Deliverable D.2.2

Bio-Lexicon DataBase: Architecture, Concepts and Loading Software

Project acronym:	BOOTStrep
Project full title:	Bootstrapping Of Ontologies and Terminologies STrategic REsearch Project
Proposal/Contract no.:	FP6 - 028099
Duration:	April 01, 2006 – March 31, 2009
Project coordinator:	FSU Jena
Website:	www.bootstrep.eu
Authors:	Riccardo Del Gratta, Monica Monachini, Valeria Quochi, Eva Sassolini
Additional Contributors:	Nicoletta Calzolari
Date of preparation:	30/03/2007
Dissemination level:	(RE)

Table of Contents

Table of Contents 2
Executive Summary 3
Introduction 4
From the BioLexicon Data Model to the BioLexicon Data Base: The XML Interchange Format (XIF) 5
General Structure of the XIF 5
DTD and XML entries 6
Importing Semantic Variants 10
Managing Inflected Forms 11
BioLexicon DataBase Internal Architecture 13
Typical data base concepts 13
DataBase Architecture: Three-frame Architecture 15
Dictionary Frame 16
Staging Frame 17
Target Frame 21
Software Architecture 22
Main flow 22
Packages 23
How LD works: from the XIF to target tables 25
Concepts 28
“Deep” concept 28
Synchronizing information between Dictionary and Staging Frames 30
A Graphical View of the Environment 31
Mapping between BioLexicon entities and BioLexicon tables 33
Standard Tables 33
User Extended Models 37
Mapping BioLexicon model onto standard target tables 38
Mapping BioLexicon model onto standard user tables 39
A description of the “how-to-access” methods to the BioLexicon database. 40
Appendix A: Complete DTD 41
Appendix B XIF Example 45
Appendix C: Dictionary Frame Details 47
Work Tables 47
Rule Tables 47
Appendix D: Staging and Target Frame Details 49
Staging tables columns 49
Data category columns 50
Target tables columns 51

Table of Figures

Table 2-1: A Clustering Example 5
Figure 2-2: Lexical entry and some DCs 6
Figure 2-3 *LexicalEntry* and *Lemma* 8
Figure 2-4: *Lemma* and *FormRepresentation* 9
Figure 2-5: Synonymous Entries in the BioLexicon 11
Figure 2-6: lemma, wordforms and DCs 12

0. Executive Summary

The present Deliverable D2.2 describes the steps towards the definition of the BioLexicon database, which constitutes the physical counterpart of the BioLexicon data model defined in D2.1.
The two Deliverables, D2.1 and D2.2, are to be seen as a continuum, rather than two separate documents. While the first presents the conceptual architecture underlying the BioLexicon, the latter describes the database, i.e. the "*container*" where the lexical objects of the model will be implemented and automatically populated with pieces of information (coming from WP01, WP03 and WP04) to instantiate lexical entries of the bio domain.
DB implementation has been preceded by both a specification and a design phase, which go step by step. Specification is meant for collection of requirements and constraints, whereas design concerns both the model and the database. Requirements flow from the specifications to the model and architecture chosen to meet the defined conditions (accounted for in D2.1). In the design of DB architecture, we exploited expertise accumulated in many years about lexical databases, resorting to the SIMPLE lexicon database and evaluating its adaptability to the requirements of the BioLexicon. A first prototype version has been deployed in MS Access, then we migrated to an open-source DB.
The BioLexicon DB consists of two modules: the actual MySQL database, i.e. the container of bio-terms and term related information, and a java software component for the automatic population of the database with data provided by WP01 (EBI) and WP03 (Manchester). External to the DB, but fundamental for its automatic population, is an XML Interchange Format (XIF) specifically tailored on terminological repositories organization and the BioLexicon structure to facilitate data providers to structure their input data and *populators* to correctly upload them. The design of the DB has gone hand in hand with the lexical objects conceptualized in the model. Lexical objects have been implemented as tables, whereas relation between objects as correspondence tables; the so-called Data Categories have been loaded into special tables. Java procedures for uploading lexical data have gone hand in hand with the XIF.
This Deliverable is structured as follows:
Section 1. describes the XIF developed, both for encoding terms extracted from already existing bio databases and for importing them into the BioLexicon database. Fragments of the XIF are shown together with relevant parts of the BioLexicon model.
Section 2. is devoted to the BioLexicon database architecture. Mapping of the XIF onto the DB is shown in order to allow an external configuration by users. Choices and reasons of the proposed architecture are presented to explain how the database may support a sort of parallel loading.
Section 3. is about the architecture of software. The Java APIs designed to manage data encoded into the XML interchange format and their loading into the database are shown: each paragraph is devoted to a particular API with synopsis and returned data.
Section 4. provides some "concepts" used during the software development and database design.
Section 5. presents a graphical view of the BioLexicon Environment as it will appear at the end of the Project.
The Deliverable comes with a substantial set of Appendices, which are meant to provide the reader with the necessary support to exhaustively understand the mapping between the BioLexicon data model and database.
Appendix 1 is the mapping between the Biolexicon entities and BioLexicon tables
Appendix 2 contains the XML Interchange Format DTD
Appendix 3 shows a sample entry encoded into the XIF
Appendix 4 describes the Dictionary Frame of the database architecture
Appendix 5 contains staging and target tables

1. Introduction

This deliverable describes the steps towards the definition of the BioLexicon database (T2.3), which is the "container" where the objects conceptualized at the level of the data model will be implemented to allow automatic population of term entries with data collected from existing bio terminologies (WP01) and automatic enrichment with pieces of lexical information extracted from texts (WP03 and WP04).

The requirements identified during the specification phase for the design of the conceptual model (cfr. D2.1) also constitute the conditions underlying the architecture of the BioLexicon database presented here. The major emerging issue is that any of the terminologies readily available in the bio domain is targeted to fulfil special demands for the biological research community, i.e. access to continuously produced knowledge. The BOOTStrep project brings together experts from the domains of biology, bioinformatics, computational linguistics, ontologies to develop new resources for the demands in text mining for biologists and bio-informaticians. One of these resources is the BioLexicon, which brings together data from the biomedical data resources and from the scientific literature.

The BioLexicon database is a comprehensive, continuously growing and integrated resource. *Integration* is intended in, at least, two senses:

1) Initially, it *integrates* data from terminological resources of the biomedical domain and terms extracted from the scientific literature (e.g. UniProtKb/ Swiss-Prot, ChEBI, BioThesaurus, NCBI taxonomy and other biomedical resources) into a commond standard representation that has already on other domains proven its usefulness.
2) then, it allows for further *integration* of morphological, syntactic and lexical semantic features (extracted from the bio literature) which go to enrich terms (and variants).

The enrichment of term entries with information typically belonging to computational lexicons is especially designed to target special demands for the biological research community, i.e. access bio knowledge, and offers the detail information relevant to information extraction technologies. Other richness of the BioLexicon is that it will incorporate semantic information that has been gathered from taxonomies and from the ontological resource of the BOOTStrep project (D5.1).

The BioLexicon DB is a flexible, extensible relational database, articulated into modules:

- the actual MySQL database, i.e. the container for bio-terms and term related information, which comes equipped with
- automatic population procedures, i.e. a java software which takes in input data taken from the Integrated Repository (D1.2) to populate the lexical containers.
- External to the DB but fundamental for its automatic population is a BioLexicon XML Interchange Format (XIF hereafter) specifically tailored on the BioLexicon DTD (D2.1). The XIF is a dedicated input data structure which allows data providers to encode in a structured way terms gathered from existing sources and lexicon managers to pull and appropriately store them in the database.

The database is structured into three logically distinct but strongly interconnected layers: the first two are operational, i.e. especially designed for interaction with automatic population procedures; the third one, the so-called *Target Frame* layer, contains the actual BioLexicon database. Here the lexical objects conceptualized in the model (and defined in the corresponding DTD) are implemented as tables, whereas relation between lexical objects are implemented as correspondence tables in the DB. The

BioLexicon DB ensures total reusability and direct accessibility to all of its content. The neat separation between target tables (the BioLexicon proper) and operational tables allows for the optimization of the uploading of data into the BioLexicon DB, and ensures an easy extendibility both of the database and of the uploading procedures.

2. From the BioLexicon Data Model to the BioLexicon Data Base: The XML Interchange Format (XIF)

This section provides a description of an exchange XML format (called the BioLexicon XML Interchange Format or XIF) that has been designed to facilitate the automatic population of the BioLexicon DB. The form of the XIF is defined in the XIF DTD (in appendix). This format has been originally designed to facilitate WP01 and WP03 partners as provider of data, and therefore has been primarily conceived to import terms and variants extracted from already existing databases and resources in the form of clusters. As the possibility of extracting new types of information emerged, the XIF has been enhanced to cover also semantic relations and inflected forms. By means of the XIF, we also allow for a standardization of the data extracted from the different terminological resources and, in a later stage, from texts; and for an easier efficient uploading of the final BioLexicon DB. This way, any future system/group wishing to feed new data into the BioLexicon would only need to encode this in a XIF conformant XML file.
The XML Interchange Format DTD is to be considered a simplified version of the BioLexicon DTD, as reported and described in D2.1 (and following updates), and the two must not be confused. The way information is represented in the XIF is substantially different from the representation in the BioLexicon DB, and the choice of this specific structure for the exchange format has been made also for reasons of software implementation.
The present section will:

- describe the overall structure of the XIF;
- describe the exchange format in more detail with examples and explanations;
- describe how the exchange format maps onto the BioLexicon structure.

2.1 General Structure of the XIF

The basics concepts of the organization of the XIF are, on the one hand, drawn from the work done in WP03 (D.3.1) on term variant clusterization (see and example in Table 2-1), and on the other taken from the basic elements of the BioLexicon DTD.

ClusterID (SourceID?)	Subcluster	Term	Isprefer Red
Q4U313	A	Interleukin-2	Y
Q4U313	A	IL-2	N
Q4U313	B	T-cell-growth-factor	Y
Q4U313	B	TCGF	N

Table 2-2: A Clustering Example

In table 2-1 we can see and example of term and variants clustering performed in WP03. The first field is the *ClusterID* (i.e. the id in the original DB), the second field is the sub-cluster indication; the third is the *writtenform* (i.e. the string) representing the term. The fourth field is an indication of which string in the cluster has to be considered the base form. The identification of sub-clusters within a same cluster implicitly gives

information about synonymic relation between terms. The XIF is designed to represent all this kind of information in a way that is at the same time close to the extraction techniques and to the BioLexicon Model. That is to say that, from our perspective, sub-cluser A and subcluster B, in the example reported above, would be two synonymic but independent Lexical Entries in the Lexicon. The members of the same sub-clusters, instead, are treated in the Lexicon as othographic variants of the same *LexicalEntry*.

The XIF is, thus, organized in clusters of terms, i.e. in sets of coherent types of information. A cluster may contain one or more synonymous entries containing information about their lemmas, parts-of-speech, inflected forms, semantic relations and external references. In the future, the XIF may well be extended in order to meet new population needs, on the basis of the types of information that will be extracted (in particular sub-categorization frames and bio-events).

The XIF organization allows for a splitting of the input file by clusters for a parallel uploading. Also, it is easily extendible in order to face the need of adding new types of information to the lexicon. In such a case, it is only sufficient to add new elements or new attributes to the XIF DTD and to introduce the appropriate (few) modifications to the DB.

2.2 DTD and XML entries

The present XIF DTD is designed to represent terms and related information (i.e. term variants, POS, external resources …) that will fill the Lexical Entry, *Lemma*, *FormRepresentation*, *Sense* and *Sense*Relation objects in the DB (plus correspondence tables).

Relation between Lexical Entry and its Typed Data Categories

Let's consider the relation between the following classes in the BioLexicon model: *LexicalEntry*, SourceDC, and POSDC.
SourceDC and POSDC are typed Data Categories, and therefore have been represented as separate objects here, but with different colours because they are to be distinguished by the actual lexical objects of the model. DCs in fact contain the features that are used to specify real instances of such classes (i.e. the feature POS:commonNoun, in the figure below, will be actually contained by the POSDC table, and only referred to in the *LexicalEntry* instance).

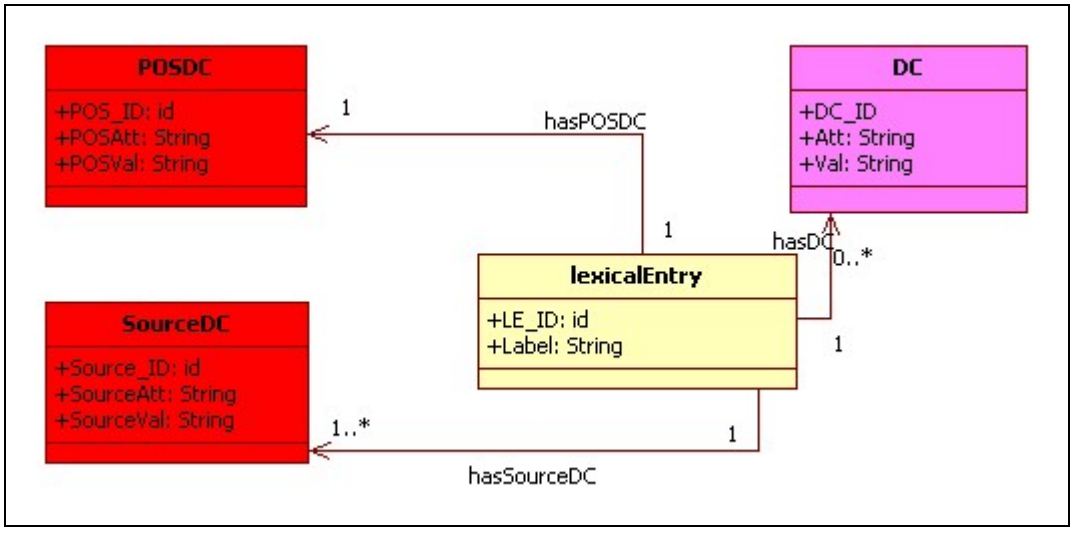


Figure 2-3: Lexical entry and some DCs

Most of these features will be imported into the BioLexicon DB through the proposed exchange format. The correspondent XIF DTD fragment is the following:

```
<!ELEMENT Entry (SOURCEDC*, DC*, POSDC, ……. )>
<!ATTLIST Entry
ENTRYID            ID #REQUIRED
……
 >
<!ELEMENT SOURCEDC EMPTY>
<!ATTLIST SOURCEDC
 SOURCENAME   CDATA #REQUIRED
SOURCEID        CDATA #REQUIRED
  >
```

Here ENTRY should represents the terms in the clusters indicated as “preferred: Y”, and SOURCEDC is the element that contains the external references of such terms, i.e. the indication of their IDs in their original resources.1

A similar definition applies to the POSDC data category:

```
<!ELEMENT POSDC EMPTY>
<!ATTLIST POSDC
POSNAME   CDATA #FIXED “POS”
POS           CDATA #REQUIRED
>
```

The following is an example of a valid entry according to the XIF:

```
…….
<Entry ENTRYID="ENTRY0002" …..>
    <SOURCEDC SOURCENAME=”UNIPROT” SOURCEID="Q4U313"/>
    <SOURCEDC SOURCENAME=”GENEDB” SOURCEID="GNDB123"/>
    <POSDC POSNAME=”POS” POS="N"></POSDC>
……..
</Entry>
…….
```

In the BioLexicon DB, the such information on the external references and POS will specify the *LexicalEntry* object.

