

# Data Model for Lexicography (DMLex) Version 1.0

## Committee Specification Draft 01

8 September 2023

### Specification URIs

**This version:**

<https://docs.oasis-open.org/lexidma/dmlex/v1.0/csd01/dmlex-v1.0-csd01.html>  
<https://docs.oasis-open.org/lexidma/dmlex/v1.0/csd01/dmlex-v1.0-csd01.pdf> (Authoritative)

**Previous version:**

N/A

**Latest version:**

<https://docs.oasis-open.org/lexidma/dmlex/v1.0/dmlex-v1.0.html>  
<https://docs.oasis-open.org/lexidma/dmlex/v1.0/dmlex-v1.0.pdf> (Authoritative)

**Technical Committee:**

OASIS Lexicographic Infrastructure Data Model and API (LEXIDMA) TC

**Chair:**

Tomaž Erjavec ([tomaz.erjavec@ijs.si](mailto:tomaz.erjavec@ijs.si)), Jozef Stefan Institute

**Editors:**

David Filip ([glorfindel@mail.muni.cz](mailto:glorfindel@mail.muni.cz)), Masaryk University  
Miloš Jakubiček ([milos.jakubicek@sketchengine.eu](mailto:milos.jakubicek@sketchengine.eu)), Lexical Computing  
Simon Krek ([simon.krek@ijs.si](mailto:simon.krek@ijs.si)), Jozef Stefan Institute  
John McCrae, University of Galway  
Michal Měchura ([michmech@mail.muni.cz](mailto:michmech@mail.muni.cz)), Masaryk University

**Additional artifacts:**

NONE AT THE MOMENT

**Related Work:**

This specification is related to:

- No related specifications.

**Declared namespaces:**

This specification declares one or more namespaces. Namespace isn't considered an XML specific feature in this serialization independent specification.

*The core and modules namespace*

- <http://docs.oasis-open.org/lexidma/ns/dmlex-1.0>

## Key words:

The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in [BCP 14 \[RFC2119\]](#) and [\[RFC8174\]](#) if, and only if, they appear in all capitals, as shown here.

## Abstract:

This document defines the 1st version of a data model in support of the high-priority technical goals described in the LEXIDMA TC's charter, including:

- A serialization-independent Data Model for Lexicography (DMLex)
- An XML serialization of DMLex
- A JSON serialization of DMLex
- A relational database serialization of DMLex
- An RDF serialization of DMLex
- An informative NVH serialization of DMLex

## Status:

This document was last revised or approved by the LEXIDMA TC on the above date. The level of approval is also listed above. Check the "Latest version" location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at [https://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=lexidma#technical](https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=lexidma#technical).

TC members should send comments on this document to the TC's email list. Others should send comments to the TC's public comment list, after subscribing to it by following the instructions at the "Send A Comment" button on the TC's web page at <https://www.oasis-open.org/committees/lexidma/>.

This specification is provided under the [Non-Assertion](#) Mode of the [OASIS IPR Policy](#), the mode chosen when the Technical Committee was established. For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the TC's web page (<https://www.oasis-open.org/committees/lexidma/ipr.php>).

Note that any machine-readable content ([Computer Language Definitions](#)) declared Normative for this Work Product is provided in separate plain text files. In the event of a discrepancy between any such plain text file and display content in the Work Product's prose narrative document(s), the content in the separate plain text file prevails.

## Citation format:

When referencing this specification the following citation format should be used:

[DMLex-1.0]

Data Model for Lexicography Version 1.0. Edited by Michal Měchura, David Filip and Simon Krek. 8 September 2023. OASIS Committee Specification Draft 01. <https://docs.oasis-open.org/lexidma/dmlex/v1.0/csd01/dmlex-v1.0-csd01.html>. Latest version: <https://docs.oasis-open.org/lexidma/dmlex/v1.0/dmlex-v1.0.html>.

## Notices:

Copyright © OASIS Open 2023.

All Rights Reserved. Distributed under the terms of the [OASIS IPR Policy](#)).

The name "OASIS" is a trademark of [OASIS](#)), the owner and developer of this specification, and should be used only to refer to the organization and its official outputs.

For complete copyright information please see the [full Notices](#) section in an Appendix below.

---

# Table of Contents

1 Introduction (Normative)	7
1.1 Modular structure of DMLex	7
1.2 Implementing DMLex	7
1.3 The metamodel behind DMLex	7
1.3.1 Object types	7
1.3.2 Properties	8
1.3.3 Tree structures	8
1.3.4 Relational structures	8
1.3.5 Unique identifiers	9
1.4 Definitions	9
2 Conformance	11
3 DMLex Core	12
3.1 lexicographicResource	12
3.2 entry	14
3.3 partOfSpeech	16
3.4 inflectedForm	17
3.5 sense	18
3.6 definition	20
3.7 label	21
3.8 pronunciation	22
3.9 transcription	23
3.10 example	24
4 DMLex OPTIONAL Modules (Normative)	27
4.1 DMLex Crosslingual Module	27
4.1.1 Extensions to lexicographicResource	27
4.1.2 translationLanguage	27
4.1.3 Extensions to sense	28
4.1.4 headwordTranslation	29
4.1.5 headwordExplanation	31
4.1.6 Extensions to example	32
4.1.7 exampleTranslation	32
4.1.8 Extensions to partOfSpeech	33
4.1.9 Extensions to label	34
4.1.10 Extensions to pronunciation	34
4.1.11 Extensions to inflectedForm	35
4.2 DMLex Controlled Values Module	35
4.2.1 Extensions to lexicographicResource	35
4.2.2 definitionTypeTag	36
4.2.3 inflectedFormTag	37
4.2.4 labelTag	39
4.2.5 labelTypeTag	41
4.2.6 partOfSpeechTag	42
4.2.7 sourceIdentityTag	44
4.2.8 transcriptionSchemeTag	45
4.2.9 sameAs	46
4.3 DMLex Linking Module	47
4.3.1 Extensions to lexicographicResource	47
4.3.2 relation	48
4.3.3 member	49
4.3.4 relationType	50
4.3.5 memberType	52
4.3.6 Extensions to sameAs	54
4.4 DMLex Annotation Module	54
4.4.1 Extensions to entry	54
4.4.2 Extensions to headwordTranslation	55

4.4.3 placeholderMarker .....	55
4.4.4 Extensions to definition .....	56
4.4.5 Extensions to example .....	57
4.4.6 Extensions to exampleTranslation .....	58
4.4.7 headwordMarker .....	59
4.4.8 collocateMarker .....	59
4.4.9 Extensions to label .....	61
4.5 DMLex Etymology Module .....	61
4.5.1 Extensions to entry .....	61
4.5.2 etymology .....	62
4.5.3 etymon .....	63
4.5.4 etymonUnit .....	64
4.5.5 Extensions to lexicographicResource .....	66
4.5.6 etymonType .....	66
4.5.7 etymonLanguage .....	67
4.5.8 Extensions to partOfSpeech .....	68
5 DMLex REQUIRED Serializations (Normative) .....	69
5.1 DMLex XML serialization (Normative) .....	69
5.1.1 Design Principles (Informative) .....	69
5.1.2 DMLex XML serialization elements (Normative) .....	69
5.2 DMLex JSON serialization (Normative) .....	78
5.2.1 Design principles .....	79
5.2.2 DMLex JSON serialization objects (Normative) .....	79
5.3 DMLex RDF serialization (Normative) .....	87
5.3.1 Design principles .....	87
5.3.2 DMLex RDF serialization elements .....	88
5.4 DMLex relational database serialization (Normative) .....	98
5.4.1 Design principles (Informative) .....	98
5.4.2 Database diagrams (Informative) .....	98
5.4.3 DMLex relational database serialization tables (Normative) .....	104

## Appendixes

A Informative material on serializations (Informative) .....	117
A.1 Examples (Informative) .....	117
A.1.1 A basic entry .....	117
A.1.2 How to use inflectedForm .....	120
A.1.3 Pronunciation given as transcription .....	121
A.1.4 Pronunciation given as a sound file .....	122
A.1.5 Pronunciation given both ways .....	122
A.1.6 How to use partOfSpeechTag and inflectedFormTag .....	123
A.1.7 Mapping controlled values to external inventories .....	125
A.1.8 Defining a bilingual lexicographic resource .....	126
A.1.9 Defining a multilingual lexicographic resource .....	127
A.1.10 How to use headwordTranslation in a bilingual lexicographic resource .....	127
A.1.11 How to use headwordTranslation in a multilingual lexicographic resource ..	129
A.1.12 How to use headwordExplanation .....	131
A.1.13 Modelling parts and wholes .....	132
A.1.14 Modelling antonyms .....	136
A.1.15 Modelling synonyms .....	138
A.1.16 Modelling variants .....	142
A.1.17 Modelling subsenses .....	144
A.1.18 Modelling subentries (at subsense level) .....	148
A.1.19 Modelling subentries (at sense level) .....	151
A.1.20 Using placeholderMarker .....	154
A.1.21 Using placeholderMarker in a bilingual lexicographic resource .....	154
A.1.22 Using headwordMarker .....	155
A.1.23 Using collocateMarker .....	157

A.1.24 Modelling etymology .....	158
A.2 DMLex NVH serialization (Informative) .....	160
A.2.1 Design Principles (Informative) .....	160
A.2.2 DMLex NVH serialization nodes (Normative) .....	160
A.2.2.1 NVH node: lexicographicResource .....	160
A.2.2.2 NVH node: entry .....	161
A.2.2.3 NVH node: partOfSpeech .....	161
A.2.2.4 NVH node: inflectedForm .....	161
A.2.2.5 NVH node: sense .....	162
A.2.2.6 NVH node: definition .....	162
A.2.2.7 NVH node: label .....	162
A.2.2.8 NVH node: pronunciation .....	162
A.2.2.9 NVH node: transcription .....	163
A.2.2.10 NVH node: example .....	163
A.2.2.11 NVH node: translationLanguage .....	163
A.2.2.12 NVH node: headwordTranslation .....	163
A.2.2.13 NVH node: headwordExplanation .....	164
A.2.2.14 NVH node: exampleTranslation .....	164
A.2.2.15 NVH node: partOfSpeechTag .....	164
A.2.2.16 NVH node: inflectedFormTag .....	165
A.2.2.17 NVH node: definitionTypeTag .....	165
A.2.2.18 NVH node: labelTag .....	165
A.2.2.19 NVH node: labelTypeTag .....	166
A.2.2.20 NVH node: sourceIdentityTag .....	166
A.2.2.21 NVH node: transcriptionSchemeTag .....	166
A.2.2.22 NVH node: forLanguage .....	166
A.2.2.23 NVH node: forPartOfSpeech .....	167
A.2.2.24 NVH node: sameAs .....	167
A.2.2.25 NVH node: relation .....	167
A.2.2.26 NVH node: member .....	167
A.2.2.27 NVH node: relationType .....	167
A.2.2.28 NVH node: memberType .....	168
A.2.2.29 NVH node: placeholderMarker .....	168
A.2.2.30 NVH node: headwordMarker .....	168
A.2.2.31 NVH node: collocateMarker .....	168
A.2.2.32 NVH node: etymology .....	168
A.2.2.33 NVH node: etymon .....	169
A.2.2.34 NVH node: etymonUnit .....	169
A.2.2.35 NVH node: etymonType .....	169
A.2.2.36 NVH node: etymonLanguage .....	169
B References (Normative) .....	170
B.1 Normative references (Normative) .....	170
B.2 Informative references (Informative) .....	171
C Machine Readable Validation Artifacts (Informative) .....	172
D DMLex UML diagram (Normative) .....	173
E DMLex plain text description (Informative) .....	174
F Specification Change Tracking (Informative) .....	178
G Acknowledgements (Informative) .....	179
H Notices (Informative) .....	180

---

# 1 Introduction (Normative)

DMLex is a data model for modelling dictionaries (here called lexicographic resources) in computer applications such as dictionary writing systems.

DMLex is a data model, not an encoding format. DMLex is abstract, independent of any markup language or formalism. At the same time, DMLex has been designed to be easily and straightforwardly implementable in XML, JSON, NVH, as a relational database, and as a Semantic Web triplestore.

## 1.1 Modular structure of DMLex

The DMLex specification is divided into a REQUIRED core with several OPTIONAL modules.

- [DMLex Core](#) allows you to model the basic entries-and-sense structure of a monolingual lexicographic resource.
- [DMLex Crosslingual Module](#) extends DMLex Core to model bilingual and multilingual lexicographic resources.
- [DMLex Controlled Values Module](#) extends DMLex Core to represent inventories of look-up values to be used as part-of-speech tags, usage label tags and others.
- [DMLex Linking Module](#) extends DMLex Core and allows you to model various kinds of relations between entries, senses and other objects, including semantic relations such as synonymy and antonymy and presentational relations such as subentries and subsenses, both within a single lexicographic resource and across multiple lexicographic resources.
- [DMLex Annotation Module](#) extends DMLex Core to allow the modelling of inline markup on various objects such as example sentences, including the modelling of collocations and corpus patterns.
- [DMLex Etymology Module](#) extends DMLex Core to allow the modelling of etymological information in dictionaries.

## 1.2 Implementing DMLex

DMLex is an abstract data model which can be implemented in many different programming environments and serialization languages. In this document, we specify REQUIRED serializations for:

[XML](#),  
[JSON](#),  
[RDF](#), and  
[relational databases](#).

An informative serialization specification is provided for:

[NVH](#).

## 1.3 The metamodel behind DMLex

The DMLex data model is defined in this document through the following concepts.

### 1.3.1 Object types

DMLex defines the existence of certain object types such as `entry` and `definition`. In every DMLex-conformant data set, each “object” (for example each XML element or each database table) must be an instance of one of these object types.

The names of object types are unique within the scope of the entire data model, including all the modules. An object type defined in one module is guaranteed not to name-conflict with another object type in another module.

### 1.3.2 Properties

For every object type, DMLex defines which properties it can have, whether the properties are required or optional, what their arities are (for example “zero or more”), and what kinds of values it can contain.

There are two kinds of properties: those that contain literal values and those that contain objects.

Some properties are defined to contain literal values such as strings and numbers. The following types of literal values are used in DMLex:

non-empty string,  
number,  
unique identifier  
and reference to a unique identifier

Some properties are defined to contain objects (of types defined in DMLex), for example an object of type `entry` can contain objects of type `sense`. In such cases, the name of the property is the same as the name of the object type. For example, the `entry` object type has a property called `sense` which (with an arity of “zero or more”) contains `sense` objects.

