FID Documentation

Institute of Applied Computer Linguistics - Goethe University Frankfurt

Frank Abromeit

May 21, 2020

Contents

1 Annohub Editor

1.1 Main Resource Table

The main data table keeps the results of all processed language resources. Each row in the table represents a single file that is included within a resource file. In the following the content and meaning of each individual column of the table is explained in detail.

- *Status* : Color encoded process state information of each resource file

yellow	Basic state of a file after processing that has *good* results
grey	Basic state of a file after processing that has *weak* results
red	Indicates that the model or language results have been edited
green	Indicates that a file has been marked as *Accepted*, and should be included in the Annohub export
black	Indicates that a file has been marked as *Discarded*
blue	Indicates an error during processing

- Linghub URL : Pointer to `http://linghub.com` where metadata about the resource can be found (not available for language resources that do not originate from `http://linghub.com`)
- *Data URL* : Download URL for the raw data
- *File* : The relative path of a file in a language resource
- *Format* : File format (by now XML files are shown as format CONLL, s.t.c.)
- *ISO-639-3* : Language codes for languages that have been detected (— for none)
- *Model* : Annotation models that have been detected (numbers indicate CoNLL columns)
- *Vocabulary* : LLOD-RDF vocabularies that have been detected
- *Metadata Source* : Source of resource metadata like (description, author, etc.) (LINGHUB|CLARIN|USER)
- *Metadata State* : Indicates the quality of harvested metadata (or manually entered)

complete	title, description, creator, year, license, email, webpage
sufficient	title, description, creator, year
incomplete	less fields than required for *sufficient* are present
empty	no metadata is available

- *Mbytes*: File size in megabytes
- *Type* : Automatically detected type (Corpus, Lexicon, Ontology, Wordnet, Unknown)
- *Comment* : A comment text that is provided by the reviewer
- *Processed* : Date of processing
- *Duration* : Time spend for processing, e.g. PT30.72S means 32.72 seconds
- *Accepted* : Date of acceptance
- *Last Updated* : Shows the most recent date were a found model or language was *automatically* updated. Sorting resources by *Last Updated* can help to identify changed results after a *Rescan* or a *OLiA model update* operation. As this is a new feature the results from previous processing are unknown. Therefor the dummy value 1.1.1970 is used as default date for *last updated* if no update information is available.
- *Last Updated text* : A comment with info about the update. Possible values are *added*, in case a new model or language was detected, or *changed* if a existing result (model, language) has been altered during a update operation.

1.1.1 Visibility of main table columns

Via the *Columns* button on the upper left of the main table individual columns can be selected to be displayed. The selection of columns is automatically saved in the *gui.properties* file on the server, and will be restored on any server restart.

1.2 Basic Editing Functions

1.2.1 Language Editing

From the edit window (context menu *Edit*) and further via the *pencil* button one can access the *Edit RDF/CoNLL languages* view. It shows all languages in a file that have been detected automatically by the application together with manually added languages by a user. For a more detailed description of the meaning of each column we refer to the publication [AFG20]. Additional language entries can be added by entering a ISO639-3 code in the ISO field on top and the *Add* button. Each list entry can be furthermore selected/deselected or deleted via a context menu.

1.2.2 Model Editing

From the edit window (context menu *Edit*) and further via the *pencil* button one can access the *Edit RDF/CoNLL models* view.

Editing detected models for RDF files The view shows all automatically detected annotation schemes (models). Each list entry has several result parameters :

Row expansion	Shows detailed info of matched tags/classes for a model
Model	Detected annotation scheme
Property	RDF property with model annotation info
Coverage	Percentage of values for property that could be matched
Hit types	Number of different detected annotations
Sum	Total number of instances of hit types that were found
Exclusive Hit types	Like *hit types* but only matched by that model
Sum	Total number of instances of excl. hit types that were found
Detected-by	AUTO | MANUAL
From	ANNOMODEL | SELECTION
Selected	true | false
Date	Date of detection
Update text	added | changed

Via a context menu it is possible to select/deselect and delete an entry in the list.

Editing detected models for CoNLL files The view for editing CoNLL models contains the same information as described above. In addition to the RDF view different columns of a CoNLL file can be selected, that have been identified to contain annotation model information. For a more detailed discussion on the structure of CoNLL data we refer to [AFG20]. Under *Options* it is possible to delete a column that was incorrectly classified as a *model* column, which can happen if the language detection fails to identify the language in a FORM (LEMMA) column. In order to correct the error it is first required to first delete the model column and then to assign manually the correct language to it in the *Language Edit* view. Occasionally the opposite happens when a model column was incorrectly detected as a text column. Accordingly, one has to delete the misclassified CoNLL language column first, and in order to assign a model, add the appropriate model column for it. Remark : Adding new models to the list is not possible by now.

