KBSET/NER – User Guide

Christoph Wernhard

Draft – March 2, 2021

Contents

1 Introduction

KBSET/NER is an experimental system to investigate possibilities of knowledge-based support for scholarly editing and text processing. The system should help to develop precise notions of knowledge processing tasks that arise in advanced scholarly text editing in presence of today's large semantic fact bases such as *Wikipedia* extracts or the *Gemeinsame Normdatei (GND)*.

The system is implemented as free software, available from

`http://cs.christophwernhard.com/kbset/`.

It runs on Linux, Mac/OS X and with some restrictions also on Windows. (Offline caches of external knowledge bases can only be precomputed on a Linux or Mac/OS X platform, but once prepared, these cached data can be used on Windows.) A rare German book from the 19th century, *Max Stirner: Geschichte der Reaction, Band 1. Berlin, 1852*, has been used as example during development. It is enclosed in full with the system to provide further user documentation.

For the future, a systematic redesign of the configuration language, the internal query processing as well as the caching of external knowledge sources is planned.

In a typical setting, the system takes as inputs:

1. A source text file, possibly in LaTeX format.
2. *Annotation documents*, that is, text files with annotations, possibly in LaTeX format. The associated places in the source text to which they are referring are specified abstractly.
3. Large fact bases, currently in particular *GND* and *GeoNames* as well as extracts from *YAGO2* and *DBpedia*.
4. A so-called *assistance document*, that is, a configuration file, where, among other things, the fact bases are specified and information is given to bias or override automated inferencing such that fully correct results are obtained.

The system produces a variety of outputs, including:

- LaTeX documents where annotations and inferred information are merged in. By passing unrestricted access to LaTeX to the user, high-quality layouts can be achieved.
- Support during development by possibilities to highlight and inspect entities recognized by the system.
- An export possibility to visualize detected locations mentioned in the source text with the *Dariah* geobrowser.

A typical application would be the development of an annotated essay or book, where the source text is edited in LaTeX and the configuration evolves step-by-step until the inferred information is fully correct. A simpler application is to present a text file with location names to the system, let it detect the locations and produce an input file for the *Dariah* geobrowser.

A user interface integrates the system into the *Emacs* editor, which is free software. *Emacs* is then used to edit the source texts as well as to give access to the full functionality of the *KBSET* system, through menus, key bindings and by issuing commands to the underlying interpreter of the *Prolog* programming language. *Emacs* further supports editing and activating configuration files, which are written in Prolog readable syntax.

The system includes a component for named entity recognition that currently applies to persons, locations and dates. Phrases are not just classified according to their type, but designated entities are actually *identified*. The technical basis is rule processing. For each type of entity the text is scanned separately. Recognition of dates is essentially done by parsing. Recognition of persons and locations is performed with a gazetteer that combines information in a configurable way from sources such as *GND*, *GeoNames* and *DBpedia*. It operates with respect to given single words, but with access to the preceding and succeeding text as well as to other information. A special mode is used to identify persons that are specified by a functional role, e.g., *king of France*. A fixed series of features is evaluated for each candidate entity. Entities are then ranked by lexical comparison of their associated series of values. This is performed in two steps: First some of the candidate entities are quickly ruled out based on features that are cheap to evaluate. The second step then takes further features into account.

Results of named entity recognition are presented to the user who works on a text edition by highlighting words and displaying a ranked series of candidate entities. If one entity is ranked strictly above the others, the item is considered as identified. For each ranked entity also a representation of the rationale that led the system to choose it as candidate is shown in form of the a listing of the relevant feature values. On this basis, the assistance document can now be refined by the user: The features to consider as well as their ordering can be modified. Information about the general context, such as the years concerned by the object text can be given. Additional information about entities that is not present in the imported large knowledge bases, such as particular alternate names or functional roles, can be supplemented. The ranking of particular entities can be promoted by explicitly referencing to them in the assistance document, possibly qualified with a precondition, e.g., another specific word occurring nearby.

