CYRMIE information extraction system to *tokenize*, *lemmatize*, *find part of speech tags*, *get token types*, *get orthographies*, *get named entities* and *get lookups performed by the CymrIE system*.

As mentioned in the help message, to following arguments will tokenize the content of the *Cam-drin_plan.txt* file and save the tokens to *outputFile.txt*

***java -jar wnlt.jar lemmatize "Cam-drin_plan.txt" "outputFile.txt"***

The results in the output files are the same as described in chapter 2.

# 5. WNLT API

The CymrIE information extraction system and all of its processing resources can be utilised in Java code by using the WNLT API. The WNLT API also provides higher level functionality such as tokenization and lemmatisation without using the GATE API directly, see 5.3. The user can access the WNLT API using the classes located in the Java package ***wnlt.api***

The following sections outline the setup process for including the WNLT API in an existing Java project, how to use the WNLT API to create and reuse components of the CymrIE information extraction system and how to use the *CymrIE* class which hides the GATE API from the user.

Users who do not wish to use the GATE API or the data structures provided by the GATE API should first follow the instructions outlined in section 5.1 then skip to section 5.3.

## 5.1. Setting up the WNLT API

WNLT has been developed using the framework provided by the GATE API (GATE Embedded). In order to use the WNLT API, the GATE API also needs to be included in the build path of the user's Java project.

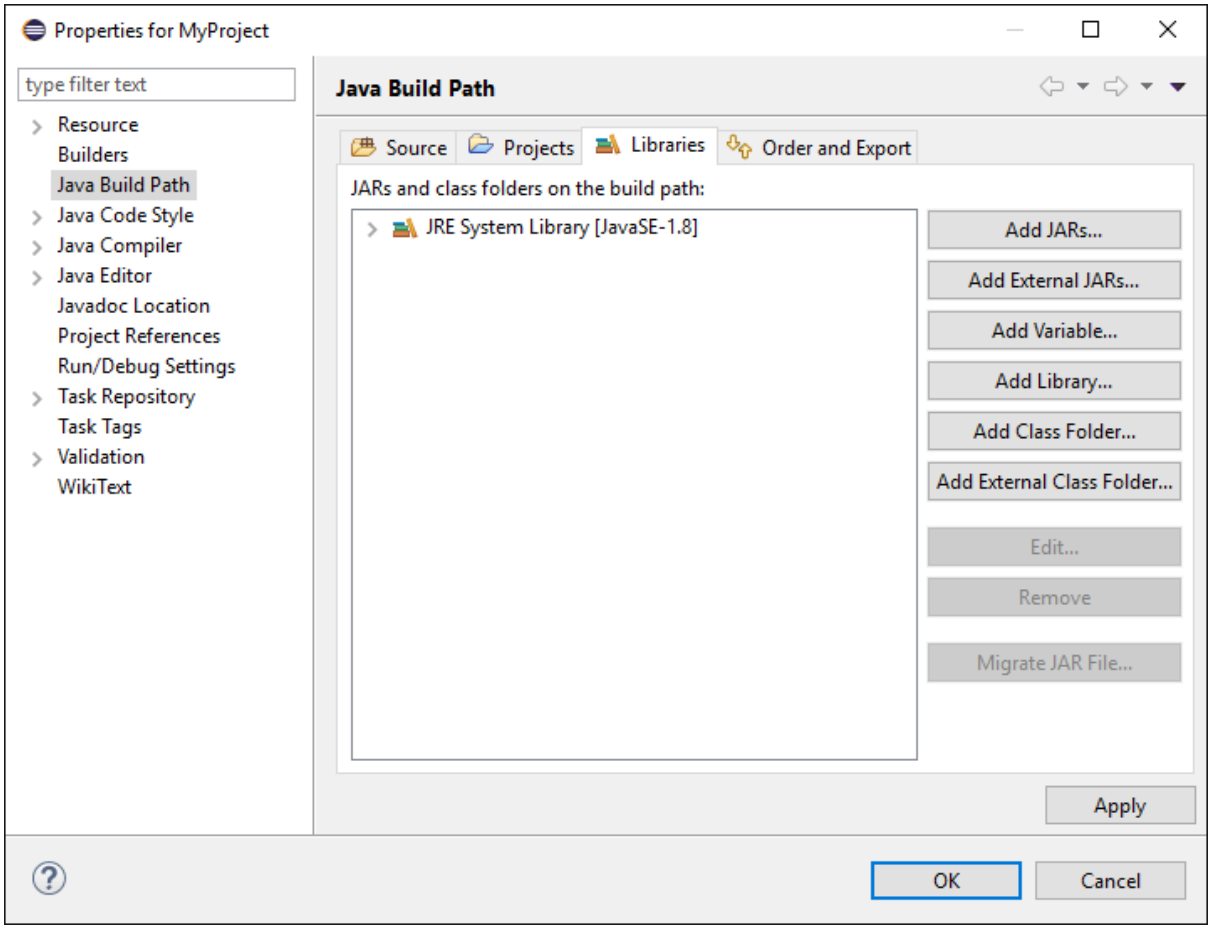
The following sections will outline the setup process using an Eclipse IDE. The quickest way to setup a Java project for the WNLT API is outlined in section 5.1.1. To setup your Java project to use the latest or an existing version of GATE, refer to section 5.1.2.

### 5.1.1. Quick setup

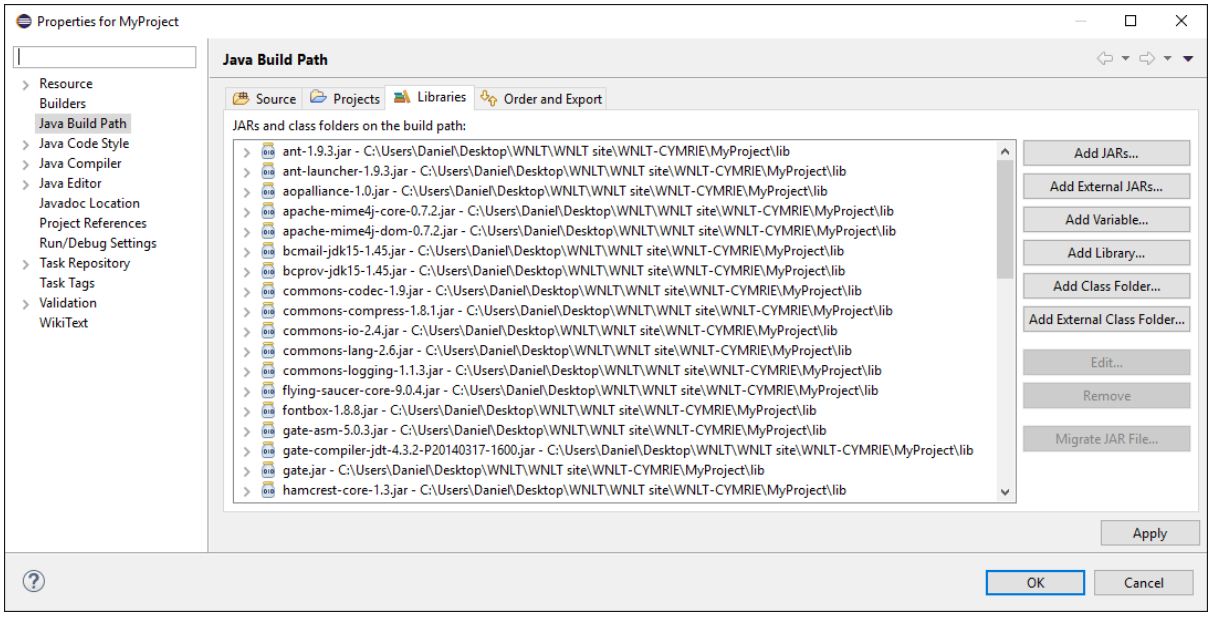
**Step 1:** Include all of the necessary ***.jar*** files in your project's build path. These are all the ***.jar*** files located in the WNLT's ***lib*** folder and the ***wnlt.jar*** file located in the root directory of the WNLT folder.