Relation between *LexicalEntry* and *Lemma*

The figure 2-3 below shows the relation between the *LexicalEntry* and *Lemma* objects in the BioLexicon model.
As briefly mentioned above, in the XIF the preferred form of a cluster will be promoted to the status of *LexicalEntry* (and by virtue of the 1:1 relations, of *Lemma*). Therefore, the string representing the preferred term will become the value of the “baseform” attribute.

[1] Please note that, representing external references in a separate element allows for the possibility of adding as many external references as needed to entries, without the need to fix a maximum number.

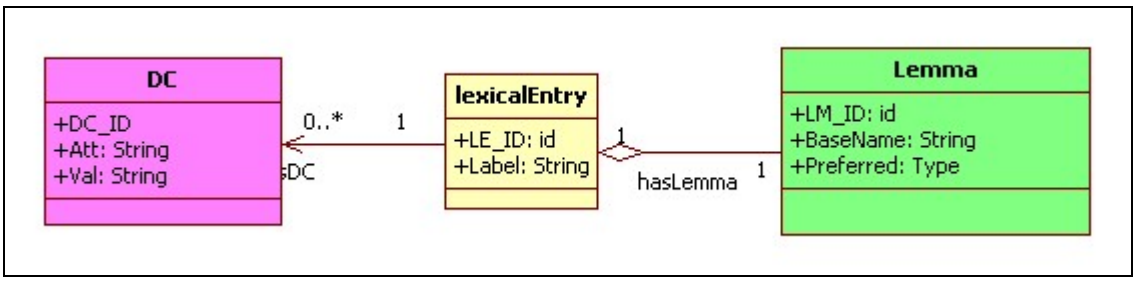


Figure 2-4 *LexicalEntry* and *Lemma*

The relative DTD fragment of the XIF is:

```
<!ELEMENT ENTRY (SOURCEDC*,DC*,POSDC,…..)>
<!ATTLIST ENTRY
ENTRYID     ID  #REQUIRED
BASEFORM CDATA #REQUIRED
                ……
TYPE          CDATA #REQUIRED >
```

An Example of this is the following XML:

```
<Entry          ENTRYID="ENTRY0001"          BASEFORM="Interleukin-2"
TYPE="PREFERRED" >
……..
</Entry>
```

Preferred forms in Clusters will become independent *Lemma*s (thus, LexicalEntries) in the BioLexicon). By means of this XIF structure, then, we can fill the *Lemma* table with the relevant baseform.

Managing Variant and Variant-Type

Orthographic variants, i.e. acronyms, abbreviations and short forms in general, in the BioLexicon will be stored in the *FormRepresentation* object, together with the wordform of the preferred term, indicated as “fullform”. Therefore the information encoded in the XIF will be distributed in various lexical objects (i.e. tables in the DB) as described in the previous sections. The attributes
“fullform”, “shortform”, “acronym” etc. are the values of the VariantType DC that, in the BioLexicon structure, specifies the *FormRepresentation*, whereas in the exchange format, they will specify the Variant element.

Figure 4 below shows the relation between *Lemma* and *FormRepresentation* in the Lexicon:

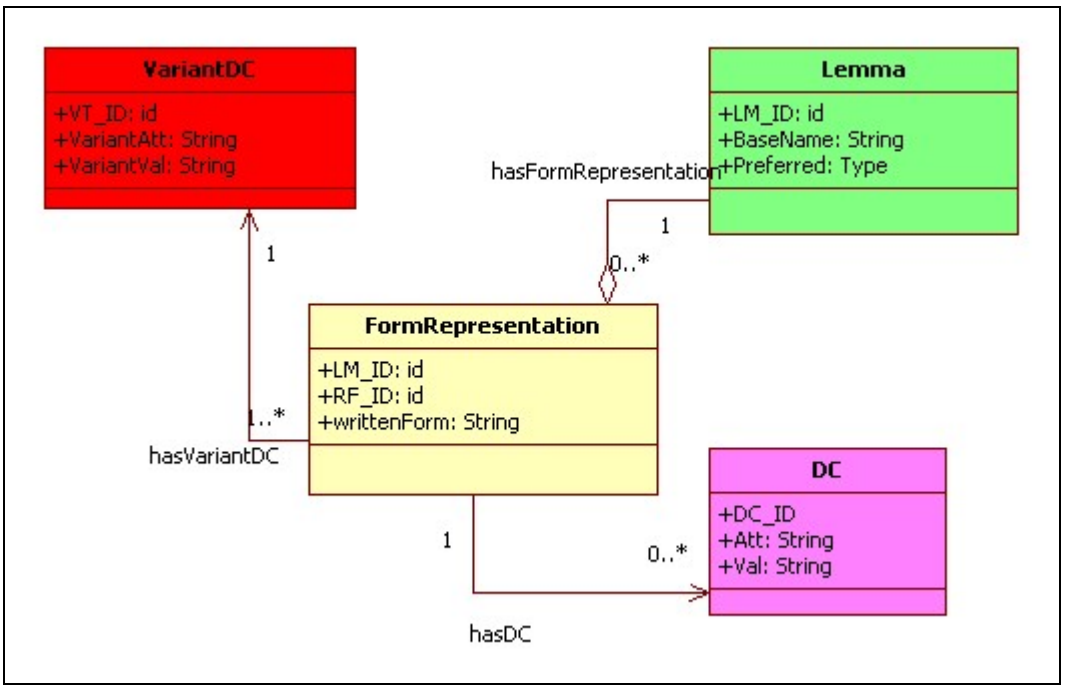


Figure 2-5: *Lemma* and *FormRepresentation*

In summary, through the XIF, as described so far, we can import:

- The wordforms (i.e. text strings) of the preferred terms, their (ortho)graphic variants, and information on the type of variant2.
- Morpho-syntactic features, like POS.

Given the clustering presented at the beginning, we know which terms are (ortho)graphic variants and which terms are the lemmas (the preferred terms). This way, it is possible to implicitly infer the semantic variants (i.e. terms that are synonymous, but do not share wordforms). From a BioLexicon proper perspective, *Lemma*s would also be represented in the *FormRepresentation* as "fullform". Other terms in the same sub-cluster (variant elements) would only be encoded in the *FormRepresentation* with their wordform and *variantTypeDC*, i.e. "non-preferred" members become *Variants*.

Here below is the DTD fragment that formalize the above description.

```
<!ELEMENT ENTRY (SOURCEDC*,DC*, POSDC, VARIANT* )>
 <!ATTLIST ENTRY
 ENTRYID      ID  #REQUIRED
 BASEFORM  CDATA #REQUIRED
 ……..
 TYPE          CDATA #REQUIRED
 >
<!ELEMENT VARIANT (DC*) >
 <!ATTLIST VARIANT
 WRITTENFORM CDATA #REQUIRED
 TYPE               CDATA #REQUIRED >
```

The following XML is an example of what the Entry in the interchange format (XIF) would look like:

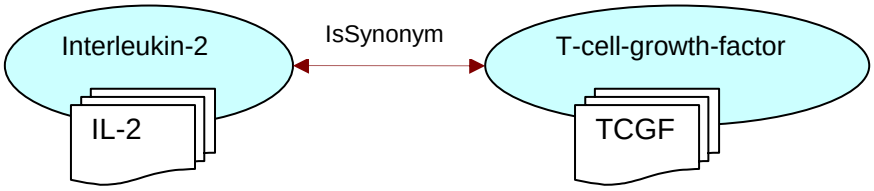
[2] Potentially we can distinguish between acronyms, abbreviations, short forms, morphological variants… However, for the moment the only automatic distinction provided by WP03 is between full forms and orthographic variants.

```
<entry entryid="entry0002" baseform="t-cell-growth-factor" type="preferred" >
        <variant writtenform="tcgf" type="acronym">
        </variant>
</entry>
```

This XML permits to store in the *FormRepresentation* both term variants and the preferred term, with their wordform (string) and VariantType.
The Data Categories used in the XIF, especially those related to variant types, will be reused in the final BioLexicon; that is, they will enrich the DC Selection defined for the BioLexicon.

2.3 Importing Semantic Variants

As previously mentioned, from our point of view, terms marked as preferred that belong to the same cluster are to be considered semantic variants (i.e. Synonyms).
Semantic Variants will be represented in the lexicon and as separate LexicalEntries, linked to one another by a synonymy relation.
From the figure above, we can infer the following information:



Which translates into a tabular representation like the table 2-2 below:

Entry	IsSynonymOF
Interleukin-2	T-cell-growth-factor

Tabella 2-1: Synonymy Relation

In fact, the two terms, Interleukin-2 and T-cell-grow-factor, belong to the same cluster but to two different sub-clusters (that is, entries in the exchange format); and they are both marked as “preferred”.

Inferring Synonymy in the XIF

Consider the figure below, which represents a part of the BioLexicon Structure.

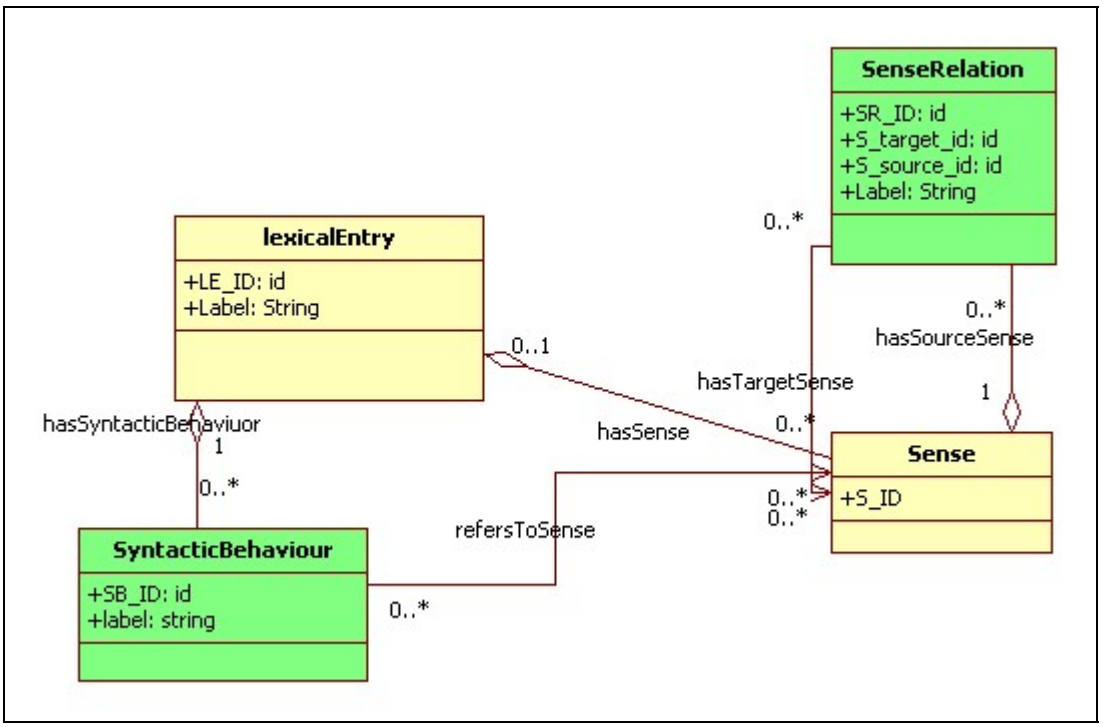


Figure 2-6: Synonymous Entries in the BioLexicon

In the BioLexicon Model, synonymy is defined at the *Sense* level, i.e. one Lexical Entry is a synonym of another Lexical Entry because their *Sense*s are linked to each other by a sense relation of the type isSynonymof.
At the XIF level, however, synonymy is implicit in the cluster structure (i.e. preferred terms in the same cluster are synonyms, and terms marked as "non-preferred" or "orthographic" are variants of the "preferred" one in the same sub-cluster).

2.4 Managing Inflected Forms

Since the need to represent also inflected forms emerged, the BioLexicon DTD and the XIF have been enhanced accordingly. A Wordform element has been added to the XIF DTD, which contain the string of the inflected form and its related grammatical feature3.
Let's consider the part of the BioLexicon model relative to the relation between the following classes: *Lemma*, Wordform, and typed data categories (fig. 2-6 below).
INFLECTEDDC is a typed Data Category that stores information about the grammatical properties of the inflected forms of terms.

[3] The shape and label of this element is not to be considered stable at this stage.

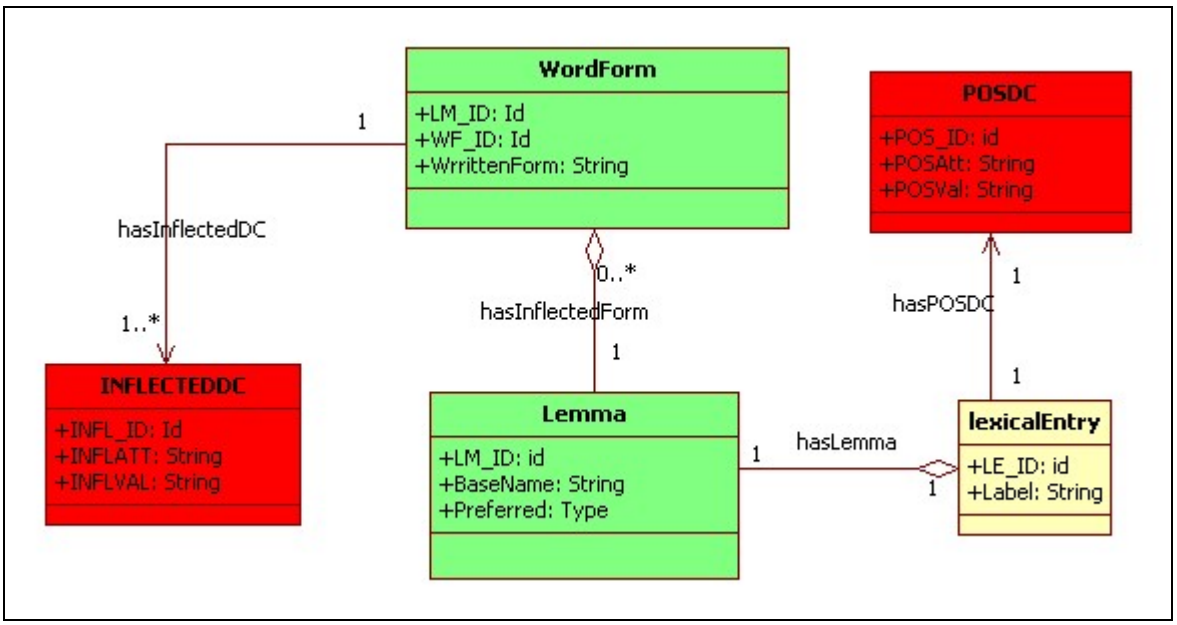


Figure 2-7: lemma, wordforms and DCs

These features will be imported into the BioLexicon DB through the proposed exchange format. The correspondent DTD fragment is the following:

```
<!ELEMENT Entry (….., POSDC,……..,WordForm*,……. )>
 <!ATTLIST Entry
 entryid        ID  #REQUIRED
 BASEFORM CDATA #REQUIRED
 issynonym    IDREF #IMPLIED
 type           CDATA #REQUIRED
 >
……..
<!ELEMENT WordForm (DC*) >
 <!ATTLIST WordForm
 INFLECTEDFORM     CDATA #REQUIRED
 GRAMDATA             CDATA #REQUIRED
 >
```

The following is an example of a valid entry according to the above portion of the XIF DTD:

```
<Cluster CLSID="V1SP12">
                <Entry entryid="V1SP12_1" baseform="carbamylate"
type="PREFERRED">
                        <SOURCEDC sourceName="SwissProt" sourceid=""/>
                        <POSDC posname="POS" pos="V"></POSDC>
                        <WordForm INFLECTEDFORM="carbamylate"
                                GRAMDATA="VVP"></WordForm>
                        <WordForm INFLECTEDFORM="carbamylated"
                                GRAMDATA="VVD"></WordForm>
                        <WordForm INFLECTEDFORM="carbamylated"
                                GRAMDATA="VVN"></WordForm>
                        <WordForm INFLECTEDFORM="carbamylates"
                                GRAMDATA="VVZ"></WordForm>
                        <WordForm INFLECTEDFORM="carbamylating"
```

```
                    GRAMDATA="VVG"></WordForm>
            </Entry>
        </Cluster>
```

3. BioLexicon DataBase Internal Architecture

This section describes the BioLexicon data base (from now BioLexicon) architecture, in particular the data dictionary frame, staging frame and target frame.

