Module 6: Tools and Plugins

Topics covered

- This module is about adapting ANNIE to create your own applications, and to look at more advanced techniques within applications
  - Using different gazetteers
  - Adapting ANNIE to different languages
  - Using conditional applications
  - Section-by-section processing
  - Using multiple annotation sets
  - Separating useful content in a document
  - Schema Enforcer
  - Using Groovy
  - Modular Pipelines

Using different gazetteers

Why?

- The standard gazetteer in ANNIE only performs exact matching against the text
- An entry in a gazetteer list must match the word exactly in the text (with the exception of capitalisation issues depending on if the case-sensitive parameter is switched on)
- But what if we want to match a plural word in the text with a singular word in the gazetteer?
- Or different forms of a verb (says, saying, say, said etc.)
- It would be nice not to have to specify alternative forms of each word in the lists
- Luckily, we have ways to do this

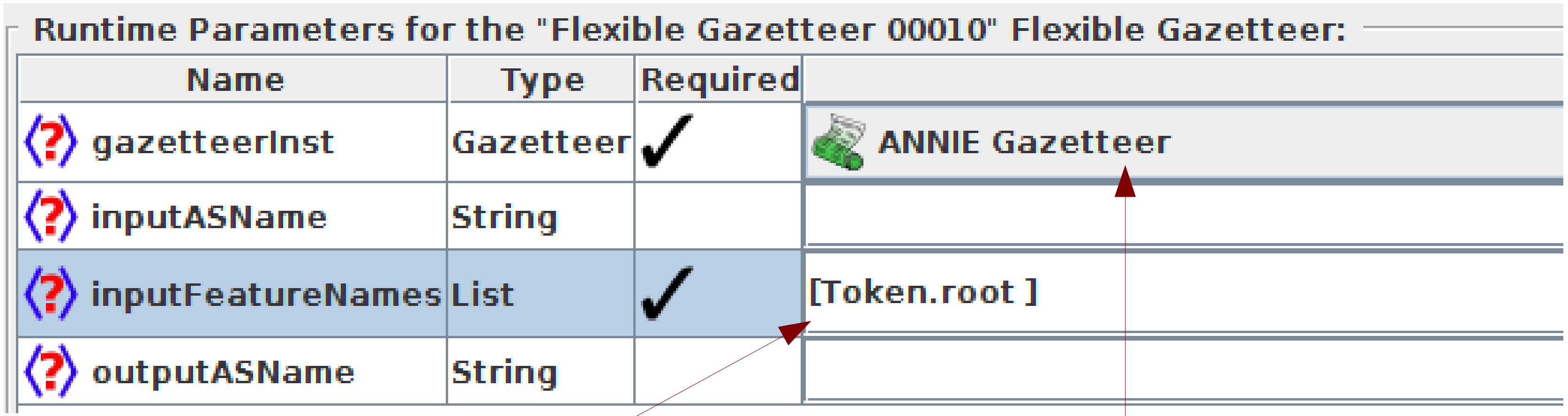
Advanced Gazetteers

- There are several different gazetteers which let you do more complex matching
  - **Flexible Gazetteer**: enables matching against features on an annotation (typically the Token's root feature)
  - **Feature Gazetteer**: enables matching against features on an annotation, but also enables adding/removing annotations and features when a match is found
  - **Extended Gazetteer**: as for the flexible gazetteer, but also provides features for more powerful matching of partial words, annotating prefixes and suffixes, and more versatile handling of word boundaries and white space.

Flexible Gazetteer

- Found in the Tools plugin
- Requires a regular gazetteer to be loaded in memory **but** not in the pipeline itself (the regular gazetteer gets wrapped in the flexible one)
- Run-time parameters let you specify:
    - the regular gazetteer to use
    - the annotations and features to match on
    - input and output annotation sets
- A typical use for this is to match against the root form of a word (e.g. dogs -> dog; laughing -> laugh)
- To do this, you need to specify Token.root as the annotation and feature to match. You also need to make sure you have run the morphological analyser first, so you have root features on your Tokens

Flexible gazetteer run-time params



Runtime Parameters for the "Flexible Gazetteer 00010" Flexible Gazetteer:

Name	Type	Required	
gazetteerInst	Gazetteer	✓	ANNIE Gazetteer
inputASName	String		
inputFeatureNames	List	✓	[Token.root]
outputASName	String		

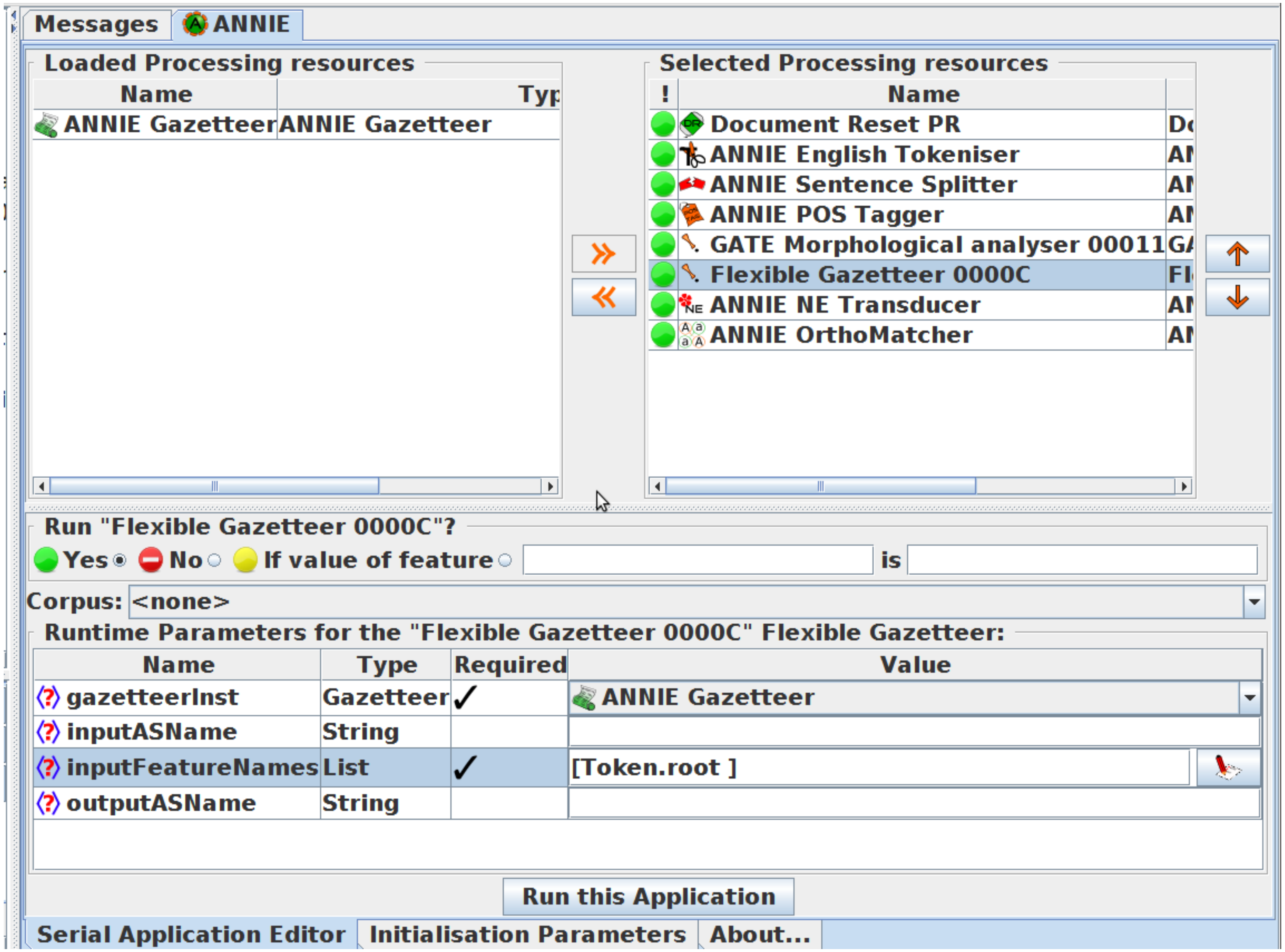
Choose the annotation name and feature that you want to match on

Select a gazetteer that you have loaded, e.g. the default ANNIE one

Hands-on with flexible gazetteer

- Load ANNIE
- Load the Tools plugin
- Create a new Flexible Gazetteer and a new Morphological Analyser
- Go to the ANNIE application and add the morphological analyser and flexible gazetteer to the pipeline after the POS tagger
- Select the ANNIE gazetteer as the gazetteer instance to use in the Flexible Gazetteer; set Token.root as the input feature name (one item in the list)
- Remove the ANNIE gazetteer from the application (but don't remove it from GATE) or switch it off
- Try it on some text!

Hands-on with flexible gazetteer



Loaded Processing resources

Name	Typ
ANNIE Gazetteer	ANNIE Gazetteer

Selected Processing resources

!	Name	
●	Document Reset PR	Do
●	ANNIE English Tokeniser	AN
●	ANNIE Sentence Splitter	AN
●	ANNIE POS Tagger	AN
●	GATE Morphological analyser 00011	GA
●	Flexible Gazetteer 0000C	Fl
●	ANNIE NE Transducer	AN
●	ANNIE OrthoMatcher	AN

Run "Flexible Gazetteer 0000C"?

[✓ Yes] No If value of feature ___ is ___

Corpus: <none>

Runtime Parameters for the "Flexible Gazetteer 0000C" Flexible Gazetteer:

Name	Type	Required	Value
gazetteerInst	Gazetteer	✓	ANNIE Gazetteer
inputASName	String		
inputFeatureNames	List	✓	[Token.root]
outputASName	String		

Extended Gazetteer

- Found in the StringAnnotation plugin
- Faster loading, uses much less memory than regular gazetteer
- Needs annotations that identify words and whitespace (usually *Token* and *SpaceToken*).
- Can limit matching to just within containing annotations
- This PR can be used for direct matching of document text or indirect matching of feature values
- Can specify separately whether to match at the beginning and/or the end of words
- Can use (gzip) compressed list files (.lst.gz)

Init parameters

- **caseSensitive**: false if case should be ignored for matching
- **configFile URL**: specify the definition/config file – similar to the “listsURL” parameter on the ANNIE gazetteer
- **caseConversionLanguage**: Specify the language to use for converting characters to upper case when case-insensitive matching (e.g. ß→SS for de) . Default is en (English)
- **gazetteerFeatureSeparator**: same as for the ANNIE gazetteer (but “\t” tab character is the default here)
- => no encoding parameters, list files have to be UTF-8 encoded

Run-time parameters

- **containingAnnotationType**: if an annotation type is given, then matching is done only fully within the span of such annotations. E.g. DocumentContent, Sentence.
- **longestMatchOnly**: if set to true, then only the longest match is used and all shorter matches are ignored.
- **matchAtWordEndOnl**y: if true, then the end of a match can only occur at the end of a word annotation. Typically set to true.
- **matchAtWordStartOnly**: if true, then the start of a match can only occur at the start of a word annotation. Typically set to true.
- **textFeature**: feature of the word annotation to match on (as for FlexibleGazetteer). Typically left empty or set to root.

More run-time parameters

- **outputAnnotationType**: in case you want to change the name of the annotation to be created on a match (instead of Lookup)
- **spaceAnnotationType**: the annotation type that identifies space between words. Default is SpaceToken.
- **splitAnnotationType**: the annotation type that identifies positions in the document that should not be crossed by matches. Default is Split.
- **wordAnnotationType**: type of annotations that define the word boundaries of the text that should be used for matching or if matching by feature is used, the annotations containing the feature. Default is Token.

Extended gazetteer cache files

- When a gazetteer is first loaded from a .def file, then the ExtendedGazetteer will create a new gazetteer cache file.
- This cache file has the same name as the .def file but with a file extension ".gazbin" instead of ".def".
- When the gazetteer gets loaded and such a cache file exists, the cache file will be loaded instead of the original files.
- NOTE: if a cache file exists, it will always be used, even if the .def or any .lst file has been changed in the meantime. If you update the gazetteer data, make sure you select “Remove cache and re-initialise” in the GUI

Feature gazetteer

- Found in the StringAnnotation plugin
- Enables adding/removing annotations/features when a match is found
  - For example, if tokens have a root feature and there is a gazetteer list that has as a feature the frequencies of English word roots in some corpus, the "add features" action can be used to enrich the token annotations with word frequencies.
  - **filter annotations**: if there is a gazetteer of stopwords, the string or root feature of existing token annotations can be matched and the "remove annotation" action can be used to remove these annotations if a stopword is matched.