It is recommended that the user creates a folder called ***lib*** in the root directory of the user's Java project then copy all of the ***.jar*** files into that folder.

**Step 2:** In Eclipse, click "*Project*" from the menu bar -> "*Properties*" -> "*Java Build Path*" -> "*Libraries*"




Click "*Add External JARs*" then add all the **.jar** files in step 1.



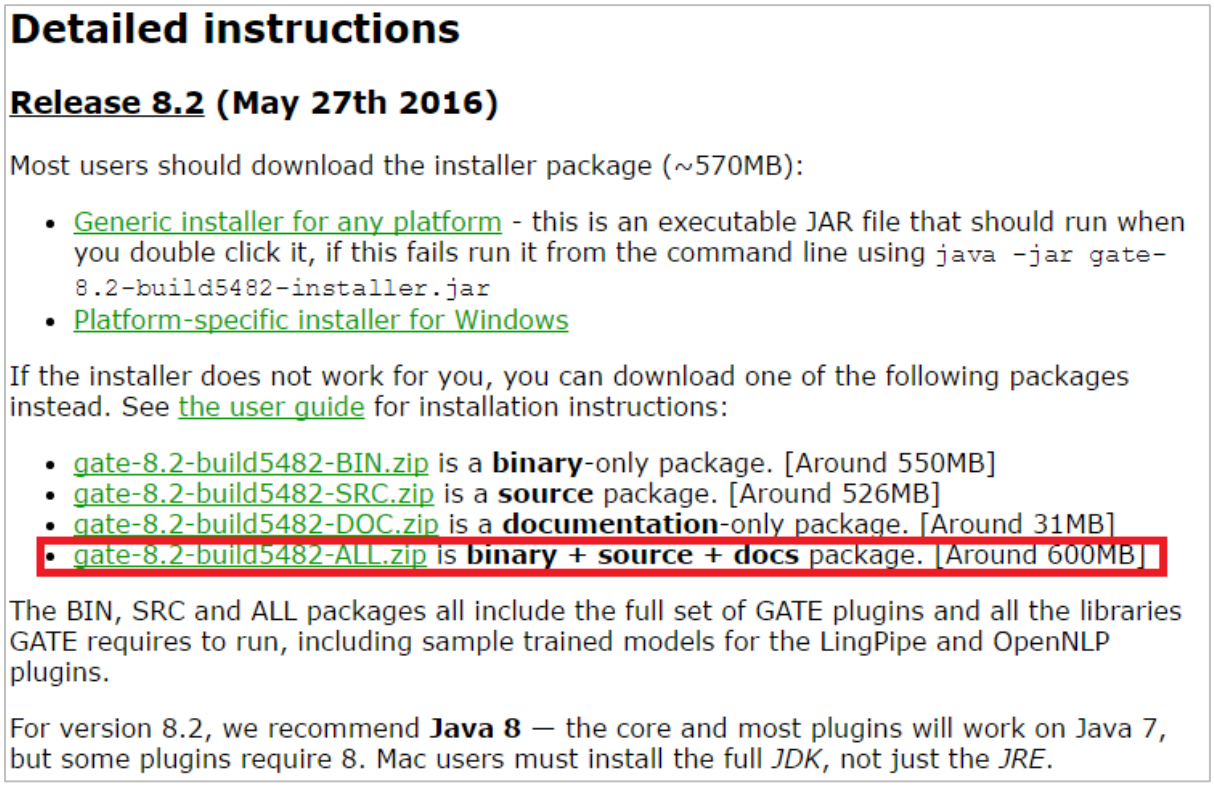

Click "*Apply*" then "*OK*"

**Step 3:** Copy the 'plugins' and 'resources' folder into your Java project's root directory.

### 5.1.2. Updating to the latest version of GATE

If the steps in section 5.1.1 have been followed then ignore the following steps as your project is already setup.

**Step 1:** Download the latest version of GATE's binary, source and documentation via https://gate.ac.uk/download/



## Detailed instructions

### Release 8.2 (May 27th 2016)

Most users should download the installer package (~570MB):

- Generic installer for any platform - this is an executable JAR file that should run when you double click it, if this fails run it from the command line using `java -jar gate-8.2-build5482-installer.jar`
- Platform-specific installer for Windows

If the installer does not work for you, you can download one of the following packages instead. See the user guide for installation instructions:

- gate-8.2-build5482-BIN.zip is a **binary**-only package. [Around 550MB]
- gate-8.2-build5482-SRC.zip is a **source** package. [Around 526MB]
- gate-8.2-build5482-DOC.zip is a **documentation**-only package. [Around 31MB]
- gate-8.2-build5482-ALL.zip is **binary + source + docs** package. [Around 600MB]

The BIN, SRC and ALL packages all include the full set of GATE plugins and all the libraries GATE requires to run, including sample trained models for the LingPipe and OpenNLP plugins.

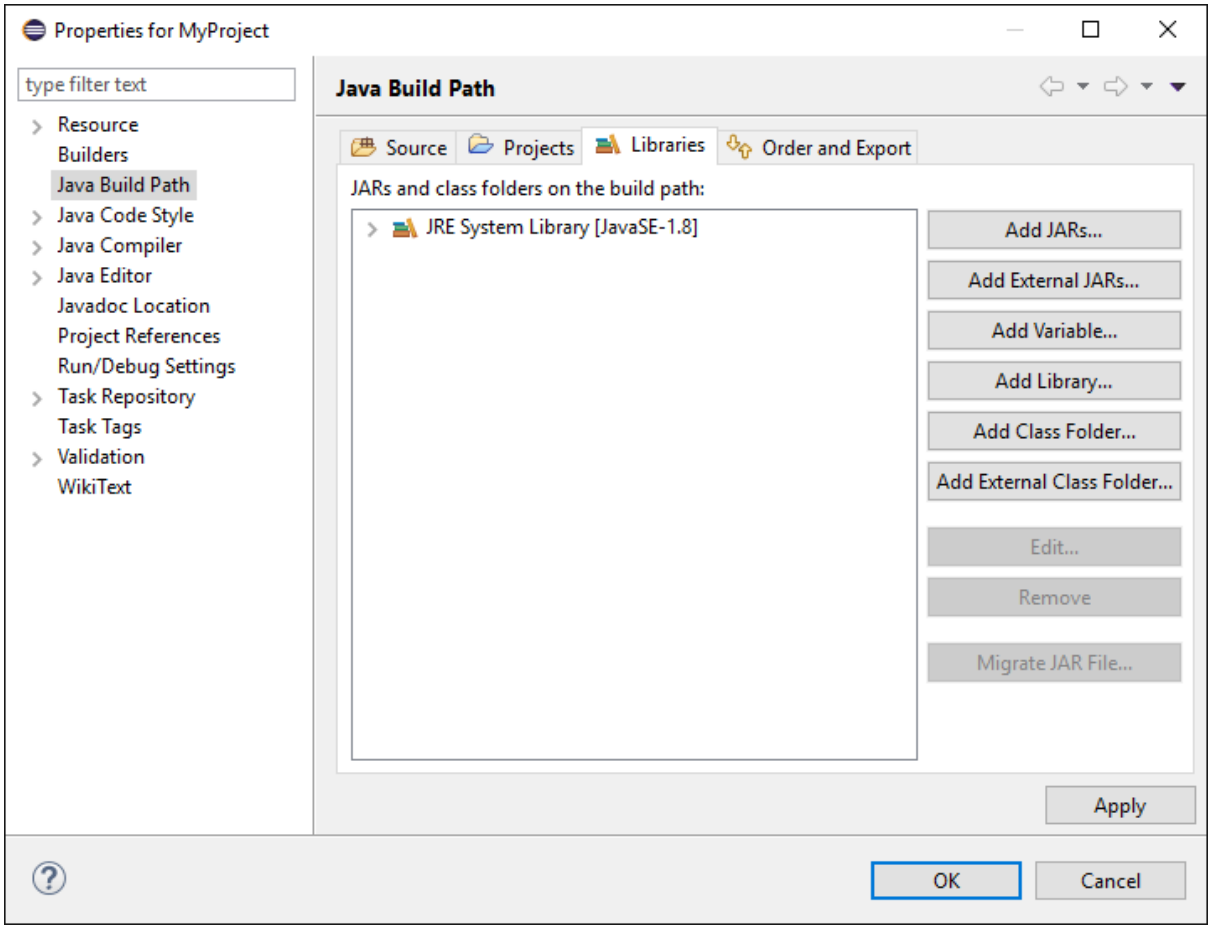
For version 8.2, we recommend **Java 8** — the core and most plugins will work on Java 7, but some plugins require 8. Mac users must install the full *JDK*, not just the *JRE*.