Also we'll see how a XIF file (described in the previous section) maps data onto tables.

At the moment, we have developed methods to insert data into tables: the initial loading will be performed using the "load" feature.4 Possibly, these methods could be useful in the future for updating the tables and incremental loading.
First of all, we present a list of database concepts both standard, i.e. typical concepts used in databases environments, and specific, that's to say concepts characteristic of data Ware-House.
Also notations used within the project are shown.
Then we explain the database architecture.

3.1 Typical Data Base Concepts

This paragraph lists some typical data base concepts used in this document.

Standard Concepts

Concept	Explanation
Primary Key (PK)	The unique identifier of a record within a table or a whole data base.
Foreign Key (FK)	A particular column in a table (child) pointing to a particular row in another table (father).
Index	A constraint on a table, usually defined on a PK. It may be unique or not.
Commit	Commit means making a change definitive in one table.
Rollback	Rollback means undoing a change.
Unit of Work (UoW)	The UoW takes care of data consistency managing rollback and commit.

Specific Concepts

Concept	Explanation
Star-Table	Star table is a typical data warehouse concept: simply speaking a star-table is a table whose fact columns are identifiers pointing to external feature-table.
Feature-table	This concept stands for tables of the form identifier-attribute-name-attribute-value.

[4]The LOAD instruction is typical of various databases.

Concept	Explanation
Staging Tables	Staging tables are tables in which data are initially loaded: they have no PK. They may have some indexes to speed up join instruction.
Target Tables	Target tables are the last tables to be loaded. Usually they are similar to staging tables, but are normalized and *starred.*
Initial Loading	In a loading process it represents the first time that the data are loaded into tables. The target tables are empty and the number of rows to load is large.
Incremental Loading	In a loading process it represents any insert/update operation into data base tables. The loading is incremental because some data are present into the tables, and new data only update existent data or simply are added. The target tables are not empty and the number of rows to load is small.
Data Consistency	In a loading process, it is important to care about the consistency of data, i.e. all records must be loaded and all relations must be implemented.

Annotation for the Project

Annotation	Explanation
Dictionary Tables	Dictionary tables are tables containing information about the mapping between the xml and the staging tables. Also, they contain information about identifier generation and the mapping with real entities.
Standard Tables	These are the tables of the BioLexicon model; actually they are a *realization* of the BioLexicon model.
User Tables	These are tables external to BioLexicon model, but useful to record relation between Standard Tables.
Rules tables	Rules tables store information about the mapping of the staging tables and the target tables.
Audit Tables	Audit tables store information about the flux of the loading process.[5]
Exception Tables	These tables store data that are not well-formed to be loaded into a given table.[6]

[5] These tables are for future use only.
[6] Exception tables are automatically managed (for a lot of data base engines) by load instruction, but must be manually managed in insert/update operations.

3.2 DataBase Architecture: Three-frame Architecture

This paragraph describes the architecture chosen to implement the BioLex.
Possible Enhancements of this architecture for future uses and for parallelizing features are also shown.
Figure 2-1 below shows the BioLex architecture

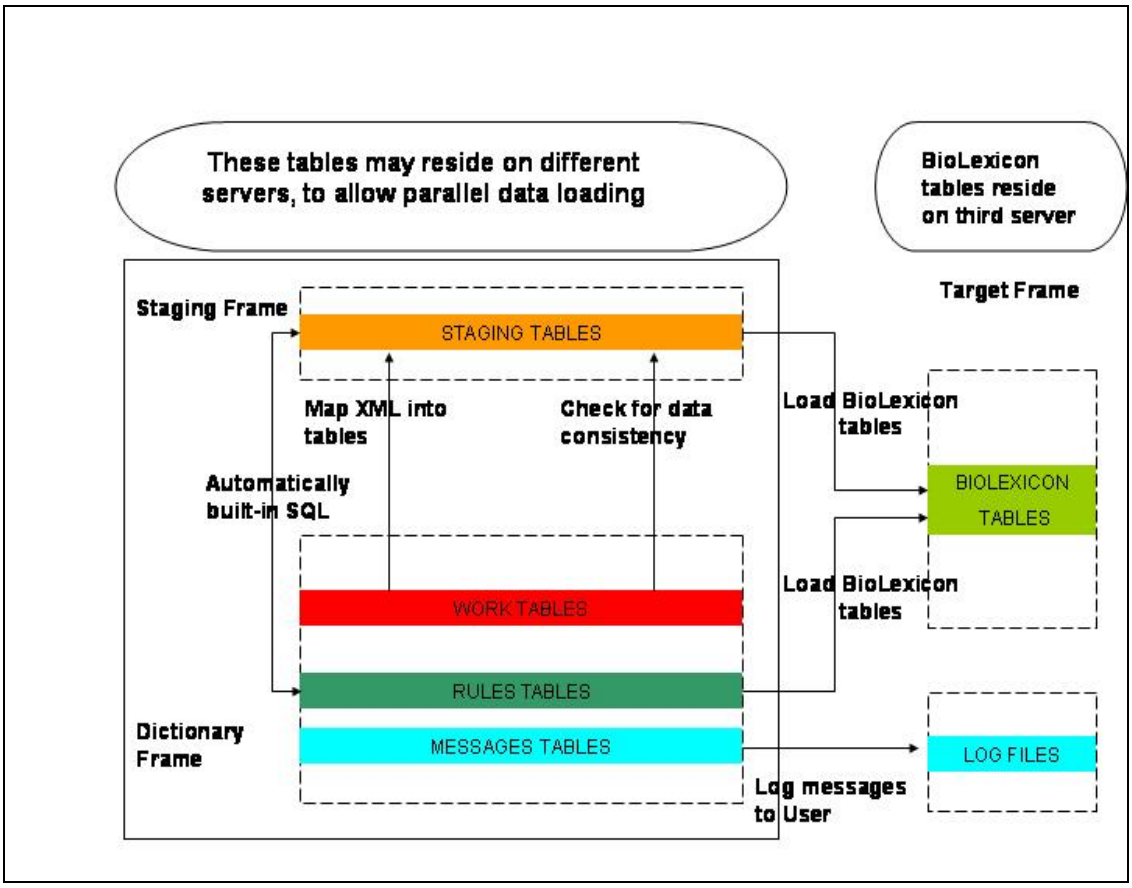


Figure 3-1: Architecture

The 3-frames Structure

The database is structured into three logically distinct but strongly interconnected layers (see fig. 2.1 above):
It has been chosen to logically separate the BioLexicon into three conceptual frames:

- Dictionary tables
- Staging tables
- Target tables

The DICTIONARY FRAME is a middle layer between staging and target frames. Tables belonging to this layer are dedicated to the mapping of the XML Interchange Format elements and attributes onto the staging tables, and it contains the rules used to automatically build SQL instructions that populate target tables with data contained into staging tables.
The STAGING FRAME level is an intermediate set of hybrid tables for volatile data: staging table columns consists of attributes of the XIF and attributes of target tables. In addiction staging frame is the level dedicated to data-cleaning and normalization.
The TARGET FRAME layer contains the actual BioLexicon tables, i.e. tables that directly instantiate the lexical objects and the relations between objects in the BioLexicon meta model.

We separated the whole database into three different levels both because of the peculiar format of the xml exchange format. Besides, target tables are not in a one-to-one relation with the staging tables.

- Dictionary tables contain information about “how-to-fill” staging and target tables, their joins, input files, and so on.
- Dictionary Frame is a middleware between the XIF and the BioLexicon database. The loading software parses the XIF applying rules defined in the Dictionary Frame. These rules drive the software to load data into the right tables.

3.3 Dictionary Frame

This frame contains functional tables such as:

- tables mapping between xml elements (and attributes) and BioLexicon tables (and columns);
- tables with rules to populate target tables;
- tables with messages;
- tables for user interfaces for the end user7

The dictionary frame is thought to be manually updated, with respect to the mapping and rules information: the format of the xml exchange format may vary, adding or deleting elements or attributes.
Face to the need to add a new table (i.e. a new lexical object), or a new attribute to an existing table, it is sufficient to add only the properties of the new tables or attributes and to modify accordingly only the relevant portion of the Dictionary Frame layer to handle the novelties.

Work Tables

Work tables store information and instructions that allows for reasoning on the xml file. In particular, their function is to drive the software to build a given output from a given input. These tables also manage both standard tables and user tables.
Work tables map XIF elements onto the staging tables and XIF attributes into tables’ columns.
In addiction these tables rule whether an element has an identifier or it does not; map prefixes to define table identifiers and so on.
An exhaustive list of defined Work Tables and their use is provided in appendix C.

Rule Tables

Rule tables lead the loading process from the staging tables to target tables.
As said before, there is no one-to-one relationship between staging and target tables, so we need, for loading purposes, a map between sources and targets to automatically know whether a record is already in the BioLexicon or not.
In addiction , this set of tables drive the software to build SQL instructions automatically.
An exhaustive list of defined Rule Tables and their use is provided in appendix C.

Example

This example shows how one single target table (i.e. LexicalEntry) is related to many staging tables.

[7] These tables will serve to the GUI to display the user labels, comments and data in a more human-readable way.

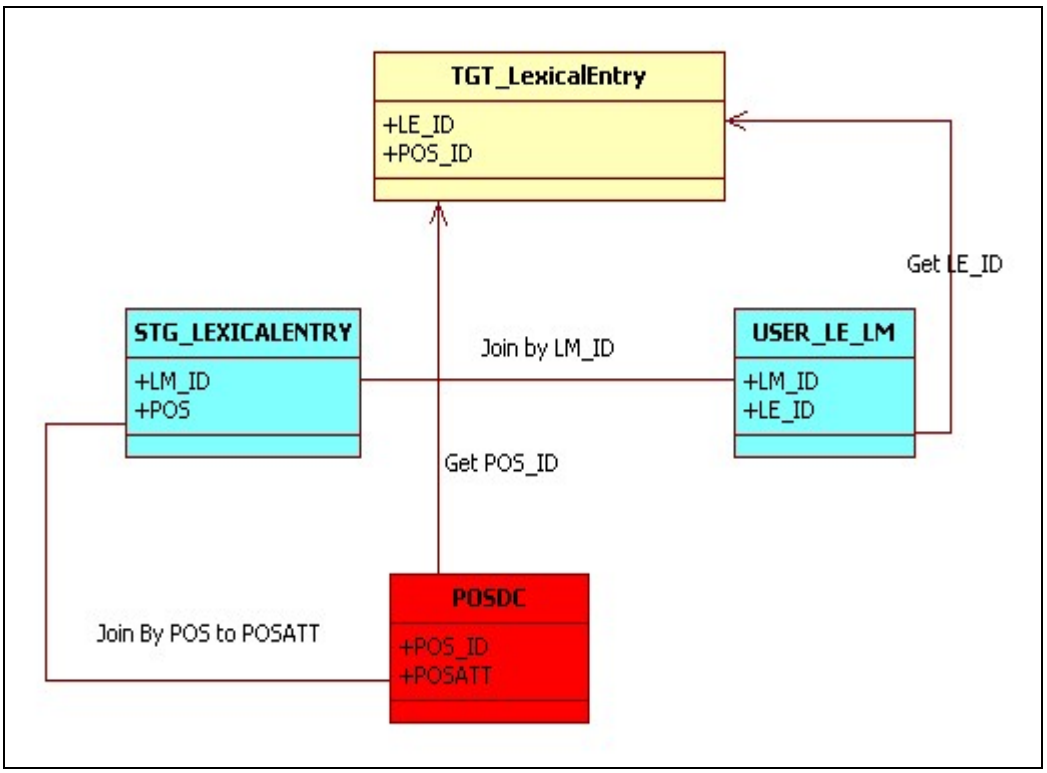


Figure 3-2: Lexical Entry Loading Example

Messages Tables

Message tables are designed from which to retrieve text messages from error identifiers. Message tables are useful also to manage multilingual text messages.

GUI tables

In the next release of the software we'll provide a GUI.
The GUI will help users search for information. It will rest on configuration tables.

Graphical User Interface tables record labels and comments linked to target tables.
We decided to separate typical lexical information (such part of speech, data category, lemmas, used basically by an application) from end user information such as labels and comments.
These labels and comment may be written in native languages. TO BE BETTER SPECIFIED.

3.4 Staging Frame

Staging Frame is a middle tier between the input xml file and the final target tables; it is a layer that maps the input data onto the target tables; also, the staging frame takes care of cleaning tables and of data consistency.
The structure of staging tables is quite similar to the structure of target tables, both for standard and for user tables. Obviously the differences (TO BE DETAILED) are due to the fact that not all the information needed for target tables are easily retrieved from the input file (As fig 2.2 explains). So, it is easier to load an intermediate level.
Each staging table has its corresponding exception table that records not well-formed data.
These exception tables, at this level, allow the loading process to check for data error and for rubbish entries.

Since the column type for staging and target tables are the same, data loaded from staging to target tables will be, surely, correct and clean.

Staging Frame and initial loading

Many data bases use the “load” method to load data into tables.
This method has a parameter [INSERT|REPLACE] that tells the data base engine to clear table before loading new data [REPLACE] or to append new data to existent ones[INSERT].
Staging tables must be cleaned out before loading new data, so we’ll use the load method with [REPLACE] option. 8

Staging Frame and incremental loading

During incremental loading, staging tables must be cleaned to allow new data be checked before loading them into target tables.
Moreover, at this time we’ll use the load method with [REPLACE] option in order to record only new data.9
The staging frame is useful to manage quasi-volatile tables, i.e. tables that exist only for a purpose in the process but that can be truncated (cleared) after data they contain are read.

Data consistency

The staging frame, together with the WORK_LIST_TABLES10, manages data consistency. The WORK_LIST_TABLES records information on the insert order of the tables: the inserting order is the order in which tables are loaded. The lower the insert order the greater the priority.
If a table with a given priority is successfully loaded then the following table may be loaded, and, conversely, if a table of a given priority is unsuccessfully loaded then the previous tables must be roll-backed.11
Following the record in the (simplified) WORK_LIST_TABLES:

WORK_LIST_TABLES				
XMLNAME	**TABNAME**	**INSERTORDER**	**DEEP**	**FLAG**
CLUSTER	STG_USER_SYNCLUSTER	1	2	1
ENTRY	STG_LEMMA	1	4	1
POSDC	STG_*LEXICALENTRY*	1	8	1
VARIANT	STG_*FORMREPRESENTATION*	2	8	1

Work list tables

it may be decided, for example, not to load the STG_LEXICALENTRY table if the STG_LEMMA has not been successfully loaded.