Init parameters

- exactly the same as for the ExtendedGazetteer
- Note: this gazetteer uses the cache, .def and .lst files in exactly the same way as the ExtendedGazetter. If the ExtendedGazetteer and/or FeatureGazetteer load from the same files using the same Init-parameters, only one shared copy is used in memory.

Feature gazetteer run-time parameters

- **containingAnnotationType**: If an annotation type is given, then matching is done only within the span of such annotations.
- **InputAnnotationSet**: the set that contains the annotations to be updated, if annotations are updated
- **matchAtStartOnly**: if true, then a match must be found at the start of the value of the feature, if false, a match may start anywhere.
- **matchAtEndOnly**: if true, then a match must be found that ends at the end of the value of the feature, if false, a match may end anywhere.
- **outputAnnotationType**: in case you want to change the name of the annotation to be created on a match, if annotations are created (instead of Lookup)

More run-time parameters

- **wordAnnotationType**: the annotation type that is used for matching. For example Token or Lookup.
- **textFeature**: the name of a feature of the word annotation which is used for matching, e.g. root or id
- **processingMode**: select an option from:
  - **AddFeatures**: add all features from the def file or the gazetteer entry which are not already present in the annotation
  - **OverwriteFeatures**: overwrite all features (if any existing) from the def file or the gazetteer entry with the new values
  - **RemoveAnnotation**: delete the annotation from the input AS
  - **AddNewAnnotation**: add a new annotation to the output AS
  - **KeepAnnotation**: keep annotation if match, else remove

Gazetteer features

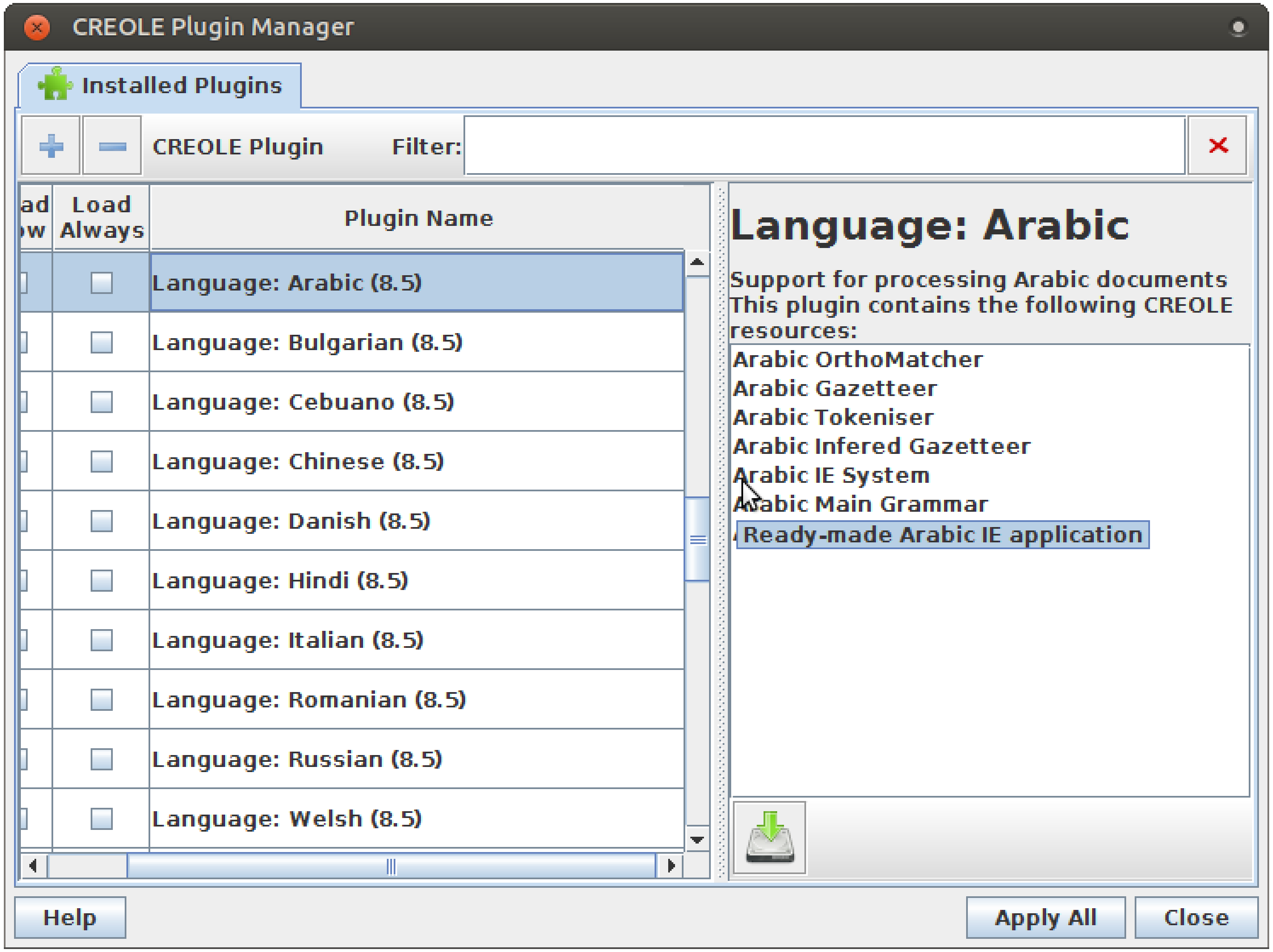
- All GATE gazetteers allow arbitrary feature values to be associated with particular entries in a single list
- Values are separated by the gazetteerFeatureSeparator character which for historical reasons defaults to ;
- We advise setting this to the tab character (\t) when creating a new gazetteer so that it is never confused with part of a list entry
- It also means you can directly use tab-separated value (.tsv) files from your favourite spreadsheet editor
- You do not have to provide the same features for every line in the file, e.g. you can provide extra features for some lines in the list but not others.
- Use the “+cols” option in the ANNIE gazetteer editor to add new sets of features and values

Developing IE for other languages

Finding available resources

- When creating an IE system for new languages, it's easiest to start with ANNIE and then work out what needs adapting
- Check the resources in GATE for your language (if any)
  - Check the GATE plugin manager (hint: the language plugins begin with “Language:”)
  - Check the user guide for things like POS taggers and stemmers which have various language options
- Check which PRs you can reuse directly from ANNIE
  - Existing tokeniser and sentence splitter will work for most European languages. Asian languages may require special components.
- Collect any other resources for your language, e.g POS taggers. These can be implemented as GATE plugins.

Language plugins available



ad ow	Load Always	Plugin Name
☐	☐	Language: Arabic (8.5)
☐	☐	Language: Bulgarian (8.5)
☐	☐	Language: Cebuano (8.5)
☐	☐	Language: Chinese (8.5)
☐	☐	Language: Danish (8.5)
☐	☐	Language: Hindi (8.5)
☐	☐	Language: Italian (8.5)
☐	☐	Language: Romanian (8.5)
☐	☐	Language: Russian (8.5)
☐	☐	Language: Welsh (8.5)

Language: Arabic

Support for processing Arabic documents This plugin contains the following CREOLE resources:

- Arabic OrthoMatcher
- Arabic Gazetteer
- Arabic Tokeniser
- Arabic Infered Gazetteer
- Arabic IE System
- Arabic Main Grammar
- Ready-made Arabic IE application

Which resources need modifying?

We can divide the PRs into 3 types depending on how much modification they need to work with other languages:

- **language-independent**: work with different languages with little or no modification
- **easily modifiable**: can be easily modified for a different language with little programming skill
- **language-dependent**: these need to be replaced by an entirely new PR

Language-independent resources

- ANNIE PRs which are totally language-independent are the **Document Reset** and **Annotation Set Transfer**
- They can be seen as “language-agnostic” as they just make use of existing annotations with no reference to the document itself or the language used
- The **tokeniser** and **sentence splitter** are (more or less) language-independent and can be re-used for languages that have the same notions of token and sentence as English (white space, full stops etc)
- Make sure you use the Unicode tokeniser, not the English tokeniser (which is customised with some English abbreviations)
- Some tweaking could be necessary - these PRs have resources that are easy to modify (with no Java skills needed)

Easily modifiable resources

- **Gazetteers** are normally language-dependent, but can easily be translated or equivalent lists found or generated
  - Lists of numbers, days of the week etc. can be translated
  - Lists of cities, countries, etc., can be found on the web
- Gazetteer modification requires no programming or linguistic skills
- The **Orthomatcher** will work for other languages where similar rules apply, e.g. John Smith --> Mr Smith
- Might need modification in some cases: some basic Java skills and linguistic knowledge are required

Language-dependent resources

- **POS taggers** and **grammars** are highly language-dependent
- If no POS tagger exists, a hack can be done by replacing the English lexicon for the Hepple tagger with a language-specific one
- Some grammar rules can be left intact, but many will need to be rewritten
- Many rules may just need small modifications, e.g., component order needs to be reversed in a rule
- Knowledge of some linguistic principles of the target language is needed, e.g., agglutination, word order
- No substantial programming skills are required, but knowledge of JAPE and basic Java are necessary

Adding a POS tagger for a new language

- If you already have a POS tagger for your language with a Java API, you can write a “wrapper” PR for it
  - This enables you to feed sentences/tokens to the tagger, and map the output back to GATE annotations
  - See the Stanford_CoreNLP plugin for an example
- Or for a non-Java POS tagger, use the Tagger_Framework PR (to pipe data through an external process and back)
- If you have a POS-tagged corpus, you can translate it into “traditional” tagged format with one line per sentence, e.g.

  The_DT cat_NN sat_VBD on_IN the_DT mat_NN ._.

  - You can use this to train a model for the LingPipe POS Tagger PR
  - This is how we POS-tagged Bulgarian in GATE

Tree Tagger

- Language-independent POS tagger supporting English, French, German, Spanish in GATE
- Needs to be installed separately (binary executable + models)
- Also supports Italian and Bulgarian, but not in GATE
- Tagger framework should be used to run the TreeTagger
- This provides a generic wrapper for various taggers
- In addition to TreeTagger, sample applications for
  - GENIA (English biomedical tagger)
  - HunPos (English and Hungarian)
  - Stanford Tagger (English, German and Arabic)
- More details in the GATE User Guide

Conditional Processing

What is conditional processing?

- In GATE, you can set a processing resource in your application to run or not depending on certain circumstances
- You can have several different PRs loaded, and let the system automatically choose which one to run, for each document.
- This is very helpful when you have texts in multiple languages, or of different types, which might require different kinds of processing
- For example, if you have a mixture of German and English documents in your corpus, you might have some PRs which are language-dependent and some which are not
- You can set up the application to run the relevant PRs on the right documents automatically.

Conditional processing with different languages

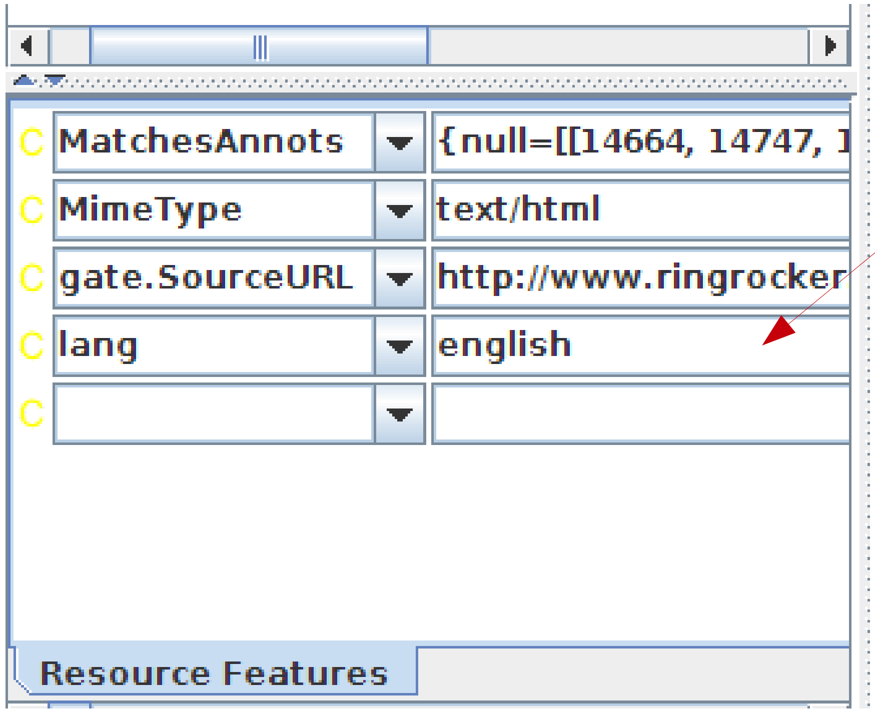
- Suppose we have a corpus with documents in German and English, and we only want to process the English texts.
- First we must distinguish between the two kinds of text, using a language identification tool
- For this we can use TextCat, which has a GATE plugin (Language Identification)
- We use this (in default mode) to add a feature on each document, telling us which language the document is in
- Then we run a conditional processing pipeline, that only runs the subsequent PRs if the value of the language feature on the document is English
- The other documents will not be processed

Hands-on with multilingual corpora

- Create a new corpus in GATE and populate it with the two documents found in corpus/rar-english-german-corpus/
- Select utf-8 as the encoding when you populate the corpus
- You should have one English and one German document loaded
- Load the Language Identification plugin and create a TextCat Language Identification PR
- Create a new application
- Add TextCat to the end of the application and run it on the corpus
- Examine the document features for both documents

Check the document language

- Click on a document
- In the bottom left corner is the document features pane
- TextCat will add a language feature here



MatchesAnnots	{null=[[14664, 14747, 1
MimeType	text/html
gate.SourceURL	http://www.ringrocker
lang	english

What if we want to process the German and the English?