The names of properties are unique within the scope of the object type they belong to but not necessarily outside it.

### 1.3.3 Tree structures

The mechanics of objects which (through their properties) contain other objects mimics the tree-like structure commonly used in lexicography to represent dictionary entries, typically using XML.

Because of this, this document sometimes refers to objects being the “children”, “parents”, “descendants” or “ancestors” of other objects, or objects “belonging to” other objects. This is to be understood as follows.

- “Object A is the parent of object B” and “object B is a child of object A” both mean that object A has a property whose value is object B.
- “Object A is an ancestor of object B” and “object B is a descendant of object A” means that there is a chain of parent-child links between object A and object B.
- “Object B belongs to object A” is the same as “object B is a descendant of object A” and “object A is an ancestor of object B”.

The tree structures created by DMLex are guaranteed to have similar properties as in XML and many other tree-structured formalisms, including importantly the constraint that every object has no more than one parent.

Additionally, tree structures created by DMLex are guaranteed to be non-recursive: an object can never be an ancestor of another object of the same type.

### 1.3.4 Relational structures

In addition to tree structures, which are used in DMLex to model the basic entries-and-senses skeleton of a lexicographic resource, DMLex also allows (through the `relation` object type defined in the Linking Module) the modelling of non-tree relations between objects, that is, relations connecting objects which belong to different parents.

Non-tree relations are used in DMLex to model cross-references between entries and senses, both within a single lexicographic resource and across multiple lexicographic resources. Non-tree relations



are also used in DMLex to model certain things which have traditionally been modelled with tree structures in lexicography, such as the hierarchical arrangement of entries and subentries, or senses and subsenses. In DMLex, these phenomena are modelled as non-tree relations in order to avoid recursion in the tree structures and to minimize the number of object types in the data model.

### 1.3.5 Unique identifiers

Some object types defined in DMLex have a property or a combination of properties mandated to be unique. Designating a property as unique means that its value must be unique within the scope of its parent object type. Designating a combination of properties as unique means that the ordered composition of its values must be unique within the scope of their parent object type. Such unique properties in DMLex serve two purposes:

- To enable objects to participate in relational structures. Relational structures (as opposed to tree structures) are modelled in DMLex as tuples (triples, ...) of unique identifiers.
- To enable addressing and accessing individual objects, for example individual senses or individual example sentences, outside the context of the entries they are in. See [Fragment identification](#) for details.

Serializations of DMLex may choose to assign arbitrary unique identifiers (e.g. a random integer or a hash of the property value) to each property or combination of properties.

## 1.4 Definitions

### Agent

any application or tool that , reads, edits, writes, processes, stores, renders or otherwise handles DMLex Instances.

Agent is the most general application conformance target that subsumes all other specialized user agents disregarding whether they are defined in this specification or not.

### Enrich, Enriching

the process of adding metadata and resources to DMLex Instances

#### *Processing Requirements*

- Enriching MAY happen at the time of Creation.

### Enricher, Enricher Agent

any Agent that performs the Enriching process

### Create, Creation

the process of encoding lexicographic data as DMLex Instances

### Creator, Creator Agent

any Agent that performs the Creation process

### Modify, Modification

the process of changing core and module based structural and inline objects that were previously created by other Writers

#### *Processing Requirements*

- DMLex Instances MAY be Modified and Enriched at the same time.

## Note

Creator or Enricher knowledge of the DMLex Instance is not assumed while Modifying. However, modifications can only happen in the same serialization.

Modifier, Modifier Agent

an Agent that performs the Modification process

Writer, Writer Agent

an Agent that Creates, generates, or otherwise writes DMLex Instances for whatever purpose, including but not limited to Creator, Modifier, and Enricher Agents.

## Note

Since DMLex serializations are intended as editing and exchange formats rather than processing formats, many applications will need to generate DMLex Instances from their internal processing formats, even in cases when they are processing DMLex Instances provided by other Writers.

UNIQUE

The term UNIQUE shall be interpreted as given in the section on [Unique identifiers](#).

---

## 2 Conformance

### 1. *DMLex Instances Conformance*

- a. Conformant DMLex Instances **MUST** be well formed and valid instances according to one of the normative DMLex Serialization Specifications.
- b. DMLex Instances **MAY** contain custom extensions, provided that those extensions do not provide features or parts of features that are provided by the [DMLex Core](#) or any of the [DMLex Modules](#). Extensions **MUST** be serialized in a way conformant and compatible with the pertaining DMLex Serialization Specifications.

### 2. *Application Conformance*

- a. DMLex Writers **MUST** create conformant DMLex Instances to be considered DMLex compliant.
- b. Agents processing conformant DMLex Instances that contain custom extensions are not **REQUIRED** to understand and process non-DMLex objects or attributes. However, conformant applications **SHOULD** preserve existing custom extensions when processing conformant DMLex Instances, provided that the objects that contain custom extensions are not removed according to DMLex Processing Requirements or the extension's own processing requirements.
- c. All Agents **MUST** comply with Processing Requirements for otherwise unspecified Agents or without a specifically set target Agent.
- d. Specialized Agents defined in this specification - this is Creator, Writer, Modifier, and Enricher Agents - **MUST** comply with the Processing Requirements targeting their specifically defined type of Agent on top of Processing Requirements targeting all Agents as per point c. above.
- e. DMLex is an object model explicitly designed for exchanging data in the same serialization among various Agents. Thus, a conformant DMLex application **MUST** be able to accept DMLex Instances Created, Modified, or Enriched by a different application, provided that:
  - i. The processed files are conformant DMLex Instances according to the same DMLex Serialization Specification,
  - ii. in a state compliant with all relevant Processing Requirements.

### 3. *Backwards Compatibility*

- a. N/A (no previous versions)

## Note

DMLex Instances cannot be conformant to this specification w/o being conformant to a specific serialization.

## Warning

Exchange of DMLex Instances data between and among DMLex serializations is out of scope of the DMLex Version 1.0 specification. Although, the serializations are semantically compatible, they don't have to possess compatible addressing and fargemnt identification mechanisms, mainly due to the fact that each serialization is expected to use its own native id data types, uniqueness scopes, and addressing methods.

---

## 3 DMLex Core

The DMLex Core provides data types for modelling monolingual dictionaries (called lexicographic resources in DMLex) where headwords, definitions and examples are all in one and the same language. DMLex Core gives you the tools you need to model simple dictionary entries which consist of the following data types:

- `lexicographicResource`
- `entry`
- `partOfSpeech`
- `inflectedForm`
- `sense`
- `definition`
- `label`
- `pronunciation`
- `transcription`
- `example`

### Optional roots

When exchanging data encoded in a DMLex serialization which has the concept of a "root" or top-level object, such as XML, JSON or NVH, the object types `lexicographicResource` and `entry` can serve as such roots.

### Fragment identification

Incomplete parts of DMLex objects represent valid fragments as long as it is possible to identify their complete source DMLex object. For a particular serialization, this can be achieved by:

- Making sure the fragment contains all **unique properties** of the DMLex parent-child hierarchy that are necessary to determine its source object.
- Using a serialization-specific addressing mechanism (e.g. XPath for XML, JMESPath for JSON or an NVH query) to provide the location of the fragment in the source DMLex object.

#### 3.1 `lexicographicResource`

Represents a dictionary. A lexicographic resource is a dataset which can be used, viewed and read by humans as a dictionary and – simultaneously – ingested, processed and understood by software agents as a machine-readable dataset.

#### Note

The correct name of this data type in DMLex is `lexicographic`, not `lexical`, resource.

#### Properties

- `title` OPTIONAL (zero or one). Non-empty string. A human-readable title of the lexicographic resource.

- `uri` REQUIRED (zero or one). The URI of the lexicographic resource, identifying it on the Web.
- `langCode` REQUIRED (exactly one). The IETF language code of the language that this lexicographic resource describes.
- `entry` OPTIONAL (zero or more)

#### Comments

- `langCode` identifies the language of headwords, definitions and examples in this dictionary. DMLex is based on the assumption that all headwords in a lexicographic resource are in the same language, and that definitions and examples, if any are included in the lexicographic resource, are in that language too. The `langCode` property of `lexicographicResource` informs potential users of the lexicographic resource which language that is.
- Ideally, a lexicographic resource should include at least one entry. However, DMLex specifies that `entry` is optional in `lexicographicResource` to allow for the existence of lexicographic resources which are not yet complete.
- The `lexicographicResource` data type does not prescribe properties for detailed metadata about the lexicographic resource, implementers are advised to refer to the properties in the [Dublin Core elements namespace](#) as for basic metadata.

#### Example 1. XML

```
<lexicographicResource uri="..." langCode="...">
  <title>...</title>
  <entry.../>
</lexicographicResource>
```

#### Example 2. JSON

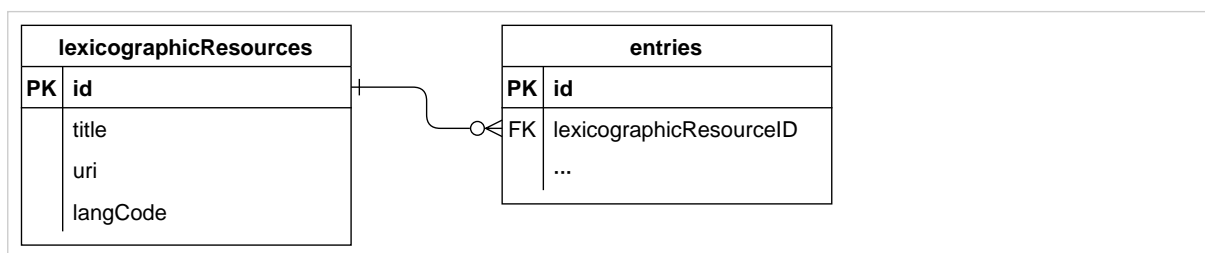
```
{
  "title": "...",
  "langCode": "...",
  "entries": [...]
}
```

#### Example 3. RDF

```
@prefix dmlex: <http://www.oasis-open.org/to-be-confirmed/dmlex> .

<#id> a dmlex:LexicographicResource ;
  dmlex:title "...";
  dmlex:uri "...";
  dmlex:langCode "...";
  dmlex:entry <entry1> , ... .
```

#### Example 4. Relational database



## 3.2 entry

Represents a dictionary entry. An entry contains information about one headword.

Property of

- [lexicographicResource](#)

Properties

- **headword** REQUIRED (exactly one) and UNIQUE (in combination with other unique properties if present). Non-empty string. The entry's headword.
- **homographNumber** OPTIONAL (zero or one) and UNIQUE (in combination with other unique properties if present). The entry's homograph number, as a guide to distinguish entries with the same headword.
- **partOfSpeech** OPTIONAL (zero or more) and UNIQUE (in combination with other unique properties if present).
- **label** OPTIONAL (zero or more).
- **pronunciation** OPTIONAL (zero or more).
- **inflectedForm** OPTIONAL (zero or more).
- **sense** OPTIONAL (zero or more).

### Note

DMLex Core does not have a concept of "subentry". To model subentries (i.e. entries inside entries) in a lexicographic resource, object types from the Linking Module should be used.

### Note

The headword can be a single word, a multi-word expression, or any expression in the source language which is being described by the entry.

### Note

DMLex allows only one headword per entry. Things such as variant headwords do not exist in DMLex. However, the [DMLex Linking Module](#) does make it possible to represent the existence of variants by treating them as separate headwords of separate entries, and linking the entries using a type of link which will cause the entries to be placed together when shown to human users. See [Section A.1.16, "Modelling variants"](#) for an example using the English words "colour" and "color".

## Note

Entries in DMLex do not have an explicit listing order. An application can imply a listing order from a combination of the headword and the homograph number, such that the headword is the primary sorting key and the homograph number (for entries that have one) is the secondary sorting key.

## Note

Ideally, each entry should have exactly one part-of-speech label. However, DMLex allows more than one `partOfSpeech` in `entry` in order to allow for exceptional cases when the lexicographer has decided to treat multiple part-of-speech readings of a headword in a single entry. Example: English words which denote nationalities ("Czech", "German") and which can function both as nouns and as adjectives.

### Example 5. XML

```
<entry homographNumber="...">
  <headword>...</headword>
  <partOfSpeech.../>
  <label.../>
  <pronunciation.../>
  <inflectedForm.../>
  <sense.../>
</entry>
```

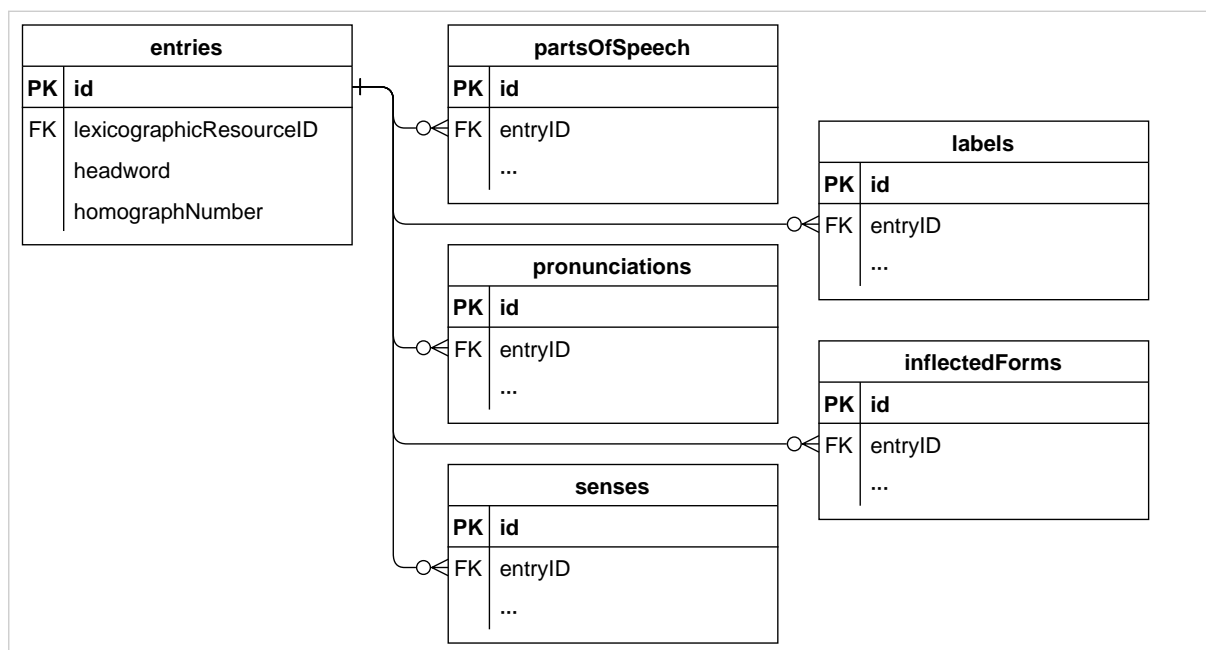
### Example 6. JSON

```
{
  "headword": "...",
  "homographNumber": "...",
  "partsOfSpeech": [...],
  "labels": [...],
  "pronunciations": [...],
  "inflectedForms": [...],
  "senses": [...]
}
```

### Example 7. RDF

```
<id> a dmlex:Entry ;
  dmlex:headword "...";
  dmlex:homographNumber ... ;
  dmlex:partOfSpeech ... ;
  dmlex:label ... ;
  dmlex:pronunciation ... ;
  dmlex:inflectedForm ... ;
  dmlex:sense ... .
```

### Example 8. Relational database



## 3.3 partOfSpeech

Represents a part-of-speech label.