Staging tables list

This paragraph lists all staging tables designed for the loading process.

[8] In mySQL data bases these two terms assume different meaning: the replace keyword means. for example: “replace the old record with this one only if the have the same PK” .
[9] in mySQL data base tables must be “truncated” before loading new data.
[10] See appendix A
[11] The roll-back depends on which table is loaded.

The description of the columns is provided in Appendix D

Staging Table	Type	Explanation
STG_CLSDC	User	This table contains relationships between clusters and their data categories. This table is filled only if a given cluster has data categories.
STG_FORMREPREENTATION	Standard	This table contains all variants for a given lemma.
STG_LE_SOURCEDC	User	This table contains relationships between lexical entries and their source data category. This table is filled only if a given lexical entry has source data categories.
STG_LEDC	User	This table contains relationships between lexical or lemmas d their data categories. This table is filled only if a given lexical entry has data categories.
STG_LEMMA	Standard	This table contains lemmas and their attributes.
STG_*LEXICALENTRY*	Standard	This table contains the lexical entries and their part of speech.
STG_REL_TYPE	User	This table contains the type of the (semantic) relations that a given entry has with other entries.
STG_SENSE	Standard	This table contains the senses of the lemma.
STG_SENSERELATION	Standard	This table contains the sense relationships between lemmas: typical relations are synonymy, ISA….
STG_USER_SYNCLUSTER	User	This table contains cluster attributes.
STG_VARIANTDC	User	This table contains relationships between variants and their data categories. This table is filled only if a given variant has data categories.
USER_CLS_LM	User	This table contains links between cluster and lemmas belonging to the cluster.
USER_FR_LM	User	This table contains links between lemma and its variants.
USER_LE_LM	User	This table contains links between lemma and lexical entry. The relation is one to one.

Data categories

This paragraph explains which data categories have been typed, that is specialized and why.
In addiction, one description of table is provided.
A Data Category is a linguistic constant, representing the basic linguistic notions: it is either an attribute name like e.g. /partOfSpeech/ or a value dedicated to populate the attribute, e.g. /noun/. Data categories represent the main building blocks which, in combination with the structures of the lexicon data-model, make it possible to design different possible lexical entries as instances of the abstract schema.
For software perspective, we found appropriate to provide a set of “typed”-data categories. During these concrete exercises, for the sake of consistency and correctness, we start defining sets of attribute-value pairs specific for different lexical layers and/or objects.

Typed Data categories

The BioLex contains many relations between entities (tables) and their attribute-name-value features. To simply the loading process we defined some typed data categories.

The table below shows these typed DC.

Type Data Category	Type	Explanation
POSDC	Typed	This data category contains all possible value of the part-of-speech attribute.
VARIANTDC	Typed	This data category contains all possible value of the variant types attribute: some values are acronym, orthographic…
RELDC	Typed	This data category contains all the relationships defined in the model, or provided by the input file[12].
SEMDC	Typed	This data category contains semantic classes so far defined.
SOURCEDC	Typed	This data category contains all the sources provided by the input file.

An exhaustive list of typed data categories is provided in Appendix D.

[12] The data categories have been loaded retrieving data from input.

3.5 Target Frame

This frame is the ctual BioLexicon; even if many tables have the same structure as the corresponding staging tables, some of target tables have a more enriched structure, since there is a many to many relationship between staging and target tables.
Exception tables have the same structure as the corresponding target tables and should contain only records violating the PK constraints, since any rubbish entries have been skipped at the staging frame level.

Target Frame and initial loading

Even target tables will be loaded using the “load” method; the target tables must not be cleared before inserting new data, so the loading option is [INSERT].

Target Frame and incremental loading

To be provided for next release.

Data consistency

Data consistency is also present in the target frame: flow and process are similar to those presented for staging tables.

Target tables list

This paragraph lists all target tables designed for BioLex.
The table below explains these tables:

Staging Table	Type	Explanation
TGT_USER_CLS_DC	User	This table contains relationships between clusters and their data categories. This table is filled only if a given cluster has data categories.
TGT_FORMREPREENTATION	Standard	This table contains all variants for a given lemma. The table contains also the preferred (basename) of the lemma as variant.
TGT_LE_SOURCEDC	User	This table contains relationships between lexical entries and their source data category. This table is filled only if a given lexical entry has source data categories.
TGT_LEMMA	Standard	This table contains lemmas and their attributes.
TGT_*LEXICALENTRY*	Standard	This table contains the lexical entries and identifiers of their part of speech.
TGT_SENSE	Standard	This table contains the senses of the lemma and their semantic types.
TGT_SENSERELATION	Standard	This table contains the sense relationships between lemmas: typical relations are synonymy, ISA…\\\.
TGT_USER_FR_DC	User	This table contains relationships between variants and their data categories. This table is filled only if a given variant has data categories.
TGT_USER_LE_DC	User	This table contains links between lemmas and their data categories.
TGT_USER_FR_LM	User	This table contains links between lemma and its variants.

Staging Table	Type	Explanation
TGT_USER_LE_LM	User	This table contains links between lemma and lexical entry. The relation is one to one.
TGT_USER_S_LE	User	This table contains relations between senses and lexical entries.

Target Tables columns are provided in Appendix D.

4. Software Architecture

This section describes the methods designed in the Software. This SW is responsible for transforming and loading data from an xml format in the DB.
For the prototype we used MySql™ as DBRMS.on Microsoft™
The software (since now we refer to the software as LD13) LD has been developed in Java, using the JDOM technology to manipulate the xml document.
Please, refer to section 1 to get more information on the xml format.

4.1 Main flow

The figure below shows the main flow of the loading process.

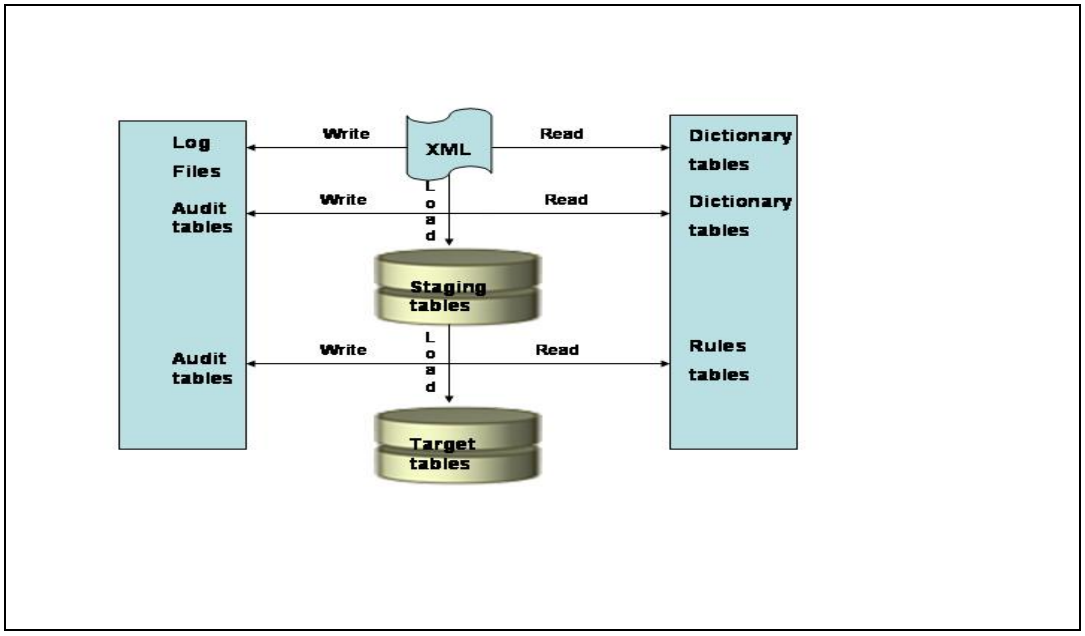


Figure 4-1: Main Flow

The loading process may be divided into three sub processes:

- Read xml file
- Load staging tables
- Load target tables14

Reading xml file

[13] LD stands for **L**oad **D**ata
[14] The Load Target sub process involved some peculiar methods.

This sub process reads the input file, and, retrieving information from dictionary tables, prepares input flat files to load staging tables.
Additionally, the sub process writes some log files.

Loading staging tables

This sub process reads the input flat files, and, retrieving information from dictionary tables loads data into staging tables.
Additionally, the sub process fills some exception tables[15] with data not well formed.

Loading target tables

This sub process loads data from staging tables into target tables; the mapping between staging and target tables are stored into rules tables, defined into the dictionary tables.
Additionally, the sub process fills some exception tables with data violating the primary keys constraints.
Target tables contain:

- standard tables, i.e. tables connected to the BioLexicon model;
- user tables to map relationships between standard tables;
- special tables, i.e. target table (standard or not) that need a particular loading process;
- final tables, i.e. tables that must be re-filled with new data after the loading process is terminated.

4.2 Packages

This section describes packages defined within the LD software.
Five packages has been shaped to accomplish to the project prerequisites.

Packages	Classes Defined within
org.bootstrep.BioLex	LoadXml2DB ManageTables
org.bootstrep. JdomMethods	AccessDataDictionary manageJDomDoc
org.bootstrep.Logger	Logger
org.bootstrep.manageTables	LoadStagingTables LoadTargetTables
org.bootstrep.Variables	ListOfVariables

Variables

This package contain only a class; this class contains no methods, but only variables and constants.

Class Name	Explanation
ListOfVariables	A simple set of public variables and constants.

[15] This operation is not yet outlined, even if potentially not well formed input lines have been trapped by exception in java methods.

Please refer to the javaDoc section under the web site
http://www.ilc.cnr.it/BootStrepD22

For software purposes we defined the following Java List objects, in order to store, step by step, into them, information about data base records and their relations with files and tables.

List	Notes
mainDeep	Contains the deep of the element
mainVariables	Contains the attribute name – attribute value relation for an element
insertStrings	Contains strings as “Insert into table (column-1….column-n)
insertTables	Contains the name of the table corresponding to the element
insertRelStrings	Contains strings as “Insert into table (column-1….column-n) for user tables only
insertRelTables	Contains the user relational tables
insertRelValues	Contains identifier’s values of related entity
fileNames	Contains data files of standard tables
fileRelNames	Contains data files of relational tables
stgExceptionFiles	Contains the name of the exception file for a given staging table
tgtTables	Contains the list of target tables
tgtInsert	Contains strings as “Insert into table (column-1….column-n) for target tables
tgtFiles	Contains the list of input files for target tables
tgtExceptionFiles	Contains the name of the exception file for a given target table

JdomMethods

This package contains two classes:

Class Name	Explanation
AccessDataDictionary	A complete set of methods to access the Data Dictionary Frame
manageJDomDoc	A complete set of methods and procedures to allow the user to access a given XML document, to load its content and to map its data into a data base format.

For more details about methods defined for these two classes, please refer to the javadoc in the web site: http://www.ilc.cnr.it/BootStrepD22

BioLex

This package contains two classes:

Class Name	Explanation
LoadXml2DB	A complete set of methods and procedures to allow the user to manipulate a given XML document to load its content into an external data base.
ManageTables	

Class Name	Explanation
	This class initializes "LoadStagingTables" , manageJDomDoc and LoadTargetTables classes

For more details about methods defined for these two classes, please refer to the javadoc in the web site http://www.ilc.cnr.it/BootStrepD22

Logger

This package contains only one class:

Class Name	Explanation
Logger	A complete set of methods to manage log files. This class declares a ListOfVariables instance to manage global variables.

For more details about methods defined for this class , please refer to the javadoc in the web site: http://www.ilc.cnr.it/BootStrepD22

Package: manageTables

This package contains only one class:

Class Name	Explanation
LoadStagingTables	This class manages loading data into staging tables.
LoadTargetTables	This class manges loading data into target tables.

For more details about methods defined for this class , please refer to the javadoc in the web site: http://www.ilc.cnr.it/BootStrepD22

4.3 How LD works: from the XIF to target tables

This paragraph describes how LD loads data into target and special tables.
In the web site http://www.ilc.cnr.it/BootStrepD22 you may find a complete log about "Enzyme Loading Process"
In what follows we presents a simplified version of loading process. The three sub-processes defined above mark each sub-section.

Let's consider the following XML:

```
<Cluster CLSID="CLS0001" SEMTYPE=”Protein Name” >
        <Entry ENTRYID="ENTRY0001" BASEFORM="Interleukin-2"
TYPE="PREFERRED" >
                <POSDC POSNAME=”POS” POS="N"></POSDC>
                <Variant WRITTENFORM="IL-2" TYPE="ACRONIM"></Variant>
                <Variant WRITTENFORM="IL2" TYPE="ACRONIM"></Variant>
        </Entry>
</Cluster>
```

This fragment will load the following tables:

Table	Type
TGT_LEMMA	Target
TGT_*FORMREPRESENTATION*	Target
TGT_*FORMREPRESENTATION*	Special
TGT_USER_FR_LM	Special
TGT_SENSE	Special
TGT_SENSE_RELATION	Special

From XIF to input Files

Methods involved is this sub process create some flat files, each files contain strings like below:

File	Record
LEMMA	**'LM_Interleukin-2','Interleukin-2','PREFERRED'**
LEXICALENTRY	**'LM_Interleukin-2','POS','N'**
FORMREPRESENTATION	**'FR_IL_2','IL-2','ACRONIM'**
FORMREPRESENTATION	**'FR_IL2','IL2','ACRONIM'**
USER_LE_LM	**'LM_Interleukin-2', 'LE_Interleukin-2'**
USER_FR_LM	**'LM_Interleukin-2','FR_IL-2'**
USER_FR_LM	**'LM_Interleukin-2','FR_IL2'**

Steps

- LD software initializes the platform getting the values of some properties (the data source and the xml file name);
- LD software tests the connection and checks the file existence;
- LD software instantiates the JDOM document and retrieves the root element;
- LD software begins to parse the document's nodes;

The first node analyzed is the <Cluster>, from its attributes LD retrieves the CLUSTERID and the SEMTYPE values; the former is important being the postfix of the flat files, the latter is the semantic category entries belong to. From dictionary frame the LD retrieves table structure of <Cluster> element. The second node corresponds to <Entry> element. Other nodes are <POSDC> and <Variants>.

For each node:

- LD software creates the "insert into" strings from dictionary information and fills insertStrings and insertTables lists;
- LD software decides (from dictionary) if a node has an identifier or inherits an ID from the parent;
- LD software creates files;

When standard table list is totally read the LD starts analyzing user table:

- LD software creates the "insert into" strings from dictionary information and fills insertRelStrings and insertRelTables lists;
- LD software creates "insert into" strings for user relational tables:
- LD software creates files;

From input files to Staging tables

Methods involved in this sub process load data into some staging tables from input flat files:

Table	From XML entity	Type
STG_LEMMA	ENTRY	Standard
STG_*LEXICALENTRY*	POSDC	Standard
STG_*FORMREPRESENTATION*	VARIANT	Standard
USER_LE_LM	POSDC	Relational
USER_FR_LM	VARIANT	Relational
USER_CLS_LM	ENTRY	Relational

Steps

From dictionary frame LD retrieves the list of staging table to load and their insertion order.
From lists, LD retrieves the stack Insert-Into, table, file to manage.