**Step 2:** Extract the contents of GATE's zip folder

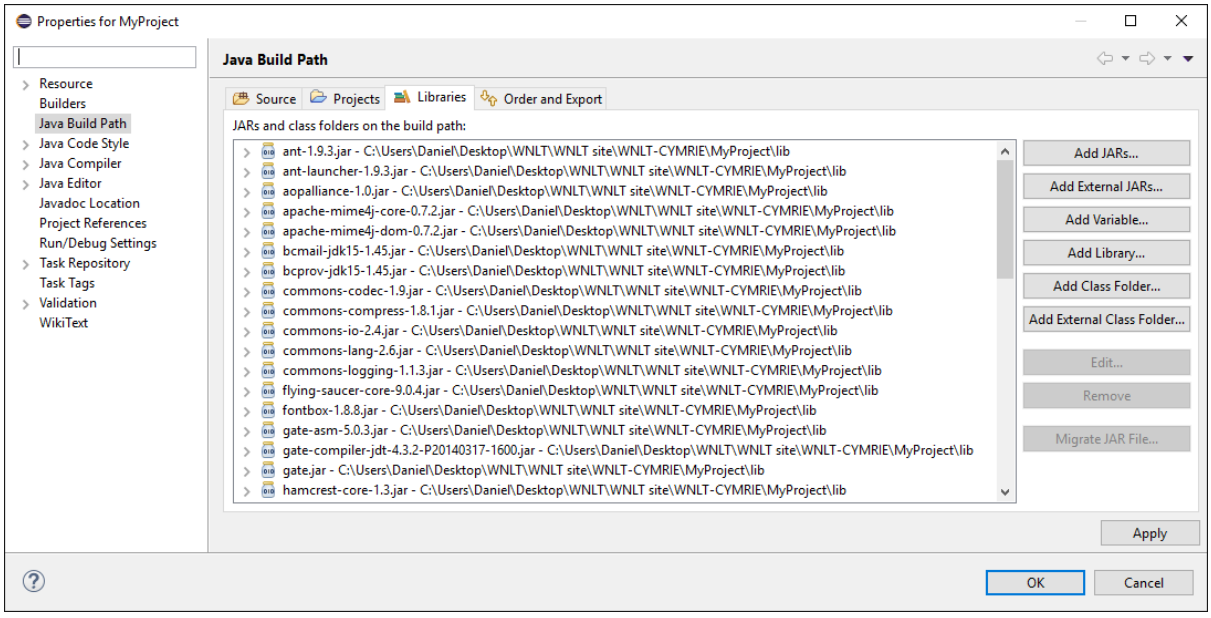
**Step 3:** Include all of the necessary ***.jar*** files in your project's build path. These are all the ***.jar*** files located in GATE's ***lib*** folder and the ***gate.jar*** file located in GATE's ***bin*** folder.

It is recommended that the user creates a folder called ***lib*** in the root directory of the user's Java project then copy all of the ***.jar*** files into that folder.

In Eclipse, click "*Project*" from the menu bar -> "*Properties*" -> "*Java Build Path*" -> "*Libraries*"




Click "*Add External JARs*" then add all the necessary **.jar** files in step 3.




Click "*Apply*" then "*OK*"

**Step 4:** Copy **wnlt.jar** located in the root directory of the WNLT folder into your Java project's lib folder and add it to the your project's build path, instructions in previous setup.

**Step 5:** Copy the 'plugins' and 'resources' folder into your Java project's root **directory**.

## 5.2. Using the *CymrIEUtilities* class

CymrIE is an information extraction (named entity recognition) system for the Welsh language. The CymrIE information extraction system is accessible to Java developers from the ***CymrIEUtilities*** class located in the Java package ***wnlt.api***. The following sections will outline key parts of the GATE API and outline the ***CymrIEUtilities*** class.

The WNLT has been developed using the framework provided by the GATE API. For this reason, the ***CymrIEUtilities*** class produces the CymrIE information extraction system and all of its processing resources as Java Objects from within the GATE framework. The ***CymrIEUtilities*** class enables GATE Embedded developers to use, reuse and adapt the components of the WNLT in new or existing GATE Embedded projects. Though the functionality exposed by the ***CymrIEUtilities*** class is targeted towards GATE Embedded developers, non-GATE Embedded developers can use the ***CymrIEUtilities*** class with help from the examples in the following sections.

The WNLT API also contains a ***CymrIE*** class which hides the GATE API code from the user. Non-GATE Embedded developers may wish to use the ***CymrIE*** class which is outlined in this chapter below in section 5.3.

### 5.2.1. Corpus (GATE API class)

The CymrIE information extraction system processes ***Corpus*** Objects located in the Java package ***gate***. The following code creates a ***Corpus*** with a ***Document***.

```
1. try {
2.     Gate.runInSandbox(true);
3.     Gate.init(); // Initialise GATE
4. } catch (GateException e) {
5.     e.printStackTrace();
6. }
7.
8. Corpus corpus = Factory.newCorpus("StandAloneAnnie corpus");
9. Document doc = Factory.newDocument(new
   URL("http://www.bbc.co.uk/cymrufyw/37655968"));
10. corpus.add(doc);
```

Line 2 tells GATE not to load any local configuration files when initialising.

Line 3 initialises GATE. This needs to be executed before functions are called by GATE's ***Factory*** class.

Line 8 creates a ***Corpus*** using GATE's ***Factory*** class. The GATE API requires GATE Objects to be created using the ***Factory*** class.

Line 9 creates a ***Document*** using GATE's ***Factory*** class.