Property of

- [entry](#)

Properties

- **tag** REQUIRED (exactly one) and UNIQUE. Non-empty string. An abbreviation, a code or some other string of text which identifies the part-of-speech label, for example *n* for noun, *v* for verb, *adj* for adjective. The [partOfSpeechTag](#) object type can be used to explain the meaning of the part-of-speech tags, to constrain which part-of-speech tags are allowed to occur in the lexicographic resource, and to map them onto external inventories and ontologies.
- **listingOrder** REQUIRED (exactly one). Number. The position of this part-of-speech label among other part-of-speech labels of the same entry. This can be implicit from the serialization.

Comments

- The way to model other grammatical properties of the headword besides part of speech, such as gender (of nouns) or aspect (of verbs) in DMLex is to combine them with the part of speech into a single part-of-speech tag, for example *noun-masc* and *noun-fem*, or *v-perf* and *v-imperf*.

Example 9. XML

```
<partOfSpeech tag="..." />
```

Example 10. JSON

```
"..."
```



### Example 11. RDF

```
<entry> dmlex:partOfSpeech [  
  a dmlex:PartOfSpeech ;  
  dmlex:tag "...";  
  dmlex:listingOrder 1 ] .
```

### Example 12. Relational database

partsOfSpeech	
PK	id
FK	entryID
	tag
	listingOrder

## 3.4 inflectedForm

Represents one (of possibly many) inflected forms of the headword. Example: [Section A.1.2, “How to use inflectedForm”](#).

#### Property of

- [entry](#)

#### Properties

- `text` REQUIRED (exactly one) and UNIQUE (in combination with other unique properties if present). Non-empty string. The text of the inflected form.
- `tag` OPTIONAL (zero or one) and UNIQUE (in combination with other unique properties if present). Non-empty string. An abbreviation, a code or some other string of text which identifies the inflected form, for example `p1` for plural, `gs` for genitive singular, `com` for comparative. The [inflectedFormTag](#) object type can be used to explain the meaning of the inflection tags, to constrain which inflection tags are allowed to occur in the lexicographic resource, and to map them onto external inventories and ontologies.
- `label` OPTIONAL (zero or more).
- `pronunciation` OPTIONAL (zero or more).
- `listingOrder` REQUIRED (exactly one). Number. The position of this inflected form among other inflected forms of the same entry. This can be implicit from the serialization.

### Example 13. XML

```
<inflectedForm tag="...">  
  <text>...</text>  
  <label.../>  
  <pronunciation.../>  
</inflectedForm>
```

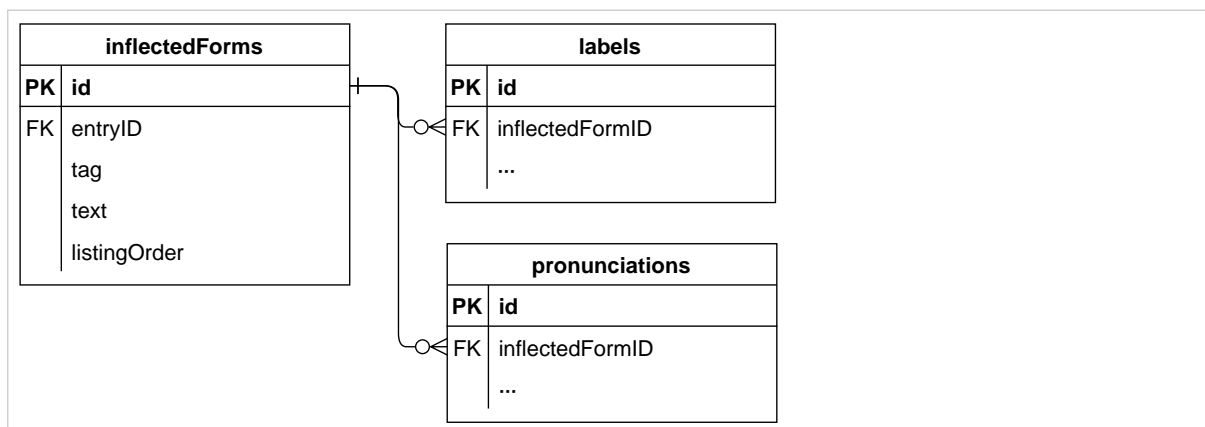
### Example 14. JSON

```
{
  "tag": "...",
  "text": "...",
  "labels": [...],
  "pronunciations": [...]
}
```

### Example 15. RDF

```
<entry> dmlex:inflectedForm [
  dmlex:text "...";
  dmlex:tag "...";
  dmlex:listingOrder 1;
  dmlex:label ...;
  dmlex:pronunciation ... ] .
```

### Example 16. Relational database



#### Comments

- The `inflectedForm` object type is intended to model the **inflectional morphology** of a headword. To model derivational morphology, for example feminine forms of masculine nouns, the recommended way to do that in DMLex is to create separate entries for the two words, and link them using the [Linking Module](#).

## 3.5 sense

Represents one of possibly many meanings (or meaning potentials) of the headword.

#### Property of

- [entry](#)

#### Properties

- `listingOrder` REQUIRED (exactly one). Number. The position of this sense among other senses of the same entry. Can be implicit from the serialization.

- `indicator` OPTIONAL (zero or one) and UNIQUE. A short statement, in the same language as the headword, that gives an indication of the meaning of a sense and permits its differentiation from other senses in the entry. Indicators are sometimes used in dictionaries instead of or in addition to definitions.
- `label` OPTIONAL (zero or more).
- `definition` OPTIONAL (zero or more) and UNIQUE.
- `example` OPTIONAL (zero or more).

#### Comments

- The properties of **entry** are, apart from `sense`, formal properties of the headword such as orthography, morphology, syntax and pronunciation. A **sense** is a container for statements about the headword's semantics. DMLex deliberately makes it impossible to include morphological information at sense level. It is impossible in DMLex to model an entry where each sense has slightly different morphological properties (e.g. a noun has a weak plural in one sense and a strong plural in another). Such phenomena need to be treated as two entries (homographs) and can be linked using the Linking Module to make sure they are always shown together to human users.

#### Example 17. XML

```
<sense>
  <indicator>...</indicator>
  <label.../>
  <definition.../>
  <example.../>
</sense>
```

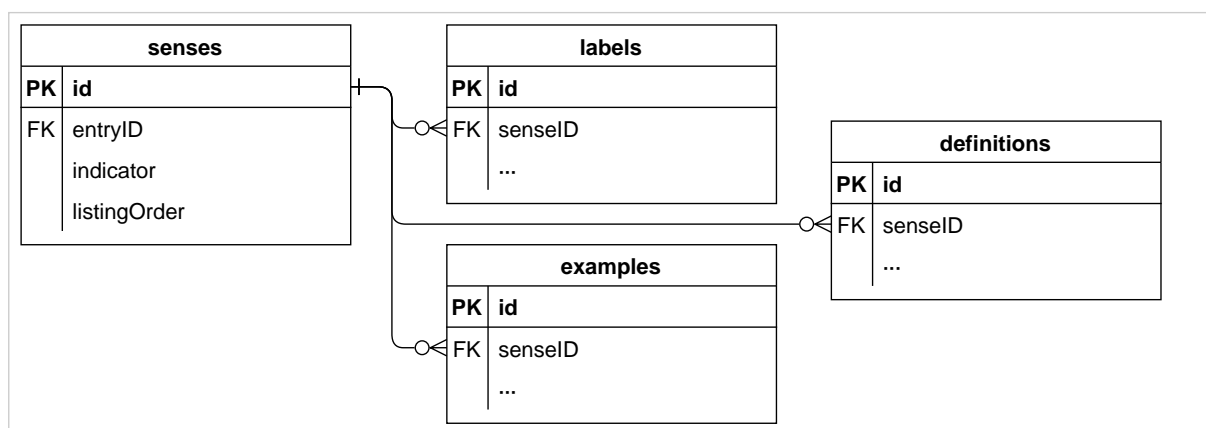
#### Example 18. JSON

```
{
  "indicator": "...",
  "labels": [...],
  "definitions": [...],
  "examples": [...]
}
```

#### Example 19. RDF

```
<id> a dmlex:Sense ;
  dmlex:listingOrder 1 ;
  dmlex:indicator "... " ;
  dmlex:label ... ;
  dmlex:definition ... ;
  dmlex:example ... .
```

### Example 20. Relational database



## 3.6 definition

Represents one of possibly several definitions of a sense.

*Property of*

- [sense](#)

*Properties*

- **text** REQUIRED (exactly one) and UNIQUE. Non-empty string. A statement, in the same language as the headword, that describes and/or explains the meaning of a sense. In DMLex, the term definition encompasses not only formal definitions, but also less formal explanations.
- **definitionType** OPTIONAL (zero or one). If a sense contains multiple definitions, indicates the difference between them, for example that they are intended for different audiences. The [definitionTypeTag](#) object type can be used to constrain and/or explain the definition types that occur in the lexicographic resource.
- **listingOrder** REQUIRED (exactly one). Number. The position of this definition among other definitions of the same sense. This can be implicit from the serialization.

### Example 21. XML

```
<definition definitionType="...">...</definition>
```

### Example 22. JSON

```
{  
  "text": "....",  
  "definitionType": "..."  
}
```

### Example 23. RDF

```
<sense> dmlex:definition [  
  a dmlex:Definition ;  
  dmlex:text "...";  
  dmlex:definitionType "...";  
  dmlex:listingOrder 1 ] .
```

### Example 24. Relational database

definitions	
<b>PK</b>	<b>id</b>
<b>FK</b>	senseID
	text
	definitionType
	listingOrder

## 3.7 label

Represents a restriction on its parent such as temporal (old-fashioned, neologism), regional (dialect), register (formal, colloquial), domain (medicine, politics) or grammar (singular-only).

#### Property of

- [entry](#)
- [sense](#)
- [inflectedForm](#)
- [pronunciation](#)
- [example](#)

#### Properties

- `tag` REQUIRED (exactly one) and UNIQUE. Non-empty string. An abbreviation, a code or some other string of text which identifies the label, for example `neo` for neologism, `colloq` for colloquial, `polit` for politics. The `labelTag` object type can be used to explain the meaning of the labels, to constrain which labels are allowed to occur in the lexicographic resource, and to map them onto external inventories and ontologies.
- `listingOrder` REQUIRED (exactly one). Number. The position of this label among other labels of the same entry. This can be implicit from the serialization.

#### Comments

- A label applies to the object that it is a child of. When the label is a child of [entry](#), then it applies to the headword in all its senses. When the label is a child of [sense](#), then it applies to the headword in that sense only (**not** including any subsenses linked to it using the [Linking Module](#)). When the label is a child of [inflectedForm](#), then it applies only to that inflected form of the headword (in all senses). When the label is a child of [pronunciation](#), then it applies only to that pronunciation of the headword (in all senses).

### Example 25. XML

```
<label tag="..." />
```

### Example 26. JSON

```
"..."
```

### Example 27. RDF

```
<entry> dmlex:label [  
  a dmlex:Label ;  
  dmlex:tag "..." ;  
  dmlex:listingOrder 1 ] .
```

### Example 28. Relational database

labels	
<b>PK</b>	<b>id</b>
FK	entryID
FK	senseID
FK	inflectedFormID
FK	pronunciationID
FK	exampleID
	tag
	listingOrder

## 3.8 pronunciation

Represents the pronunciation of its parent. Examples: [Section A.1.3, "Pronunciation given as transcription"](#), [Section A.1.4, "Pronunciation given as a sound file"](#), [Section A.1.5, "Pronunciation given both ways"](#).

#### Property of

- [entry](#)
- [inflectedForm](#)

#### Properties

- At least one of:
  - [soundFile](#) OPTIONAL (zero or one) and UNIQUE. A pointer to a file, such as a filename or a URI, containing a sound recording of the pronunciation
  - [transcription](#) OPTIONAL (zero or more) and UNIQUE.

- `listingOrder` REQUIRED (exactly one). Number. The position of this pronunciation object among other pronunciation objects of the same entry. This can be implicit from the serialization.
- `label` OPTIONAL (zero or more).