2 Installation

2.1 Requirements

2.1.1 Further Software

- *SWI-Prolog* (tested with version 8.2.4)
- A TEX distribution (tested with *TeX Live 2020*)
- *GNU Emacs* (tested with version 25.2.2)

2.1.2 Data Files

A TAR archive with preprocessed extract of the data required for the enclosed example can be downloaded from `http://cs.christophwernhard.com/kbset/datasets.html`. This should be unpacked to a writable directory (*SWI-Prolog* compiles the files to quick load format files that are stored in the same directory). The space needed is around 1.2GB.

Alternatively, the data can be downloaded and preprocessed. See Sect. 2.5. Around 5GB of disk space are then needed for the data files and preprocessing outputs.

2.1.3 Platforms

The system has been tested on Linux and (in previous versions) on OS/X. It also runs on Microsoft Windows with some restrictions and workarounds (see Sect. 2.6).

2.2 Configuration

The system configuration is specified in `examples/stirner_reaction/assistance01.pl`. In particular, the following lines might need adaption:

```
:- register_file_path(tmp(_), '~/tmp/').
:- register_file_path(bigdata(_), '~/space/bigdata_stirner/').
```

These point to a directory where the system can store temporary files and the directory with the data files (Sect. 2.1.2). The symbol `~` is understood by the system as the user's home directory (determined by `getenv('HOME', X)` in *SWI-Prolog*).

In `elisp/kbset.el` set `*kbset-tmpdir*` to the directory for temporary files specifed in `assistance01.pl`. (Alternatively, it is possible to leave `elisp/kbset.el` unmodified and set `*kbset-tmpdir*` *before* loading `elisp/kbset.el`).

For processing sources with more than 2,000 entities, the Emacs variable `*kbset-high-timeout*` might need adaption. See `elisp/kbset.el`.

2.3 Suggested *Emacs* Setup

1. Create an executable script file, e.g., `/home/ch/bin/swi-kbset` with the following content (possibly after replacing f$HOME/kbset with the place of the `kbset` directory at your system):

   ```
   #!/bin/bash
   swipl --no-tty --stack_limit=12g -s ${HOME}/kbset/core/load.pl
   ```

2. Insert this (with pathnames adapted) into your `~/.emacs` file:

   ```
   (setq *kbset-tmpdir* "/home/ch/tmp")
   (load "/home/ch/kbset/elisp/kbset.el")
   (defun run-kbset ()
     (interactive)
     (kbset-minor-mode 1)
   ```

```
(let ((prolog-program-name "/home/ch/bin/swi-kbset"))
  (run-prolog nil)))
```

2.4 Invoking the System

1. Start *Emacs*, start the system Prolog shell with `M-x run-kbset`.
2. Consult an assistance file, e.g., `examples/stirner_reaction/assistance01.pl`.
3. Load data from the Prolog shell:

   ```
   ?- load_data.
   ```

   The predicate `load_data`/0 is defined in the example assistance file `assistance01.pl`.

   For the example, this takes about 1.5 minutes. At the first time it takes about 6 minutes because *SWI-Prolog* then compiles the data files to its quick-load format.
4. Consult the assistance file again to merge supplementary information with the loaded data.
5. Visit the sourcetext file in a buffer (e.g., `examples/stirner_reaction/src/reaction_01_16.tex`).

   `M-x kbset-minor-mode` toggles the *Kbset* minor mode.

   In that mode, various actions are offered in the *Kbset* Menu. *Process file* (re-)reads the sourcetext file and enables the other actions.

   Process file takes about 20 seconds for the example. (At the first run somewhat longer as *SWI-Prolog* internally sets up predicates.)

2.5 Downloading and Preprocessing External Data

This step is not needed for the example document, as the data can be directly retrieved in preprocessed form (see Sect. 2.1.2).

As this step invokes several shell commands, it is only implemented for Linux systems and requires `7z` installed as shell command.

1. Start *Emacs*, start the system Prolog shell with `M-x run-kbset`.
2. Consult an assistance file, e.g., `examples/stirner_reaction/assistance01.pl`.
3. Start the downloading and preprocessing from the Prolog Shell:

   ```
   ?- prepare_data.
   ```

For the example, `prepare_data`/0 is defined in `assistance01.pl`. `prepare_data`/0 takes already downloaded and preprocessed files that are present in the configured "bigdata" directory, along with their timestamps, into account, such that it can simply be called again after an error or other interruption. (In some situations incompletely generated files in the "bigdata" directory need to be removed before re-invoking `prepare_data`, such that they are re-created).