Line 10 adds the ***Document*** to the ***Corpus***.

A ***Corpus*** Object holds a collection of ***Documents***. More information can be obtained from GATE Embedded's user guide

### 5.2.2. CymrIEUtilities

The ***CymrIEUtilities*** class has a function called ***getCymrIE()*** that creates a new instance of the CymrIE information extraction system in one line of code, see line 2 of code snippet below. This CymrIE Object mimics the functionality seen in the CymrIE application in the GATE Developer graphical user interface, see chapter 3.

```
1. try {
2.     ConditionalSerialAnalyserController cymrie = CymrIEUtilities.getCymrIE();
3.     Corpus corpus = getCorpus();
4.     cymrie.setCorpus(corpus);
5.     cymrie.execute();
6.
7. } catch (GateException e) {
8.     e.printStackTrace();
9. } catch (IOException e) {
10.    e.printStackTrace();
11. }
```

Line 4 sets a ***Corpus*** of documents to the CymrIE system then line 5 executes the CymrIE system on those documents within the ***Corpus***.

After the ***execute()*** function is run, the documents within the ***Corpus*** are annotated with metadata by the CymrIE information extraction system. The user can extract information from those annotations, such as *tokens*, *lemmas*, *part of speech tags*, *token types*, *orthographies*, *named entities* and *lookups performed by the CymrIEU system*. More information on how to extract this information from annotations can be found in section 5.2.3.

The ***CymrIEUtilities*** class also allows the user to get preconfigured processing resources used in the CymrIE information extraction system. Each processing resource in the CymrIE system can be obtained from the following functions in the ***CymrIEUtilities*** class,

- ***getAnnotationDelete()***
- ***getWelshTokeniser()***
- ***getWelshSentenceSplitter()***
- ***getWelshPOSTagger()***
- ***getWelshMorph()***
- ***getFlexibleGazetteer()***
- ***getANNIETransducer()***

The ***CymrIEUtilities*** class provides functions to get the default configurations of each processing resource, for example, the ***getWelshTokeniserDefaultParameters()*** function returns the default parameters used in the Welsh tokenizer processing resource of the CymrIE information extraction system. The ***CymrIEUtilities*** class also provides functionality that creates the processing resource given user specified parameters, for example ***getWelshTokeniser(FeatureMap)***. The following snippet of code reuses the CymrIE system and changes the '***listsURL***' parameter of the ***DefaultGazetteer*** process resource.

```
try {
    ConditionalSerialAnalyserController cymrieController = new ConditionalSerialAnalyserController();
    cymrieController.setName("Modified CymrIEUtilities");

    // Replace the listsURL parameter's value using CymrIEUtilities's default parameters
    FeatureMap defaultGazetteerParameters = CymrIEUtilities.getDefaultGazetteerDefaultParameters();
    defaultGazetteerParameters.replace("listsURL", new File("gazetteer/lists.def").toURI().toURL());
    DefaultGazetteer defaultGazetteer = (DefaultGazetteer) CymrIEUtilities.getDefaultGazetteer(defaultGazetteerParameters);

    // Create flexible gazetteer using the new default gazetteer
    FeatureMap flexibleGazetteerParameters = CymrIEUtilities.getFlexibleGazetteerDefaultParameters(defaultGazetteer);
    ProcessingResource flexibleGazetteer = CymrIEUtilities.getFlexibleGazetteer(flexibleGazetteerParameters);

    // Copied and modified from the getCymrIEUtilities() function
    ArrayList<ProcessingResource> prsArrayList = new ArrayList<ProcessingResource>();
    prsArrayList.add(CymrIEUtilities.getAnnotationDelete());
    prsArrayList.add(CymrIEUtilities.getWelshTokeniser());
    prsArrayList.add(CymrIEUtilities.getWelshSentenceSplitter());
    prsArrayList.add(CymrIEUtilities.getWelshPOSTagger());
    prsArrayList.add(CymrIEUtilities.getWelshMorph());
    prsArrayList.add(flexibleGazetteer); // add modified ProcessingResource
    prsArrayList.add(CymrIEUtilities.getANNIETransducer());

    // Set Processing resources and initialise
    cymrieController.setPRs(Collections.synchronizedList(prsArrayList));
    cymrieController.init();
} catch (GateException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

### 5.2.3. AnnotationSet (GATE API class)

After the CymrIE information extraction system has been executed on a corpus, the documents within the corpus are annotated with metadata by the CymrIE information extraction system, such as *tokens*, *lemmas*, *part of speech tags*, *token types*, *orthographies*, *named entities* and *lookups performed by the CymrIE system*. The following snippet of code shows how the AnnotationSet (which contains many annotations) can be obtained.

```
1. try {
2.     ConditionalSerialAnalyserController cymrie = CymrIEUtilities.getCymrIE();
3.     Corpus corpus = getTestCorpus();
4.     cymrie.setCorpus(corpus);
5.     cymrie.execute();
6.
7.     Iterator<Document> iterator = corpus.iterator();
8.
9.     while(iterator.hasNext()) {
10.           Document document = iterator.next();
11.           AnnotationSet annotationSet = document.getAnnotations();
12.    }
13. } catch (GateException e) {
14.    e.printStackTrace();
15. } catch (IOException e) {
16.    e.printStackTrace();
17. }
```

The following snippet of code shows how to retrieve all the lemmas of each token of a Document.