*Example 29. XML*

```
<pronunciation soundFile="...">
  <transcription.../>
  <label.../>
</pronunciation>
```

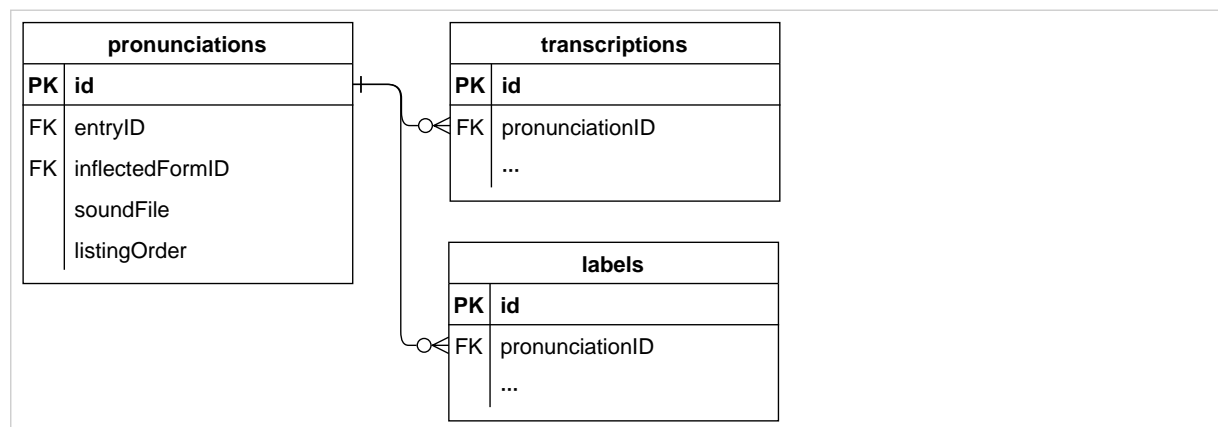
*Example 30. JSON*

```
{
  "soundFile": "...",
  "transcriptions": [...],
  "labels": [...]
}
```

*Example 31. RDF*

```
<entry> dmlex:pronunciation [
  a dmlex:Pronunciation ;
  dmlex:soundFile <...> ;
  dmlex:transcription ... ;
  dmlex:listingOrder 1 ;
  dmlex:label ... ] .
```

*Example 32. Relational database*



## 3.9 transcription

Represents the transcription of a pronunciation in some notation such as IPA.

### Property of

- [pronunciation](#)

### Properties

- `text` REQUIRED (exactly one) and UNIQUE. Non-empty string. The actual transcription.
- `scheme` OPTIONAL (zero or one). IETF language tag. Identifies the transcription scheme used here. Example: `en-fonipa` for English IPA. This can be implicit if the lexicographic resource uses only one transcription scheme throughout. The [transcriptionSchemeTag](#) object type can be used to define which transcription schemes are allowed in the lexicographic resource.
- `listingOrder` REQUIRED (exactly one). Number. The position of this transcription object among transcriptions of the same pronunciation. This can be implicit from the serialization.

### Example 33. XML

```
<transcription scheme="...">...</transcription>
```

### Example 34. JSON

```
{
  "text": "...",
  "scheme": "..."
}
```

### Example 35. RDF

```
<pronunciation> dmlex:transcription [
  a dmlex:Transcription ;
  dmlex:scheme "...";
  dmlex:listingOrder 1 ] .
```

### Example 36. Relational database

transcriptions	
<b>PK</b>	<b>id</b>
<b>FK</b>	pronunciationID
	text
	scheme
	listingOrder

## 3.10 example

Represents a sentence or other text fragment which illustrates the headword being used.

### Property of

- [sense](#)



## Properties

- `text` REQUIRED (exactly one) and UNIQUE. Non-empty string. The example itself.
- `sourceIdentity` OPTIONAL (zero or one). An abbreviation, a code or some other string of text which identifies the source. The `sourceIdentityTag` object type can be used to explain the meaning of the source identifiers, to constrain which source identifiers are allowed to occur in the lexicographic resource, and to map them onto external inventories and ontologies.
- `sourceElaboration` OPTIONAL (zero or one). Non-empty string. A free-form statement about the source of the example. If `sourceIdentity` is present, then `sourceElaboration` can be used for information about where in the source the example can be found: page number, chapter and so on. If `sourceIdentity` is absent then `sourceElaboration` can be used to fully name the source.
- `label` OPTIONAL (zero or more).
- `soundFile` OPTIONAL (zero or one). A pointer to a file, such as a filename or a URI, containing a sound recording of the example.
- `listingOrder` REQUIRED (exactly one). Number. The position of this example object among examples of the same sense. This can be implicit from the serialization.

### Example 37. XML

```
<example sourceIdentity="..." sourceElaboration="..." soundFile="...">
  <text>...</text>
  <label.../>
</example>
```

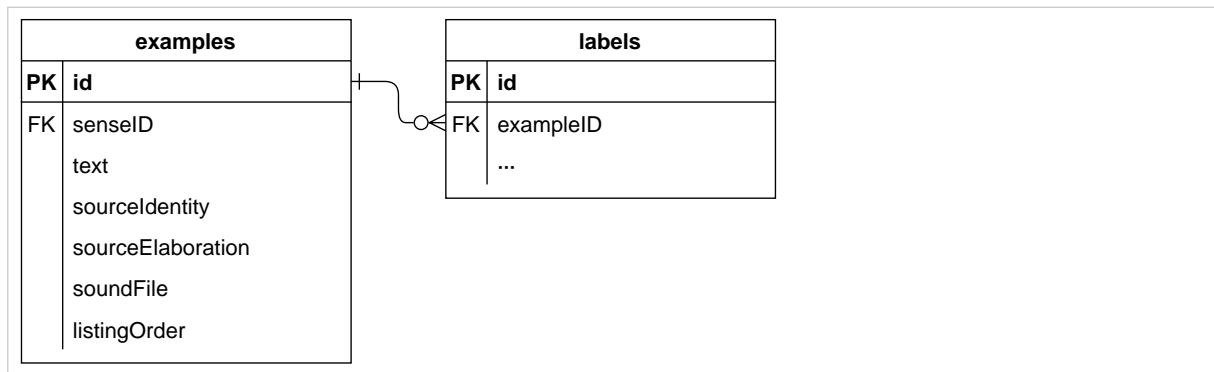
### Example 38. JSON

```
{
  "text": "...",
  "sourceIdentity": "...",
  "sourceElaboration": "...",
  "labels": [...],
  "soundFile": "..."
}
```

### Example 39. RDF

```
<sense> dmlex:example [
  a dmlex:Example ;
  dmlex:text "...";
  dmlex:sourceIdentity "...";
  dmlex:sourceElaboration "...";
  dmlex:label ... ;
  dmlex:soundFile <...> ;
  dmlex:listingOrder 1 ] .
```

Example 40. Relational database



---

## 4 DMLex OPTIONAL Modules (Normative)

### 4.1 DMLex Crosslingual Module

DMLex's Multilingual Module extends the Core and turns a monolingual lexicographic resource into a bilingual or multilingual one. A bilingual or multilingual lexicographic resource is a lexicographic resource with multiple (two or more) languages: the headwords and the examples are in one language (called "the language of the lexicographic resource" in DMLex) and their translations are in one or more other languages (called the translation languages in DMLex).

#### 4.1.1 Extensions to `lexicographicResource`

Extends the `lexicographicResource` object type from the [Core](#).

*Additional properties*

- `translationLanguage` REQUIRED (one or more)

*Example 41. XML*

```
<lexicographicResource ...>
  ...
  <translationLanguage.../>
</lexicographicResource>
```

*Example 42. JSON*

```
{
  ...,
  "translationLanguages": [...]
}
```

*Example 43. RDF*

```
<#lexicographicResource> dmlex:translationLanguage ...
```

#### 4.1.2 `translationLanguage`

Represents one of the languages in which translations are given in this lexicographic resource. Examples: [Section A.1.8, "Defining a bilingual lexicographic resource"](#), [Section A.1.9, "Defining a multilingual lexicographic resource"](#).

*Property of*

- `lexicographicResource`

*Properties*

- `langCode` REQUIRED (exactly one) and UNIQUE. The IETF language code of the language.

- `listingOrder` REQUIRED (exactly one). Number. Sets the order in which translations (of headwords and examples) should be shown. It outranks the listing order given in `headwordTranslation`, `headwordExplanation` and `exampleTranslation` objects.

*Example 44. XML*

```
<translationLanguage langCode="" />
```

*Example 45. JSON*

```
"..."
```

*Example 46. RDF*

```
<#lexicographicResource> dmlex:translationLanguage [
  dmlex:langCode ... ;
  dmlex:listingOrder 0 ] .
```

*Example 47. Relational database*

translationLanguages	
PK	langCode
FK	lexicographicResourceID listingOrder

### 4.1.3 Extensions to sense

Extends the `sense` object type from the [Core](#).

*Additional properties*

- `headwordExplanation` OPTIONAL (zero or more)
- `headwordTranslation` OPTIONAL (zero or more)

*Example 48. XML*

```
<sense ...>
  ...
  <headwordExplanation.../>
  <headwordTranslation.../>
  ...
</sense>
```

#### Example 49. JSON

```
{
  ...
  "headwordExplanations": [...],
  "headwordTranslations": [...],
  ...
}
```

#### Example 50. RDF

```
<#sense>
  dmlex:headwordExplanation ... ;
  dmlex:headwordTranslation ... .
```

### 4.1.4 headwordTranslation

Represents one of possibly multiple translations of a headword. Examples: [Section A.1.10, “How to use headwordTranslation in a bilingual lexicographic resource”](#), [Section A.1.11, “How to use headwordTranslation in a multilingual lexicographic resource”](#).

#### Property of

- [sense](#)

#### Properties

- `text` REQUIRED (exactly one) and UNIQUE (in combination with `langCode` if present). Non-empty string.
- `langCode` OPTIONAL (zero or one) if only one translation language exists in the lexicographic resource, REQUIRED (exactly one) and UNIQUE (in combination with `text`) otherwise. IETF language tag. Indicates the language of this translation. The [translationLanguage](#) datatype can be used to explain the meaning of the language codes that appear here and/or to constrain which language codes are allowed.
- `listingOrder` REQUIRED (exactly one). Number. The position of this translation among other translations of the same sense in the same language. Can be implicit from the serialization.
- `partOfSpeech` OPTIONAL (zero or more).
- `label` OPTIONAL (zero or more).
- `pronunciation` OPTIONAL (zero or more).
- `inflectedForm` OPTIONAL (zero or more).

*Example 51. XML*

```
<headwordTranslation langCode="...">
  <text>...</text>
  <partOfSpeech.../>
  <label.../>
  <pronunciation.../>
  <inflectedForm.../>
</headwordTranslation>
```

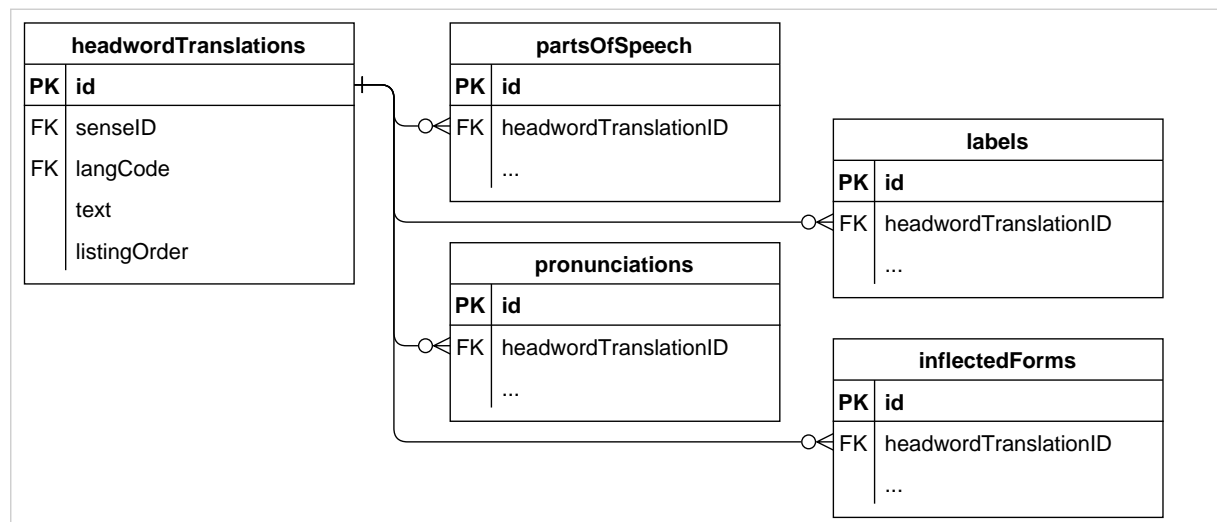
*Example 52. JSON*

```
{
  "langCode": "...",
  "text": "...",
  "partsOfSpeech": [...],
  "labels": [...],
  "pronunciations": [...],
  "inflectedForms": [...]
}
```

*Example 53. RDF*

```
<#sense> dmlex:headwordTranslation [
  dmlex:langCode "...";
  dmlex:text "...";
  dmlex:partOfSpeech ...;
  dmlex:label ...;
  dmlex:pronunciation ...;
  dmlex:inflectedForm ... ] .
```

*Example 54. Relational database*



## 4.1.5 headwordExplanation

Represents a statement in the translation language which explains (but does not translate) the meaning of the headword. Example: [Section A.1.12, "How to use headwordExplanation"](#).

*Property of*

- [sense](#)

*Properties*

- `text` REQUIRED (exactly one) and UNIQUE (in combination with `langCode` if present). Non-empty string.
- `langCode` OPTIONAL (zero or one) if only one translation language exists in the lexicographic resource, REQUIRED (exactly one) and UNIQUE otherwise (in combination with `text`). IETF language tag. Indicates the language in which this explanation is written. The `translationLanguage` datatype can be used to explain the meaning of the language codes that appear here and/or to constrain which language codes are allowed.

*Comments*

- It is assumed that there will always be a maximum of one `headwordExplanation` per translation language in each sense. For this reason, `headwordExplanation` does not have a `listin-gOrder`.

*Example 55. XML*

```
<headwordExplanation langCode="...">...</headwordExplanation>
```

*Example 56. JSON*

```
{  
  "langCode": "...",  
  "text": "...",  
}
```

*Example 57. RDF*

```
<#sense> dmlex:headwordExplanation [  
  dmlex:langCode "...";  
  dmlex:text "..."] .
```

*Example 58. Relational database*

headwordExplanations	
PK	id
FK	senseID
FK	langCode
	text

## 4.1.6 Extensions to example

Extends the [example](#) object type from the [Core](#).

*Additional properties*

- [exampleTranslation](#) OPTIONAL (zero or more)

*Example 59. XML*

```
<example ...>
  ...
  <exampleTranslation.../>
</example>
```

*Example 60. JSON*

```
{
  ...,
  "exampleTranslations": [...]
}
```

*Example 61. RDF*

```
<#example> dmlex:exampleTranslation ... .
```

## 4.1.7 exampleTranslation

Represents the translation of an example.

*Property of*

- [example](#)

*Properties*

- `text` REQUIRED (exactly one) and UNIQUE (in combination with `langCode` if present). Non-empty string.
- `langCode` OPTIONAL (zero or one) if only one translation language exists in the lexicographic resource, REQUIRED (exactly one) and UNIQUE (in combination with `text`) otherwise. IETF language tag. Indicates the language of this translation. The [translationLanguage](#) datatype can be used to explain the meaning of the language codes that appear here and/or to constrain which language codes are allowed.
- `label` OPTIONAL (zero or more).
- `soundFile` OPTIONAL (zero or one). A pointer to a file, such as a filename or a URI, containing a sound recording of the translation.