- LD software starts loading relational tables;
- LD software clears staging tables;
- LD software manages exception tables for bad-formed entries[16];
- LD software manages standard tables;

From Staging to Target tables

Methods involved in this sub process load data from staging tables into target tables:

Table	Type
TGT_LEMMA	Target
TGT_*FORMREPRESENTATION*	Target
TGT_*FORMREPRESENTATION*	Special
TGT_USER_FR_LM	Special
TGT_SENSE	Special
TGT_SENSE_RELATION	Special

From Dictionary Frame (Rule tables) LD retrieves the mapping between staging and target tables, for example, about the TGT_*LEXICALENTRY*:

RULE_MAP_STG_TO_TGT		
TGT_TABLE	**STG_TABLE**	**STGORDER**
TGT_*LEXICALENTRY*	USER_LE_LM	1
TGT_*LEXICALENTRY*	STG_*LEXICALENTRY*	2
TGT_*LEXICALENTRY*	POSDC	3

Steps

- LD software gets information on some properties and retrieves table list;
- LD creates the stack for tables;
- LD software inserts into table;

[16] Only if a specific parameter is set to true in the property file.

- LD software manages variants;
- LD software manages special tables: TGT_SENSE, TGT_USER_FR_LM.
- LD software manages final table: TGT_SENSERELATION
- LD software stops processing.

5. Concepts

In this document we have used many times the “deep” concept, information lining-up, that is the stacks and information retrieval form dictionary frame.
This section provides a description of these concepts.

5.1 “Deep” concept

This paragraph describes the *deep* concept and how it is used in parsing xml document.
Deep is a number that simulates the graphical indentation of elements in xml file:

```
<level1>
        <level2/>
                <level4>
</level1>
```

Deep, as number, is 2 exp {the child level of the element}; in the example above: 2 exp {0} (that is 1) for level1 element, 2 exp {1} (that is 2) for level2 and 2 exp {2} (that is 4) for level3.

The DTD and the XML fragment

The following dtd provides some guidelines to validate XIF.
For example this dtd states that the root element has at least one child; and that that child has at least one child, that has at least one child:

- inputData has at least one Cluster;
- Cluster has at least one Entry;
- Entry has at least one POSDC;

```
<!ELEMENT INPUTDATA (CLUSTER+)>
<!ATTLIST INPUTDATA
   DTDVERSION CDATA     #FIXED "1.1">
<!ELEMENT CLUSTER (ENTRY+,DC*)>
<!ATTLIST CLUSTER
CLSID ID #REQUIRED
SEMTYPE CDATA #REQUIRED >
 <!ELEMENT ENTRY (SOURCEDC*,DC*,POSDC,GRAMMATICALDC*,VARIANT* )>
 <!ATTLIST ENTRY
 ENTRYID ID  #REQUIRED
 BASEFORM CDATA #REQUIRED
 ISSYNONIM IDREF #IMPLIED
 TYPE  CDATA #REQUIRED
 >
 <!ELEMENT VARIANT (DC*) >
```

```
 <!ATTLIST VARIANT
 WRITTENFORM CDATA #REQUIRED
 TYPE CDATA #REQUIRED
 >
…..
```

Since we'll encounter some problems when managing the element DC, that may belong to various elements, with, of course, different meaning, we have to manage these element carefully.
These problems have been resolved by using the *deep* concept.

To introduce the *deep* concept, let's look at the xml fragment below: This fragment is a valid xml file for the above dtd:

```
<!DOCTYPE InputData SYSTEM "BioLex_DataExchange.dtd" >
<InputData>
  <Cluster CLSID="CLS0001" SEMTYPE="Protein name" >
    <Entry ENTRYID="ENTRY0001" BASEFORM="Interleukin-2" TYPE="PREFERRED"
>
       <POSDC POSNAME="POS" POS="N"></POSDC>
       <Variant WRITTENFORM="IL-2" TYPE="ACRONIM">
          <DC att="DCATT1" val="DCVAL1"></DC>
       </Variant>
       <Variant WRITTENFORM="IL2" TYPE="ACRONIM"></Variant>
    </Entry>
    <DC att="CLSDC1" val="CLSDC1"></DC>
  </Cluster>
</InputData>
```

From the above xml fragment we retrieve the following information:

- This xml fragment has five levels;
- More than one element may belong to the same level
- Elements at the same level have the same father;

One way to identify elements with same name but different father is to look at the parent element of the child element.
If the child element is DC, for example in this fragment, to get the parent element may return two different fathers, depending on the level of the DC element within the xml fragment, i.e. depending on which node of the xml tree the software is analyzing.
To get the parent we need a "string equalizing" evaluation between the father's name retrieved and the father's name encoded in the code, to know which operation to perform on the element itself.
One different way to manage this ambiguity is introducing the *deep* concept.
Deep behaves like graphical indentation, that is the more an element is far from left side the greater is its *deep*.
Obviously, *deep* must be chosen in a clever way: we choose the "2 power", exponential of base 2, to manage the *deep* values.

We set the initial *deep* as one (1) (1 is 2^0), and if n=number_of_level, the maximum *deep* is 2^ {n-1}.

In the xml fragment, the tree has five levels, so the maximum *deep* is sixteen (16).

The uniqueness of the element is guaranteed by the *deep* value:
has*Deep*(DC) =4 ↔ isChildOf(DC, Cluster).

In other words the *deep* of DC is 4 (iff) if and only if DC is a child of element Cluster.

Deep value is doubled every time a child node is parsed, and set to initial value when all children of a node have been parsed.
The information about *deep* is explicitly stored in the data dictionary data base, and basically is used to reason about an element and its attributes:
the element, deep 2-plet is a “trigger” that teaches the software what to do.

An example

As stated before the uniqueness of an element is guaranteed by the 2-plet constituted by an element and its deep:

[<Element_name>, <Element_*deep*>].

The xml fragment mapped on this 2-plet is the following:

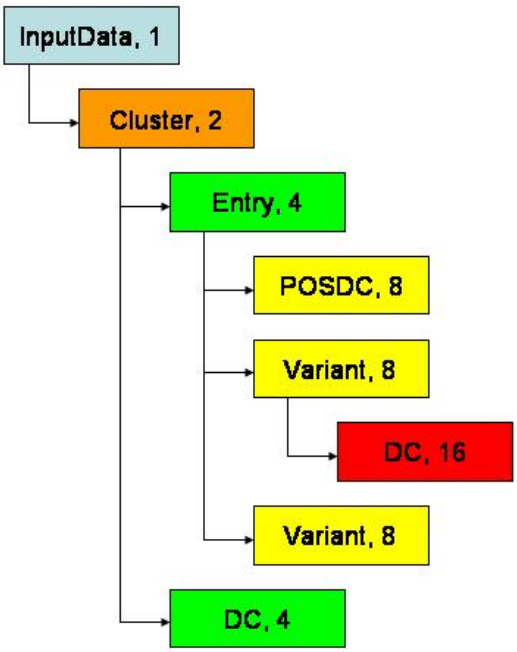


Figure 5-1: Indentation and Deep

Here, different colours mean different levels within the xml fragment, and, at the end, different meanings of the elements.

5.2 Synchronizing information between Dictionary and Staging Frames

The Dictionary and Staging Frames build a synchronized framework that aligns the stack:

- Input file-corresponding staging table-insert into statement

This framework is built using Java List Objects and manages correspondences between a single xml fragment, the corresponding input file and staging table to load. In other words the xml exchange format is fragmented at cluster level, and, since each cluster contains coherent data (such lemmas, sources variants), each fragment defines aligned stacks between tables and input files.
The Work tables set in responsible of this alignment.
Once stacks are aligned staging tables may be loaded, since we are sure to read data from the right input file. See figure 4.2 below,

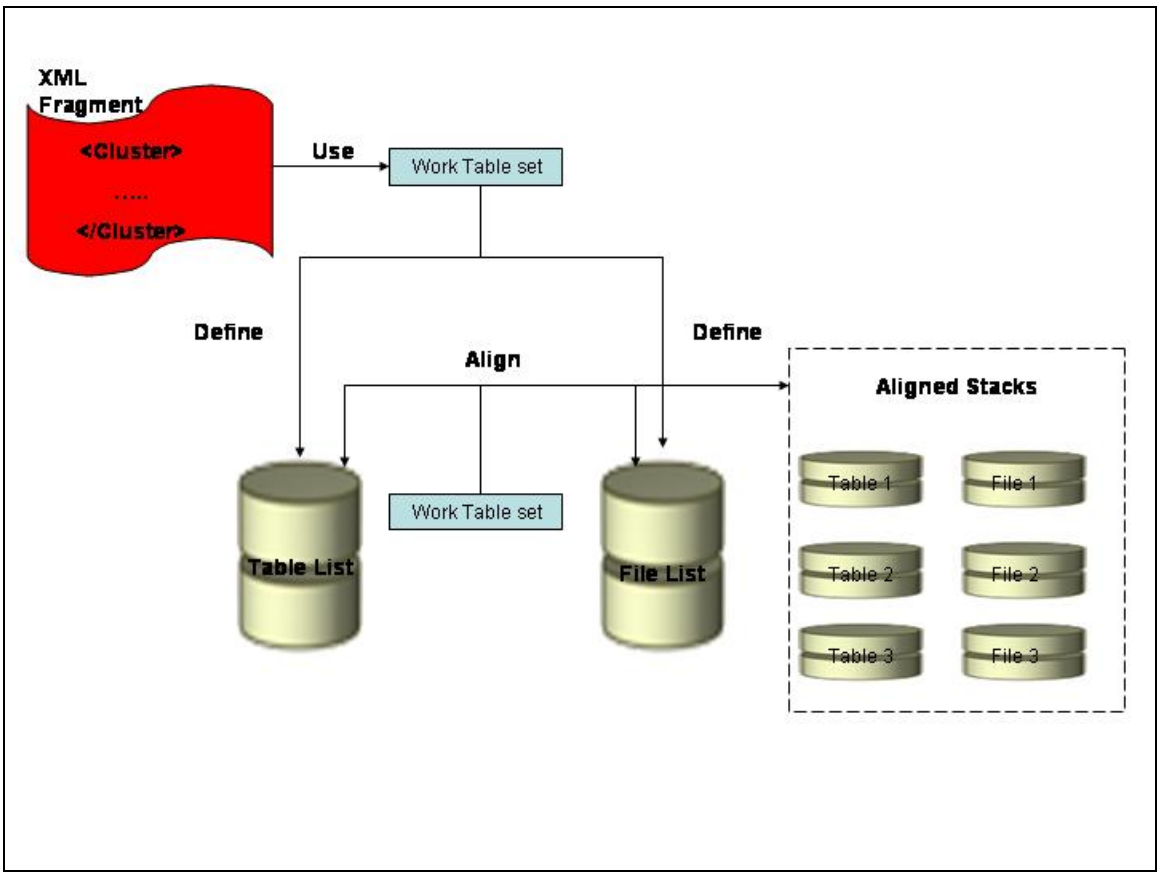


Figure 5-2: Aligned stacks

A similar alignment is used also to manage loading into target frame.

6. A Graphical View of the Environment

The figure below shows the final “status” of the BioLex within the BootStrep propject.
The DICT_TABLE simplifies the dictionary frame, an user accesses the BioLex via a GUI that shows him the information in human readable fashion.
In addiction an application may access the same frame to read and write information into BioLex.
Having a complete separation between typical linguistic information and human readable (the DICT_TABLE maps entities identifier information with their attributes) permit to manage entries in a more direct way.

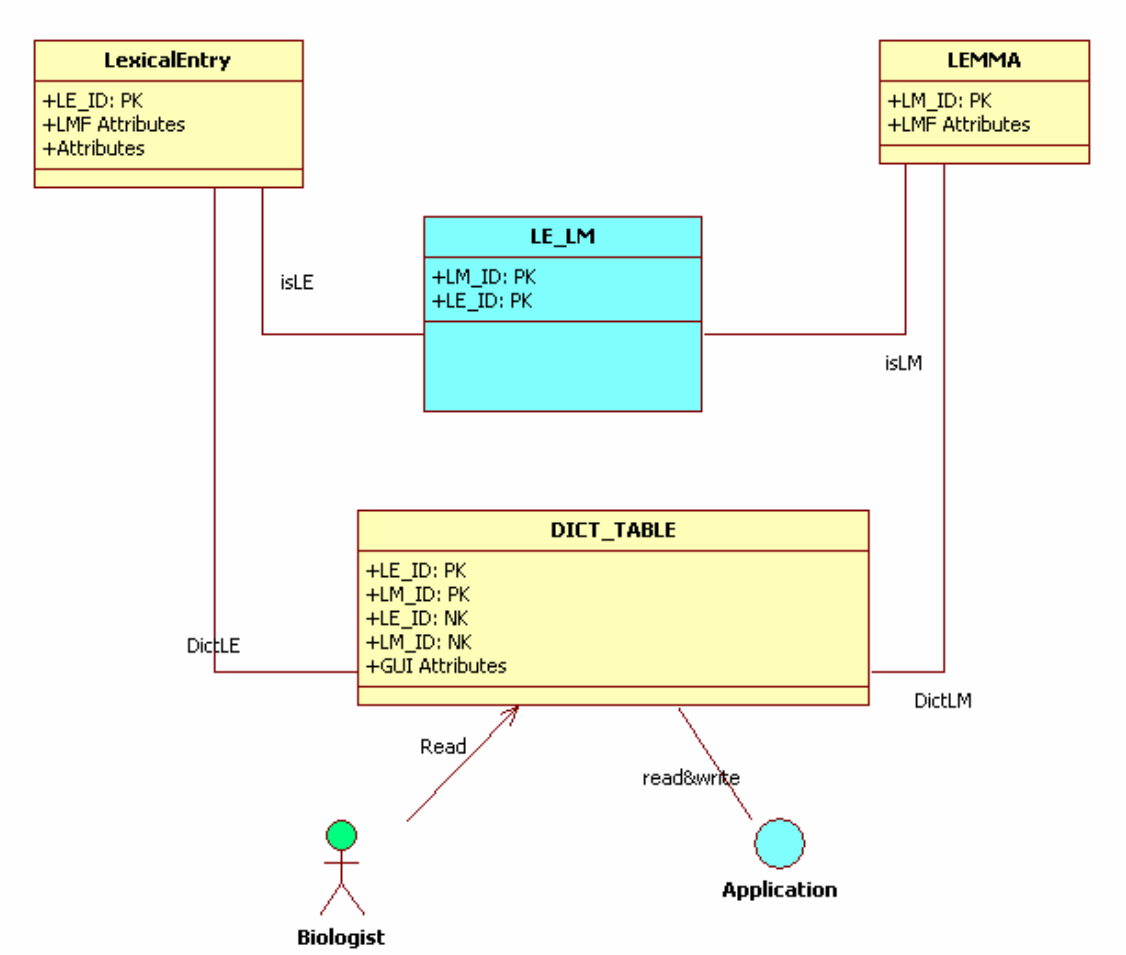


Figure 6-1: Graphical View

Mapping between BioLexicon Entities and BioLexicon Tables

This section explains the differences between the standard and the *user* tables.
We may say that in one BioLex there are two BioLexes:

- tables belonging to the BioLexicon model (standard)
- tables that implement relations between the objects of the model (user)

6.1 Standard Tables

This paragraph shows the model and the corresponding database architecture designed so far.
Two figures are provided: one shows conceptual model in UML and the other one is a view on the physical database tables.