2.6 Installation Hints for Microsoft Windows Platforms

Note: In the following examples some scripts and auxiliary files are stored in places which would be found on a Unix system and are provided by Cygwin for Windows. However, Cygwin is not required for KBSET and these files could also be stored in other places.

Example of the Settings in the `~/.emacs` File

```
(setq *kbset-tmpdir* "c:tmp\_kbset")
(load "c:kbset/elisp/kbset.el")
(defun run-kbset ()
  (interactive)
  (kbset-minor-mode 1)
  (let ((prolog-program-name
         "c:\\cygwin\\home\\ch\\bin\\swipl-kbset.bat"))
    (run-prolog nil)))
```

Example of the `swipl-kbset.bat` Script

In the script, all of the following has to be in one line:

```
"c:\Programme\swipl\bin\swipl.exe"
    -f c:\cygwin\home\ch\swipl-emacs-init.pl
    --stack_limit=12g -s d:/kbset/core/load.pl
```

Example of `swipl-emacs-init.pl`

Note: On Windows, *SWI-Prolog* when run within *GNU Emacs* does by default not properly run in TTY mode.

```
:- set_stream(user_output, tty(true)).
:- set_stream(user_error, tty(true)).
:- set_stream(user_input, tty(true)).
```

Excerpts of an Assistance File that Show the Respective Filename Syntaxes

```
:- register_file_path(tmp(_), 'D:/kbset_tmp/').
:- register_file_path(data(_), 'D:/kbset_data/bigdata_stirner/').

annotated_by('c:/cygwin/home/ch/kbset_example/src/reaction_01_16.tex',
             'c:/cygwin/home/ch/kbset_example/apparat/annot_reaction_01.tex').
```

3 *Emacs* Interface

3.1 Involved Mode, Buffers and Outputs

The `kbset` minor mode provides a menu and key bindings which give access to the main user functionality of *KBSET*. This minor mode can be activated and deactivated by

`kbset-minor-mode`. It can be used together with other modes, for example to support LaTeX.

The *Emacs* command `run-kbset` invokes *SWI-Prolog* with the *KBSET* system loaded. The `*prolog*` buffer provides a command line interface to the Prolog interpreter.

Outputs of the *KBSET* system are placed in the following respective locations:

- Printed in the `*prolog*` buffer.
- Highlighted in the source text buffer.
- Shown in one of the auxiliary buffers, which are displayed read-only in a second window,
  - `*kbset-info*`,
  - `*kbset-results*`.
- Generated ("rendered") files in different formats.

3.2 Menus and Key Bindings

The following menu and key bindings are associated with the `kbset` minor mode.

Some of the menus invoke Prolog commands, written in typewriter font in the descriptions below. The actual parameterized predicates submitted to Prolog are displayed in the echo area of *Emacs* and placed in the command line history of the `*prolog*` buffer such that they can be called again or reviewed with M-p and M-n in that buffer.

- **Process source**

  Call `process_source` on the buffer's current file and highlight recognized items in the buffer. The source file is parsed there and named entity recognition is performed, according to the active settings as described in Sect. 4.

  The source file is expected to be encoded in UTF-8. It can be a plain text document, a LaTeXdocument or an XML document. Advanced parsing and operations are supported for LaTeX. XML documents are identified by the file extension *.xml* or *.tei*. They are simply treated like plain text with the exception that all text enclosed in angle brackets is taken as "opaque", that is, not considered in operations such as entity identification, but preserved and passed through to the output.

- **Goto: Next recognized item (C-.)**

  Move cursor to next highlighted (i.e. recognized) item and show information about it in the `*kbset-info*` buffer.

  The shown information is similar as described for menu *Summary: Persons*.

- **Goto: Previous recognized item (M-.)**

  Analogous to *Goto: Next recognized item*.