- `listingOrder` REQUIRED (exactly one). Number. The position of this translation among other translations of the same example in the same language. Can be implicit from the serialization.

*Example 62. XML*

```
<exampleTranslation langCode="..." soundFile="...">
  <text>...</text>
  <label.../>
</exampleTranslation>
```

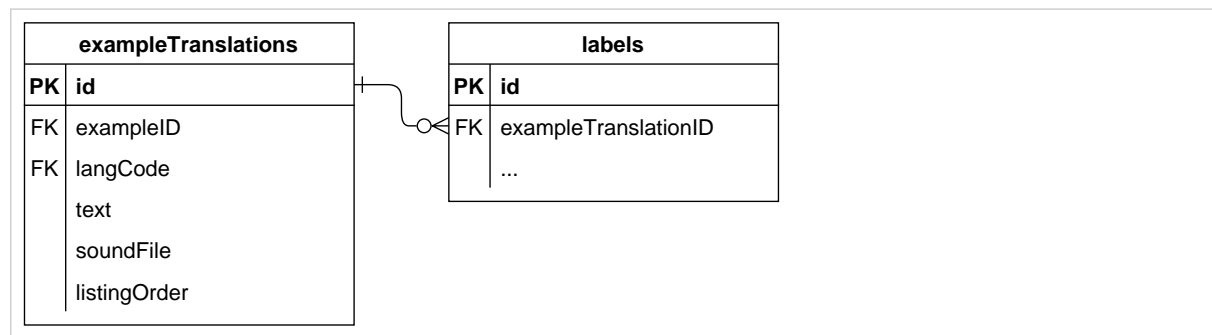
*Example 63. JSON*

```
{
  "langCode": "...",
  "text": "...",
  "labels": [...],
  "soundFile": "..."
}
```

*Example 64. RDF*

```
<#example> dmlex:exampleTranslation [
  dmlex:langCode "...";
  dmlex:text "...";
  dmlex:label ...;
  dmlex:soundFile "..."] .
```

*Example 65. Relational database*



## 4.1.8 Extensions to partOfSpeech

Extends the `partOfSpeech` object type from the [Core](#).

Can additionally be a property of

- [headwordTranslation](#)

### Example 66. Relational database

partsOfSpeech	
PK	id
FK	entryID
FK	headwordTranslationID
	tag
	listingOrder

## 4.1.9 Extensions to label

Extends the [label](#) object type from the [Core](#).

Can additionally be a property of

- [headwordTranslation](#)
- [exampleTranslation](#)

### Example 67. Relational database

labels	
PK	id
FK	entryID
FK	senseID
FK	inflectedFormID
FK	pronunciationID
FK	exampleID
FK	headwordTranslationID
FK	exampleTranslationID
	tag
	listingOrder

## 4.1.10 Extensions to pronunciation

Extends the [pronunciation](#) object type from the [Core](#).

Can additionally be a property of

- [headwordTranslation](#)

### Example 68. Relational database

pronunciations	
PK	id
FK	entryID
FK	inflectedFormID
FK	headwordTranslationID
	soundFile
	listingOrder

## 4.1.11 Extensions to `inflectedForm`

Extends the `inflectedForm` object type from the [Core](#).

Can additionally be a property of

- [headwordTranslation](#)

*Example 69. Relational database*

inflectedForms	
PK	id
FK	entryID
FK	headwordTranslationID
	tag
	text
	listingOrder

## 4.2 DMLex Controlled Values Module

DMLex's Controlled Values Module extends the Core and makes it possible to represent inventories from which the values of various properties come from, such as [parts of speech](#), [labels](#), [inflected form tags](#) and others.

*Comment*

- Treating controlled values as constraints in an implementation of DMLex, for example as business rules in a dictionary-writing system, is OPTIONAL.

### 4.2.1 Extensions to `lexicographicResource`

Extends the `lexicographicResource` object type from the [Core](#).

*Additional properties*

- [definitionTypeTag](#) OPTIONAL (zero or more)
- [inflectedFormTag](#) OPTIONAL (zero or more)
- [labelTag](#) OPTIONAL (zero or more)
- [labelTypeTag](#) OPTIONAL (zero or more)
- [partOfSpeechTag](#) OPTIONAL (zero or more)
- [sourceIdentityTag](#) OPTIONAL (zero or more)
- [transcriptionSchemeTag](#) OPTIONAL (zero or more)

### Example 70. XML

```
<lexicographicResource ...>
  ...
  <definitionTypeTag.../>
  <inflectedFormTag.../>
  <labelTag.../>
  <labelTypeTag.../>
  <partOfSpeechTag.../>
  <sourceIdentityTag.../>
  <transcriptionSchemeTag.../>
</lexicographicResource>
```

### Example 71. JSON

```
{
  ...,
  "definitionTypeTags": [...],
  "inflectedFormTags": [...],
  "labelTags": [...],
  "labelTypeTags": [...],
  "partOfSpeechTags": [...],
  "sourceIdentityTags": [...],
  "transcriptionSchemeTag": [...]
}
```

### Example 72. RDF

```
<#lexicographicResource>
  dmlex:definitionTypeTag ... ;
  dmlex:inflectedFormTag ... ;
  dmlex:labelTag ... ;
  dmlex:labelTypeTag ... ;
  dmlex:partOfSpeechTag ... ;
  dmlex:sourceIdentityTag ... ;
  dmlex:transcriptionSchemeTag ... .
```

## 4.2.2 definitionTypeTag

Represents one (of many) possible values for definitionType of [definition](#).

*Property of*

- [lexicographicResource](#)

*Properties*

- tag REQUIRED (exactly one) and UNIQUE. Non-empty string. An abbreviation, a code or some other string of text.

- `description` OPTIONAL (zero or one). Non-empty string. A human-readable description of what the tag means.
- `sameAs` OPTIONAL (zero or more).

*Example 73. XML*

```
<definitionTypeTag tag="..."
  <description>...</description>
  <sameAs.../>
</definitionTypeTag>
```

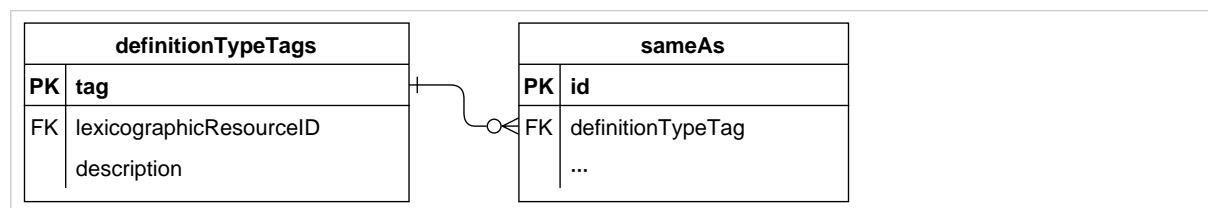
*Example 74. JSON*

```
{
  "tag": "...",
  "description": "...",
  "sameAs": [...]
}
```

*Example 75. RDF*

```
<lexicographicResource> dmlex:definitionTypeTag [
  a dmlex:DefinitionTypeTag ;
  dmlex:tag "...";
  dmlex:description "...";
  owl:sameAs ... ] .
```

*Example 76. Relational database*



## 4.2.3 inflectedFormTag

Represents one (of many) possible values for `tag` of `inflectedForm`. Example: [Section A.1.6, “How to use partOfSpeechTag and inflectedFormTag”](#).

*Property of*

- `lexicographicResource`

*Properties*

- `tag` REQUIRED (exactly one) and UNIQUE. Non-empty string. An abbreviation, a code or some other string of text.

- `description` OPTIONAL (zero or one). Non-empty string. A human-readable description of what the tag means.
- `forHeadwords` OPTIONAL (zero or one). Boolean. If present, indicates whether this tag is intended to be used on the headword side of the lexicographic resource: as `inflectedForm` of `entry`.
- `forTranslations` OPTIONAL (zero or one). Boolean. If present, indicates whether this tag is intended to be used on the translation side of the lexicographic resource: as `headwordTranslation` of `headwordTranslation`.
- `forLanguage` OPTIONAL (zero or more). If present, says that:
  - If this tag is being used inside a `headwordTranslation` object, then it is intended to be used only inside a `headwordTranslation` object labelled with this language.
- `forPartOfSpeech` OPTIONAL (zero or more). If present, says that:
  - If this tag is used as a `inflectedForm` of a `headwordTranslation`, then the `headwordTranslation` must have this part of speech.
  - If this tag is used as a `inflectedForm` of an `entry`, then the `entry` must have this part of speech.
- `sameAs` OPTIONAL (zero or more).

*Example 77. XML*

```
<inflectedFormTag tag="..." forHeadwords="true" forTranslations="true">
  <description>...</description>
  <forLanguage langCode="..."/>
  <forPartOfSpeech tag="..."/>
  <sameAs.../>
</inflectedFormTag>
```

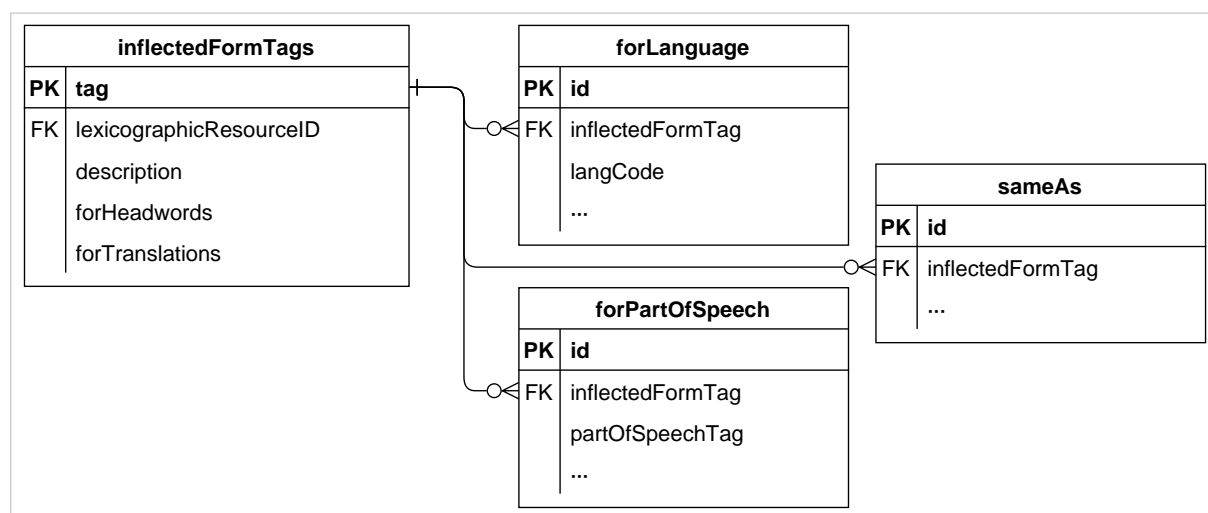
*Example 78. JSON*

```
{
  "tag": "...",
  "description": "...",
  "forHeadwords": true,
  "forTranslations": true,
  "forLanguages": ["..."]
  "forPartsOfSpeech": ["..."],
  "sameAs": [...]
}
```

### Example 79. RDF

```
<lexicographicResource> dmlex:inflectedFormTag [
  a dmlex:InflectedFormTag ;
  dmlex:tag "...";
  dmlex:description "...";
  dmlex:forHeadwods "...";
  dmlex:forTranslations "...";
  dmlex:forLanguage "...";
  dmlex:forPartOfSpeech "...";
  owl:sameAs ... ] .
```

### Example 80. Relational database



## 4.2.4 labelTag

Represents one (of many) possible values for **tag** of [label](#).

Property of

- [lexicographicResource](#)

Properties

- **tag** REQUIRED (exactly one) and UNIQUE. Non-empty string. An abbreviation, a code or some other string of text.
- **description** OPTIONAL (zero or one). Non-empty string. A human-readable description of what the tag means.
- **typeTag** OPTIONAL (zero or one). Non-empty string. An abbreviation, a code or some other string of text which identifies the type of the label, for example `temporal` for temporal labels (archaic, neologism etc) or `dialect` for labels of dialects. The [labelTypeTag](#) object type can be used to explain the meaning of the type tags, to constrain which type tags are allowed to occur in the lexicographic resource, and to map them onto external inventories and ontologies.
- **forHeadwods** OPTIONAL (zero or one). Boolean. If present, indicates whether this tag is intended to be used on the headword side of the lexicographic resource: as `inflectedForm` of [entry](#).

- `forTranslations` OPTIONAL (zero or one). Boolean. If present, indicates whether this tag is intended to be used on the translation side of the lexicographic resource: as `headwordTranslation` of `headwordTranslation`.
- `forCollocates` OPTIONAL (zero or one). Boolean. If present, indicates whether this tag is intended to be used on the collocates annotated inside examples: inside a `collocateMarker` object.
- `forLanguage` OPTIONAL (zero or more). If present, says that if this tag is being used inside a `headwordTranslation` object, then it is intended to be used only inside a `headwordTranslation` object labelled with this language.
- `forPartOfSpeech` OPTIONAL (zero or more). If present, says that:
  - If this tag is used inside a `headwordTranslation`, then it is intended to be used only inside a `headwordTranslation` labelled with this part of speech.
  - If this tag is used outside a `headwordTranslation`, then it is intended to be used only inside entries that are labelled with this part of speech.
- `sameAs` OPTIONAL (zero or more).