BioLex: Morphological Layer

The model is the following:

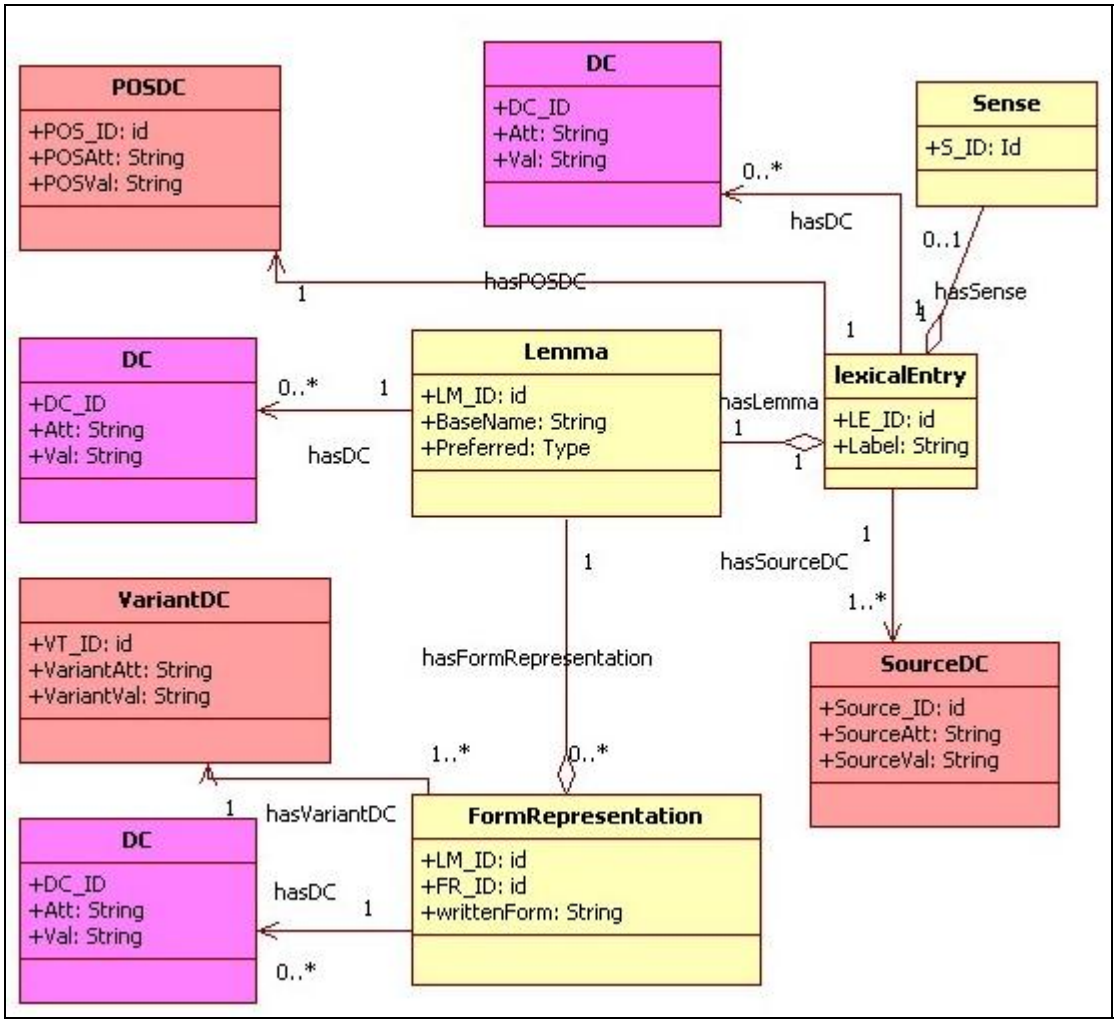


Figure 0-1: BioLexicon: Morphological Layer

Data Base: Morphological Layer

The above model is implemented in the database as follows:

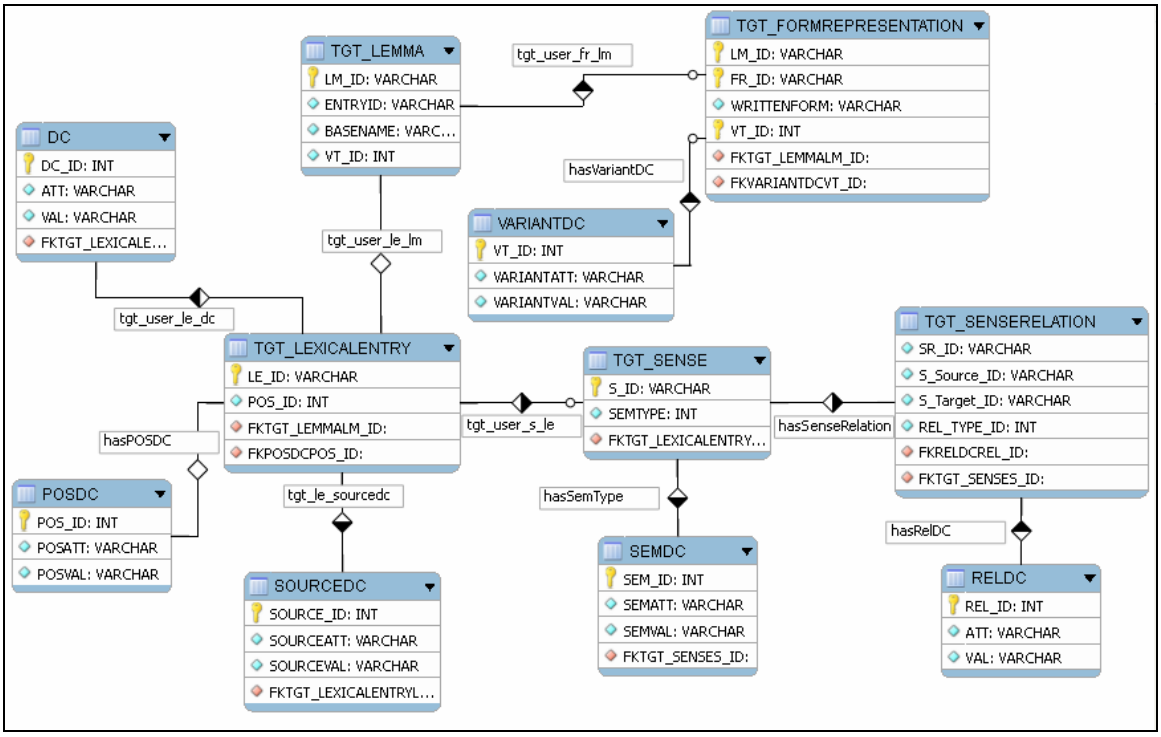


Figure 0-2: Data Base Morphological Layer

Mapping between Properties and Relational Tables

Tables like TGT_USER_* map relation between two standard tables. Below we can find which relational tables map which relation.

Table	Relation	Note
TGT_USER_LE_LM	has*Lemma*	Relation between lemma and lexical entry
TGT_LE_SOURCEDC	hasSourceDC	Relation between lexical entry and type sourceDC
TGT_USER_LE_DC	hasDC	Relation between lexical entry and general DC
	hasPOSDC	Relation implemented within lexical entry
TGT_USER_FR_LM	has*FormRepresentation*	Relation between lemma and form representation
TGT_USER_FR_DC	hasDC	Relation between form representation and general DC

BioLex: Semantic and Syntactic Layer

The model is the following:

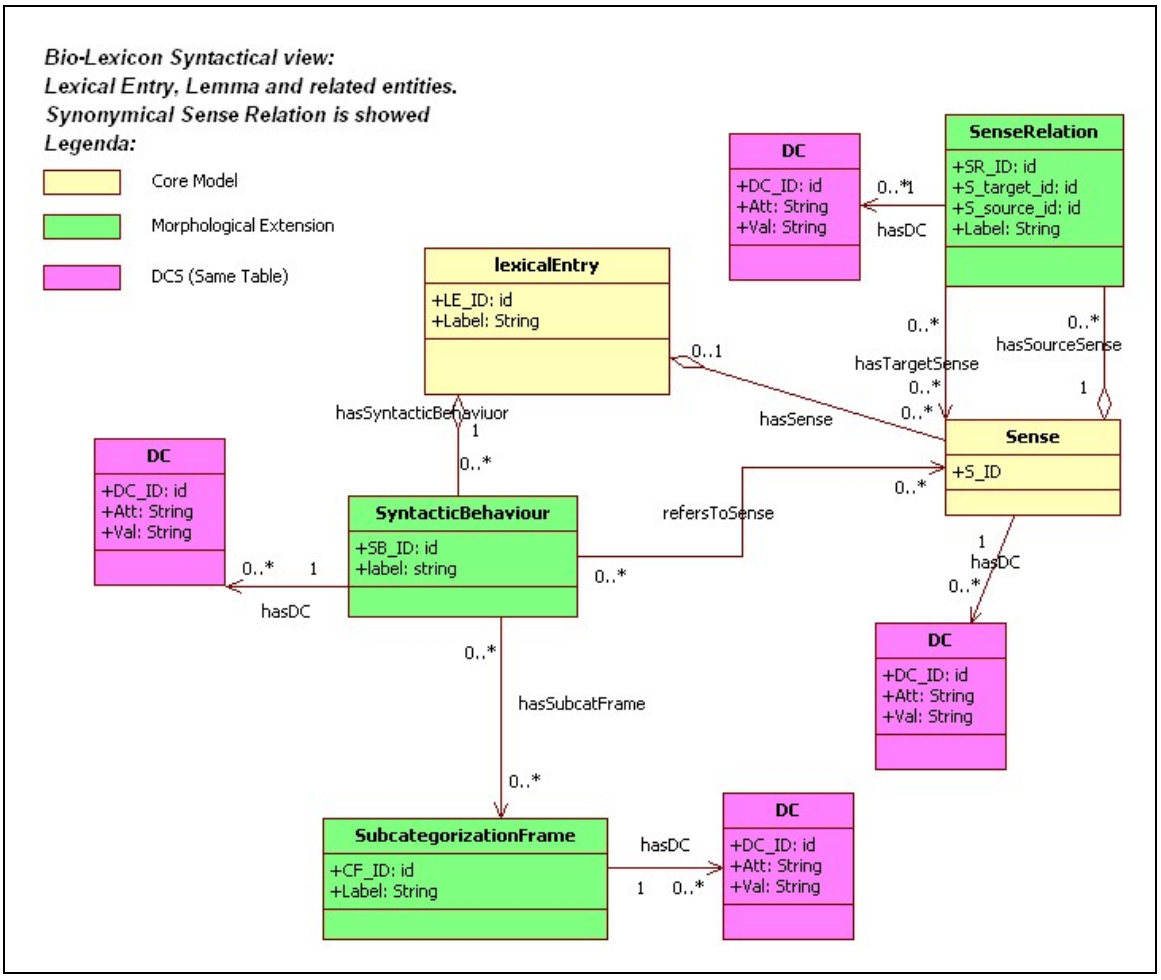


Figure 0-3: BioLex: Syntactic Layer

Data Base: Semantic Layer

The above model is realised in the following data base:

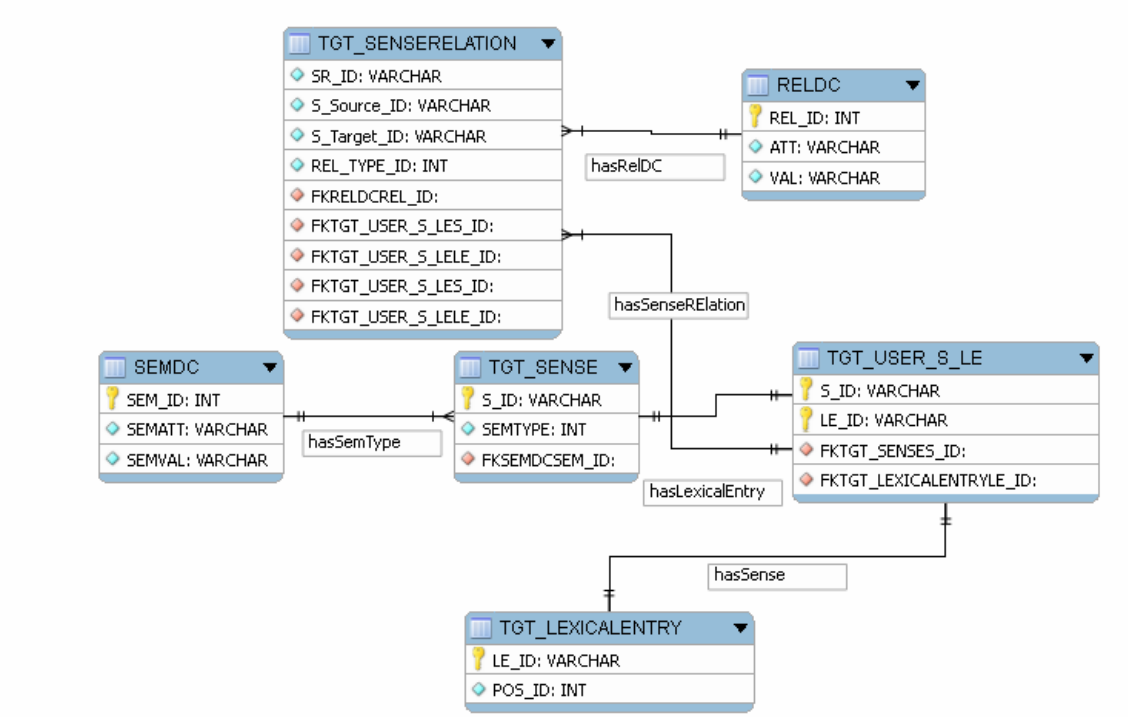


Figure 0-4: Data Base Semantic layer

Mapping between Properties and Relational tables

Tables like TGT_USER_* map relation between two standard tables. Below we can find which relational tables map which relation.

Table	Relation	Note
TGT_USER_S_LE	has*Sense*	Relation between lexical entry and sense
	has*Sense*Relation	Relation implemented within senses. A given sense may be both source and target of sense relations.

Notes on Data Base Implementation

So far not all the tables shaped in figure 7.2 have been realized; in addiction no DC have been provided for sense and sense relation.