- If we want to process both German and English documents with different resources, we have a couple of options

  1. We can just call some language-specific PRs conditionally, and use the language-neutral PRs on all documents
  2. We can call different applications from within the main application

- The following two hands-on exercises demonstrate the difference between these

Hands-on with multilingual apps

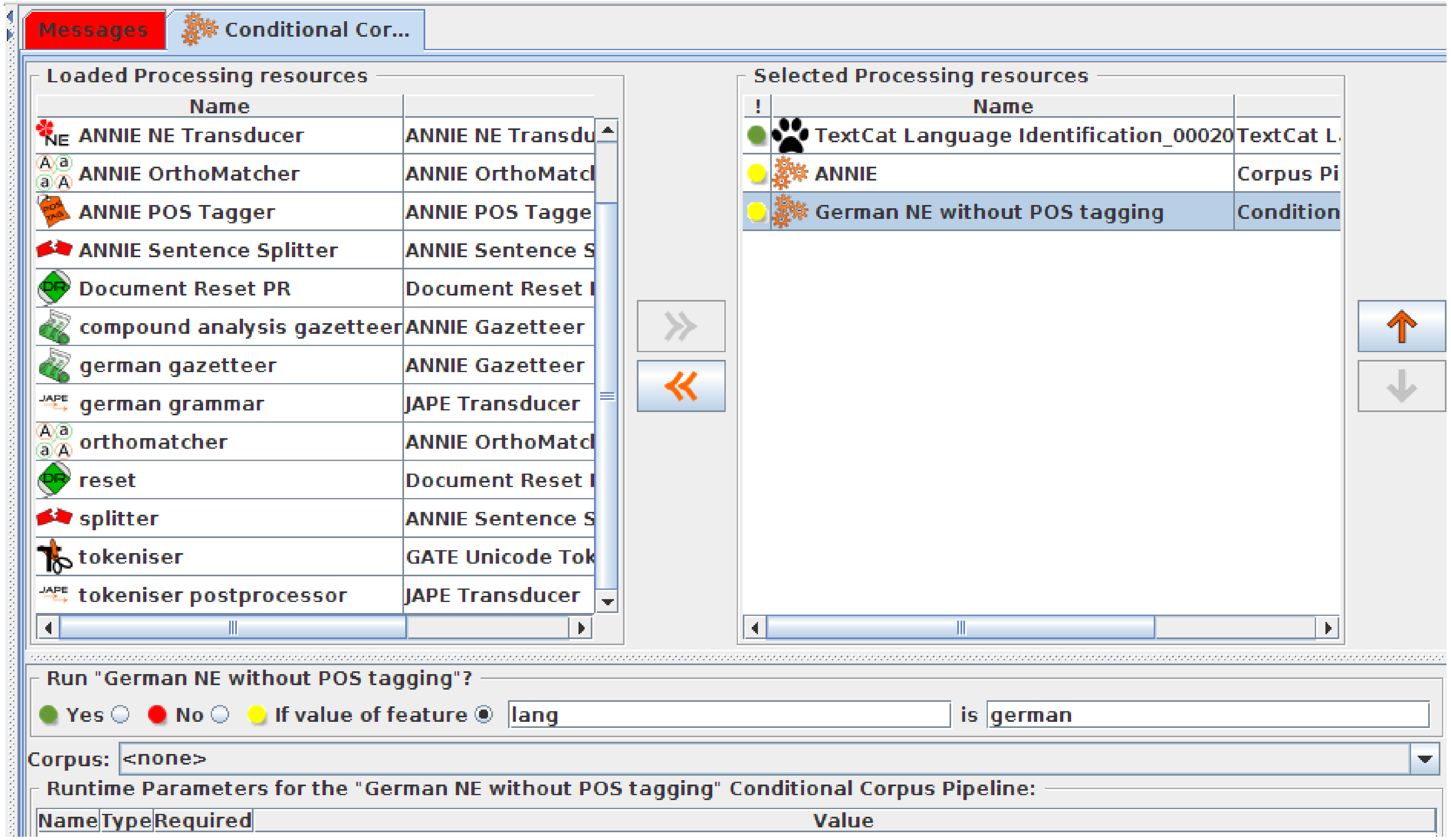
- Load the application annie+german.gapp
- Look at the various PRs in the app: some are set to run on English documents, some on German ones, and some on all documents
- Run the application on your corpus
- The German document should now be annotated with German NEs and the English document with English ones
- There will be some mistakes (we're not using a German POS tagger here so results are weaker than usual)

Hands-on with multilingual apps

- Close recursively all applications you have loaded in GATE (keep the corpus and documents)
- Load ANNIE
- Load **german-ie.gapp** from the hands-on materials
- Create a new conditional corpus pipeline
- Create a TextCat PR and add it to the new pipeline
- Add the ANNIE and German NE **applications** to the pipeline (in either order) after the TextCat

  Set ANNIE to run on English documents and the German app to run on German ones (use document features and values that match the TextCat output we have already seen)
- Run the main application on your corpus

Your application should look like this



Messages | Conditional Cor...

Loaded Processing resources

Name	
ANNIE NE Transducer	ANNIE NE Transdu
ANNIE OrthoMatcher	ANNIE OrthoMatc
ANNIE POS Tagger	ANNIE POS Tagge
ANNIE Sentence Splitter	ANNIE Sentence S
Document Reset PR	Document Reset I
compound analysis gazetteer	ANNIE Gazetteer
german gazetteer	ANNIE Gazetteer
german grammar	JAPE Transducer
orthomatcher	ANNIE OrthoMatc
reset	Document Reset I
splitter	ANNIE Sentence S
tokeniser	GATE Unicode Tok
tokeniser postprocessor	JAPE Transducer

Selected Processing resources

!	Name	
●	TextCat Language Identification_00020	TextCat L
●	ANNIE	Corpus Pi
●	German NE without POS tagging	Condition

Run "German NE without POS tagging"?

Yes ○ No ○ If value of feature [✓ ◉] lang is german

Corpus: <none>

Runtime Parameters for the "German NE without POS tagging" Conditional Corpus Pipeline:

Name	Type	Required	Value

Other uses for conditional processing

- Processing degraded text along with normal text
- For degraded text (e.g., emails, ASR transcriptions) you might want to use some case-insensitive PRs
- Another use is in combination with different kinds of files (HTML, plain text etc.) which might require different processing
- More about this later....

Another example

- In one application we developed, we found a problem when running the Orthomatcher (co-reference) on certain texts where there were a lot of annotations of the same type.
- To solve this issue, we first checked to see how many annotations of each were present in a document
- If more than a certain number were present, we added a document feature indicating this
- We then set the orthomatcher to only run on a document which did not contain this feature.

Grammar to check number of annotations

If there are more than 200 Person annotations, don't run the orthomatcher!

```
Options: control = once


Rule: CheckAnnotations
({Person}|{Organization}|{Location})
-->
{
  if (inputAS.get("Person").size() > 200) {
    doc.getFeatures().put("runOrthomatcher","false");
  } else {
    doc.getFeatures().put("runOrthomatcher","true");
  }
}
```

Section by Section Processing: the Segment Processing PR

What is it?

- PR which enables you to process labelled sections of a document independently, one at a time
- Useful for
  - very large documents
  - when you want annotations in different sections to be independent of each other
  - when you only want to process certain sections within a document

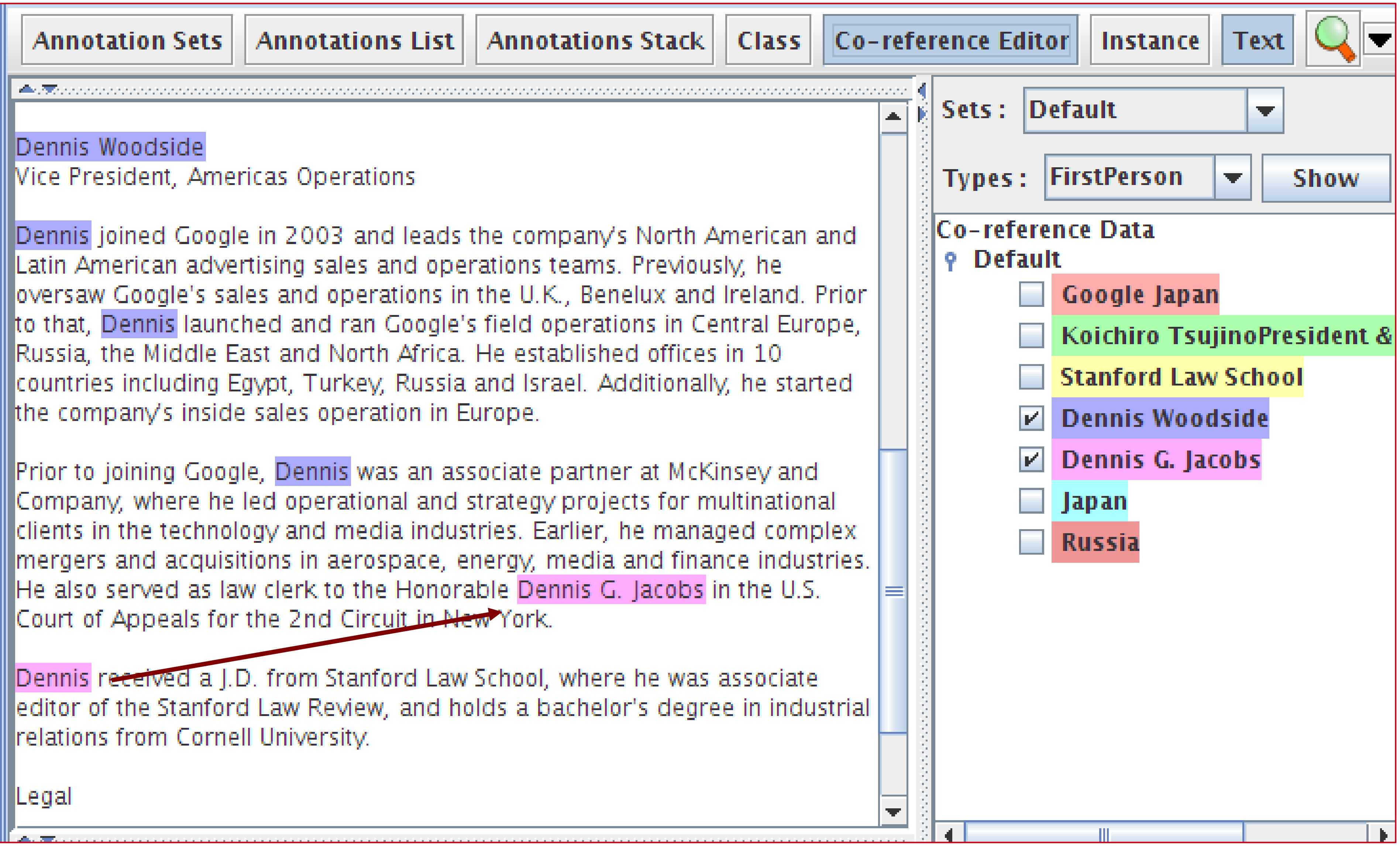
Processing large documents

- If you have a very large document, processing it may be very slow
- One solution is to chop it up into smaller documents and process each one separately, using a datastore to avoid keeping all the documents in memory at once
- But this means you then need to merge all the documents back afterwards
- The Segment Processing PR does this all in one go, by processing each labelled section separately
- This is quicker than processing the whole document in one go, because storing a lot of annotations (even if they are not being accessed) slows down the processing

Processing Sections Independently

- Another problem with large documents can arise when you want to handle each section separately
- You may not want annotations to be co-referenced across sections, for instance if a web page has profiles of different people with similar names
- Using the Segment Processing PR enables you to handle each section separately, without breaking up the document
- It also enables you to use different PRs for each section, using a conditional controller
- For example, some documents may have sections in different languages

Problematic co-references



Dennis Woodside
Vice President, Americas Operations

Dennis joined Google in 2003 and leads the company's North American and Latin American advertising sales and operations teams. Previously, he oversaw Google's sales and operations in the U.K., Benelux and Ireland. Prior to that, Dennis launched and ran Google's field operations in Central Europe, Russia, the Middle East and North Africa. He established offices in 10 countries including Egypt, Turkey, Russia and Israel. Additionally, he started the company's inside sales operation in Europe.