- **Browse URL (C-/)**

  Open URL at point, or if in the source file, associated with the point (i.e. displayed currently in the `*kbset-info*`), in Web browser.

- **At point: Show NER candidates**

  (documentation to be written)

- **At point: Show structural content**

  (documentation to be written)

- **At point: Show semantic content**

  (documentation to be written)

- **Summary: Persons**

  Invoke `show_person_summary` on current source and display the result in the `*kbset-result*` buffer. Information comes from the last run of `process_source` (see menu *Process source*).

  The starts of each entry in the summary are marked with `*` such that it is easy to skip between them with C-s and C-r.

  A headline shows general information, such as the number of recognized entities.

  Each recognized entity is then represented by an entry. Entries are ordered according to the frequency of their recognized occurrences.

  For each entry the following is shown: The corresponding words occurring in the text, along with their number of occurrences. If the recognition is ambiguous (other candidate entities are ranked equally well), this is indicated. Information about the entity is given, e.g., the preferred name, years of birth and death, profession or occupation, links to *Wikipedia* and *GND*. Then the derivation used to identify the entity is shown. If different such derivations have been used on different occurrences in the text, then all of these are shown.

- **Summary: Locations**

  Like *Summary: Persons* but invoking `show_location_summary`.

  The general information for locations includes for places with small population also information about nearby larger, and thus presumably more well-known, places. Aside of links to *Wikipedia* also links to *OpenStreetMap* and *GeoHack* are provided.

- **Summary: Dates**

  Like *Summary: Persons* but invoking `show_date_summary`.

- **Refresh display: Last result**

Refresh the `*kbset-result*` buffer with the output of the last command that affects it.

Useful since in some situations the automatic insertion of results from Prolog to the `*kbset-result*` is not properly synchronized.

- **Refresh display: Markings**

  Refresh the highlighting in the source buffer with the output of the last invocation of *Process source*.

  Useful since in some situations the automatic highlighting according to results from Prolog is not properly synchronized.

- **Clear display: Markings**

  Remove the highlighting resulting from *Process source* from the source buffer.

- **Render annotated as LaTeX**

  Invoke `render_latex_annotated` on the file associated with the buffer. A LaTeX document is produced where information from annotations and the previous run of `process_source` is merged in.

- **Render...**

  Produce output in further formats, considering information from the previous run of `process_source`.

  - **Render as plain text**

    Call `render_plain`. The result is a plain text representation of the source, without LaTeX commands.

  - **Render annotated with XML markup**

    Call `render_rxml_annotated`. The result is the input file (XML or plain text) with TEI-compatible XML markup inserted to mark identified entities.

    For the XML rendering, so far, only entities detected by person, location and date identification are considered. The scope of the inserted XML elements spans the text which is taken as associated with the entity. For persons and locations this is currently just a single word, contradicting the intended semantics of the used TEI elements, which suggests to embed whole noun phrases.

    A designator of a person or location is wrapped with an element

    `<name resp="auto" type=`*TYPE* `key=`*KEY* `ref=`*URL*`>`,

    where *TYPE* is "person" or "place", respectively, *KEY* is an identifier stemming from *GND*, *GeoNames* or from a user declaration of a new entity in an assistance document. *URL* is an URL of a Web page representing the entity, for example in *Wikipedia*, *GND* or *GeoHack*. If no such page is available, the

`ref` attribute is not provided. A date is wrapped with an element

`<date resp="auto" when=`*DATE*`>`,

where *DATE* is normalized according to *W3C Recommendation XML Schema Part 2: Datatypes Second Edition*, as specified for TEI.

– **Render entity/3 Prolog facts**

Call `render_entity_facts`. The result is a Prolog file with facts representing the recognized entities. Their form is:

`entity(`*EntityType*`,`*EntityId*`,`*EntityPrettyName*`).`

– **Render Dariah CSV**

Call `render_dariah_csv`. Produces a CSV file with recognized locations for loading into the *Dariah* geobrowser `http://geobrowser.de.dariah.eu/`, where they can be visualized. Previously recognized dates and persons are associated with the locations.

– **Render in NER style**

Call `render_ner_style`. Produces a text file with output similar to other named entity recognition systems.