*Example 81. XML*

```
<labelTag tag="..." typeTag="..." forHeadwords="true" forTranslations="true" forColloca
  <description>...</description>
  <forLanguage langCode="..."/>
  <forPartOfSpeech tag="..."/>
  <sameAs.../>
</labelTag>
```

*Example 82. JSON*

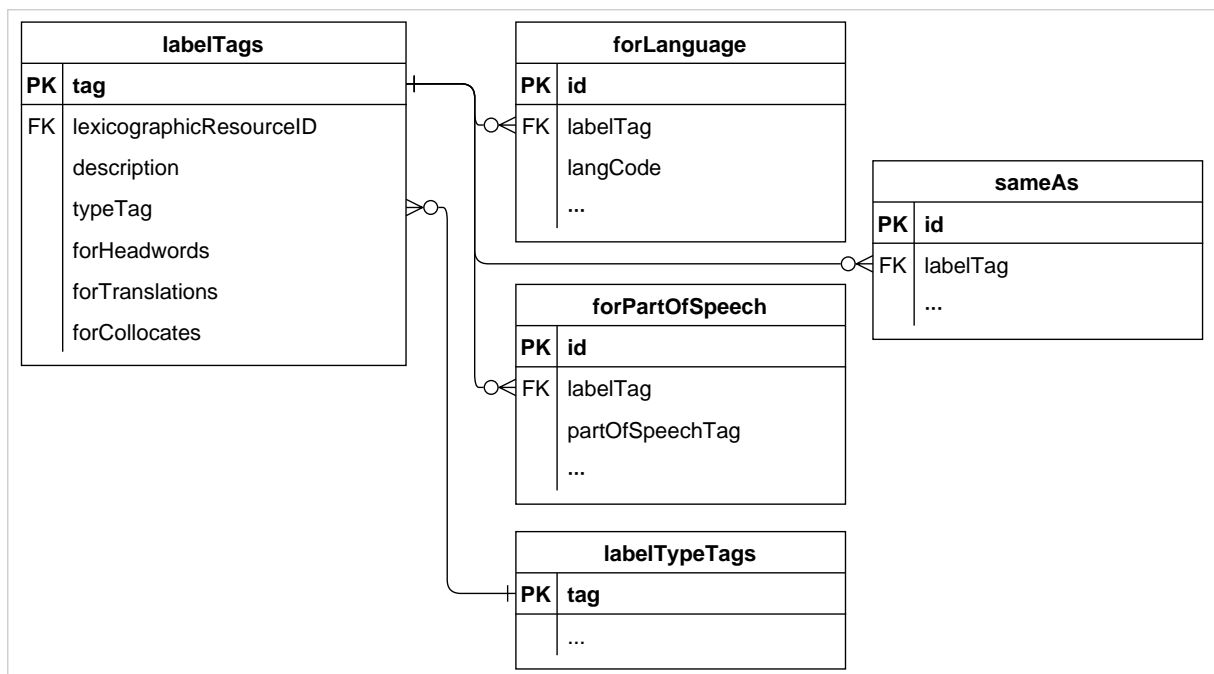
```
{
  "tag": "...",
  "description": "...",
  "typeTag": "...",
  "forHeadwords": true,
  "forTranslations": true,
  "forCollocates": true,
  "forLanguages": ["..."]
  "forPartsOfSpeech": ["..."],
  "sameAs": [...]
}
```



### Example 83. RDF

```
<lexicographicResource> dmlex:labelTag [  
  a dmlex:LabelTag ;  
  dmlex:tag "...";  
  dmlex:description "...";  
  dmlex:typeTag "...";  
  dmlex:forHeadwords "...";  
  dmlex:forTranslations "...";  
  dmlex:forCollocates "...";  
  dmlex:forLanguage "...";  
  dmlex:forPartOfSpeech "...";  
  owl:sameAs ... ] .
```

### Example 84. Relational database



## 4.2.5 labelTypeTag

Represents one (of many) possible values for `typeTag` of `labelTag`.

Property of

- `lexicographicResource`

Properties

- `tag` REQUIRED (exactly one) and UNIQUE. Non-empty string. An abbreviation, a code or some other string of text.
- `description` OPTIONAL (zero or one). Non-empty string. A human-readable description of what the type tag means.
- `sameAs` OPTIONAL (zero or more).

### Example 85. XML

```
<labelTypeTag tag="...">
  <description>...</description>
  <sameAs.../>
</labelTypeTag>
```

### Example 86. JSON

```
{
  "tag": "...",
  "description": "...",
  "sameAs": [...]
}
```

### Example 87. RDF

```
<lexicographicResource> dmlex:labelTypeTag [
  a dmlex:LabelTypeTag ;
  dmlex:tag "...";
  dmlex:description "...";
  owl:sameAs ... ].
```

### Example 88. Relational database



## 4.2.6 partOfSpeechTag

Represents one (of many) possible values for **tag** of **partOfSpeech**. Example: [Section A.1.6, "How to use partOfSpeechTag and inflectedFormTag"](#).

#### Property of

- [lexicographicResource](#)

#### Properties

- **tag** REQUIRED (exactly one) and UNIQUE. Non-empty string. An abbreviation, a code or some other string of text.
- **description** OPTIONAL (zero or one). Non-empty string. A human-readable description of what the tag means.

- `forHeadwords` OPTIONAL (zero or one). Boolean. If present, indicates whether this tag is intended to be used on the headword side of the lexicographic resource: as `partOfSpeech` of [entry](#).
- `forTranslations` OPTIONAL (zero or one). Boolean. If present, indicates whether this tag is intended to be used on the translation side of the lexicographic resource: as `partOfSpeech` of [headwordTranslation](#).
- `forEtymology` OPTIONAL (zero or one). Boolean. If present, indicates whether this tag is intended to be used in etymology: as `partOfSpeech` of [etymon](#).
- `forLanguage` OPTIONAL (zero or more). If present, says that:
  - If this tag is being used inside a [headwordTranslation](#) object, then it is intended to be used only inside a `headwordTranslation` object labelled with this language.
  - If this tag is being used inside a [etymon](#) object, then it is intended to be used only inside an `etymon` object labelled with this language.
- `sameAs` OPTIONAL (zero or more).

**Example 89. XML**

```
<partOfSpeechTag tag="..." forHeadwords="true" forTranslations="true" forEtymology="true"
  <description>...</description>
  <forLanguage langCode="..." />
  <sameAs... />
</partOfSpeechTag>
```

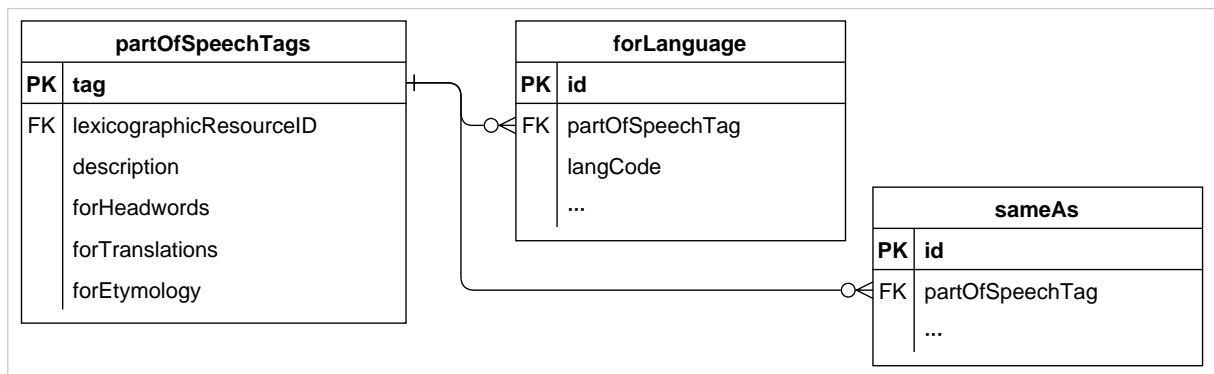
**Example 90. JSON**

```
{
  "tag": "...",
  "description": "...",
  "forHeadwords": true,
  "forTranslations": true,
  "forEtymology": true,
  "forLanguages": ["..."]
  "sameAs": [...]
}
```

**Example 91. RDF**

```
<lexicographicResource> dmlex:partOfSpeechTag [
  a dmlex:PartOfSpeechTag ;
  dmlex:tag "...";
  dmlex:description "...";
  dmlex:forHeadwods "...";
  dmlex:forTranslations "...";
  dmlex:forEtymology "...";
  dmlex:forLanguage "...";
  owl:sameAs ... ] .
```

### Example 92. Relational database



## 4.2.7 sourceIdentityTag

Represents one (of many) possible values for `sourceIdentity` of [example](#).

*Property of*

- [lexicographicResource](#)

*Properties*

- `tag` REQUIRED (exactly one) and UNIQUE. Non-empty string. An abbreviation, a code or some other string of text.
- `description` OPTIONAL (zero or one). Non-empty string. A human-readable description of what the tag means.
- `sameAs` OPTIONAL (zero or more).

### Example 93. XML

```
<sourceIdentityTag tag="..."
  <description>...</description>
  <sameAs.../>
</sourceIdentityTag>
```

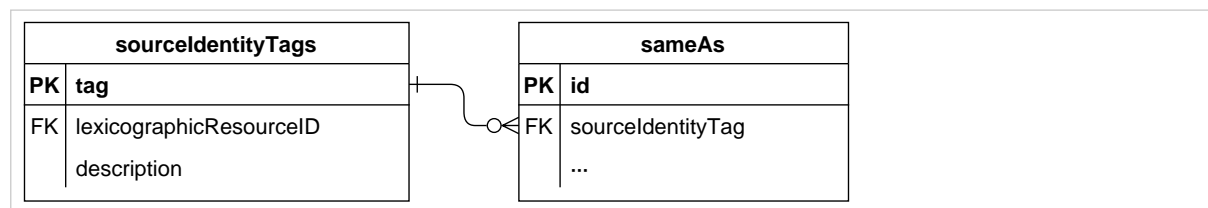
### Example 94. JSON

```
{
  "tag": "...",
  "description": "...",
  "sameAs": [...]
}
```

### Example 95. RDF

```
<lexicographicResource> dmlex:sourceIdentityTag [
  a dmlex:SourceIdentityTag ;
  dmlex:tag "... " ;
  dmlex:description "... " ;
  owl:sameAs ... ] .
```

### Example 96. Relational database



## 4.2.8 transcriptionSchemeTag

Represents one (of many) possible values for `scheme` of `transcription`.

### Property of

- [lexicographicResource](#)

### Properties

- `tag` REQUIRED (exactly one) and UNIQUE. An IETF language tag.
- `description` OPTIONAL (zero or one). Non-empty string. A human-readable description of what the tag means.
- `forHeadwords` OPTIONAL (zero or one). Boolean. If present, indicates whether this tag is intended to be used on the headword side of the lexicographic resource: inside a pronunciation of [entry](#).
- `forTranslations` OPTIONAL (zero or one). Boolean. If present, indicates whether this tag is intended to be used on the translation side of the lexicographic resource: inside a pronunciation of [headwordTranslation](#).
- `forLanguage` OPTIONAL (zero or more). If present, says that if this tag is being used inside a [headwordTranslation](#) object, then it is intended to be used only inside a `headwordTranslation` object labelled with this language.

### Comment

- The `transcriptionSchemeTag` does not have a `sameAs` property because the tag itself - which is an IETF language tag - defines fully what the tag means.

### Example 97. XML

```
<transcriptionSchemeTag tag="..." forHeadwords="true" forTranslations="true">
  <description>...</description>
  <forLanguage langCode="..." />
</transcriptionSchemeTag>
```

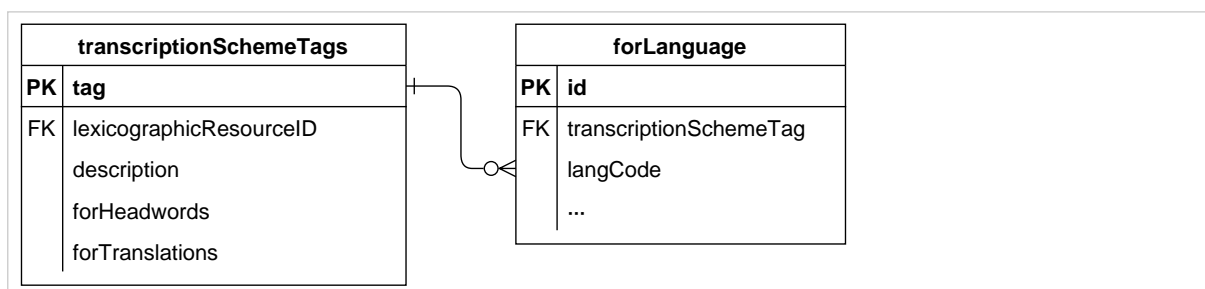
### Example 98. JSON

```
{
  "tag": "...",
  "description": "...",
  "forHeadwords": true,
  "forTranslations": true,
  "forLanguages": ["..."]
}
```

### Example 99. RDF

```
<lexicographicResource> dmlex:transcriptionSchemeTag [
  a dmlex:PartOfSpeechTag ;
  dmlex:tag "...";
  dmlex:description "...";
  dmlex:forHeadwords "...";
  dmlex:forTranslations "...";
  dmlex:forLanguage "..."] .
```

### Example 100. Relational database



## 4.2.9 sameAs

Represents the fact that the parent object is equivalent to an item available from an external authority. Example: [Section A.1.7, "Mapping controlled values to external inventories"](#).

#### Property of

- [definitionTypeTag](#)
- [inflectedFormTag](#)
- [labelTag](#)
- [partOfSpeechTag](#)
- [sourceIdentityTag](#)

#### Properties

- `uri` REQUIRED (exactly one) and UNIQUE. The URI of an item in an external inventory.

### Example 101. XML

```
<sameAs uri="..." />
```

### Example 102. JSON

```
"..."
```

### Example 103. Relational database

sameAs	
<b>PK</b>	<b>id</b>
FK	definitionTypeTag
FK	inflectedFormTag
FK	labelTag
FK	partOfSpeechTag
FK	sourceIdentityTag
	uri

## 4.3 DMLex Linking Module

DMLex's Linking Module can be used to construct relations between objects which "break out" of the tree-like parent-and-child hierarchy constructed from datatypes from the Core and from other modules. The Linking Module can be used to create relations between senses which are synonyms or antonyms, between entries whose headwords are homonyms or spelling variants, between senses which represent superordinate and subordinate concepts (eg. hypernyms and hyponyms, holonyms and meronyms), between entries and subentries, between senses and subsenses, and many others.

Each relation is represented in DMLex by an instance of the `relation` datatype. A relation brings two or more members together. The fact that an object (such as a sense or an entry) is a member of a relation is represented in DMLex by an instance of the `member` datatype.

The Linking Module can be used to set up relations between objects inside the same lexicographic resource, or between objects residing in different lexicographic resources.