Prior to joining Google, Dennis was an associate partner at McKinsey and Company, where he led operational and strategy projects for multinational clients in the technology and media industries. Earlier, he managed complex mergers and acquisitions in aerospace, energy, media and finance industries. He also served as law clerk to the Honorable Dennis G. Jacobs in the U.S. Court of Appeals for the 2nd Circuit in New York.

Dennis received a J.D. from Stanford Law School, where he was associate editor of the Stanford Law Review, and holds a bachelor's degree in industrial relations from Cornell University.

Legal

Sets : Default
Types : FirstPerson   Show

Co-reference Data
Default
- [ ] Google Japan
- [ ] Koichiro TsujinoPresident &
- [ ] Stanford Law School
- [x] Dennis Woodside
- [x] Dennis G. Jacobs
- [ ] Japan
- [ ] Russia

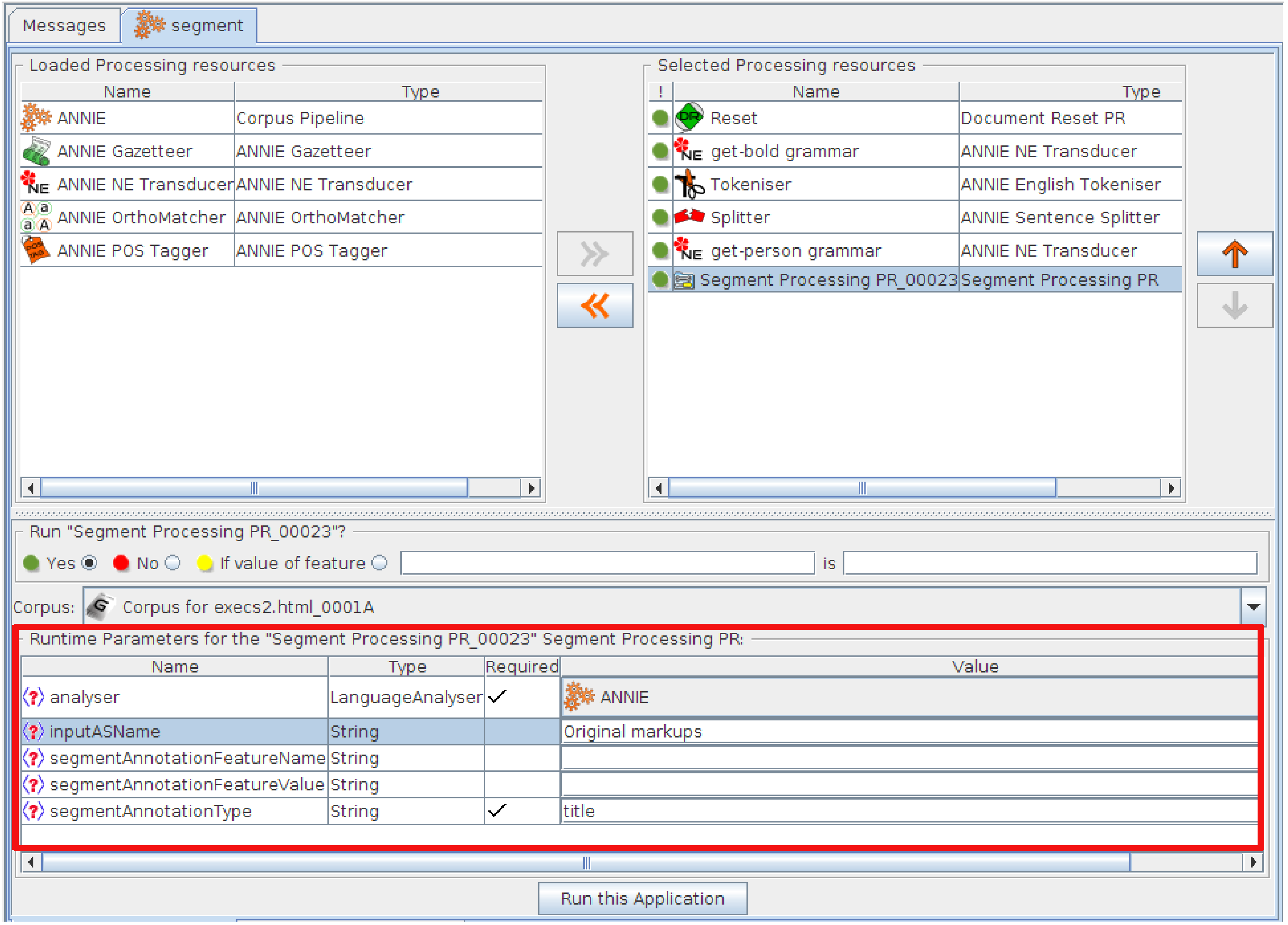
Getting rid of the junk

- Another very common problem is that some documents contain lots of "junk" that you don't want to process, e.g. HTML files contain javascript or contents lists, footers etc.
- There are a number of ways in which you can do this: you may need to experiment to find the best solution for each case
  - **Segment Processing** PR enables you to only process the section(s) you are interested in and ignore the junk
  - Using the **AnnotationSetTransfer** PR, though this works slightly differently
  - Using the **Boilerpipe** PR - this works best for ignoring standard kinds of boilerplate

How does it work?

- The PR is part of the Alignment Plugin
- A new application needs to be created, containing the Segment PR
- The PR then takes as one of its parameters, an instance of the application that you want to run on the document (e.g. ANNIE)
- You can add other PRs before or after the Segment PR, if you want them to run over the whole document rather than the specified section(s)

Running ANNIE on a title segment



Loaded Processing resources

Name	Type
ANNIE	Corpus Pipeline
ANNIE Gazetteer	ANNIE Gazetteer
ANNIE NE Transducer	ANNIE NE Transducer
ANNIE OrthoMatcher	ANNIE OrthoMatcher
ANNIE POS Tagger	ANNIE POS Tagger

Selected Processing resources

!	Name	Type
●	Reset	Document Reset PR
●	get-bold grammar	ANNIE NE Transducer
●	Tokeniser	ANNIE English Tokeniser
●	Splitter	ANNIE Sentence Splitter
●	get-person grammar	ANNIE NE Transducer
●	Segment Processing PR_00023	Segment Processing PR

Run "Segment Processing PR_00023"?

Yes [✓] No If value of feature ___ is ___

Corpus: Corpus for execs2.html_0001A

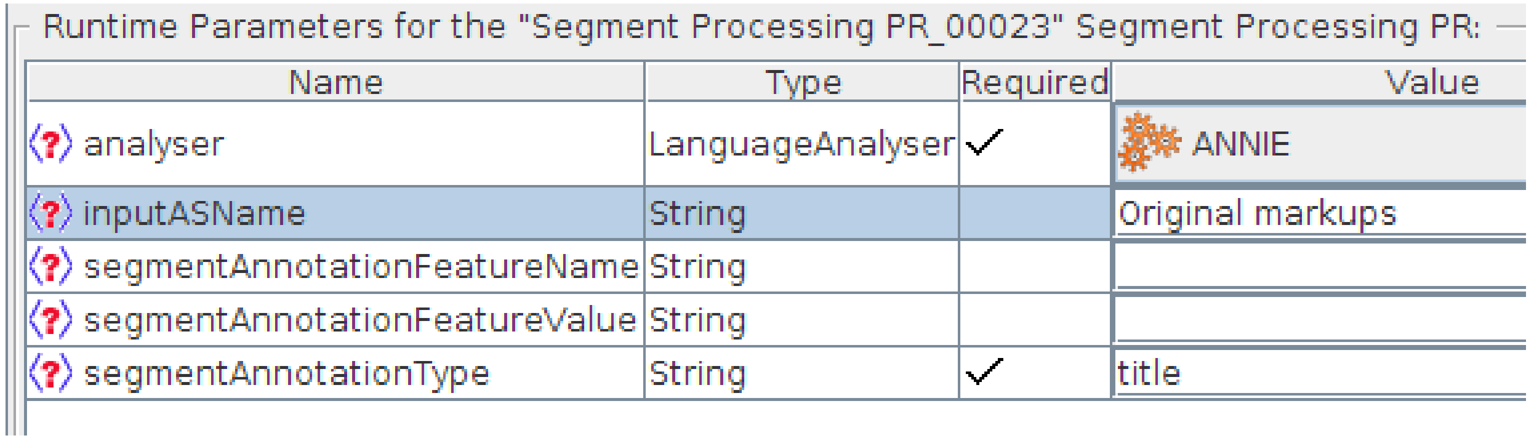
Runtime Parameters for the "Segment Processing PR_00023" Segment Processing PR:

Name	Type	Required	Value
analyser	LanguageAnalyser	✓	ANNIE
inputASName	String		Original markups
segmentAnnotationFeatureName	String		
segmentAnnotationFeatureValue	String		
segmentAnnotationType	String	✓	title

- Application contains a Segment Processing PR

- Segment Processing PR calls ANNIE application

Segment Processing Parameters

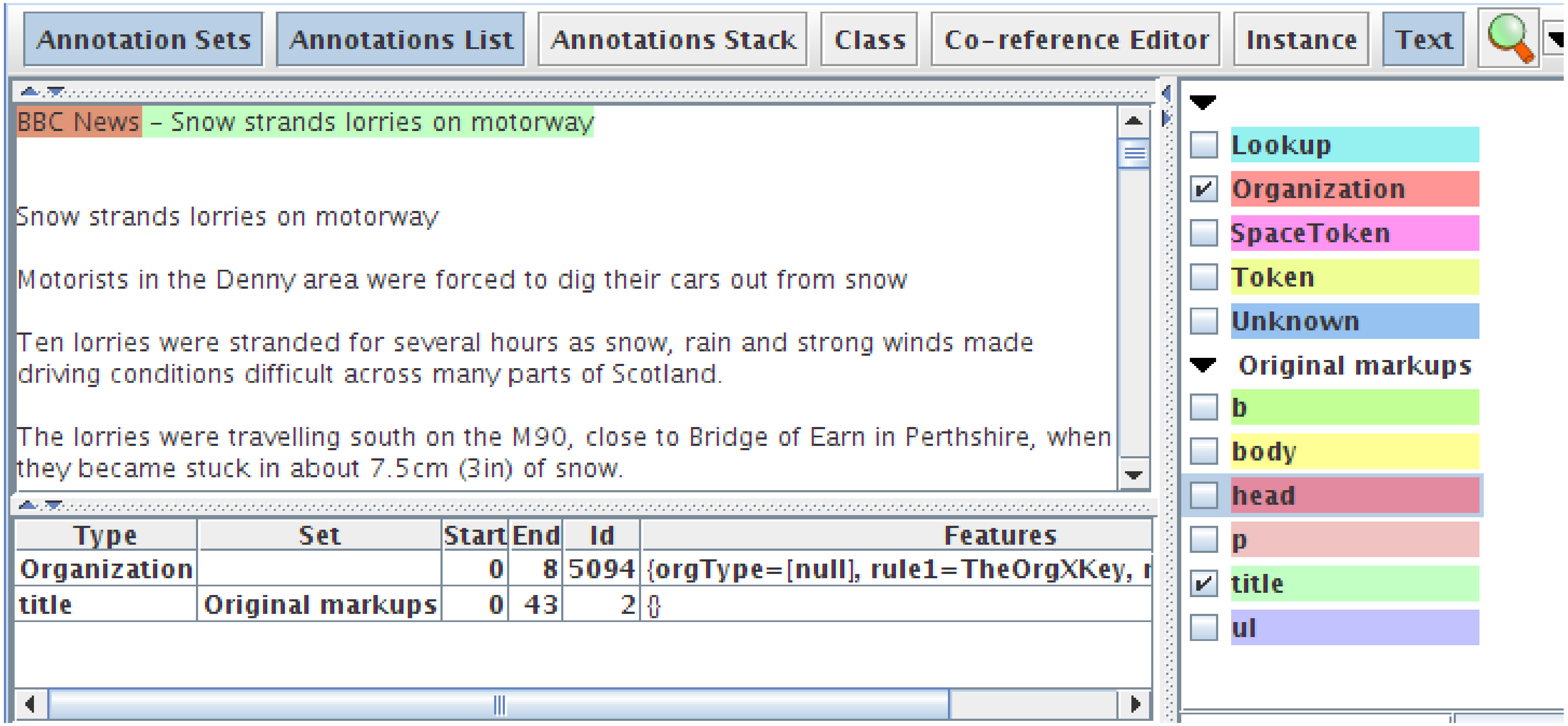

Runtime Parameters for the "Segment Processing PR_00023" Segment Processing PR:

Name	Type	Required	Value
analyser	LanguageAnalyser	✓	ANNIE
inputASName	String		Original markups
segmentAnnotationFeatureName	String		
segmentAnnotationFeatureValue	String		
segmentAnnotationType	String	✓	title

- Segment Processing PR calls the ANNIE application

- ANNIE is set to run only on the text covered by the span of the “title” annotation in the Original markups annotation set

Annotation Result



BBC News – Snow strands lorries on motorway

Snow strands lorries on motorway

Motorists in the Denny area were forced to dig their cars out from snow

Ten lorries were stranded for several hours as snow, rain and strong winds made driving conditions difficult across many parts of Scotland.