1.2.3 Saving changes after editing language or model information

After editing is finished one can save the edited results via the *SAVE* button in the main *Edit window* to the database. In order to speed up editing of multiple files the *SAVE to*

ALL button can be used to apply the previously made changes in the currently selected resource to *all files* included in a language resource. However, this will work only for such files that have the same results as the selected file before editing. Additionally only language edits, model edits or all edits can be applied.

1.2.4 Deleting a resource or individual files of a resource

Deleting a resource will destroy all information about it and all included files. When deleting a single file in a resource this will not affect other resource files as well as the resource metadata. If *delete file* is selected and the resource has only this file left, then the resource is deleted as well. Deleting single files of a resource can be useful to reduce the number of files in the database. This is an option if for example all resource files have more or less identical results. Keeping the file count low is advisable because later *update / rescan* operations will run faster, because only those files of a resource, that are currently in the database, will be reconsidered.

1.2.5 Resource sample view

The sample view shows an excerpt of a language resource. It must be noted however, that this function is only available for resources that produced at least language results. Generally, for RDF, CoNLL and XML files the first 100 lines of text will be shown. In addition, the sample for XML files also contains the converted CoNLL output.

1.2.6 Editing Resource Metadata

Via the *Metadata* context menu it is possible to review and edit general resource metadata , like title, description and author that could be harvested from `linghub.org` or any CLARIN center by the application. If the metadata view is empty this means that no such metadata was available and instead the user can enter these information by himself. All metadata will finally be included with the Annohub RDF dump.

1.3 Advanced Editing Functions

1.3.1 Editing multiple language resources

Editing a language resource, like changing a incorrectly detected language can be tedious if many files show the same error. In order to facilitate the editing process some functions (see below) can be applied to many resource files at once.

Mark Accepted	Mark files as accepted
Mark Disabled	Mark files as disabled
Mark Processed	Mark files as processed (only for files with status disabled
Rescan	Rescan resources
Delete	Delete resources
Copy Metadata	Copy metadata of the selected resource file to other files

After selecting the desired operation a dialog opens where the *URL* of the currently selected resource file is shown. It represents the target for the edit operation. If left unchanged, the selected edit operation will be carried out only on this single file. By shortening (changing) the URL expression the target set can be increased, because it is actually treated like a regular expression. The editing target set can be verified by pressing *OK* which opens another window where all affected files will be listed. After reviewing the target set the edit operation can be executed by pressing the *EXECUTE EDIT* button.

1.3.2 Reprocessing a resource

The *Rescan* operation can be used to reprocess a language resource in case the resource has been changed, configuration parameters of the application were changed, or for updating results if the implementation is changed in future releases. There are some rules : Firstly, it is only possible to rescan all files included in a resource and not single files from it. However, only those files from a language resource will be reconsidered that are currently stored in the database. In case that a resource file is found to be unchanged, rescaning will not take place at all. Otherwise rescaning will update the results for models and languages, thereby keeping all manual selections that have been previously made. This holds equally for positive as well as for negative selections. If, for example a model (language) has been automatically selected by the parser, disabling the resource later manually will have the effect that it can not be automatically selected again by the application during subsequent *rescan* or *model update* operations.

1.4 Options

1.4.1 Uploading Resources for Processing

The application also allows for the processing of language resources that can be uploaded as *URL* or as *local file*. A third option is to upload a list of language resource that is contained in a TSV file. Supported file types include RDF (e.g. .rdf, .nt) CoNLL (e.g. .conll, .conllu, .conllx) and XML. A uploaded resource enters the *Process Queue* and is processed after all previously queued processing tasks are finished. The progress of the computation can be monitored via the *Process Queue* window. A restriction on the maximal file size to be accepted for processing, can be set in the application's configuration file.

1.4.2 Process Queue

The process queue window has the resources that are awaiting processing, and the bottom of the list is showing the resource that is currently processed. In order to provide a means to stop executions that take very long, or won't stop (in a case a error happened), the *Cancel Job* button is provided. Another choice is to empty the queue via the *CANCEL ALL* button which will also affect the currently running job.

1.4.3 Refresh

Pushing the refresh button updates the main table with new results that have been generated by finished processing tasks (see Upload).

1.4.4 Publish

Starts the generation of the Annohub RDF file. This includes the results and general metadata only for files that have been marked as accepted (status green).

1.4.5 Database Statistics

Gives an overview of all processed resources so far.