6.2 User Extended Models

This paragraph shows the extended models including all the user relational tables. We can see that the user tables complete the standard model.[17]

BioLex: Morphologic Layer User Extension

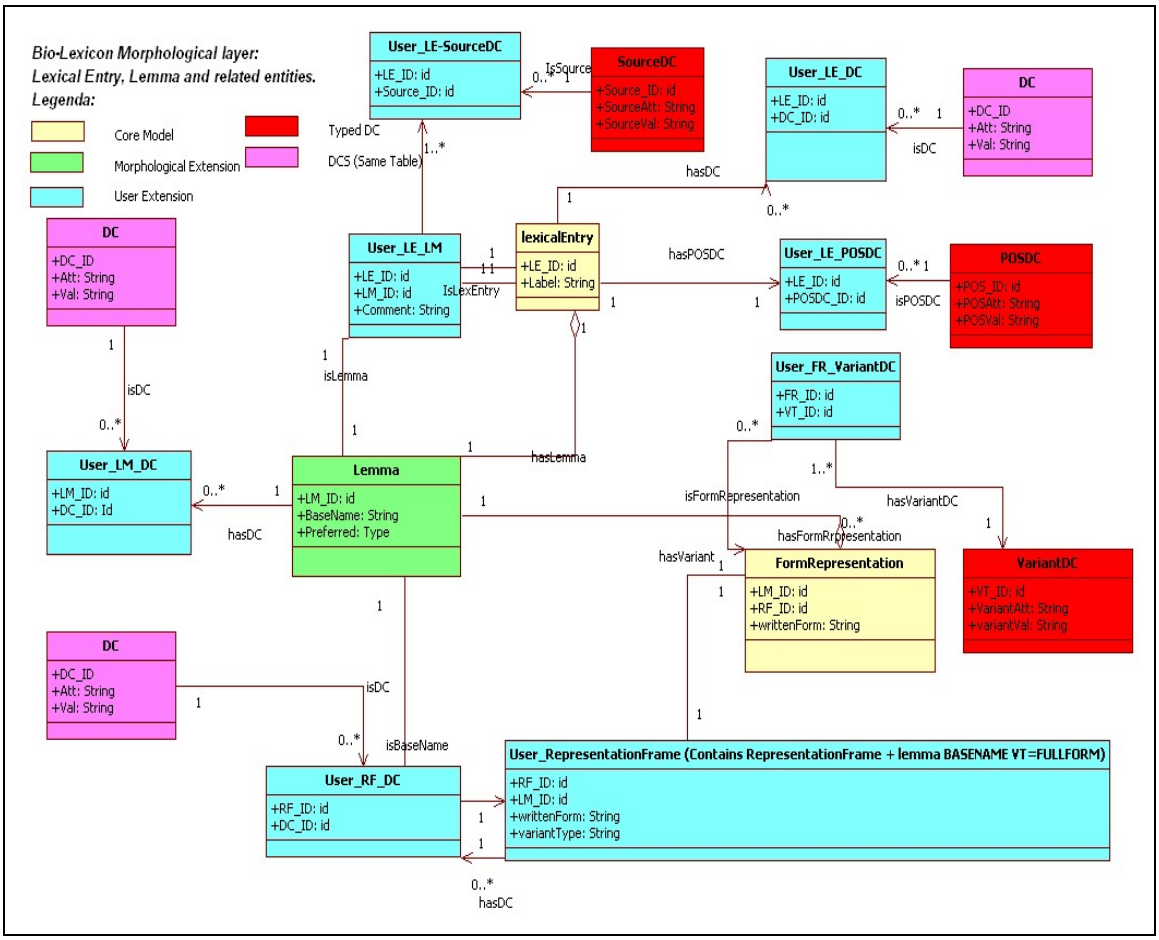


Figure 0-5: BioLex Morphologic User Extension

[17] The term user extension may be misleading: for user extension we mean any relational table realising properties defined in the BioLex Model, plus any other useful table needed for GUI or software reasons.
In the next release we'll change the name of the user tables.

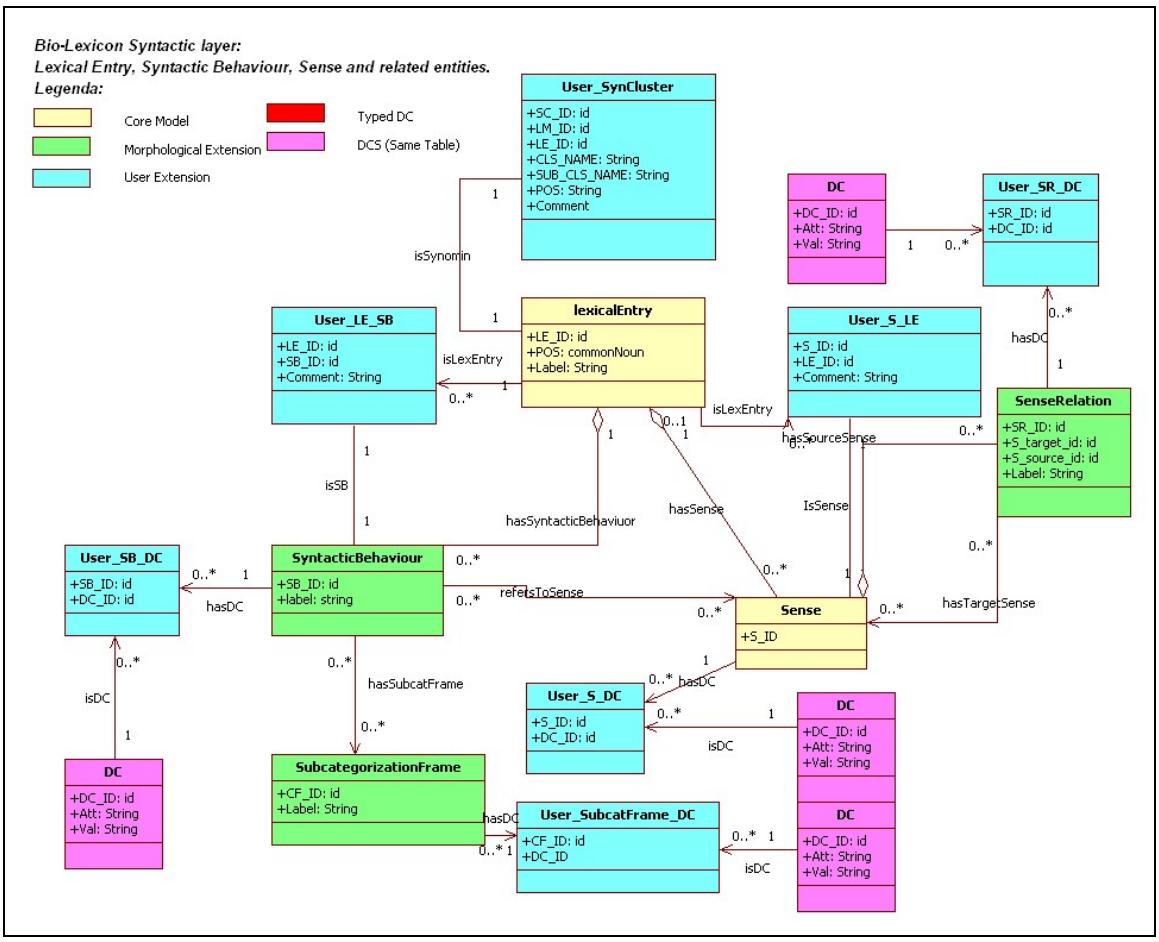


Figure 0-6: BioLex semantic User Extension

6.3 Mapping BioLexicon Model onto Standard Target Tables

The table below shows how the BioLexicon objects have been mapped onto the standard target tables in the BioLex.

BioLexicon Object	Target Table	Note
Lemma	TGT_LEMMA	Morphologic layer
LexicalEntry	TGT_*LEXICALENTRY*	Morphologic layer/Semantic layer
FormRepresentation	TGT_*FORMREPRESENTATION*	Morphologic layer
Sense	TGT_SENSE	Semantic layer
*Sense*Relation	TGT_SENSERELATION	Semantic layer
SyntacticBehaviour	Not Mapped	Not yet treated.
SubcategorizationFrame	Not Mapped	Not yet treated.

6.4 Mapping BioLexicon Model onto Standard User Tables

The table below shows how the BioLexicon objects have been mapped onto the standard target tables in the BioLex.

BioLexicon Extended User Object	Target Table	Note
User_Le_SourceDc	TGT_LE_SOURCEDC	Morphologic user extended layer
User_LE_DC	TGT_USER_LE_DC	Morphologic user extended layer
User_LE_LM	TGT_USER_LE_LM	Morphologic user extended layer
User_RepresentationFrame	TGT_USER_FR_LM	Morphologic user extended layer
User_FR_DC	TGT_USER_FR_DC	Morphologic user extended layer
User_SynCluster	TGT_USER_SYNCLUSTER	Semantic user extended layer
User_S_LE	TGT_USER_S_LE	Semantic user extended layer
User_Le_PosDc	Not Mapped	Morphologic user extended layer: the Pos identifier is a column of the TGT_*LEXICALENTRY*
User_FR_VariantDC	Not Mapped	Morphologic user extended layer: the variant type is a column of the TGT_*FORMREPRESENTATION*.
User_LM_DC	Not Mapped	Morphologic user extended layer: not yet mapped: lack of information.
User_SB_DC	Not Mapped	lack of information.[18]
User_S_DC	Not Mapped	Semantic user extended layer: lack of information.
User_LE_SB	Not mapped	lack of information.[19]
User_SubCatFrame_DC	Not mapped	lack of information.[20]
User_SR_DC	Not mapped	lack of information.[21]

[18] So far syntactic layer has not yet been treated.
[19] So far syntactic layer has not yet been treated.
[20] So far syntactic layer has not yet been treated.
[21] So far syntactic layer has not yet been treated.

Notes on TGT_USER_SYNCLUSTER

The TGT_USER_SYNCLUSTER table is realised to manage in “one-shot” which lemmas are synonyms within a given cluster.

TGT_USER_SYNCLUSTER					
SC_ID	**LM_ID**	**LE_ID**	**POS_ID**	**CLS_NAME**	**SEM_ID**
SC_CLS0001	LM_Interleukin-2	LE_Interleukin-2	1	CLS0001	6
SC_CLS0001	LM_T-Cell-Growth	LE_T-Cell-Growth	1	CLS0001	6
SC_CLS0002	LM_Alcohol dehydrogenase	LE_Alcohol dehydrogenase	1	CLS0002	6
SC_CLS0002	LM_Aldehyde Reductase 2	LE_Aldehyde Reductase 2	2	CLS0002	6
SC_CLS10	LM_fluoride	LE_fluoride	1	CLS10	2
SC_CLS10794	LM_fluoride salts	LE_fluoride salts	1	CLS10794	2
SC_CLS13474	LM_Alcohol dehydrogenase	LE_Alcohol dehydrogenase	1	CLS13474	4
SC_CLS13475	LM_Alcohol dehydrogenase (NADP(+))	LE_Alcohol dehydrogenase (NADP(+))	1	CLS13475	4

In addiction this table is useful to create synonymy in a second step: that is given a new entry belonging to a cluster already loaded, we can implement the synonymycal relation between the new entry and any other entries belonging to the same cluster.

6.5 A Description of the “How-to-Access” Methods to the BioLexicon Database.

The BioLexicon data base defines unique identifiers that let any object within the data base to be independently accessible from queries or software procedures.
The policy of id-building is the following:
We prefix any entry with a set of characters representing the table, entries belong to. For example the same entry *Kynureninase* will become *LM_ Kynureninase* for lemma; *FR_ Kynureninase* for form representation and so on.

The following list shows some of the access methods to the data base:

- BioLexicon may be accessed by variants, i.e. querying *FormRepresentation* table; the id build policy immediately returns the lemma of the variants. From the lemma we can retrieve information on part of speech, relations, senses.
- Biolexicon may be accessed by sense relations, i.e. querying *senseRelation* table; the id-build policy immediately returns relation, the given entry belongs to.
- BioLexicon may be accessed by lemma or lexical entries, to retrieve information on syntactic properties.

7. Conclusions

The BioLexicon constitutes the expected lexical resource of the BOOTStrep project, a resource that integrates features of both terminologies and lexicons, meets the bio domain needs, complies with the most recent standards for lexical representation, the ISO Lexical Markup Framework (Francopoulo et al. 2006a), and is built according to the design principles of state-of-the-art NLP computational lexicons. It constitutes one of the major outcomes and novelties of the BOOTStrep Consortium, with respect to related works, e.g. *Termino* (Harkema *et al.* 2004).

It also is a tangible proof that the state-of-the-art in these fields is mature enough for us to create a terminological-lexical resource which is candidate to become *the* standard for the Bio domain. Although specifically designed for a given domain, thanks to conformity to most accredited international standards, the BioLexicon allows for interoperability with other lexicons and extendibility to other terminological areas.

8. Availability

The BioLexicon and related documentation are available for download[22] at the following URL:

http://www.ilc.cnr.it/BootStrepD22_files/BioLexDataBase.zip

This is a MySQL dump zip file which includes data and DDL. Loaded data are relevant to enzymes.

[22] Because of the restricted nature of this first delivery, the files are password protected. The password will be given to authorised people by the ILC partner.

9. References

[1] Francopulo G., et al. 2006a. Lexical Markup Framework (LMF). *Proceedings of the LREC 2006*, Genova, Italy.

[2] Francopoulo G., Declerck T., Monachini M., Romary L. 2006b. The relevance of standards for research infrastructure. *Proceeding of the LREC 2006*, Genoa, Italy.

[3] Hahn U., Markó K. 2001. Joint Knowledge Capture for Grammars and Ontologies. *Proceedings of the 1st international conference on Knowledge capture* Victoria, British Columbia, Canada.

[4] Harkema H., Gaizauskas R., Hepple M., Angus R., Roberts I., Davis N., Guo Y. et al. 2004. A Large Scale Teminology Resource for Biomedical Text Processing. *HLT-NAACL 2004 Workshop: BioLINK 2004, Linking Biological Literature, Ontologies and Database*, Boston, Massachusetts, USA.

[5] Kors J. A., Schuemie M. J., Schijvenaars B. J. A., Weeber M., Mons B. et al. 2005. Combination of Genetic Databases for Improving Identification of Gens and Proteins in Text, Rotterdam, Netherlands.

[6] ISO-12620. 2006. Terminology and other content language resources – Data Categories – Specifications of data categories and management of a Data Category Registry for language resources. ISO/TC37/SC3/WG4.

[7] Nenadic, G., Ananiadou, S. & McNaught, J. 2004. Enhancing Automatic Term Recognition through Term Variation, in *Proceedings of 20 th Int. Conference on Computational Linguistics, Coling 2004*, Geneva, Switzerland.

[8] Ruimy N., Monachini M., Gola E., Calzolari N., Del Fiorentino M. C., Ulivieri M., Rossi Sergio. 2002. In *Linguistica Computazionale*, Vol.XVIII-XIX, I.L.C. and *Computational Linguistics*, special issue, A. Zampolli, N. Calzolari, L. Cignoni, (Eds.), I.E.P.I., Pisa-Roma.

[9] Wright S.E. 2004. A global data category registry for interoperable language resources. *Proceedings of the LREC 2002*, Lisbon, Portugal.