The lorries were travelling south on the M90, close to Bridge of Earn in Perthshire, when they became stuck in about 7.5cm (3in) of snow.

Type	Set	Start	End	Id	Features
Organization		0	8	5094	{orgType=[null], rule1=TheOrgXKey, r
title	Original markups	0	43	2	{}

- ☐ Lookup
- ✓ Organization
- ☐ SpaceToken
- ☐ Token
- ☐ Unknown

Original markups
- ☐ b
- ☐ body
- ☐ head
- ☐ p
- ✓ title
- ☐ ul

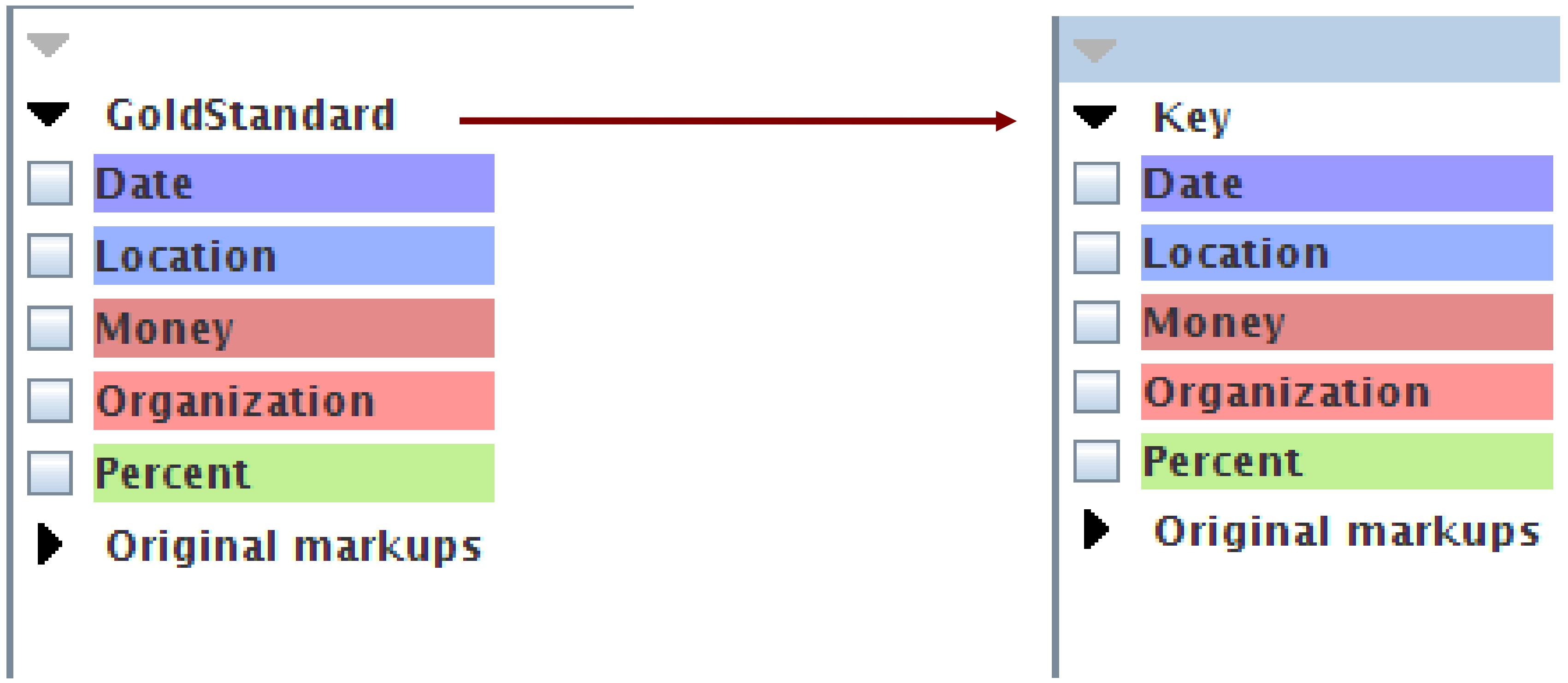
- Green shading shows the title, which spans the section to be annotated
- The only NE found is the Organization “BBC News” in the title
- Tokens in the rest of the document are not annotated

Using multiple annotation sets

Annotation Set Transfer

- This PR enables copying or moving annotations from one set to another
- As with the Segment Processing PR, you can specify a covering annotation to delimit the section you're interested in
- One use for this is to change annotation set names or to move results into a new set, without rerunning the application
- For example, you might want to move all the gold standard annotations from Default to Key annotation set

Transferring annotations



The annotations remain the same, they're just stored in a different set

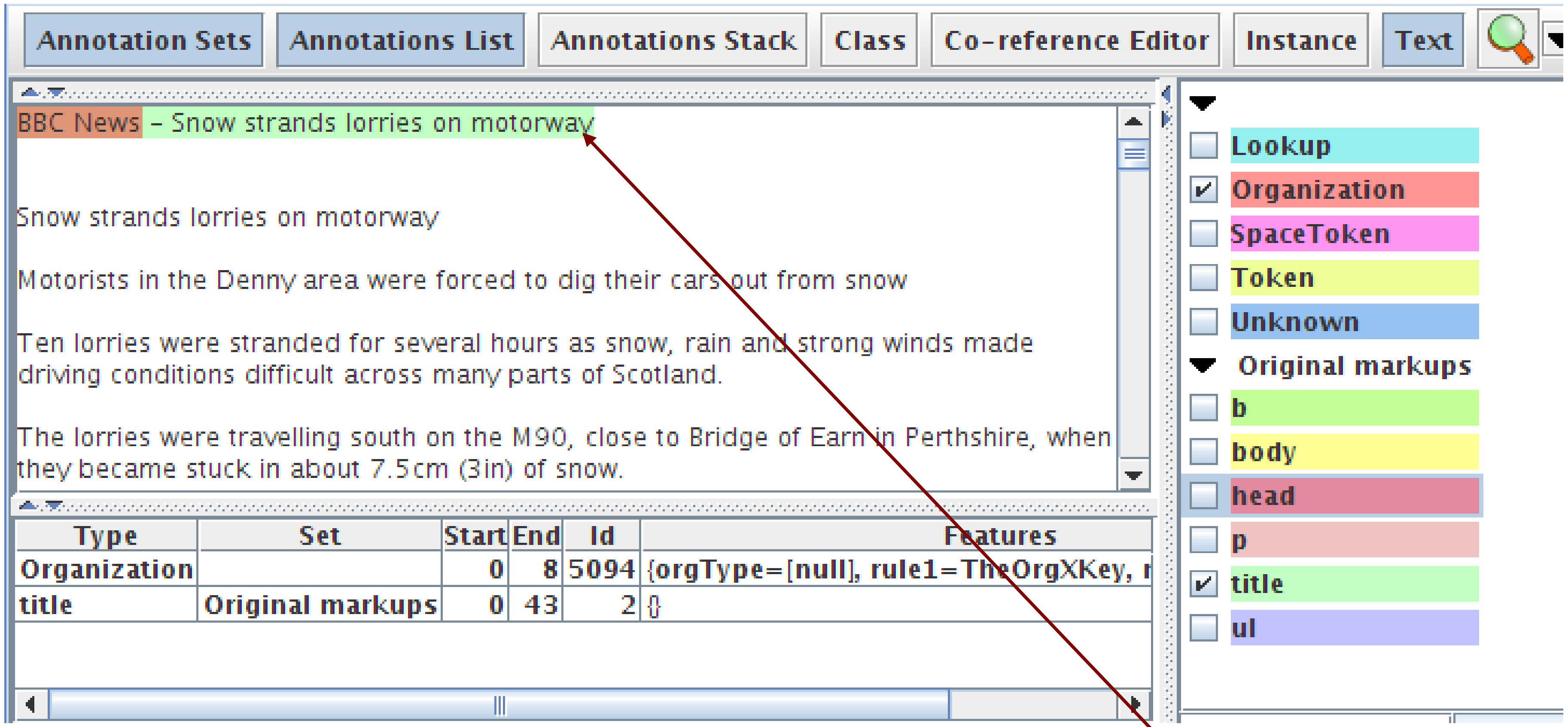
Hands-on Exercise

- Objective: move all the annotations from the Default set into the Key set
- Clear GATE of all previous documents, corpora, applications and PRs
- Load document self-shearing-sheep-marked.xml and create an instance of an AST (you may need to load the Tools plugin)
- Have a look at the annotations in the document
- Add the AST to a new application and set the parameters to move all annotations from the default AS to Key
- Make sure you don't leave the originals in the default AS!
- Run the application and check the results

Delimiting a section of text

- Another use is to delimit only a certain section of text in which to run further PRs over
- Unlike with the Segment Processing PR, if we are dealing with multiple sections within a document, these will not be processed independently
- So co-references will still hold between different sections
- Also, those PRs which do not consider specific annotations as input (e.g. tokeniser and gazetteer), will run over the whole document regardless

Processing a single section



BBC News – Snow strands lorries on motorway

Snow strands lorries on motorway

Motorists in the Denny area were forced to dig their cars out from snow

Ten lorries were stranded for several hours as snow, rain and strong winds made driving conditions difficult across many parts of Scotland.

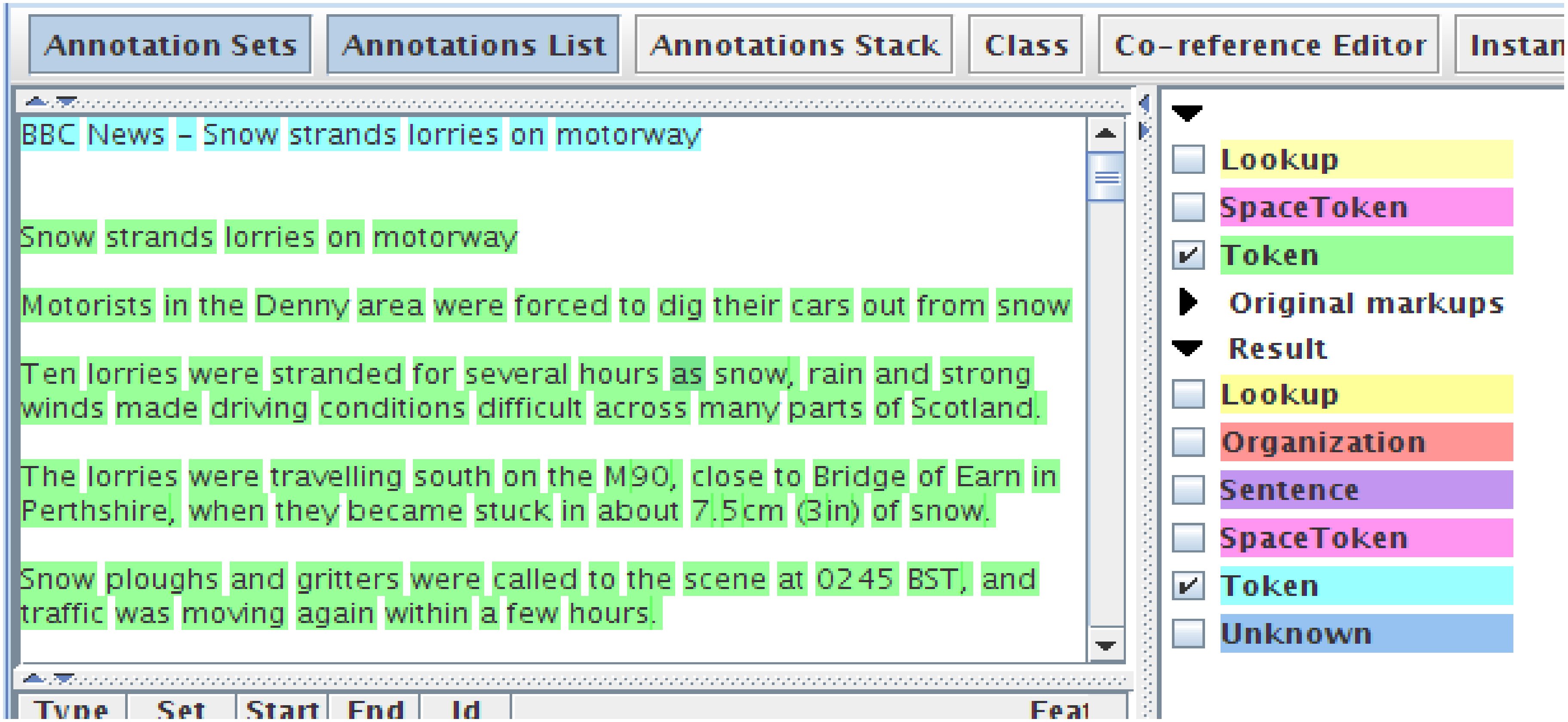
The lorries were travelling south on the M90, close to Bridge of Earn in Perthshire, when they became stuck in about 7.5cm (3in) of snow.