1.4.6 Error Log

If the processing of a language resource (see Upload) did not generate any results (no model or language was detected), or an error occurred during the processing, it will not be listed in the main table view, but here. The columns of the error table provide nearly the same info as the main table view. Additionally the reason for failure is presented in the result, error and error message columns. The error messages that are listed here can be further used to debug, improve or to adjust the application configuration.

1.4.7 Help

Shows this file.

2 Data Model

The application uses two data stores, one for managing all processed resources (REG-DB) and in a second the computation of results takes place (MOD-DB). We employ Apache-Tinkerpop[1], a framework for graph databases, that allows the use of various graph database implementations. We choose *Neo4j* for both data stores as backend, in particular with gremlin-server for the REG-DB, and the embedded version for MOD-DB.

2.1 Registry Database

The registry database has information on any resource that has been processed so far by the NLP pipeline. It contains :

- HTTP-Header info like Mime type, resource size, last-modified-date, etc.
- Info about all included files of a resource and individual file information like size, type, format, etc.
- Results including *annotation models*, *languages* and *RDF vocabularies* that have been detected, for each file

In the database, each language resource is represented by a *resource vertex* that is connected to one or more *file vertices*. Latter can connect to *model*, *language* and *vocabulary* vertices. Metadata about a resource with bibliographical data (e.g. author, publisher, etc.) is stored in *meta-data* vertices that are connected to resource vertices.

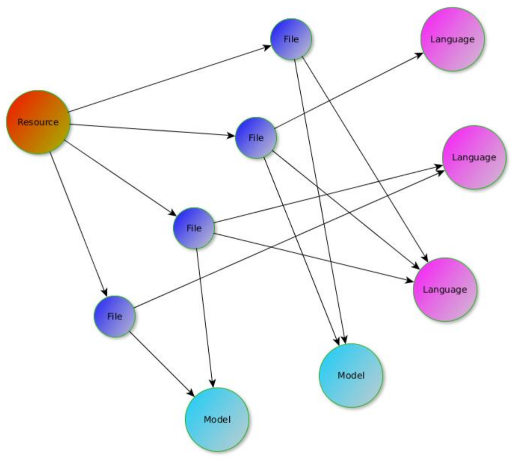

Figure 1: Basic Registry database structure

[1]http://tinkerpop.apache.org/

2.2 Model Database

The model database (MOD-DB) is used to compute the results that are later stored into the REG-DB. It is a graph that includes information of OLiA annotation classes that have been imported from OLiA RDF annotation models[2]. For the linking of results to BLL concepts, the BLL ontology[3] is imported as well. Finally the graph stores all annotations that have been found in language resources, and that could be matched with OLiA annotation classes.

- RDF models of OLiA models
- RDF models of the BLL-Ontology
- Annotations that were found in language resources, that could be matched with OLiA annotation classes

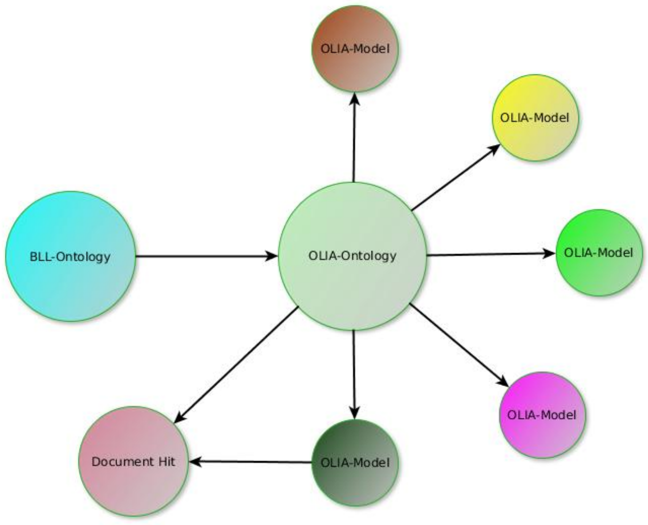

Figure 2: Basic Model database structure

OLiA models linguistic entities from syntax and morphology in a hierarchical class structure with RDF properties (e.g. *subClassOf*, *intersection*, *union*, *complement*, *equivalentClass*). These relations are preserved in the model graph. Annotations from LLOD resources will then be connected with an edge to any matching OLiA class. By traversing the resulting graph it is possible to determine all matching annotations of a file and to make a prediction on the annotation schemes beeing used. A second result are BLL concepts that can be associated with matching annotations.