3.3 Prolog Predicates Underlying the *Emacs* Interface

The Prolog predicates listed below are invoked from *Emacs*. They are described in Sect. 3.2 in the context of the menus that invoke them. In the system they are gathered in the module `kbset_ner(ner_interface)`.

```
process_source/1
process_source/2
inspect_item_at_srcpos/2
show_person_summary/1
show_location_summary/1
show_date_summary/1
show_semantic_context_at_point/2
show_structural_context_at_point/2
render_latex_annotated/1
render_plain/1
render_rxml_annotated/1
render_entity_facts/1
render_dariah_csv/1
render_ner_style/1
set_verbosity/1
```

4 Assistance Documents

An *assistance document* specifies a system configuration of *KBSET*. It has the form of a file in Prolog readable syntax and is loaded into the *KBSET* system the with the normal Prolog mechanism for loading source files, that is, the `consult` predicate.

An assistance document can be reloaded at any time, where the specifications loaded previously are replaced.

A convenient way to work with an assistance document is by editing it in the *Emacs* editor, where common Prolog modes provide a key binding or a menu to load (*"consult"*) the file associated with a buffer.

4.1 Conventions

File and directory names are atoms. Also words of text are represented by atoms. A *PropertyList* is a list of *Key*=*Value* pairs, where *Key* is an atom.

4.2 General Setup

The module `kbset_ner(ner_kbset)` provides auxiliary predicates for use in assistance documents. They are discussed below in the context of their application. To make these predicates available, they have to be imported with the following statement at the beginning of the assistance file:

```
:- use_module(kbset_ner(ner_kbset)).
```

The verbosity of the system can be controlled by statements

```
:- set_verbosity(Number).
```

Suggested values for *Number* are 50 and 1, respectively.

4.2.1 Specifying Paths to Auxiliary Directories

The system needs two auxiliary directories, one for temporary files and one for caching downloaded external knowledge bases in source and preprocessed form. This has to be set up with the following statements:

```
:- register_file_path(tmp(_), Directory).
:- register_file_path(data(_), Directory).
```

Directory is a path name specifier interpreted as follows:

- If the specifier begins with `/`, then it is taken as an absolute filename.
- If the specifier is of the form `~/`*PathName*, then *PathName* is taken relative to the user's home directory (as determined by `getenv('HOME', X)` in *SWI-Prolog*).

4.2.2 Specifying File Locations

Various files used by components of the system are addressed internally with symbolic names. Their mapping to actual files is done with statements of the form:

`:- register_file(`*SymbolicFileName*`,` *File*`).`

SymbolicFileName is an atom. *File* is an absolute pathname or a term of the form `tmp(`*RelativeFile*`)` `data(`*RelativeFile*`)`, representing a file in the temporary or data directory.

4.3 Specifying External Knowledge Sources and their Preprocessing

By convention, a predicate `prepare_data/0` is specified in the assistance file. It downloads external knowledge sources and preprocesses them, ensuring that `load_data/0` (Sect. 4.4) can afterwards be successfully performed. A *make*-like mechanism to specify and maintain dependencies is provided. It is invoked by `dep_ensure/1`, dependencies are specified with `dep/1`, which has to be defined in the assistance file. See source of module `kbset_ner(prepare)` for details.

4.4 Loading Preprocessed External Data into the System

By convention, a predicate `load_data/0` is specified in the assistance file.

4.5 Configuration of the Source Document Parsing

The system parses LaTeX documents. It properly recognizes many common LaTeX commands. Statement of the following form provide a hook to specify further commands, either from LaTeX or some of its packages or defined by in the source document:

`:- register_latex_command(`*LaTeXCommand*`,` *LaTeXCommandMode*`,` *PlainText*`).`

See source of module `kbset_ner(readwrite)` for documentation.

The following predicate specifies LaTeX commands that delimit section-like units considered at keyword inference.

`kwd_section_commands(`*ListOfLaTeXCommands*`).`

4.6 Associating Annotation and Source Documents

An "annotated by" relationship between documents is specified by statements of the following form:

`annotated_by(`*Source*`,`*Annotation*`).`

Source and *Annotation* are absolute file names. Associated annotations are also discussed in Sect. 5.