Type	Set	Start	End	Id	Features
Organization		0	8	5094	{orgType=[null], rule1=TheOrgXKey, r
title	Original markups	0	43	2	{}

- Only the “title” section is annotated with NEs

Transferring title annotations

- But the rest of the document remains tokenised
- These Tokens remain in the Default set because they weren't moved.



BBC News – Snow strands lorries on motorway

Snow strands lorries on motorway

Motorists in the Denny area were forced to dig their cars out from snow

Ten lorries were stranded for several hours as snow, rain and strong winds made driving conditions difficult across many parts of Scotland.

The lorries were travelling south on the M90, close to Bridge of Earn in Perthshire, when they became stuck in about 7.5cm (3in) of snow.

Snow ploughs and gritters were called to the scene at 0245 BST, and traffic was moving again within a few hours.

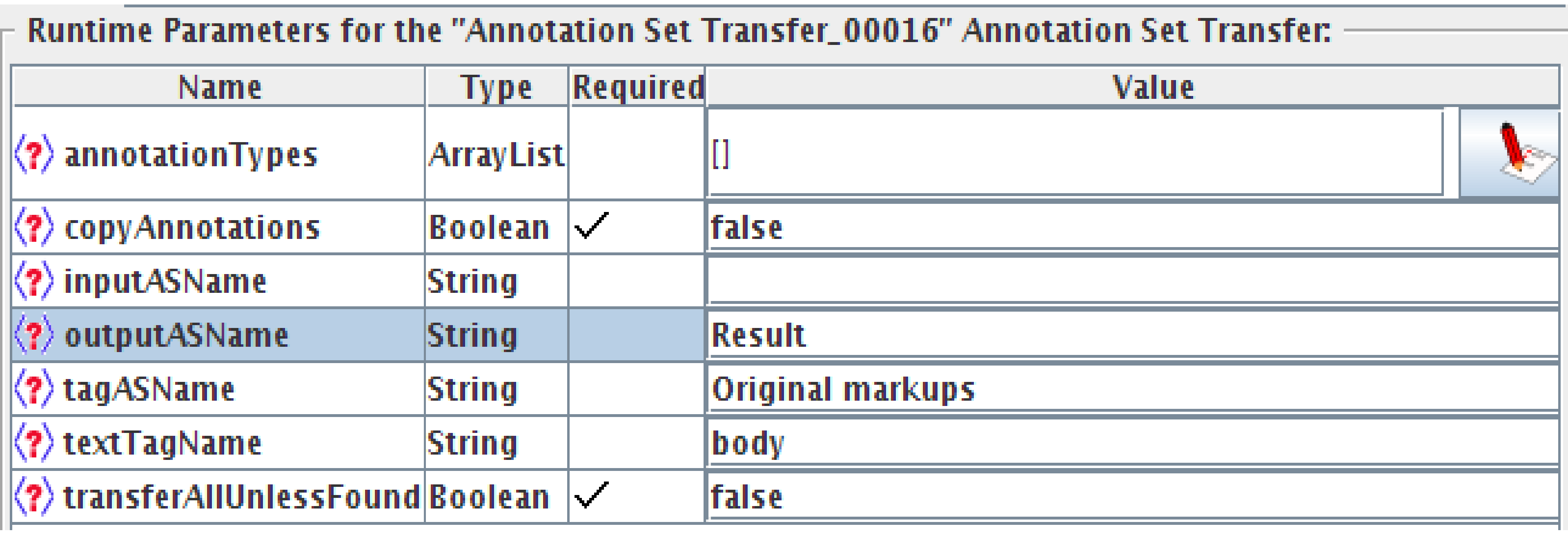
Setting the parameters

- Let's assume we want to process only those annotations covered by the HTML “body” annotation (i.e. we don't want to process the headers etc.).
- We'll put our final annotations in the “Result” set.
- We need to specify as parameters
  - **textTagName**: the name of the covering annotation: “body”
  - **tagASname**: the annotation set where we find this: “Original markups”
  - **annotationTypes**: which annotations we want to transfer
  - **inputASname**: which annotation set we want to transfer them from: “Default”
  - **outputASname**: which annotation set we want to transfer them into: “Result”

Additional options

- There are two additional options you can choose
  - **copyAnnotations**: whether to copy or move the annotations (i.e. keep the originals or delete them)
  - **transferAllUnlessFound**: if the covering annotation is not found, just transfer all annotations. This is a useful option if you just want to transfer all annotations in a document without worrying about a covering annotation.

Parameter settings


Runtime Parameters for the "Annotation Set Transfer_00016" Annotation Set Transfer:

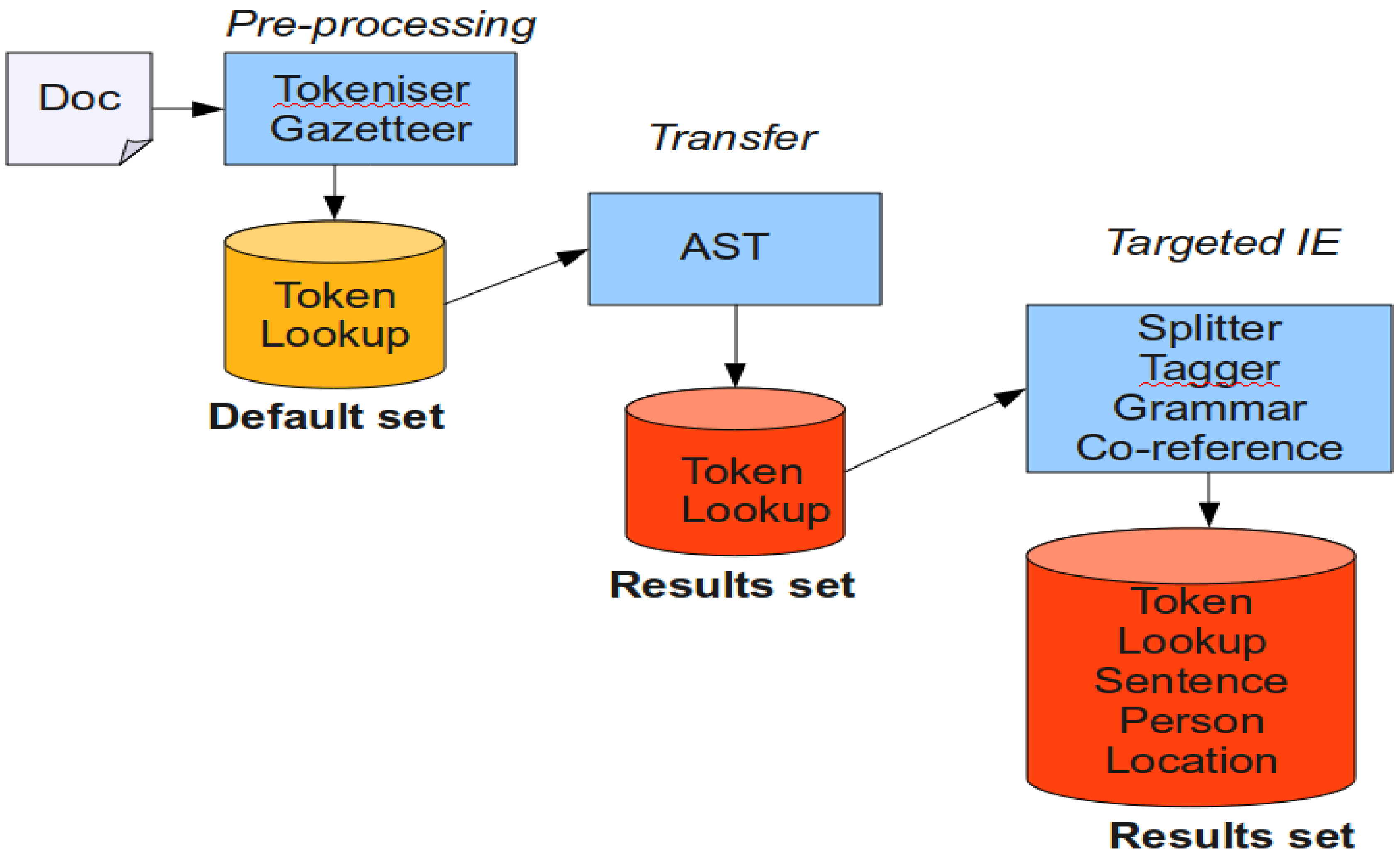
Name	Type	Required	Value
annotationTypes	ArrayList		[]
copyAnnotations	Boolean	✓	false
inputASName	String		
outputASName	String		Result
tagASName	String		Original markups
textTagName	String		body
transferAllUnlessFound	Boolean	✓	false

- Move all annotations contained within the “body” annotation (found in the Original markups set), from the Default set to the Result set.
- If no “body” annotation is found, do nothing.

Using it within an application

- We want to run ANNIE over only the text contained within the “title” text
- Apart from the tokeniser and gazetteer, the other ANNIE PRs all rely on previous annotations (Token, Lookup, Sentence, etc.)
- We run the tokeniser and gazetteer first on the whole document
- Then use the AST to transfer all relevant Token and Lookup annotations into the new set
- Then we can run the rest of the ANNIE PRs on these annotations
- To do this, we use for inputAS and outputAS the name of the new set “Result”

Application architecture



Hands-on: processing a document section

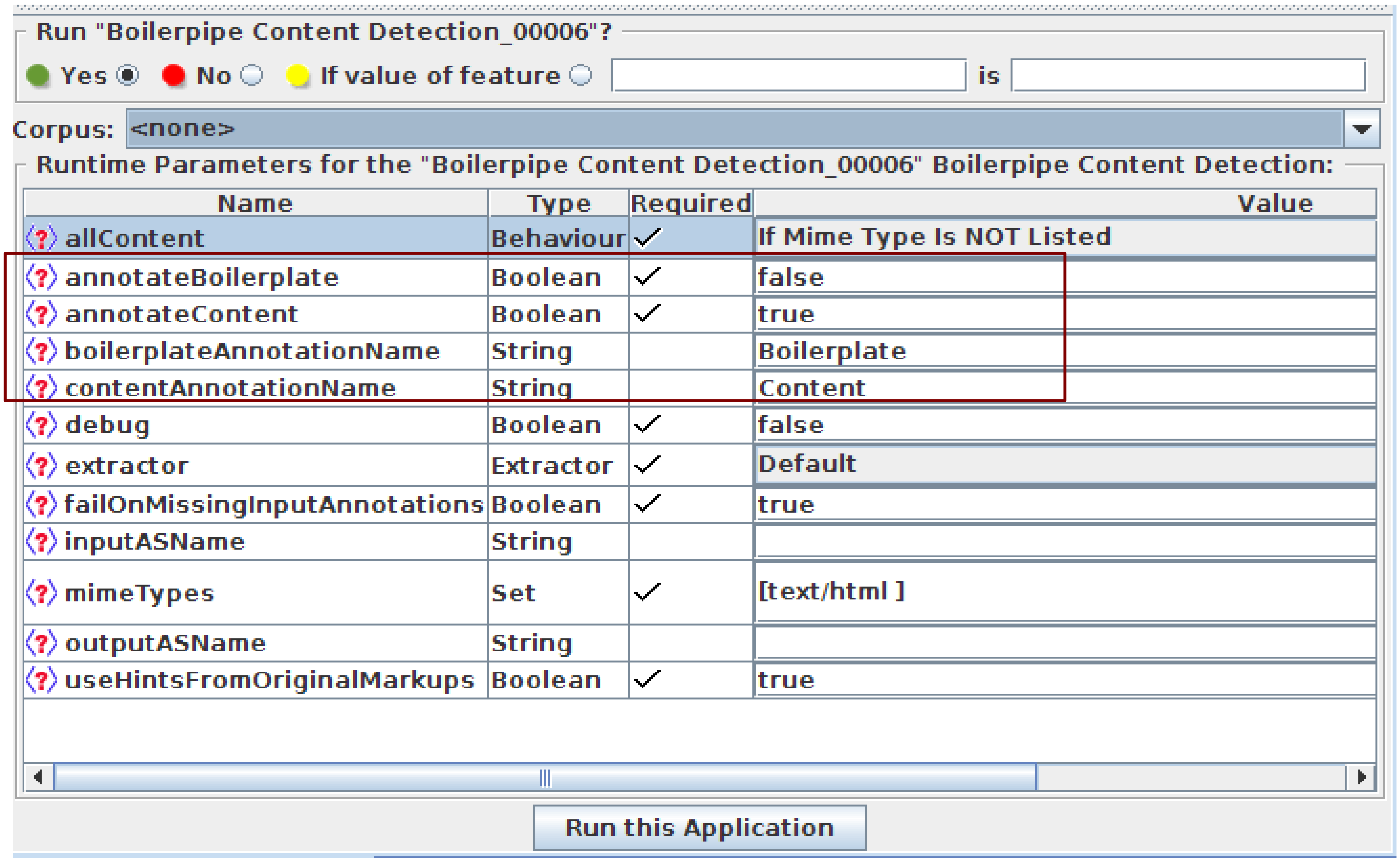
- We will modify ANNIE to only run over the title of the document
- Close old applications, PRs, and LRs
- Load the document cricket.html and create a corpus with it
- Load ANNIE
- Add an AST PR immediately after the sentence splitter
- Set the AST parameters to move all annotations contained within the “title” annotation (found in the Original markups set), from the default set to the Result set. If you get stuck, check the slide “Setting the Parameters”
- Modify the input and output sets of all <u>following</u> PRs to “Result”
- Run on the corpus and inspect the results
- Now try moving the AST to earlier place in the application—what happens when you run it?