Appendix A: Complete DTD

```
<?xml version='1.0' encoding="UTF-8"?>
<!ELEMENT InputData (Cluster+)>
<!ATTLIST InputData
    dtdVersion CDATA     #FIXED "1.1">
<!ELEMENT Cluster (Entry+,DC*)>
<!ATTLIST Cluster
CLSID ID #REQUIRED
SEMTYPE CDATA #IMPLIED
>

 <!ELEMENT Entry
(SOURCEDC*,POSDC,GRAMMATICALDC*,Variant*,WordForm*,RELATION*,DC* )>
 <!ATTLIST Entry
 entryid ID  #REQUIRED
 BASEFORM CDATA #REQUIRED
 issynonym IDREF #IMPLIED
 type  CDATA #REQUIRED
 >
 <!ELEMENT Variant (DC*) >
 <!ATTLIST Variant
 WRITTENFORM CDATA #REQUIRED
 type CDATA #REQUIRED
 >
 <!ELEMENT GRAMMATICALDC EMPTY>
          <!-- att=constant to be taken from the DCR -->
          <!-- val=free string or constant to be taken from the DCR-->
<!ATTLIST GRAMMATICALDC
  Gender    CDATA #REQUIRED
  Number CDATA #REQUIRED>

  <!ELEMENT DC EMPTY>
          <!-- att=constant to be taken from the DCR -->
          <!-- val=free string or constant to be taken from the DCR-->
<!ATTLIST DC
  att   CDATA #REQUIRED
  val CDATA #REQUIRED>

  <!ELEMENT SOURCEDC EMPTY>
          <!-- att=constant to be taken from the DCR -->
          <!-- val=free string or constant to be taken from the DCR-->
<!ATTLIST SOURCEDC
  sourceName CDATA #REQUIRED
  sourceid   CDATA #REQUIRED
  >

  <!ELEMENT POSDC EMPTY>
          <!-- att=constant to be taken from the DCR -->
          <!-- val=free string or constant to be taken from the DCR-->
<!ATTLIST POSDC
  posname CDATA #FIXED "POS"
  pos   CDATA #REQUIRED
  >
  <!ELEMENT RELATION (DC*) >
```

```
  <!ATTLIST RELATION
  TYPE CDATA #REQUIRED
  TARGET   CDATA #REQUIRED
  >

  <!ELEMENT WordForm (DC*) >
<!ATTLIST WordForm
INFLECTEDFORM CDATA #REQUIRED
GRAMDATA CDATA #REQUIRED
>
```

Appendix B XIF Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE InputData SYSTEM "BioLex_DataExchange.dtd" >
<InputData>
        <Cluster CLSID="CLS0001" SEMTYPE="Protein name" >
                <Entry ENTRYID="ENTRY0001" BASEFORM="Interleukin-2"
TYPE="PREFERRED" >
                        <POSDC POSNAME="POS" POS="N"></POSDC>
                        <Variant WRITTENFORM="IL-2" TYPE="ACRONIM"></Variant>
                </Entry>
                <Entry ENTRYID="ENTRY0002" BASEFORM="T-Cell-Grow-Factor"
TYPE="PREFERRED" >
                        <POSDC POSNAME="POS" POS="N"></POSDC>
                        <Variant WRITTENFORM="TCGF"
TYPE="ACRONIM"></Variant>
                        </Entry>
                        <Entry ENTRYID="ENTRY0003" BASEFORM="Aldehide
Reductase" TYPE="PREFERRED" >
                        <POSDC POSNAME="POS" POS="N"></POSDC>
                        <Variant WRITTENFORM="ALDRED"
TYPE="ACRONIM"></Variant>
                        </Entry>
        </Cluster>
<Cluster CLSID="CLS10794" SEMTYPE="ChebiChem">
                <Entry entryid="CLS10794_ENTRY1" BASEFORM="fluoride salts"
type="PREFERRED">
                        <SOURCEDC sourceName="CHEBI" sourceid="CHEBI:24060"/>
                        <POSDC posname="POS" pos="N"></POSDC>
                        <Variant WRITTENFORM="fluorides" type="orthographic"/>
                </Entry>
        </Cluster>
        <Cluster CLSID="CLS10" SEMTYPE="ChebiChem">
                <Entry entryid="CLS10_ENTRY1" BASEFORM="fluoride"
type="PREFERRED">
                        <SOURCEDC sourceName="CHEBI" sourceid="CHEBI:17051"/>
                        <POSDC posname="POS" pos="N"></POSDC>
                        <Variant WRITTENFORM="F(-)" type="orthographic"/>
                        <RELATION TYPE="CHEBI_is_part_of"
TARGET="CHEBI:24060"></RELATION>
                        <RELATION TYPE="CHEBI_is_a"
TARGET="CHEBI:24060"></RELATION>
                </Entry>
        </Cluster>
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE InputData SYSTEM "Inflected.dtd">
<InputData>
<Cluster CLSID="CLSID" SEMTYPE="CLSSEMTYPE">
                <Entry entryid="ENTRY1" BASEFORM="accelerate"
type="PREFERRED">
                        <SOURCEDC sourceName="SOURCE" sourceid="SOURCEID"/>
                        <POSDC posname="POS" pos="V"></POSDC>
                        <WordForm INFLECTEDFORM="acceletates" GRAMDATA
="VVZ">
                        </WordForm>
```

```
                        <WordForm INFLECTEDFORM="Acceletate" GRAMDATA
="VVP">
                        </WordForm>
                        <WordForm INFLECTEDFORM="acceletating" GRAMDATA
="VVG">
                        </WordForm>
                </Entry>
                <Entry entryid="ENTRY02" BASEFORM="speed-up"
type="PREFERRED">
                        <POSDC posname="POS" pos="V"></POSDC>
                        <WordForm INFLECTEDFORM="speeds-up" GRAMDATA
="VVZ">
                        </WordForm>
                </Entry>
        </Cluster>
</InputData>
```

Appendix C: Dictionary Frame Details

This annex shows data dictionary tables used in the loading process.

Work Tables

The table below show the work tables currently used.

Work Table	Explanation
WORK_LIST_TABLE	Contains all the staging and user tables. If a new table is added to the BioLex it must be recorded in this table.
WORK_MAP_TABLE	Maps the xml element with the corresponding BioLex table, and maps each element attribute with columns of corresponding table. Only used attributes are mapped.
WORK_MAP_COLUMNS	This table contains information about standard staging biolexicon tables only. It contains the table structure and the link to the xml elements.
WORK_HASID_TABLE	This table contains only the xml elements that hold an identifier. This identifier will be the PK(FK) for target tables(s). The table provides also information on the position of the identifier within the list of attributes of a given element.
WORK_ID_TABLE	The table contains information on the prefix of the identifier. The identifier is strictly linked to the table it refers to[23].
WORK_REL_TABLE	This table is similar to the WORK_MAP_COLUMNS table, and contains information on user table structures.
WORK_REL_ FILES	The table contains a mapping between the xml elements and their (eventual) related user files. This tables also stores information on the prefixes of the identifier of tables composing the relational files.
WORK_PREFERRED	This table contains identifier-to-identifier mapping to manage variants and sense relations.

Rule Tables

The table below show some rule tables sketched.

Rule Table	Explanation
RULE_LIST_TGT_TABLE	This table contains the list of target tables and the order followed by software to load them.
RULE_MAP_TGT_TABLE	This table contains mapping between

[23] For example: the lemma table will be prefixed by LM_

Rule Table	Explanation
	staging tables and target tables.
RULE_JOIN_TGT_TABLE	This table contains join between staging table(s) for a given target table.
RULE_MAP_STG_TO_TGT	This table contains the list of staging table(s) for a given target table; also it contains the order of the staging table(s). The order is used to automatically build sql instruction.
RULE_LIST_SPECIAL_TABLE	This table contains the list of special tables and the order followed by software to load them.
RULE_MAP_SPECIAL_TABLE	This table contains mapping between input tables and special tables.
RULE_JOIN_SPECIAL_TABLE	This table contains join between input table(s) for a given special table.
RULE_MAP_SOURCE_TO_SPECIAL	This table contains the list of input table(s) for a given special table; also it contains the order of the input table(s). The order is used to automatically build sql instruction.
RULE_LIST_FINAL_TABLE	This table contains the list of final tables and the order followed by software to load them.
RULE_MAP_FINAL_TABLE	This table contains mapping between input tables and final tables.
RULE_JOIN_FINAL_TABLE	This table contains join between input table(s) for a given final table.
RULE_MAP_SOURCE_TO_FINAL	This table contains the list of input table(s) for a given final table; also it contains the order of the input table(s). The order is used to automatically build sql instruction.

Appendix D: Staging and Target Frame Details

This annex describes staging and target tables, in terms of columns.
It also provides a list of typed data category.

Staging tables columns

This paragraph lists all staging tables' columns:

Staging Table	Column	Explanation
STG_CLSDC	SC_ID	The identifier of the cluster.
STG_CLSDC	ATT	The attribute of data category
STG_CLSDC	VAL	The value of data category.
STG_FORMREPREENTATION	FR_ID	The identifier of the variant.
STG_FORMREPREENTATION	WRITTENFORM	The form of the variant.
STG_FORMREPREENTATION	VARIANTTYPE	The type of the writtenform of the variant.
STG_LE_SOURCEDC	LM_ID	The identifier of the lemma of the lexical entry.
STG_LE_SOURCEDC	SOURCENAME	The name of the source. It is a fixed value: SOURCENAME
STG_LE_SOURCEDC	SOURCEVAL	The value of the source.
STG_LEDC	LM_ID	The identifier of the lemma of the lexical entry.
STG_LEDC	ATT	The attribute of data category
STG_LEDC	VAL	The value of data category.
STG_LEMMA	LM_ID	The identifier of the lemma.
STG_LEMMA	ENTRYID	The identifier of the lemma in the xml file.
STG_LEMMA	BASENAME	The preferred form for the lemma.
STG_LEMMA	VARIANTTYPE	The type of the form. It is a fixed value: PREFERRED.
STG_*LEXICALENTRY*	LE_ID	The identifier of the lemma for a given lexical entry.
STG_*LEXICALENTRY*	POS	A fixed value: POS, the part of speech.
STG_*LEXICALENTRY*	POSVAL	The value of the part of speech.
STG_REL_TYPE	LM_ID	The identifier of the lemma.

Staging Table	Column	Explanation
STG_REL_TYPE	REL_TYPE	The type of the relation.
STG_REL_TYPE	TARGET	The partner of the relation, usually a target entry in the xml input file.
STG_SENSE	LM_ID	The identifier of the lemma.
STG_SENSE	BASENAME	The preferred form for the lemma.
STG_SENSE	SEMTYPE	The semantic class of the lemma.
STG_SENSERELATION	LM_ID	The identifier of the lemma.
STG_SENSERELATION	LE_ID	The identifier of the lexical entry for a given lemma.
STG_SENSERELATION	S_ID	The identifier of sense for a given lemma.
STG_SENSERELATION	BASENAME	The preferred form for the lemma.
STG_USER_SYNCLUSTER	SC_ID	The identifier of the cluster.
STG_USER_SYNCLUSTER	CLS_NAME	The name of the cluster.
STG_USER_SYNCLUSTER	SC_SEMTYPE	The semantic type of the cluster.
STG_VARIANTDC	FR_ID	The identifier of the variant for a given entry.
STG_VARIANTDC	ATT	The attribute of data category
STG_VARIANTDC	VAL	The value of data category.
USER_CLS_LM	SC_ID	The identifier of the cluster.
USER_CLS_LM	LM_ID	The identifier of the lemma.
USER_FR_LM	LM_ID	The identifier of the lemma.
USER_FR_LM	FR_ID	The identifier of the variant for a given lemma.
USER_LE_LM	LM_ID	The identifier of the lemma.
USER_LE_LM	LE_ID	The identifier of the lexical entry for a given lemma.

Data category columns

This paragraph data category columns:

Data category	Column	Explanation
POSDC	POS_ID	The identifier of the part of speech.
POSDC	POSATT	A fixed value: POS

Data category	Column	Explanation
POSDC	POSVAL	The domain of the part of speech: N,V,A.
VARIANTDC	VT_ID	The identifier of the variant type.
VARIANTDC	VARIANTATT	A fixed value: VARIANTTYPE
VARIANTDC	VARIANTVAL	The domain of the variant type.
RELDC	REL_ID	The identifier of the relation type.
RELDC	ATT	A fixed value: RELATION
RELDC	VAL	The domain of the relation types.
SEMDC	SEM_ID	The identifier of the semantic type.
SEMDC	SEMATT	A fixed value: SEMTYPE
SEMDC	SEMVAL	The domain of the semantic types.
SOURCEDC	SOURCE_ID	The identifier of the source.
SOURCEDC	SOURCEATT	A fixed value: SOURCENAME
SOURCEDC	SOURCEATT	The domain of the sources.

Target tables columns

This paragraph lists all target tables' columns:

Target Table	Column	Explanation
TGT_USER_CLS_DC	SC_ID	The identifier of the cluster.
TGT_USER_CLS_DC	DC_ID	The identifier of the attribute-name-value feature, as generated in the DC table.
TGT_*FORMREPRESENTATION*	LM_ID	The identifier of the lemma.
TGT_*FORMREPRESENTATION*	FR_ID	The identifier of the variant.
TGT_*FORMREPRESENTATION*	WRITTENFORM	The form of the variant.
TGT_*FORMREPRESENTATION*	VT_ID	The identifier of the variant type, as generated in the VARIANTDC table.
TGT_SENSE	S_ID	The identifier of the senses.
TGT_SENSE	SEMTYPE	The identifier of the semantic type as generated by the SEMDC.
TGT_LE_SOURCEDC	LE_ID	The identifier of the lexical entry.
TGT_LE_SOURCEDC	SOURCE_ID	The value of the sourceid attribute in the input file for a given lexical entry.
TGT_LE_SOURCEDC	SOURCE_REF	The identifier of the source, as generated in the

Target Table	Column	Explanation
		SOURCEDC table.
TGT_USER_LE_DC	LE_ID	The identifier of the lexical entry.
TGT_USER_LE_DC	DC_ID	The identifier of the attribute-name-value feature, as generated in the DC table.
TGT_LEMMA	LM_ID	The identifier of the lemma.
TGT_LEMMA	ENTRYID	The identifier of the lemma in the xml file.
TGT_LEMMA	BASENAME	The preferred form for the lemma.
TGT_LEMMA	VT_ID	The identifier of the "PREFERRED" variant type , as generated in the VARIANTDC table.
TGT_*LEXICALENTRY*	LE_ID	The identifier of the lemma for a given lexical entry.
TGT_*LEXICALENTRY*	POS_ID	The identifier of the part of speech, as generated in the POSDC table.
TGT_SENSERELATION	SR_ID	The identifier of the relationship.
TGT_SENSERELATION	S_SOURCE_ID	The identifier of the source sense for a given lexical entry.
TGT_SENSERELATION	S_TARGET_ID	The identifier of the target sense for a given lexical entry..
TGT_SENSERELATION	REL_TYPE_ID	The identifier of the relation type, as generated in the RELDC table.
TGT_USER_SYNCLUSTER	SC_ID	The identifier of the cluster.
TGT_USER_SYNCLUSTER	LM_ID	The identifier of the lemma.
TGT_USER_SYNCLUSTER	LE_ID	The identifier of the lexical entry.
TGT_USER_SYNCLUSTER	POS_ID	The identifier of the part of speech, as generated in the POSDC table.
TGT_USER_SYNCLUSTER	CLS_NAME	The name of the cluster
TGT_USER_SYNCLUSTER	SEM_ID	The identifier of the semantic type as generated by the SEMDC.
TGT_USER_FR_DC	FR_ID	The identifier of the variant.
TGT_USER_FR_DC	DC_ID	The identifier of the attribute-name-value feature, as generated in the DC table.
TGT_USER_FR_LM	LM_ID	The identifier of the lemma.
TGT_USER_FR_LM	FR_ID	The identifier of the variant for a given lemma.
TGT_USER_LE_LM	LM_ID	The identifier of the lemma.
TGT_USER_LE_LM	LE_ID	The identifier of the lexical entry for a given lemma.

Target Table	Column	Explanation
TGT_USER_S_LE	S_ID	The identifier of the sense.
TGT_USER_S_LE	LE_ID	The identifier of the lexical entry.