Examples: [Section A.1.13, "Modelling parts and wholes"](#), [Section A.1.14, "Modelling antonyms"](#), [Section A.1.15, "Modelling synonyms"](#), [Section A.1.16, "Modelling variants"](#), [Section A.1.17, "Modelling subsenses"](#), [Section A.1.18, "Modelling subentries \(at subsense level\)"](#), [Section A.1.19, "Modelling subentries \(at sense level\)"](#).

### 4.3.1 Extensions to `lexicographicResource`

Extends the `lexicographicResource` object type from the [Core](#).

#### *Additional properties*

- `relation` OPTIONAL (zero or more)
- `relationType` OPTIONAL (zero or more)

#### Example 104. XML

```
<lexicographicResource ...>
  ...
  <relation.../>
  <relationType.../>
</lexicographicResource>
```

#### Example 105. JSON

```
{
  ...,
  "relations": [...],
  "relationTypes": [...]
}
```

#### Example 106. RDF

```
<#lexicographicResource>
  dmlex:relation ... ;
  dmlex:relationType ... .
```

### 4.3.2 relation

Represents the fact that a relation exists between two or more objects.

#### Property of

- [lexicographicResource](#)

#### Properties

- `type` REQUIRED (exactly one) and UNIQUE (in combination with all `member`) properties. Non-empty string. Specifies what type of relation it is, for example a relation between synonyms or a relation between a sense and a subsense. Optionally, [relationType](#) objects can be used to explain those types and to constrain which types of relations are allowed to exist in the lexicographic resource.
- `description` OPTIONAL (zero or one). Non-empty string. A human-readable explanation of this relation.
- `member` REQUIRED (two or more) and UNIQUE (in combination with `type`).

#### Example 107. XML

```
<relation type="...">
  <description>...</description>
  <member.../>
</relation>
```



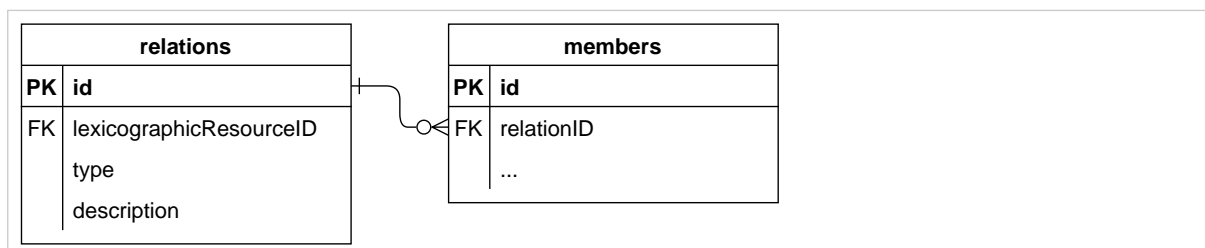
### Example 108. JSON

```
{
  "type": "...",
  "description": "...",
  "members": [...]
}
```

### Example 109. RDF

```
<#lexicographicResource> dmlex:relation [
  a dmlex:Relation ;
  dmlex:type "...";
  dmlex:description "...";
  dmlex:member ... ] .
```

### Example 110. Relational database



## 4.3.3 member

Represents the fact that an object, such as an entry or a sense, is a member of a relation.

#### Property of

- [relation](#)

#### Properties

- `memberID` REQUIRED (exactly one) and UNIQUE (in combination with other unique properties if present). Reference to an object, such as an entry or a sense.
- `role` OPTIONAL (zero or one) and UNIQUE (in combination with other unique properties if present). Non-empty string. An indication of the role the member has in this relation: whether it is the hypernym or the hyponym (in a hyperonymy/hyponymy relation), or whether it is one of the synonyms (in a synonymy relation), and so on. You can use [memberType](#) objects to explain those roles and to constrain which relations are allowed to contain which roles, what their object types are allowed to be (eg. entries or senses) and how many members with this role each relation is allowed to have.
- `listingOrder` REQUIRED (exactly one). Number. The position of this member among other members of the same relation. When showing members of the relation to human users (for example: when listing the synonyms in a synonymy relation), the members should be listed in this order. This can be implicit from the serialization.
- `obverseListingOrder` REQUIRED (exactly one). Number. The position of this relation among other relations this member is involved in. When an object - such as an entry or a sense - is a member

of several relations (for example: when a sense is a member of a synonymy relation and also of an antonymy relation) then, when showing the object (the entry or the sense) to human users, the relations should be listed in this order (for example: the synonyms first, the antonyms second).

*Example 111. XML*

```
<member memberID="..." role="..." obverseListingOrder="..."/>
```

*Example 112. JSON*

```
{  
  "memberID": "...",  
  "role": "...",  
  "obverseListingOrder": "..."  
}
```

*Example 113. RDF*

```
<#relation> dmlex:member [  
  a dmlex:Member ;  
  dmlex:memberID "...";  
  dmlex:role "...";  
  dmlex:listingOrder 0 ;  
  dmlex:obverseListingOrder 0 ] .
```

*Example 114. Relational database*

members	
<b>PK</b>	<b>id</b>
FK	relationID
FK	memberEntryID
FK	memberSenseID
	role
	listingOrder
	obverseListingOrder

## 4.3.4 relationType

Represents one of possible values for the `type` of [relation](#).

*Property of*

- [lexicographicResource](#)

*Properties*

- `type` REQUIRED (exactly one) and UNIQUE. Non-empty string.

- `description` OPTIONAL (zero or one). Non-empty string. A human-readable explanation of this relation type.
- `scopeRestriction` OPTIONAL (zero or one). Non-empty string. Specifies restrictions on member of relations of this type. The possible values are:
  - `sameEntry`: members must be within of the same entry
  - `sameResource`: members must be within the same `lexicographicResource`
  - `any`: no restriction
- `memberType` OPTIONAL (zero or more).
- `sameAs` OPTIONAL (zero or more).

*Example 115. XML*

```
<relationType type="..." scopeRestriction="...">
  <description>...</description>
  <memberType.../>
  <sameAs.../>
</relationType>
```

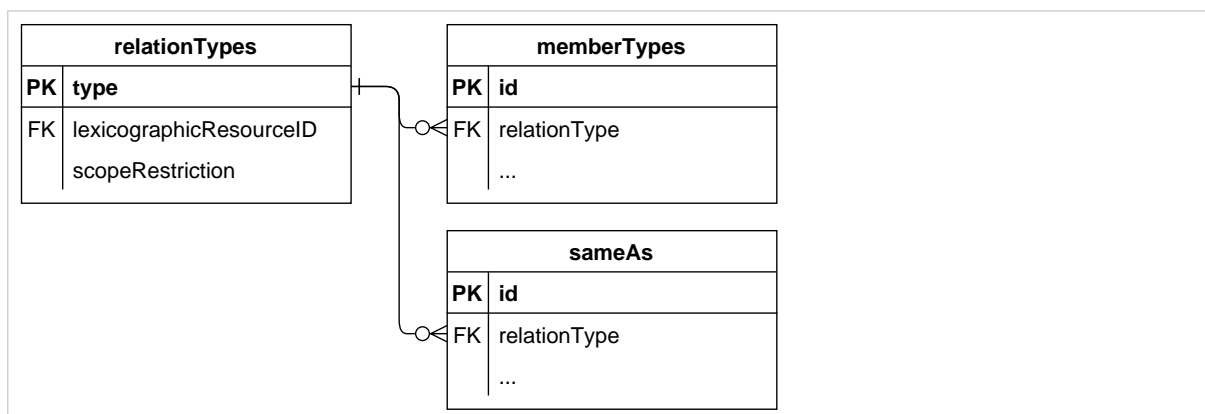
*Example 116. JSON*

```
{
  "type": "...",
  "scopeRestriction": "...",
  "memberTypes": [...],
  "sameAs": ["..."]
}
```

*Example 117. RDF*

```
<#lexicographicResource> dmlex:relationType [
  a dmlex:RelationType ;
  dmlex:type "...";
  dmlex:scopeRestriction "...";
  dmlex:memberType ... ;
  owl:sameAs ... ] .
```

### Example 118. Relational database



## 4.3.5 memberType

Represents one of possible values for the `role` of `member`, as well as various restrictions on members having this role.

Property of

- `relationType`

Properties

- `role` REQUIRED (exactly one) and UNIQUE. String. If the value is empty, then members having this role do not need to have a `role` property.
- `description` OPTIONAL (zero or one). Non-empty string. A human-readable explanation of this member role.
- `type` REQUIRED (exactly one) and UNIQUE. Non-empty string. A restriction on the types of objects that can have this role. The possible values are:
  - `sense`: the object that has this role must be a `sense`.
  - `entry`: the object that has this role must be an `entry`.
  - `collocate`: the object that has this role must be an `collocateMarker` (from the [Linking module](#)).
- `min` OPTIONAL (zero or one). Number. Says that relations of this type must have at least this many members with this role. If omitted then there is no lower limit (effectively, zero).
- `max` OPTIONAL (zero or one). Number. Says that relations of this type may have at most this many members with this role. If omitted then there is no upper limit.
- `hint` OPTIONAL (zero or one). Non-empty string. Suggests what machine agents should do when showing this member to a human user (in the context of another member of the same relation). The possible values are:
  - `embed`: Members that have this role should be shown in their entirety, i.e. the entire entry or the entire sense. This is suitable for relations between entries and subentries, or senses and subsenses.
  - `navigate`: Members that have this role should not be shown in their entirety. A navigable (e.g. clickable) link should be provided instead. This is suitable for relations between synonyms, antonyms, holonyms/heteronyms and similar.
  - `none`: Members that have this role should not be shown.

- `sameAs` OPTIONAL (zero or more).

*Example 119. XML*

```
<memberType role="..." type="..." min="..." max="..." hint="...">
  <description></description>
  <sameAs.../>
</memberType>
```

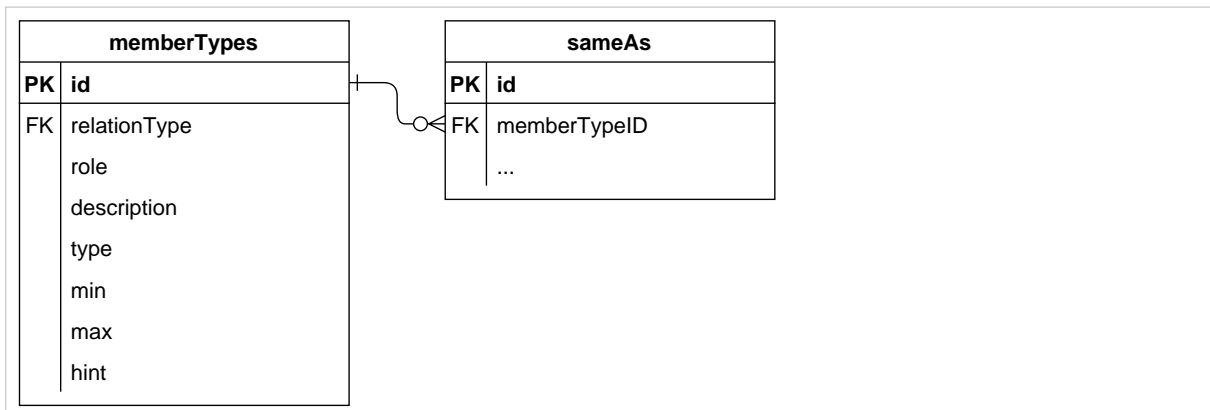
*Example 120. JSON*

```
{
  "role": "...",
  "description": "...",
  "type": "...",
  "min": "...",
  "max": "...",
  "hint": "...",
  "sameAs": [...]
}
```

*Example 121. RDF*

```
<#relationType> dmlex:type [
  a dmlex:MemberType ;
  dmlex:role "...";
  dmlex:description "...";
  dmlex:memberType "...";
  dmlex:min 0 ;
  dmlex:max 0 ;
  dmlex:hint "...";
  owl:sameAs ... ] .
```

*Example 122. Relational database*



### 4.3.6 Extensions to sameAs

Extends the `sameAs` object type from the [Controlled Values Module](#).

Can additionally be a property of

- `relationTypeTag`
- `memberRoleTag`

*Example 123. Relational database*

sameAs	
PK	id
FK	relationType
FK	memberTypeID
	uri

## 4.4 DMLex Annotation Module

This module makes it possible to mark up substrings inside the string values of certain objects and to attach properties to them.

It is up to the implementer to decide how to implement annotations, whether as inline markup (as XML) or as stand-off annotations (for example through start and end indexes).

### 4.4.1 Extensions to entry

Extends the `entry` object type from the [Core](#).

*Additional properties*

- `placeholderMarker` OPTIONAL (zero or more)

*Example 124. XML*

```
<headword>
  ...<placeholderMarker>...</placeholderMarker>...
</headword>
```

*Example 125. JSON*

```
{
  ...,
  "headword": "...",
  "placeholderMarkers": [...],
  ...
}
```

#### Example 126. RDF

```
<id> a dmlex:Entry ;
  dmlex:headword "... " ;
  dmlex:placeholderMarker ... .
```

### 4.4.2 Extensions to headwordTranslation

Extends the [headwordTranslation](#) object type from the [Crosslingual module](#).

#### Additional properties

- [placeholderMarker](#) OPTIONAL (zero or more)

#### Example 127. XML

```
<headwordTranslation>
  <text>
    ...<placeholderMarker>...</placeholderMarker>...
  </text>
</headwordTranslation>
```

#### Example 128. JSON

```
{
  "text": "...",
  "placeholderMarkers": [...],
  ...
}
```

#### Example 129. RDF

```
<id> a dmlex:HeadwordTranslation ;
  dmlex:text "... " ;
  dmlex:placeholderMarker ... .
```

### 4.4.3 placeholderMarker

Marks up a substring inside a headword or inside a headword translation which is not part of the expression itself but stands for things that can take its place. An application can use the inline markup to format the placeholders differently from the rest of the text, to ignore the placeholder in full-text search, and so on. Examples: [Section A.1.20, “Using placeholderMarker”](#), [Section A.1.21, “Using placeholderMarker in a bilingual lexicographic resource”](#).

#### Property of

- [entry](#)
- [headwordTranslation](#)

#### Example 130. XML

```
<placeholderMarker>...</placeholderMarker>
```

#### Example 131. JSON

```
{  
  "startIndex": ...,  
  "endIndex": ...  
}
```

#### Example 132. RDF

```
<#headword> dmlex:placeholderMarker [  
  dmlex:startIndex 0 ;  
  dmlex:endIndex 1 ] .
```

#### Example 133. Relational databases

placeholderMarkers	
PK	id
FK	entryID
FK	headwordTranslationID
	startIndex
	endIndex

### 4.4.4 Extensions to definition

Extends the [definition](#) object type from the [Core](#).