Content Detection using Boilerpipe

What is the Boilerpipe PR?

- In a closed domain, you can often write some JAPE rules to separate real document content from headers, footers, menus etc.
- In many cases, or when dealing with texts of different kinds or in different formats, it can get much trickier
- Boilerpipe PR provides algorithms to separate the surplus “clutter” (boilerplate, templates) from the main textual content of a web page.
- Applies the Boilerpipe Library to a GATE document in order to annotate the content, the boilerpipe, or both.
- Due to the way in which the library works, not all features from the library are currently available through the GATE PR

Boilerpipe Parameters



Run "Boilerpipe Content Detection_00006"?

● [✓ Yes] ● No ● If value of feature ___ is ___

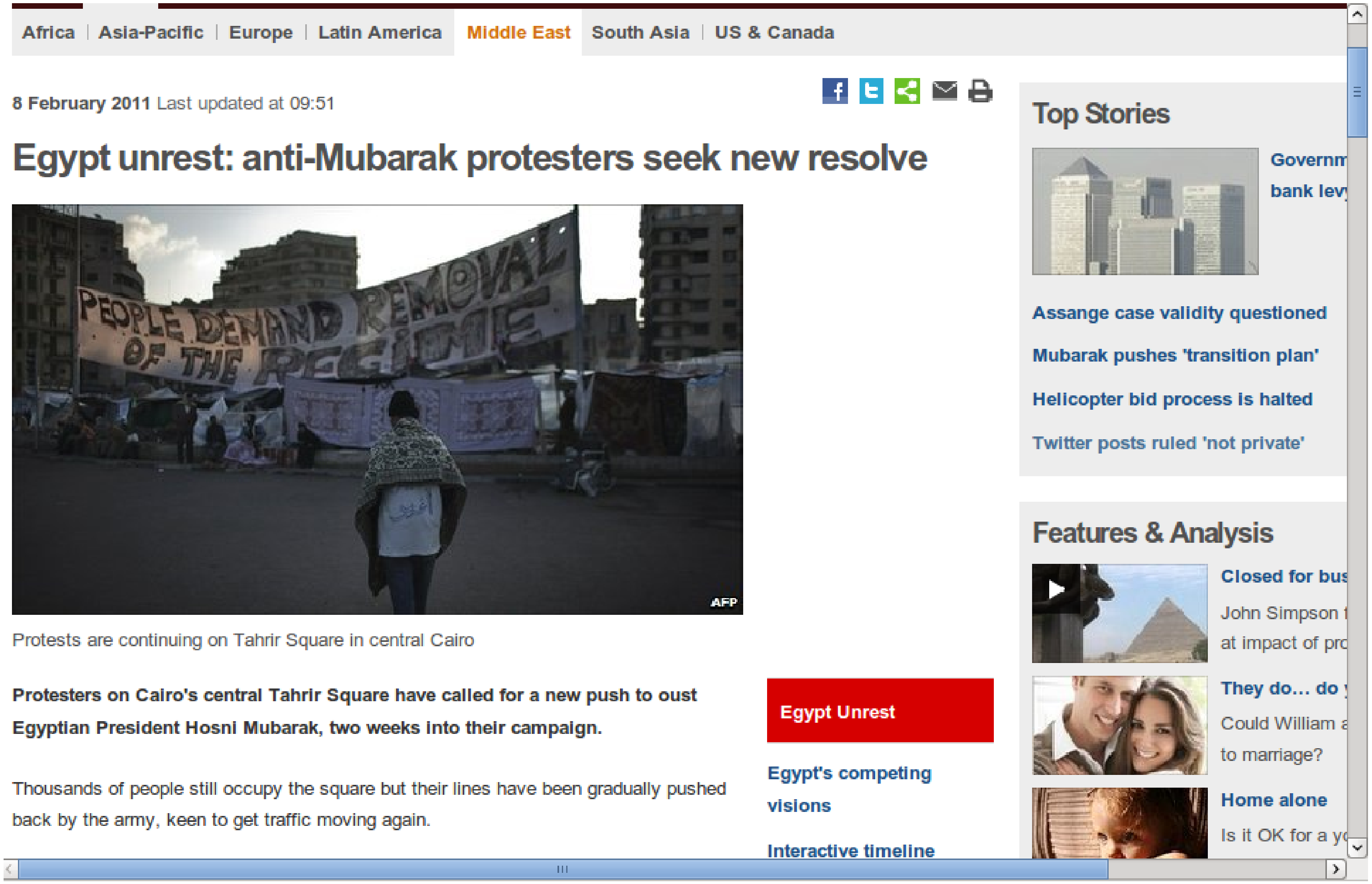
Corpus: <none>

Runtime Parameters for the "Boilerpipe Content Detection_00006" Boilerpipe Content Detection:

Name	Type	Required	Value
allContent	Behaviour	✓	If Mime Type Is NOT Listed
annotateBoilerplate	Boolean	✓	false
annotateContent	Boolean	✓	true
boilerplateAnnotationName	String		Boilerplate
contentAnnotationName	String		Content
debug	Boolean	✓	false
extractor	Extractor	✓	Default
failOnMissingInputAnnotations	Boolean	✓	true
inputASName	String		
mimeTypes	Set	✓	[text/html]
outputASName	String		
useHintsFromOriginalMarkups	Boolean	✓	true

Run this Application

Original HTML document



Africa | Asia-Pacific | Europe | Latin America | Middle East | South Asia | US & Canada

8 February 2011 Last updated at 09:51

Egypt unrest: anti-Mubarak protesters seek new resolve

Protests are continuing on Tahrir Square in central Cairo

Protesters on Cairo's central Tahrir Square have called for a new push to oust Egyptian President Hosni Mubarak, two weeks into their campaign.

Thousands of people still occupy the square but their lines have been gradually pushed back by the army, keen to get traffic moving again.

Egypt Unrest

Egypt's competing visions

Interactive timeline

Top Stories

Governm
bank levy

Assange case validity questioned

Mubarak pushes 'transition plan'

Helicopter bid process is halted

Twitter posts ruled 'not private'

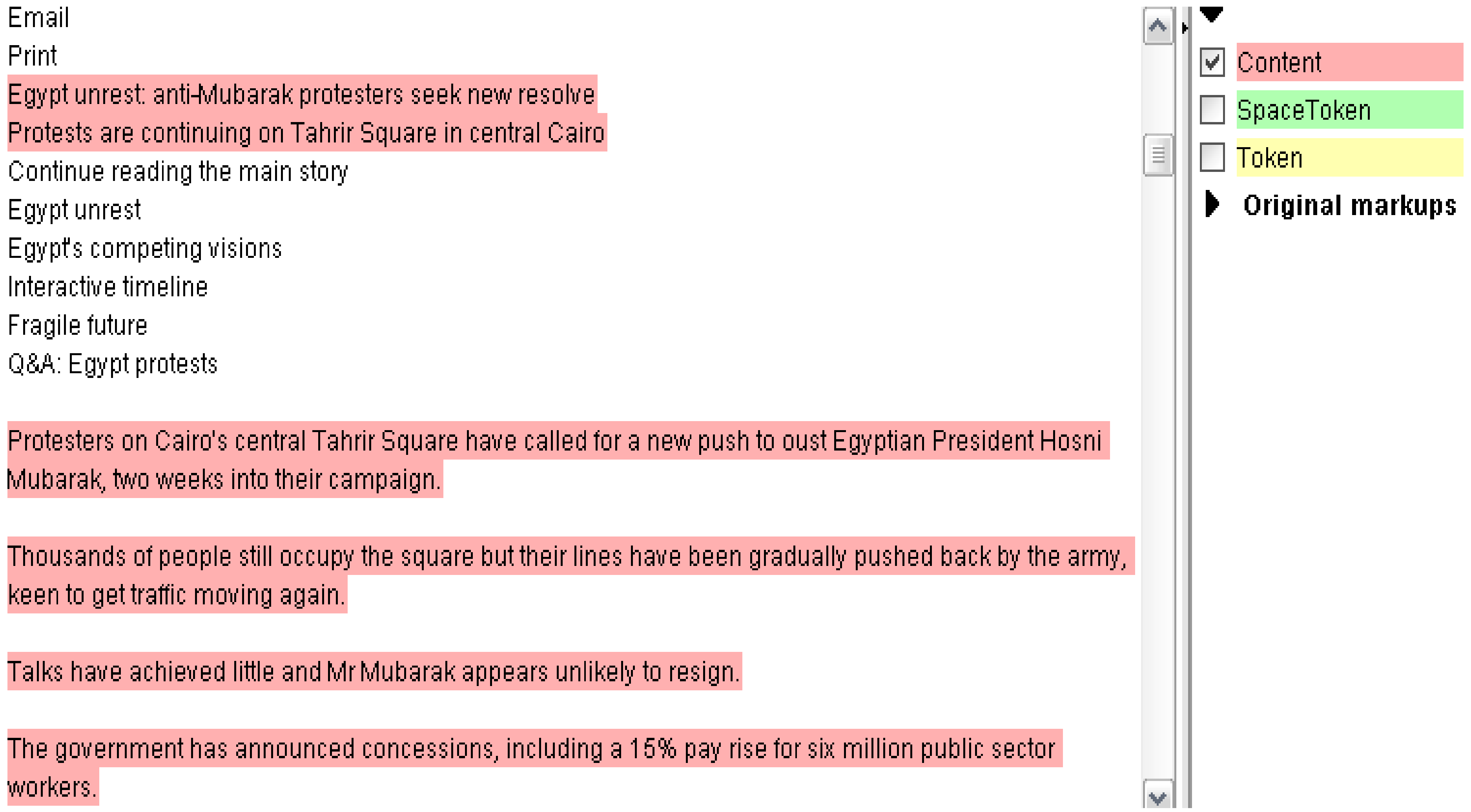
Features & Analysis

Closed for bus
John Simpson
at impact of pro

They do... do
Could William a
to marriage?

Home alone
Is it OK for a yo

Processed Document



Email
Print
Egypt unrest: anti-Mubarak protesters seek new resolve
Protests are continuing on Tahrir Square in central Cairo
Continue reading the main story
Egypt unrest
Egypt's competing visions
Interactive timeline
Fragile future
Q&A: Egypt protests

Protesters on Cairo's central Tahrir Square have called for a new push to oust Egyptian President Hosni Mubarak, two weeks into their campaign.

Thousands of people still occupy the square but their lines have been gradually pushed back by the army, keen to get traffic moving again.

Talks have achieved little and Mr Mubarak appears unlikely to resign.

The government has announced concessions, including a 15% pay rise for six million public sector workers.