```
1. Document document = corpus.get(0); // first Document
2. AnnotationSet annotationSet = document.getAnnotations(); // get all annotations
3. Node node = annotationSet.firstNode(); // start from the beginning of the document
4. ArrayList<String> lemmas = new ArrayList<String>(); // create an ArrayList to store all lemmas
5.
6. while(node != null) {
7.        AnnotationSet orderedAnnotation = annotationSet.get(node.getOffset()); // get annotations given the current
   position in the document
8.
9.        if(orderedAnnotation.getAllTypes().contains("Token")) { // if AnnotationSet contains a Token annotation then
10.             Iterator<Annotation> annotationIter = orderedAnnotation.get("Token").iterator(); // get Token annotations
11.             lemmas.add((String)annotationIter.next().getFeatures().get("lemma")); // There is only one annotation, get
   the lemma value
12.       }
13.
14.       node = annotationSet.nextNode(node); // move to next node (next position in the document)
15. }
```

The AnnotationSet and Annotation Objects are data structures generated by the GATE API. Documentation for the AnnotationSet and Annotation data structures can be found in GATE Embedded's user guide.

### 5.2.4. Known Issues

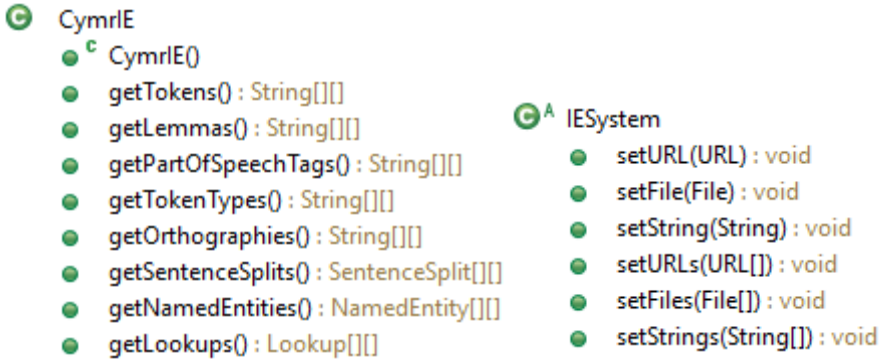
When creating a new corpus or document, developers must create these resources using GATE's ***Factory*** class. GATE creates references to those resources so to delete those references (and free up memory for the Garbage collector), developers must use the ***Factory*** class's ***deleteResource()*** function and must not have any references to the corpus or document within their own code.

Some processing resources such as CymrIEUtilitie's ***ConditionalSerialAnalyserController*** cannot be completely dereferenced using the ***Factory*** class's ***deleteResource()*** function. For this reason, the developer should limit the number of created instances. For most developer use cases, only one instance of CymrIEUtilitie's ***ConditionalSerialAnalyserController*** is needed.

## 5.3. Using the *CymrIE* class

The ***CymrIE*** class is located in the Java package ***wnlt.api***. This class provides functionality for typical use cases of the CymrIE information extraction system. This class is targeted towards non-GATE Embedded developers and hides all GATE API code from the user.

This class contains self-explanatory functions that accept File(s), URL(s) or String(s) (String(s) of Welsh text) and functions that return the CymrIE information extraction system's *tokenization*, *lemmatization*, *part of speech tags*, *token types*, *orthographies*, *sentences*, *named entities* and *lookups* performed by the CymrIE system. The functions are outlined in the following image.

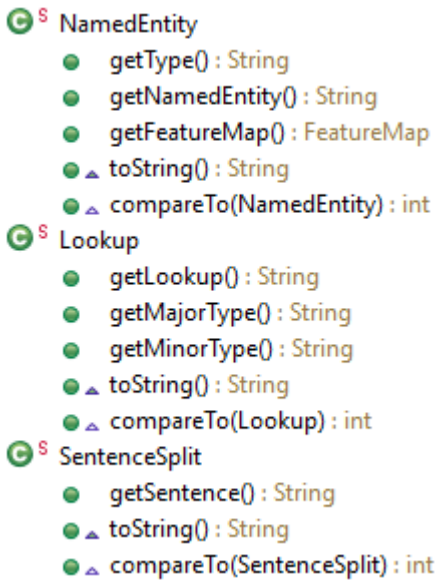



***CymrIE*** inherits from ***IESystem***.

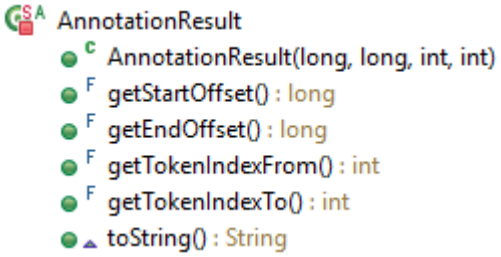
The following code finds the tokens and named entities in the content of two Files. The returned result is a two dimensional array where the first index indexes the File (or URL or String) and the second index indexes the result for that File. In the example below, the second index indexes the token (Line 8) and the named entity (Line 9).