4.7 General Control of Named Entity Recognition

Three predicates defined in the assistance document control how named entity recognition is performed:

`ner_processing_options(`*ListOfSpecifiers*`)`

ListOfSpecifiers is a list of atoms from `quotes`, `dates`, `function_persons`, `persons` and `locations`. Named entity recognition passes through this list from left to right and performs the recognition for each indicated entity type one by one. Thus, the order of items in the list might be relevant.

`entity_features(`*EntityType*`,`*ListOfFeatures*`)`

EntityType is one of `person`, `function_person`, `location` or `date`. *ListOfFeatures* is a list of atoms or term patterns, each corresponding to a feature implemented for the respective entity type. Candidate entities for a given word in a given context are compared according to the list of features, lexically from left to right.

`entity_threshold_features(`*EntityType*`,`*ListOfFeatures*`)`

Similar to `entity_features/2`, however only features represented by atoms are allowed in the list of features. Features specified by this predicate are used to reject candidate entities straight away, before features whose values are more expensive to determine are taken into consideration.

4.8 Specifying Global Context Settings

Named entity recognition is performed with a given *context* at hand, which contains information that depends on the particular location in the text, that has been determined previously by named entity recognition, and that is globally specified with the following predicate:

`def_context(+`*Name*`, +`*PropertyList*`)`

It associates with *Name* (an atom) a property list that specifies global context settings. The predicate `make_effective` (Sect. 4.12) can then be used to let these context settings take effect in named entity recognition.

Some commonly used properties set here are:

Key	Value	Explanation
`user_language`	`de`, `en`	Language of the user interface
`text_language`	`de`, `en`	Language of the object text
`text_language_variant`	e.g., `de(19)`	Variant of the object text language, e.g., German of the 19th century
`stemming_language`	e.g., `de(19)`	Language used for stemming
`context_years`	*ListOfNumbers*	Years on which the object text is about, used to disambiguate entities

4.9 Supplementing Word Data

In named entity recognition databases of words that commonly occur with a lower case first character and of words which are commonly used as substantive are used. Statements of the following form can be used to extend these databases:

```
:- register_common_downcase(Word).
:- register_common_substantive(Word).
```

4.10 Supplementing Further Entities

Statements of the following form can be used to extend the database with further entities of type person.

```
:- register_ext_person(Id, PropertyList).
```

Id is an atom, used to identify the person. The keys allowed in *PropertyList* correspond to the GND, for example:

```
:- register_ext_person(lubersac_1740,
    [preferredNameForThePerson = 'Lubersac, Jean-Baptiste-Joseph de',
     dateOfBirth = '1740-04-15',
     dateOfDeath = '1822-08-30',
     gender=male,
     biographicalOrHistoricalInformation = lang(de,'Bischof von Chartres (1780-1790)'),
     urL = 'https://fr.wikipedia.org/wiki/Jean-Baptiste-Joseph_de_Lubersac']).
```

4.11 Specifying Assistance to Named Entity Recognition

Named entity recognition takes manually specified information into account at disambiguation. That information can be specified with the following predicate:

```
def_assistance(Name, ListOfAssistanceStatements) .
```

It associates with *Name* (an atom) a list of assistance statements. The predicate `make_effective` (Sect. 4.12) can then be used to let the statements take effect in named entity recognition.

Some of the assistance statements have a *Condition* argument. This is a list of condition statements that are evaluated in the context of the word to be recognized. An empty list

represents a condition that is always satisfied. The following condition statements are recognized:

Key	Value	Condition
`word_in`	list of words	current word is member of the list
`near_word_in`	list of words	a nearby word is in the list
`following_words`	list of words	the subsequent words are as specified, in same order, punctuation and space is ignored
`followed_by_words`	list of words	the previous words are as specified, in same order, punctuation and space is ignored

Some of the assistance statements have an *EntityDesignator* argument. This is a property list that uniquely identifies an entity. If this fails after `load_data` has been invoked, an error is raised.

Assistance statements of the following forms are accepted, where *EntityType* is one of *person* or *location*.