Try it yourself

- Load the “Tagger: Boilerpipe” and ANNIE plugins
- Create a Boilerpipe Content Detection PR
- Create a new application, and add to it a Document Reset, a Tokeniser, and the Boilerpipe PR
- Leave all the parameters as default
- Load a document, e.g. the document cricket.html from your hands-on material, or a page from http://bbc.co.uk/news, and add to a corpus
- Run the application and examine the “Content” annotations on the document (in the default set)
- Change the annotateBoilerplate parameter from false to true; rerun the application; examine the “Boilerplate” annotations

Schema Enforcer

- When creating an application, you often end up with lots of annotations and features which are not needed in the final output
- If pushing the output into a MIMIR index, or if storage space is an issue, it's particularly important to get rid of these
- You can tidy up the output using the AnnotationSetTransfer PR to move selected annotation types to a new set, but there's still the problem of the features
- Schema Enforcer PR will ensure that annotations and features will only appear in the final output set if they adhere strictly to the annotation schemas used
- Straightforward to use - load Schema Tools plugin and just list the schemas to be used in the runtime parameters (they must be loaded in GATE already)

Modular Pipelines

- With a normal application (corpus pipeline) you can load other applications as sub-components, as we have seen
- The problem with this is that when you make changes to any of these sub-components and then save your main application, the original application is not saved.
- So if you want to use these sub-components separately, you have to remember to save separately any changes to them.
- The modular pipelines method gets round this by saving the individual applications separately.
- “Modular Pipelines” is now included in the GATE plugin manager.

How to use it

- Load the Modular Pipelines plugin
- Create a new Parametrized Corpus Controller from the Applications menu
- Load an application (sub-pipeline) by creating a new Processing Resource of type “Pipeline”
- Select a .gapp file as the value of PipelineFileURL in the loadtime parameters
- This will load the application into GATE
- Add the pipeline to your Parametrized Corpus Controller application
- Add more sub-pipelines or PRs as you wish

The Groovy PR

Groovy Scripting PR

- Groovy is a dynamic programming language based on Java.
  - http://groovy.codehaus.org/
- The GATE Groovy plugin provides a powerful scripting PR that can be included in a corpus pipeline and run over each document.
- The script has full access to the current document and corpus through the GATE API, like a Java JAPE RHS but more powerful
- Unlike a JAPE Transducer, this PR does not have to match anything in the document in order to “fire the rules”

Groovy Scripting PR

- Two init parameters:
  - **scriptURL**, the path to the script
  - **encoding** (default UTF-8)
- Once the PR is created, the path to the file cannot be changed
- Just like JAPE, you can edit the file outside of GATE, save it, and re-initialize the PR to reload the file (and get syntax error messages)
- Three runtime parameters:
  - **inputASName** and **outputASName** (annotation sets)
  - **scriptParams** (key-value pairs)

Groovy Scripting PR

- Inside the script, you get 6 automatic variables "free of charge":
  - **doc**, the current document (as in JAPE)
  - **corpus**, the current corpus
  - **content**, the string content of this document
  - **inputAS** and **outputAS**, the annotation sets for the current document named in the runtime parameters (as in JAPE)
  - **scriptParams**, a FeatureMap with the keys and values from the scriptParams runtime parameter, which lets you pass your own simple configuration options to the PR and change them from the pipeline interface without editing the script

Groovy Scripting PR

- What can you do with it?
  - Anything you can do in a JAPE Java RHS, and more
  - Read/write access to the document (features, content, all annotation sets)
  - Read/write access to the corpus (features, size, contents) but be careful
  - Control structures (loops, if then else, etc.)
  - No need to match a pattern of annotations
- Example: check each document for certain things and set its features accordingly
  - features can be used to regulate conditional PRs later in a conditional corpus pipeline, for example

Hands-on: Groovy Scripting PR

- Remove all existing documents, corpora, resources and applications from GATE
- Create a new corpus and populate it from **corpus/ft-corpus** in the hands-on materials
- Load the ANNIE application and the Groovy plugin
- Create a new Groovy Scripting PR from the file **groovy/Example.groovy** in the hands-on materials, and add it to end of the ANNIE pipeline.

Groovy Scripting PR

```
// Print the name of the current document
println doc.getName()

// Print the text underlying each Person annotation
inputAS["Person"].each{ anno ->
  println "   " + doc.stringFor(anno)
}

// Record the number of Person annotations
doc.getFeatures().put("nbr_persons",
   inputAS["Person"].size())

// Flag whether the document contains any Person annotations;
// this feature can be used in a Conditional Corpus Pipeline.
doc.getFeatures().put("has_persons", !
   inputAS["Person"].isEmpty())
```

What do you think this will do?

Groovy Scripting PR

- Run the pipeline and note the output in the Messages tab.
- Open a few documents, examine the document features, and compare them with the annotations in the default AS.

Groovy script to delimit parts of a document

```
// set the annotation type from an annotation in the document

intStart = inputAS["Interesting"].start()

intEnd = inputAS["Interesting"].end()


// add the new annotations

outputAS.add(doc.start(), doc.end(), "WholeDoc", Factory.newFeatureMap())

outputAS.add(doc.start(), intStart, "BeforeInt", Factory.newFeatureMap())

outputAS.add(intStart, doc.end(), "IntZone", Factory.newFeatureMap())

outputAS.add(intEnd, doc.end(), "AfterInt", Factory.newFeatureMap())
```

Try it out

- Load the JAPE grammar from the file grammar/get-interesting.jape and add it to the end of your ANNIE application
- You can remove or turn off the other Groovy PR if you want
- Create a new Groovy Scripting PR from the file **groovy/Interesting.groovy** in the hands-on materials, and add it after the new JAPE grammar you just added
- Load the document from the file corpus/report.pdf and add it to a new corpus
- Run the application over this corpus and check the annotations
- Bonus points: change your application so that ANNIE only runs over the IntZone part of the corpus
- More bonus points: change the groovy script to get the annotation type from the script parameters

Putting it all together

- You can combine ideas from all these topics (and more) when creating your applications
- Here's a real example of an IE application we created for a project on business intelligence
- It involved different kinds of HTML texts, which required different processing methods
- As you will see, it's important to keep calm and do things one step at a time
- When you have complex applications like this, keeping things in separate annotation sets (and removing unwanted annotations) can help you keep track of what's going on

Complex IE application

Pre-process all documents

Add document features depending on text type

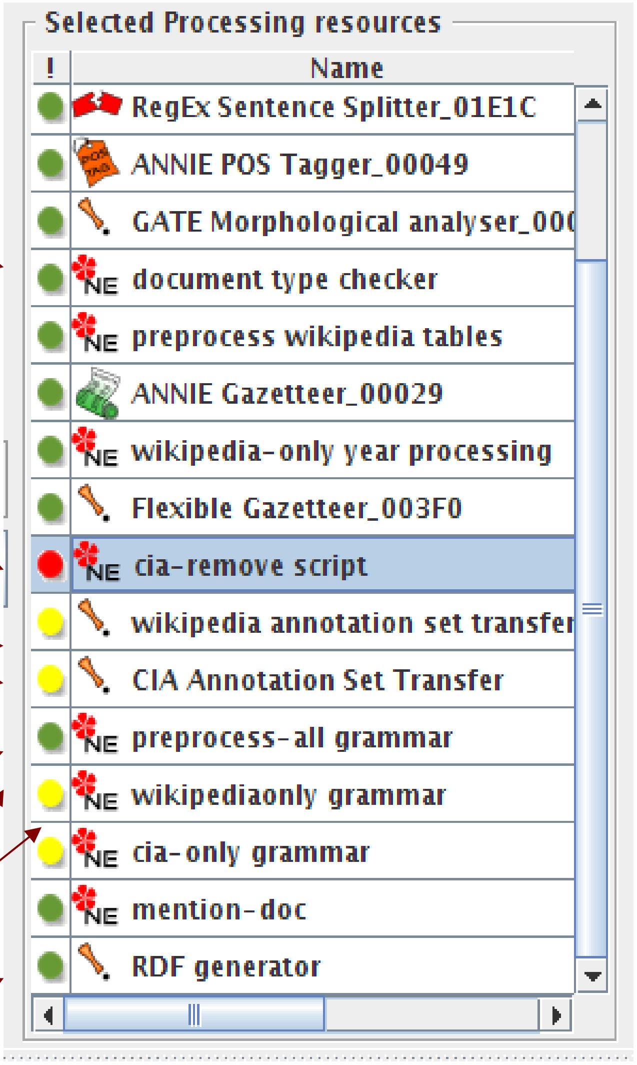
Tried this grammar out, but didn't use it ultimately

For each text type, copy the pre-processing annotations from the relevant section to a new annotation set

Pre-processing is same for all document types

Run a text-specific grammar on the documents

Do something with the results of all documents



Selected Processing resources

!	Name
●	RegEx Sentence Splitter_01E1C
●	ANNIE POS Tagger_00049
●	GATE Morphological analyser_00
●	document type checker
●	preprocess wikipedia tables
●	ANNIE Gazetteer_00029
●	wikipedia-only year processing
●	Flexible Gazetteer_003F0
🔴	cia-remove script
🟡	wikipedia annotation set transfer
🟡	CIA Annotation Set Transfer
●	preprocess-all grammar
🟡	wikipediaonly grammar
🟡	cia-only grammar
●	mention-doc
●	RDF generator

Everybody loves spreadsheets (input)

- GATE has built-in support for loading Microsoft Office and LibreOffice spreadsheets
- They are internally converted to HTML, so the “Original markups” AS has **tr** (table row) and **td** (table data) annotations marking the rows and cells
- Open a few of your favourite spreadsheets in GATE to see how they are processed

Everybody loves spreadsheets (output)

- The Configurable Exporter (in the Tools plugin) produces CSV or TSV output with a row for each matched annotation (or document) and a cell for each value—suitable for importing into spreadsheets
- Create a corpus and populate it from corpus/ft-corpus
- Load the application exporter/annie+tsv.gapp
- The exporter has one init parameter for the config file (re-initialize the PR to reload a changed file)
- It has runtime parameters for the input AS, the instance (annotation to generate one line of output), and output file location

Everybody loves spreadsheets (output)

- Look at the config file exporter tsv.conf
- The annotation types in the config file usually match the instance parameter, but they don’t have to
- Specifying an annotation type without a feature prints the text covered
- The groovy PR shows an example of copying information from elsewhere onto the instance annotations
- Set the output file, run the application, and examine the output
- If you leave the output file unset, output is printed in “Messages”
- TSV is usually the easiest format to generate consistently and load into a spreadsheet

Summary of this module

- You should now have some ideas about how to take ANNIE a step further and do more interesting things in GATE than just IE on English news texts.
- Porting an IE system to a different language
- Processing multilingual corpora
- How to process “difficult” texts, e.g. keeping sections independent, processing only parts of a document, processing large documents.
- How to manipulate existing annotated documents
- This should enable you now to start building up more complex applications with confidence

Take-home message for today

- Don't be afraid to try weird and wonderful things in GATE!
- We do it all the time...sometimes it even works :-)

Extra hands-on for the super keen

- Some more exercises to try out!

Extra hands-on

- Modify ANNIE for a language of your choice, by adapting some gazetteer lists and adding some grammar rules
- If this isn't feasible for your language (e.g. Chinese) then just make an application with some simple gazetteer lists for your language and some rules which convert the Lookups into annotations
- Create a small corpus containing a combination of documents in your chosen language and in English
- Create an application that processes the documents separately but which merges the results from both into a single final annotation set
- Use any method you like to only annotate certain parts of those documents

Extra hands-on segment processing (1)

- Clear GATE of all PRs, applications and resources
- Load the application segment-processing.gapp
- Load the document execs2.html and add it to a corpus
- Run the application on the corpus
- This application first creates an annotation type “bold” in the default annotation set, using the “b” annotations from the Original markups set.
- Have a look at the grammar get-bold.jape and the runtime parameters for it to see how it was done.
- Then the application uses the get-person.jape grammar to match a bold annotation followed by a set of sentences, creating a new annotation “Content” in the default annotation set.
- Have a look at the “bold” and “Content” annotations in the document.

Hands-on segment processing (2)

- Now we have our document separated into sections with the Content annotations.
- Load ANNIE. Remove the Document Reset, Tokeniser and Sentence Splitter from it (make sure you remove the ones named ANNIE Tokeniser, etc. and not the ones previously loaded) and change the “failOnMissingInputAnnotations” parameter of the POS Tagger to false.
- Create a Segment Processing PR and add it to the end of your Segment application.
- Select the Segment Processing PR in the application and set the “analyser” value to “ANNIE”
- Set the value of “segmentAnnotationType” to “Content”
- Run the application and look at the results
- Look at the co-references created: they should not cross Content boundaries. Look at “Google” annotations for an example.