```
1. try {
2.         CymrIE cymrie = new CymrIE();
3.         cymrie.setFiles(new File[] {
4.                 new File("TestData1.txt"),
5.                 new File("TestData2.txt")
6.         });
7.
8.         String[][] tokens = cymrie.getTokens();
9.         NamedEntity[][] namedEntities = cymrie.getNamedEntities();
10. } catch (GateException | IOException e) {
11.         e.printStackTrace();
12. }
```

Named entities, lookups and sentences are returned as Objects which encapsulate all the information about the named entity, lookup and sentence. See following image for the information that can be obtained from these Objects.




The NamedEntity, Lookup and SentenceSplit Objects inherit from the AnnotationResult class which encapsulates information about the character offsets and token indexes of the NamedEntity, Lookup or SentenceSplit.



# 6. TwitterCymrIE

Twitter is a social media platform where users can share content such as tweets. A tweet a short message that is no longer than 140 characters. As tweets cannot be longer than 140 characters in length, users tend to shorten words, use slang, acronyms and abbreviations to fit their message into a tweet. Tweets are user generated making tweets prone to spelling errors.

TwitterCymrIE is an information extraction system for Welsh tweets. TwitterCymrIE reuses the processing resources found in CymrIE and adapts the processing resources found in the TwitIE information extraction system that is available in GATE Developer. The Tweet Normaliser processing resource in TwitIE is of particular interest as it attempts to correct spelling mistakes, shorten words, slang etc.

In addition to the processing resources outlined in chapter 1 for the CymrIE information extraction system, TwitterCymrIE has adapted the following **five processing resources** from TwitIE:

- Annotation set transfer
- Language identification
- Emoticons Gazetteer
- Hashtag Gazetteer
- Tweet Normaliser

Each processing resource will be discussed in more detail in subsequent sections.

## 6.1. Annotation set transfer

The annotation set transfer processing resource identifies the tweet and the metadata associated with the tweet. If the supplied document is a JSON file obtained by using the Twitter API then the tweet and metadata is extracted from the JSON file. If the file is not a JSON file the whole text is treated as a tweet. More information on what metadata is available from Twitter JSON files can be found at https://dev.twitter.com/overview/api/tweets

A ‘user mention’ is a word or phrase with no spaces starting with the @ symbol. A ‘user mention’ is a particular user on Twitter. A hashtag is a word or phrase with no spaces starting with the # symbol. A hashtag is a form of metadata that allows users to categorise their tweet. Examples of user mentions and hashtags are shown in the following figure.


This processing resource enables TwitterCymrIE to execute subsequent processing resources only on the tweet and not the whole Twitter JSON file. This processing resource generates the Tweet, Hashtag, UserID and URL annotations.

### 6.1.1. Run-time parameters

***annotationTypes*** – if annotation type names are specified for this list, only candidate annotations of those types will be transferred or copied. If an entry in this list is specified in the formOldTypeName=NewTypeName, then annotations of type OldTypeName will be selected for copying or transfer and renamed to NewTypeName in the output annotation set.

***copyAnnotations*** – this specifies whether the annotations should be moved or copied. The default value false will move annotations, removing them from the *inputASName* annotation set. If set to true the annotations will be copied.

***inputASName*** – this defines the annotation set from which annotations will be transferred (copied or moved). If nothing is specified, the Default annotation set will be used.

***outputASName*** – this defines the annotation set to which the annotations will be transferred. This default value for this parameter is 'Filtered'. If it is left blank the Default annotation set will be used.

***tagASName*** – this defines the annotation set which contains the annotation covering the relevant part of the document to be transferred. This default value for this parameter is 'Original markups'. If it is left blank the Default annotation set will be used.

***textTagName*** – this defines the type of the annotation covering the annotations to be transferred. The default value for this parameter is 'BODY'. If this is left blank, then all annotations from the *inputASName* annotation set will be transferred. If more than one

covering annotation is found, the annotation covered by each of them will be transferred. If no covering annotation is found, the processing depends on the *copyAllUnlessFound* parameter.

***transferAllUnlessFound*** – this specifies what should happen if no covering annotation is found. The default value is true. In this case, all annotations will be copied or moved (depending on the setting of parameter *copyAnnotations*) if no covering annotation is found. If set to false, no annotation will be copied or moved.

## 6.2. TextCat Language identification

The TextCat language identification processing resource attempts to identify the language of the tweet. The TextCat processing resource is available in GATE Developer. The creators of TwitIE generated their own language models to identify whether a tweet is English, Spanish, German, Dutch or French. This processing resource is intended to choose what processing resources to execute next depending on the language of the tweet. TwitterCymrIE and TwitIE assume the language (Welsh and English respectively) and do not make use of this functionality but others are welcome to make use of this functionality.

TwitIE's language identification processing resource results were replicated using the same tweets that were used in the training and out of sample datasets (though some tweets were deleted so they were excluded from the dataset). The training datasets for each language model contained between 300-400 tweets and the out of sample datasets for each language model contained between 400-500 tweets. Using the same out of sample dataset, TextCat's default language models were tested. For both language identification systems, a Welsh language model created from 400 welsh tweets was included and tested. The average performance can be seen from the following table.

| | No Welsh language model | | With Welsh language model | |
|---|---|---|---|---|
| | TextCat | TwitIE | TextCat | TwitIE |
| Accuracy | 95.48% | 95.55% | 96.78% | 96.51% |
| Precision | 89.02% | 90.26% | 90.09% | 90.45% |
| Recall | 88.57% | 88.88% | 89.78% | 89.00% |
| NPV | 97.11% | 97.19% | 98.03% | 97.89% |
| Specificity | 97.26% | 97.30% | 98.15% | 97.95% |
| F-Score | 88.65% | 89.02% | 89.80% | 89.34% |

Without the Welsh language models, TwitIE's adapted language models derived from real tweets slightly outperformed the default language models supplied by TextCat. The performance differences between the TextCat and TwitIE language identification systems can be seen in the table below. The difference in performance is very small but there is a 1.25% increase in average precision in the TwitIE model.

| | No Welsh language model | With Welsh language model |
|---|---|---|
| Accuracy | -0.07% | 0.27% |
| Precision | -1.25% | -0.37% |
| Recall | -0.31% | 0.78% |
| NPV | -0.08% | 0.14% |
| Specificity | -0.04% | 0.20% |
| F-Score | -0.37% | 0.46% |

The result is different when including the default Welsh language model in TextCat and the Welsh language model created from Welsh tweets in TwitIE. The average precision of TwitIE is only 0.37% better than TextCat and the other performance metrics show TextCat outperforming TwitIE.