`no_entity(`*EntityType*`,` *Condition*`)`

Entity recognition of the entity type is explicitly blocked for words matching the condition. (Typically used with a `word_in` condition.).

`entity(`*EntityType*`,` *EntityDesignator*`,` *Condition*`)`

The designated entity is preferred at disambiguation in contexts where the condition applies.

`entity(`*EntityType*`,` *EntityDesignator*`)`

Same as `entity(`*EntityType*`,` *EntityDesignator*`, []`.

`supplement(`*EntityType*`,` *EntityDesignator*`,` *PropertyLists*`)`

Used to supply additional information about the designated entity. Currently the properties `name` (for alternate names not in the GND) and `biographicalOrHistoricalInformation` (e.g., for adding functions not in the GND) are supported.

4.12 Activating a Combination of Named Settings

The following predicate, defined in the assistance document, specifies which of the context and assistance specifications should take effect.

`make_effective(`*Names*`).`

Names is a list of names specified by `def_context` and `def_assistance`.

The following statement actually lets the specifications in the assistence document become effective:

```
:- update_assistance.
```

4.13 Summary of Predicates Defined and Used in Assistance Documents

The following predicates are *defined* in the assistance document:

```
dep/1
kwd_section_commands/1
ner_processing_options/1
entity_features/2
entity_threshold_features/2
def_context/2
def_assistance/2
make_effective/1
```

The following predicates are exported by the `kbset_ner(ner_kbset)` module for use in the assistance document:

```
update_assistance/0
add_assistance_update_hook/1
register_latex_command/3
register_file_path/2
register_file/2
set_verbosity/1
set_warnings/1
register_common_downcase/1
register_common_substantive/1
dep_ensure/1
install_gnd_all_bornbefore/1
install_gnd_accessors/0
install_dewiki_links/0
install_wikinames/0
install_geonames/0
install_person_stuff/0
register_ext_person/2
install_function_info/0
convert_geonames_to_prolog/0
convert_wikinames_to_prolog/0
convert_gnd_to_prolog/0
convert_gnd_to_wiki/0
convert_gnd_extract_items_in_wiki/0
convert_gnd_all_bornbefore/1
convert_gnd_extract_ontology/0
convert_dewiki_links/1
```

5 Annotation Documents

KBSET supports annotations that are not statically interspersed into the object text but maintained separately in so-called *annotation documents* which can be combined with the object text automatically. The specification of annotation in annotation documents is done with static markup which uses LaTeX syntax and embedded Prolog syntax to specify attributes.

When annotation documents are combined with object texts, the portions with relevant LaTeX markup are extracted, which allows the annotation documents to contain arbitrary other text, including LaTeX code. Annotation documents can then be full fledged LaTeX documents that can be rendered on their own, which might be useful when writing annotations.

The association of annotation and source documents is controlled by the `annotated_by/2` predicate. See Sect. 4.6.

The markup in annotation documents is done with the LaTeX command `\xabout`:

`\xabout{`*Type*`}{`*PositionSpecifier*`}{Text}`

Here is an example:

```
\xabout{source}{txt='An dieser ersten revolutionären Umwandlung haben also'}
       {Auszugsweise Übersetzung von \volcite{5}[S.~585--589]{comte:cours}.}
```

The *Type* argument is passed to the rendered LaTeX code and can be used there, e.g., to determine the color of the annotation.

PositionSpecifier specifies the positions(s) in the source text with which the annotation is associated. It is in Prolog syntax. Currently the following is supported:

- `txt=`*ListOfWords*, where list specifies a sequence of words. If there is not exactly a single match in the source text, at processing the annotations an error is raised.
- `txt_multi=`*ListOfWords*. Like `txt`, but the annotation is associated with all matches. It is no error if there is no such match.
- `txt_first=`*ListOfWords*. Like `txt`, but the annotation is associated with the first match. It is no error if there are several matches, but it is an error if there is no match.

Text is the text of the annotation.

To obtain an annotation document that can independently processed by LaTeX some definition for the f\xabout command has to be provided in the document, for example:

```
\newcommand{\xabout}[3]{\par #3\par}
```