Quick Guide for CTAP Language Extensions

March 2019, by Zarah Weiss

Important Links

This document only elaborates on how to extend CTAP to other languages. For further documentation on how to use CTAP, how to set it up locally or on a server, and where to find an example system please consult the following additional sources:

- Explanation videos how to run CTAP locally and on a server: https://www.youtube.com/playlist?list=PLbyPrDZ9AZI9auLKdQMs7ifiqskEwq_k9&feature=em-share_playlist_user
- Explanation video on how to use CTAP: https://www.youtube.com/watch?v=wVHo0hzPUyU&feature=youtu.be
- Tübingen CTAP version: http://samos.sfs.uni-tuebingen.de:8080/ctapml/

Additional documentation may be found at in the git repository at https://github.com/zweiss/multilingual-ctap-feature/tree/ctap_german/src/main/resources/doc/

Frontend

A. Include Language in Feature Selecton Buttons in Front-End

To add the language selection button to the left and right panel in the Feature Set Manager

1. **FeatureSetManager.java**: Create method parallel to onSelectGermanClick(ClickEvent e)
2. **FeatureSetManager.java**: Create method parallel to onSelectGermanRightPanelClick(ClickEvent e)
3. **FeatureSetManager.ui.xml**: Create new anchor parallel to

   ```
   <li><g:Anchor ui:field="selectGerman">German</g:Anchor></li>
   ```

4. **FeatureSetManager.ui.xml**: Create new anchor parallel to

   ```
   <li><g:Anchor ui:field="selectGermanRightPanel">German</g:Anchor></li>
   ```

B. Include Language in Analysis Generator

1. **AnalysisGenerator.ui.xml**: Add list item to language form group parallel to

```
<g:item value="DE">German</g:item>
```

2. **Analysis.java**: Include an abbreviation string for your new language in AnalysisLanguageOptions parallel to

```
public static String GERMAN = "DE";
```

Backend

- **SupportedLanguages.java**: Add new supported language string to the set of supported languages parallel to

```
public static final String GERMAN = "DE";
```

A. Feature exists, no changes needed

1. Go to the XML Descriptor of the existing feature and add the supported language string to the list of supported languages

```
<languagesSupported>
    <language>EN</language>
    <language>DE</language>
    <!-- add new language here -->
</languagesSupported>
```

2. Check if the Java class for the feature makes any assumption in process() that does not apply to the new language or requires additional language specific information in initialize()

3. If the feature is based on an NLP annotator, also open die NLP annotator XML descriptor file and add the new language to the list of supported languages as shown in step 1.

4. Adjust the description tag in the XML descriptor if necessary

Example: NLetterAE.java; NLetterAE.xml

B. Feature exists, NLP exists, new model needed

1. Proceed as described in A.
2. In the NLP annotator XML descriptor file, add an entry for an external resource dependency for the new language as follows

```
<externalResourceDependency>
    <key>POSModelDE</key>
    <description>A German POS model to pos-tagging text.</description>
    <optional>false</optional>
</externalResourceDependency>
```

3. Add a new resource manager configuration to the NLP annotator XML descriptor file parallel to

```
<externalResource>
    <name>dePOSModel</name>
    <description>German POS model.</descripion>
    <fileResourceSpecifier>
        <fileUrl>file:model/de_pos_maxent.bin</fileUrl>
    </fileResourceSpecifier>
</externalResource>
```

4. Add the referenced model to the model/ folder referenced in Step 3

5. Link the resource dependency with the resource manager configuration in an external resource binding parallel to

```
<externalResourceBinding>
    <key>POSModelDE</key>
    <resourceName>dePOSModel</resourceName>
</externalResourceBinding>
```

6. If the change of model also results in the tool output, check if you need to make further adjustments. For example:

   - when switching POS tag models, the POS tags themselves change. In order for the feature based on these annotations to work correctly, you need to adjust the POS tag information in the respective feature descriptor XML file within the configuration parameter tag. Search for configurations ending in "DE" or "EN" to identify which changes might be necessary
   - in particular for POS based features you might want to define the POS tag mapping by extending the abstract class WordCategories.java parallel to GermanWordCategories.java and EnglishWordCategories.java.
   - similarly, when using a new dependency parse model, you will need to extend the abstract class DependencyLabelCategories.java parallel to GermanDependencyLabels.java

7. Adjust the description tag in the XML descriptor if necessary

C. Feature exists, NLP infrastructure exists, but different NLP API needed

1. Proceed as described in B as far as applicable
2. Check in the NLP Annotator Java file if it already contains an abstract wrapper class, such as, for

example, ConstituencyParser in ParseTreeAnotator.java
  - if yes, extend this class with your new NLP similar to the other tools and add the correct initalization in initialize()
  - if no, create an abstract class and write an extension for the existing NLP API as well as the new one; handle the initialization in initialize() similar to ParseTreeAnnotator.java

3. Make sure to perform Steps 6 and 7 from C

D. New feature based on existing NLP infrasturcture

1. Proceed as described in C as far as applicable (i.e. check the required NLP tools and models)
2. Create a new descriptor XML file and a new Java file in the respective featureAE folders
3. If possible, use a feature that functions similarly as a template. Make sure to use a unique annotatorImplementationName and to update the feature description tag.
    - if possible, implement it so that it also works for some or all of the other supported languages
    - if not (or when in doubt) make sure to only include the new language in the list of supported languages
    - in any case, implement the feature logic general enough to allow extensions to other languages; if you make language specific assumptions, mark them as such

4. Implement the new features logic in the process() method in the java file
5. You might need to create a new feature type XML descriptor in type_system; the java file may be auto generated through the UIMA Component Descriptor Editor
6. Create a feature TAE XML descriptor in the TAE folder parallel to the existing TAE XML descriptors in order to determine which annotations are required and need to be performed before this feature is calculated

D. New feature with new NLP infrastructure

1. Proceed as described in D as far as applicable
2. Since NLP Annotators and Features are all realized as UIMA Analysis Engines, adding a new NLP tool works identical to the integration of a new feature:
    - copy and adjust the XML descriptor and the java file of an existing NLP annotator
    - make sure to change all names
    - make sure to provide all relevant models
    - you might need to create a new linguistic type parallel to Step 6 in D
    - don’t forget to define a TAE
    - only include the new language in the set of supported languages or implement the new feature and NLP tool so that it also supports some or all of the other languages, if possible