TwitterCymrIE uses the default language models (Welsh, English, Spanish, German, Dutch and French) distributed with TextCat to classify the language of the tweet. The TextCat processing resource adds a "lang" feature to the "Tweet" annotation generated by the Annotation set transfer processing resource.

### 6.2.1. Init-time parameters

***configURL*** – The list of TextCat language models used to determine the language of the tweet.

### 6.2.2. Run-time parameters

***annotationSetName*** – The annotation set used for input and output; ignored if *annotationType* is blank.

***annotationType*** – If this is supplied, the PR classifies the text underlying each annotation of the specified type and stores the result as a feature on that annotation. If this is left blank (null or empty), the PR classifies the text of each document and stores the result as a document feature.

***languageFeatureName*** – The name of the document or annotation feature used to store the results.

## 6.3. Emoticons Gazetteer

The emoticons gazetteer processing resource uses a gazetteer to normalise emoticons such as :-) and :} to the emoticon :). This processing resource generates the Emoticon annotation.

### 6.3.1. Init-time parameters

***caseSensitive*** – By default the gazetteer looks for matches irrespective of capital letters. If the user selects *'caseSensitive'* to be true, words are no longer converted into lowercase.

***encoding*** – The character encoding to be used for reading the input.

***gazetteerFeatureSeparator*** – The unique sequence of characters that separates the key and value in the *listsURL* file.

***listsURL*** – The path to the gazetteer for normalizing emoticons.

### 6.3.2. Run-time parameters

***annotationSetName*** – By default, left blank. The annotation set used for input and output; ignored if *annotationType* is blank.

***longestMatchOnly*** – If *true* then the longest match in the gazetteer is used.

***wholeWordsOnly*** – If *true* then only match whole words.

## 6.4. Hashtag tokenizer

A hashtag is a word or phrase that is used as a form of metadata to help categorise the tweet, see image below. The only constraints for hashtags are that they must only be alphanumeric characters with no spaces. Users can use hashtags to indicate the event or theme of the tweet or help others find their tweet by searching for the hashtag.


This processing resource attempts to split the hashtag into multiple words, to do this the hashtag tokenizer does two things:

- Uses JAPE rules to identify camel casing (there is no constraint that the user must use camel casing in their tweets).
- Uses the Eurfa dictionary and named entities used in the CymrIE information extraction system to break up the hashtag into words.

### 6.4.1. Init-time parameters

***gazetteerURL*** – The path to the gazetteer for Welsh words.

### 6.4.2. Run-time parameters

***inputASName*** – The name of the annotation set used for input. It is optional, if left blank then the 'default' annotation set is assigned.

***outputASName*** – The name of the annotation set used for output. This is an optional parameter. If user does not provide any value, new annotations are created under the default annotation set.

## 6.5. Tweet Normaliser

This processing resource attempts to correct spelling mistakes as well as normalising slang and shortened words.

Spelling mistakes are corrected by using the Eurfa dictionary and other gazetteers using in the CymrIE information extraction system. Spelling mistakes are corrected by comparing the Levenshtein distance between each word, if a word is found in the gazetteers with a Levenshtein distance of 2 or less then it is corrected to that word.

To normalise slang a specialised gazetteer is used that contains Welsh slang and English slang normalizations. This gazetteer takes words such as 'l8r' and normalises it to 'later'. After this processing resource is executed, it replaces the feature 'string' of the 'Token' annotation with the normalised word and moves the original word to a new feature called "origString" within the "Token" annotation.

### 6.5.1. Init-time parameters

***dictURL*** – Path to dictonary of words for correcting words with some Levenshtein distance.

***orthURL*** – Path to common normalisation terms. For example, 'b4' -> 'berfore'.

### 6.5.2. Run-time parameters

***initialTextFeature*** – Feature on Token annotations in the input AS that contains the token string.

***inputASName*** – The name of the annotation set used for input. It is optional, if left blank then the 'default' annotation set is assigned.

***maxDistance*** – Maximum Levenshtein distance to correct words.

***normTextFeature*** – Feature to which the normalized text should be saved.

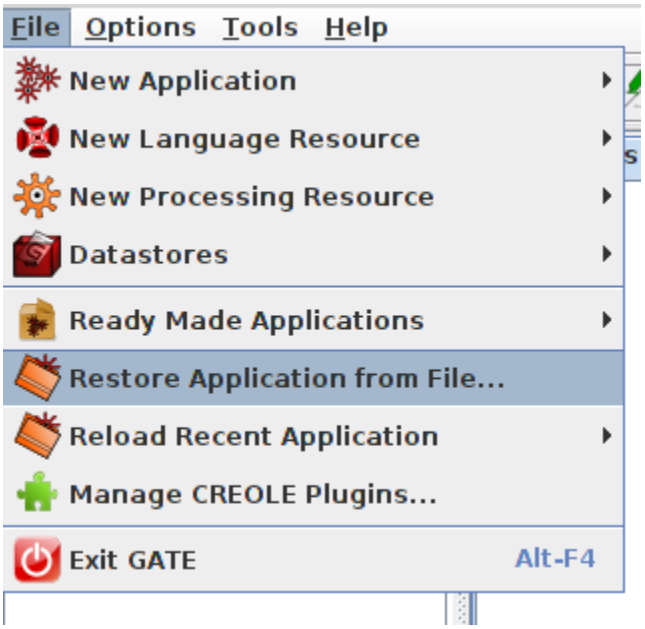
***origTextFeature*** – Feature to which the original text should be saved.

***outputASName*** – The name of the annotation set used for output. This is an optional parameter. If user does not provide any value, new annotations are created under the default annotation set.

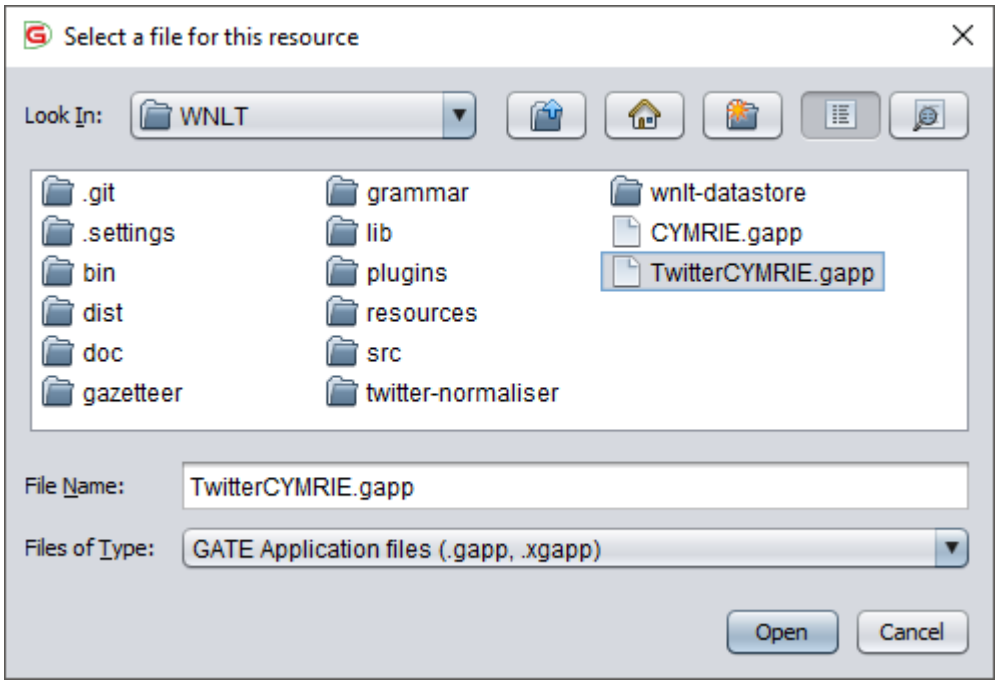
## 6.6. GATE Developer

TwtterCymrIE can be used within the GATE Developer graphical user interface by following these instructions.

**Step 1:** From the File menu in GATE choose the Restore Application from File option.



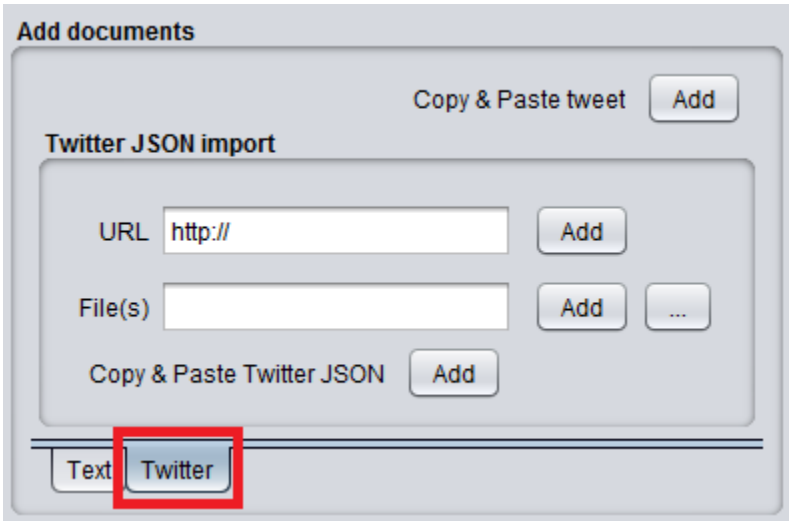

**Step 2:** Select the file TwitterCYMRIE.gapp file located in the WNLT folder and click Open




The TwitterCymrIE application is now loaded into GATE Developer and can be used in a similar fashion as outlined in 2.2 Loading the CymrIE system in GATE. There is a corpus of Welsh JSON tweets in the wnlt-datastore, to open the datastore follow the instructions outlined in section 2.3 Adding a New Corpus in GATE.

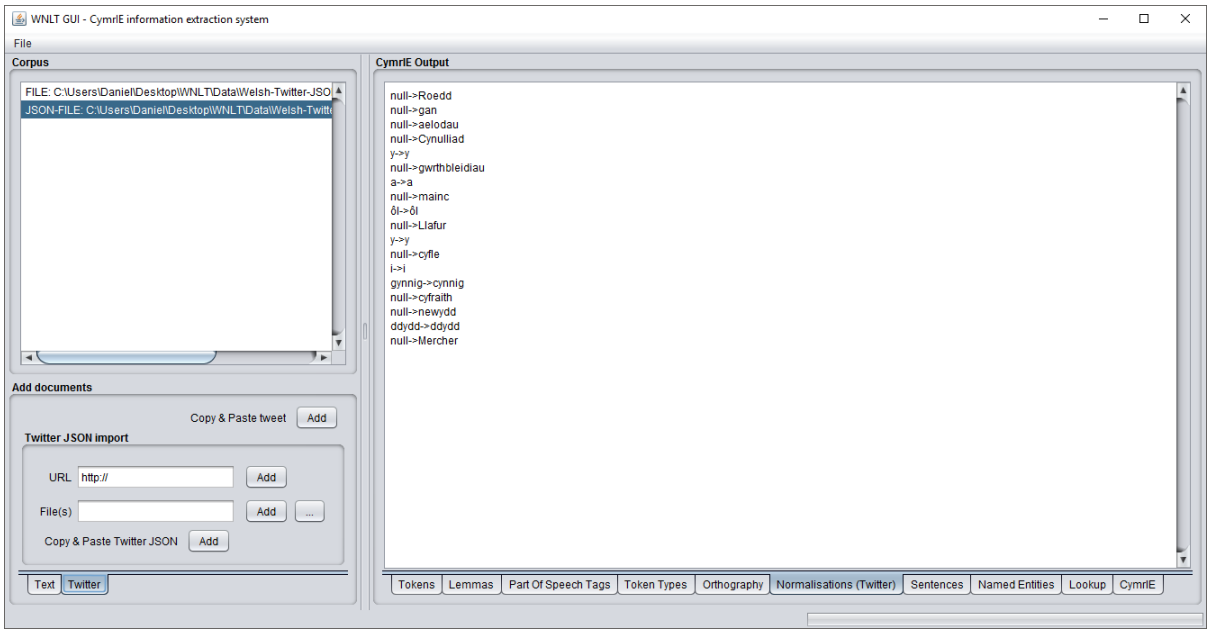
## 6.7. WNLT's GUI

The main functionality of CymrIE in WNLT's GUI is described in chapter 3. As well as the CymrIE information extraction system the TwitterCymrIE information extraction system can be used. Tweets can be imported by selecting the Twitter tab in the bottom left corner of the application, see image below.



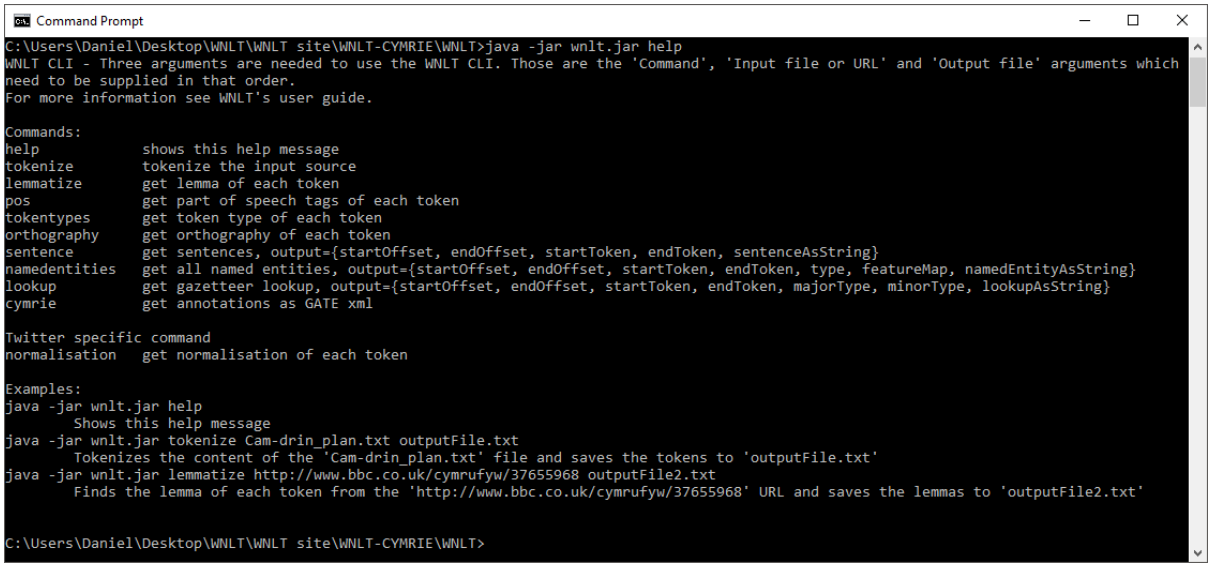

The “Copy & Paste” tweet button allows the user to paste a raw tweet in a separate dialog and the other buttons and text fields surrounded by the text “Twitter JSON import” allow the user to load JSON sources supplied by Twitter’s API.

After selecting a document in the corpus an additional tab called “Normalisation (Twitter)” is visible which displays all of the normalisations that took place using the TwitterCymrIE information extraction system.




## 6.8. WNLT’s CLI

Chapter 4 outlines how to use the WNLT’s command line interface. As can be seen from the help message produced by the command line interface below, there is an additional command for Twitter JSON sources called ‘normalisation’. This command saves all of the normalisations that took place using the TwitterCymrIE information extraction system. As the TwitterCymrIE information extraction system is based on CymrIE, all of the other commands can be used on Twitter JSON sources.




## 6.9. WNLT API

Instructions on how to setup the WNLT API and use CymrIE is detailed in chapter 5. This section expends on chapter 5 and outlines the functionality for Twitter.

### 6.9.1. Using the *TwitterCymrIEUtilities* class

The ***TwitterCymrIEUtilities*** class is located in the Java package ***wnlt.api.twitter***. The class has a function called ***getTwitterCymrIE()*** that creates a new instance of the TwitterCymrIE information extraction system in one line of code, see line 2 of code snippet below.