[2]`http://purl.org/olia`
[3]`https://www.linguistik.de/de/lod/`

3 Installation

For the installation it is not required to install *Neo4j* natively. Instead a Neo4j driver needs to be installed for usage with the Apache Tinkerpop framework. At the moment the installation on windows operation systems is not supported.

3.1 Requirements

- A Linux/Unix distribution
- Java >= 1.8 (tested)
- Apache Tinkerpop >= 3.3.3 (tested)
- Apache TomEE >= 7.1 (tested)
- 7z (7za) File archiving utility
- rapper RDF utility (`http://librdf.org/raptor/`)

3.2 Configuration file (FIDConfig.xml)

There are several choices to provide the application's configuration file

1. By setting the environment variable FID_CONFIG_FILE
2. Alternatively, if started as a web application, the file /WEB.INF/classes/FIDConfig.xml is used
3. When running from the command-line the -CX option can be used

The table below shows the required parameters that need to be set in order to start the application. An example configuration file is provided as well.

3.2.1 Basic configuration options (required)

Databases		
GremlinServer.home	Directory	Home directory of Tinkerpop installation
GremlinServer.conf	File	Path to gremlin server configuration file (gremlin-server-neo4j.yaml)
Data.Neo4jDirectory	Directory	Neo4j database directory for MOD-DB
RunParameter		
downloadFolder	Directory	Download folder for language resources
ServiceUploadDirectory	Directory	Folder to be used for uploaded language resources from the web-app

decompressionUtility	Path	path to 7z (7za) file archive tool

3.2.2 Advanced configuration options (optional)

RunParameter		
urlFilter	String	Constrains processing to language resources with given type(s) (RDF | CONLL | XML | ARCHIVE), e.g. *RDF, CONLL*
updatePolicy	String	Manages how already processed language resources will be treated : UPDATE_ALL | UPDATE_NEW | UPDATE_CHANGED
maxArchiveFileCount	Integer	Skip archives with very large file count
compressedFileSizeLimit	Integer	Skip very large archives (*in Bytes*)
uncompressedFileSizeLimit	Integer	Skip very large archives (*in Bytes*)
Processing		
ConllParser.conllFileMinLineCount	Integer	Exclude very small CoNLL files from processing
ConllParser.conllFileMaxLineCount	Integer	Process at least conllFileMaxLineCount lines (-1 = unlimited)
Sampling		
Rdf.maxSamples	Integer	Maximum number of files to be processed from all folders in a RDF resource (-1 for all files)
Rdf.activationThreshold	Integer	If file count is smaller than activationThreshold all files will be parsed
Rdf.thresholdForGood	Integer	Stop parsing more files after thresholdForGood files have been parsed successfully
Rdf.thresholdForBad	Integer	Stop parsing more files after thresholdForBad files have been parsed unsuccessfully
Xml.maxSamples	Integer	Maximum number of files to be processed from all folders in a XML resource (-1 for all files)
Xml.activationThreshold	Integer	If file count is smaller than activationThreshold all files will be parsed
Xml.thresholdForGood	Integer	Stop parsing more files after thresholdForGood files have been parsed successfully
Xml.thresholdForBad	Integer	Stop parsing more files after thresholdForBad files have been parsed unsuccessfully

Conll.maxSamples	Integer	Maximum number of files to be processed from all folders in a CoNLL resource (-1 for all files)
Conll.activationThreshold	Integer	If file count is smaller than activationThreshold all files will be parsed
Conll.thresholdForGood	Integer	Stop parsing more files after thresholdForGood files have been parsed successfully
Conll.thresholdForBad	Integer	Stop parsing more files after thresholdForBad files have been parsed unsuccessfully
OWL		
modelDefinitionFile	File	Path to custom OLiA model definition file *ModelDef.json*

4 Commandline Interface

After the installation of components has been finished the databases need to be initialized with the *fid -init* command. This will load the model definitions from the ModelDef.json file into the database and will erase all results from the database. There are two ways to provide language resources for processing.

- By using the -SD option on the command-line
- By uploading resources from the web-application via the *UPLOAD* function

After any processing has finished result can be viewed in the web-application.

-CX, –config	Provide the configuration file FidConfig.xml
-IN, –init	Initialize the application (will destroy existing data)
-SD, –seed	Provide a file with a list of language resource URLs to be processed
-EX, –execute	Run the application

References

[AFG20] Abromeit, Frank ; Fäth, Christian ; Glaser, Luis: Annohub – Annotation Metadata for Linked Data Applications Data. In: *Proceedings of the Thirteenth International Conference on Language Resources and Evaluation (LREC 2020)*. Marseille, France : European Language Resources Association (ELRA), 2020