```
1. try {
2.     ConditionalSerialAnalyserController cymrie =
   TwitterCymrIEUtilities.getTwitterCymrIE();
3.     Corpus corpus = getCorpus();
4.     cymrie.setCorpus(corpus);
5.     cymrie.execute();
6.
7. } catch (GateException e) {
8.     e.printStackTrace();
9. } catch (IOException e) {
10.    e.printStackTrace();
11. }
```

The ***TwitterCymrIEUtilities*** class also allows the user to get preconfigured processing resources used in the TwitterCymrIE information extraction system. In addition to the processing resources in the CymrIE information extraction system, other processing resources in TwitterCymrIE can be obtained from the following functions in the ***TwitterCymrIEUtilities*** class:

- ***Annotation set transfer()***
- ***Language identification()***
- ***Emoticons Gazetteer()***
- ***Hashtag Gazetteer()***

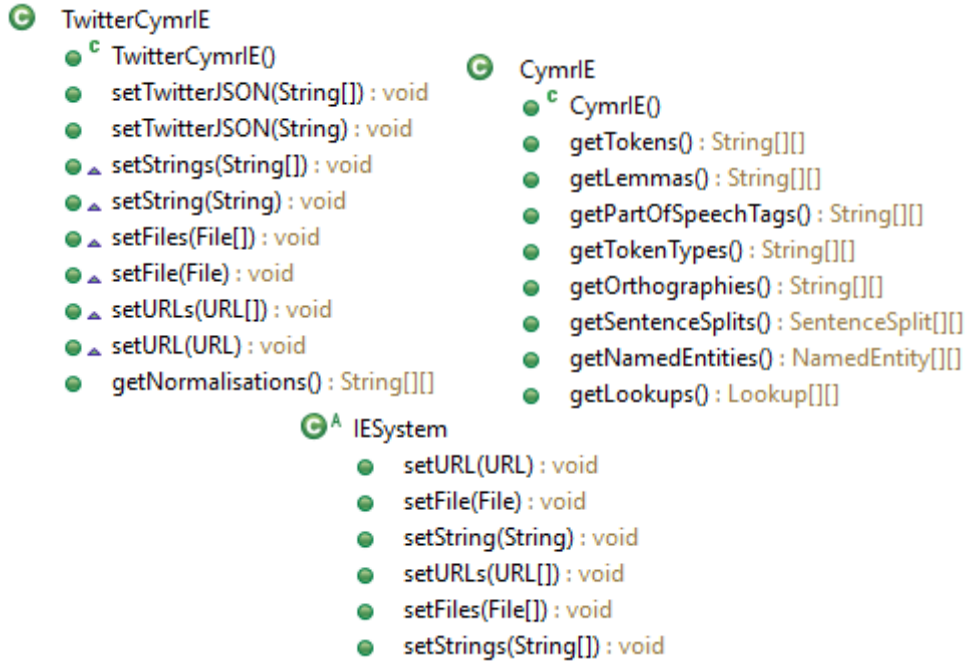
- ***Tweet Normaliser()***

The default parameter settings for each of these processing resources can also be obtained by calling the functions with the *DefaultParameters* suffix.

### 6.9.2. Using the *TwitterCymrIE* class

The ***TwitterCymrIE*** class is located in the Java package ***wnlt.api.twitter***. This class provides functionality for typical use cases of the TwitterCymrIE information extraction system. This class is targeted towards non-GATE Embedded developers and hides all GATE API code from the user.

This TwitterCymrIE Object behaves similarly to ***CymrIE*** with the addition of an extra method to obtain the normalisations performed by TwitterCymrIE and some functions in the ***IESystem*** class were overridden. The functions are outlined in the following images.




***TwitterCymrIE*** inherits from ***CymrIE*** which inherits from ***IESystem***.

The functions in the ***IESystem*** class were overridden to explicitly set each data source's mime type to be of the Twitter JSON format. The only exception is the ***setString*** and ***setStrings*** methods accept tweet(s) and place each String into the Twitter JSON format so that it can be treated as Twitter JSON. More information about each method in the ***TwitterCymrIE*** class can be found in the supplied Javadoc documentation.