#### Additional properties

- [headwordMarker](#) OPTIONAL (zero or more)
- [collocateMarker](#) OPTIONAL (zero or more)

#### Example 134. XML

```
<definition...>  
  ...  
  <headwordMarker>...</headwordMarker>  
  ...  
  <collocateMarker...>...</collocateMarker>  
  ...  
</definition>
```



#### Example 135. JSON

```
{
  "text": "...",
  "headwordMarkers": [...],
  "collocateMarkers": [...],
  ...
}
```

#### Example 136. RDF

```
<#sense> dmlex:definition [
  dmlex:text "...";
  dmlex:headwordMarker ...;
  dmlex:collocateMarker ... ] .
```

### 4.4.5 Extensions to example

Extends the [example](#) object type from the [Core](#).

#### Additional properties

- [headwordMarker](#) OPTIONAL (zero or more)
- [collocateMarker](#) OPTIONAL (zero or more)

#### Example 137. XML

```
<example>
  <text>
    ...
    <headwordMarker>...</headwordMarker>
    ...
    <collocateMarker...>...</collocateMarker>
    ...
  </text>
</example>
```

#### Example 138. JSON

```
{
  "text": "...",
  "headwordMarkers": [...],
  "collocateMarkers": [...],
  ...
}
```

### Example 139. RDF

```
<#sense> dmlex:example [  
  dmlex:text "...";  
  dmlex:headwordMarker ...;  
  dmlex:collocateMarker ... ] .
```

## 4.4.6 Extensions to exampleTranslation

Extends the [exampleTranslation](#) object type from the [Crosslingual module](#).

### Additional properties

- [headwordMarker](#) OPTIONAL (zero or more)
- [collocateMarker](#) OPTIONAL (zero or more)

### Example 140. XML

```
<exampleTranslation>  
  <text>  
    ...  
    <headwordMarker>...</headwordMarker>  
    ...  
    <collocateMarker...>...</collocateMarker>  
    ...  
  </text>  
</exampleTranslation>
```

### Example 141. JSON

```
{  
  "text": "...",  
  "headwordMarkers": [...],  
  "collocateMarkers": [...],  
  ...  
}
```

### Example 142. RDF

```
<#example> dmlex:exampleTranslation [  
  dmlex:text "...";  
  dmlex:headwordMarker ...;  
  dmlex:collocateMarker ... ] .
```

## 4.4.7 headwordMarker

Marks up a substring inside an example, inside an example translation or inside a definition which corresponds to the headword (or to a translation of the headword). An application can use the inline markup to highlight the occurrence of the headword for human readers through formatting. Example: [Section A.1.22, “Using headwordMarker”](#).

Property of

- [definition](#)
- [example](#)
- [exampleTranslation](#)

Example 143. XML

```
<headwordMarker>...</headwordMarker>
```

Example 144. JSON

```
{  
  "startIndex": ...,  
  "endIndex": ...  
}
```

Example 145. RDF

```
<#definition> dmlex:headwordMarker [  
  dmlex:startIndex 0 ;  
  dmlex:endIndex 0 ] .
```

Example 146. Relational databases

headwordMarkers	
PK	id
FK	definitionID
FK	exampleID
FK	exampleTranslationID
	startIndex
	endIndex

## 4.4.8 collocateMarker

Marks up a substring other than the headword inside an example, inside an example translation or inside a definition. An application can use the inline markup to highlight collocates or constituents. Example: [Section A.1.23, “Using collocateMarker”](#).

### Property of

- [definition](#)
- [example](#)
- [exampleTranslation](#)

### Properties

- `lemma` OPTIONAL (zero or one) and UNIQUE. Non-empty string. The lemmatized form of the collocate. An application can use it to provide a clickable link for the user to search for the lemma in the rest of the lexicographic resource or on the web. (If you want to link the collocate explicitly to a specific entry or to a specific sense in your lexicographic resource, or even in an external lexicographic resource, you can use the Linking Module for that.)
- `label` OPTIONAL (zero or more). Can be used to communicate facts about the role or type of the item in the sentence, for example its syntactic role (subject, direct object etc.), its semantic role (agent, affected etc) or its semantic type (human, institution etc.) The `labelTag` object type can be used to explain and/or constrain the collocate labels that are allowed to appear in the lexicographic resource.

### Example 147. XML

```
<itemMarker id="..." lemma="...">
  ...
  <label tag="..." />
</itemMarker>
```

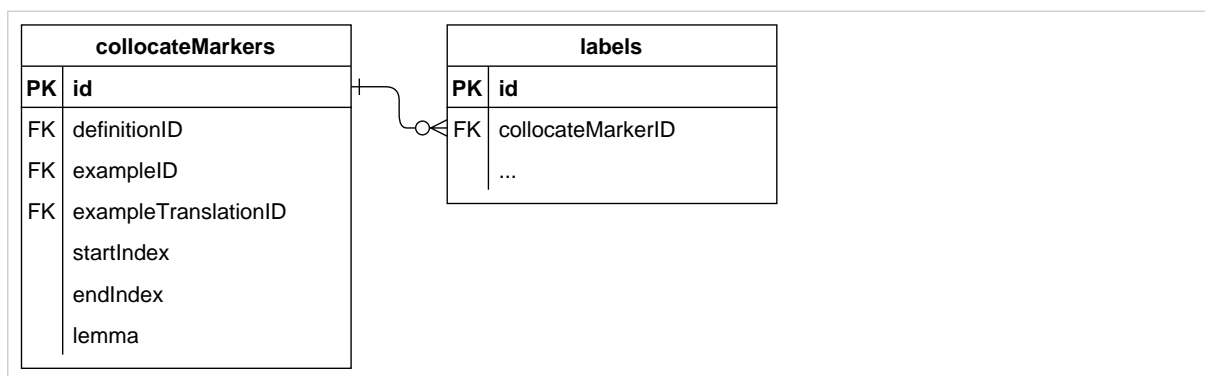
### Example 148. JSON

```
{
  "id": "...",
  "startIndex": ...,
  "endIndex": ...,
  "lemma": "...",
  "labels": ["..."]
}
```

### Example 149. RDF

```
<#definition> dmlex:itemMarker [
  dmlex:id "...";
  dmlex:startIndex 0;
  dmlex:endIndex 0;
  dmlex:lemma "...";
  dmlex:label ... ] .
```

### Example 150. SQL



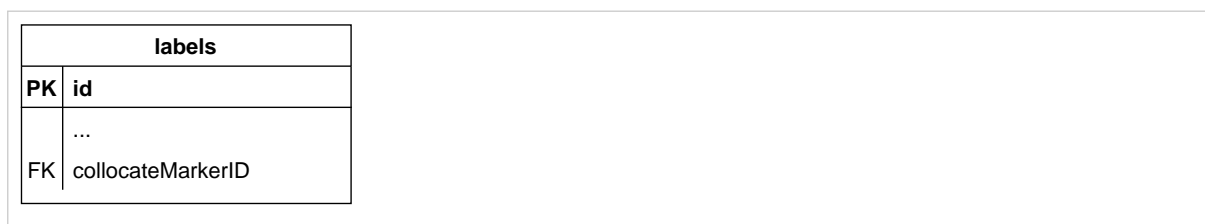
## 4.4.9 Extensions to label

Extends the `label` object type from the [Core](#).

Can additionally be a property of

- `collocateMarker`

### Example 151. Relational database



## 4.5 DMLex Etymology Module

Extends DMLex Core to support the modelling of etymological information in lexicographic resources.

Example: [Section A.1.24, "Modelling etymology"](#).

### 4.5.1 Extensions to entry

Extends the `entry` object type from the [Core](#).

*Additional properties*

- `etymology` OPTIONAL (zero or more) Represents a chain of historical derivations of the entry's headword.

#### Note

If an entry can contains more than one etymology object, then the different etymology objects represent different hypotheses about the origin of the headword.

### Example 152. XML

```
<entry>
  ...
  <etymology>...</etymology>
</entry>
```

### Example 153. JSON

```
{
  ...,
  "etymology": {...}
}
```

### Example 154. RDF

```
<#entry> dmlex:etymology ... .
```

## 4.5.2 etymology

Represents a chain of historical derivations of a word.

*Property of*

- [entry](#)

*Properties*

- `description` OPTIONAL (zero or one) and UNIQUE. A plain-text form of the etymology, which may contain notes about the etymology. This may be used instead of or alongside a structured list of origin and etymon objects.
- `etymon` OPTIONAL (zero or more).
- `listingOrder` REQUIRED (exactly one). Number. The position of this etymology among other etymologies of the same entry. Can be implicit from the serialization.

### Example 155. XML

```
<etymology>
  <description>...</description>
  <etymon>...</etymon>
  <etymon>...</etymon>
  <etymon>...</etymon>
  ...
</etymology>
```

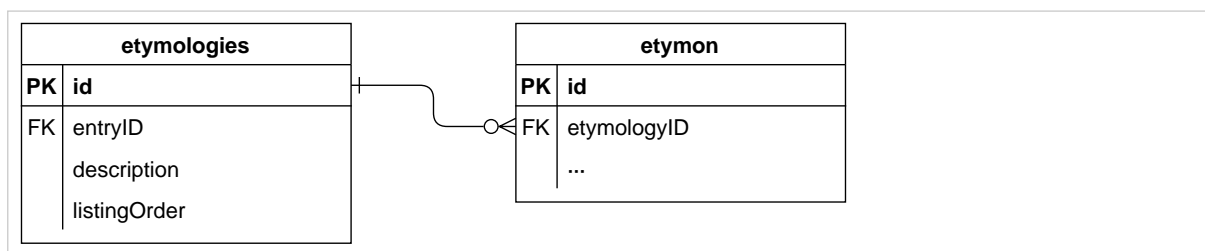
### Example 156. JSON

```
{
  "description": ... ,
  "etymons": [...]
}
```

### Example 157. RDF

```
<#entry> dmlex:etymology [
  a dmlex:Etymology ;
  dmlex:description "... " ;
  dmlex:etymon ... ;
  dmlex:listingOrder: ...] .
```

### Example 158. Relational database



## 4.5.3 etymon

Represents one stage (of possibly several) in the etymological history of the headword.

#### Property of

- [etymology](#)

#### Properties

- `when` OPTIONAL (zero or one) and UNIQUE (in combination with other unique properties if present). Indicates the time period during which this etymological origin is valid. The value is an open-text indication of the time period, in the language of the dictionary.
- `type` OPTIONAL (zero or one). The type of the etymological process that occurred at this stage of the headword's etymological history such as derivation, cognate, borrowing. The values can be explained and constrained using the [etymonType](#) object type.
- `note` OPTIONAL (zero or one). Any additional information about this stage of the headword's etymological history.
- `etymonUnit` REQUIRED (one or more) and UNIQUE (in combination with other unique properties if present).
- `translation` OPTIONAL (zero or one). A translation or gloss of the etymon in the language of the lexicographic resource.
- `listingOrder` REQUIRED (exactly one). Number. The position of this origin among other origins listed in the etymology. Can be implicit from the serialization.

### Example 159. XML

```
<etymon when="..." type="...">
  <note>...</note>
  <etymonUnit>...</etymonUnit>
  <etymonUnit>...</etymonUnit>
  <translation>...</translation>
  ...
</etymon>
```

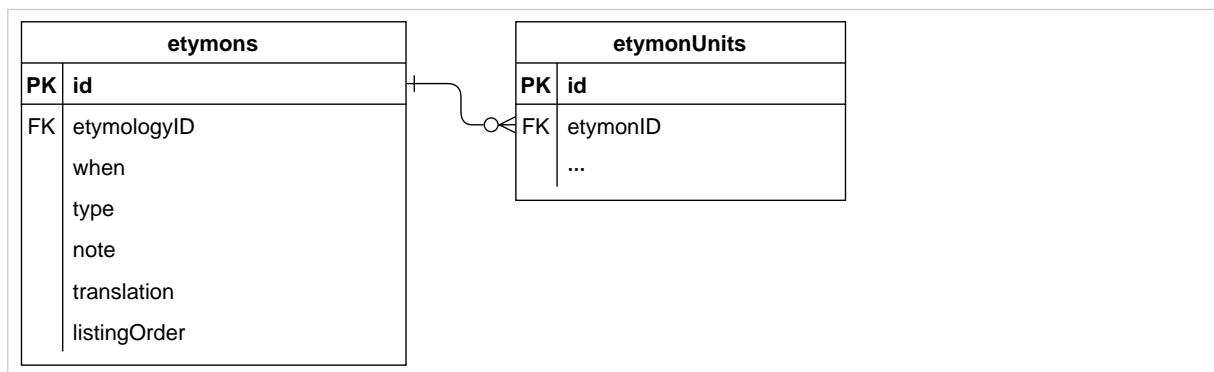
### Example 160. JSON

```
{
  "when": "...",
  "type": "...",
  "note": "...",
  "etymonUnits": [...],
  "translation": "..."
}
```

### Example 161. RDF

```
<#etymon>
  dmlex:when "...";
  dmlex:type "...";
  dmlex:note "...";
  dmlex:etymonUnit ...;
  dmlex:translation ...;
  dmlex:listingOrder 0 .
```

### Example 162. Relational database



## 4.5.4 etymonUnit

Represents a form (typically a word) which is the etymological origin of the headword, or another etymologically related form.

Property of

- [etymon](#)



## Properties

- `langCode` REQUIRED (exactly one) and UNIQUE (in combination with other unique properties if present). An IETF tag. The tags can be explained and constrained using the [etymonLanguage](#) object type.
- `text` REQUIRED (exactly one) and UNIQUE (in combination with other unique properties if present). The written form of the etymon.
- `reconstructed` OPTIONAL (zero or one). Boolean. If present and set to true, indicates that the form is reconstructed and not attested in any corpus.
- `partOfSpeech` OPTIONAL (zero or more).
- `translation` OPTIONAL (zero or one). A translation or gloss of the etymon in the language of the lexicographic resource.
- `listingOrder` REQUIRED (exactly one). Number. The position of this etymon among other etymons of the origin. Can be implicit from the serialization.

### Example 163. XML

```
<etymonUnit langCode="..." reconstructed="true">
  <text>...</text>
  <partOfSpeech tag="..."/>
  <translation>...</translation>
</etymonUnit>
```

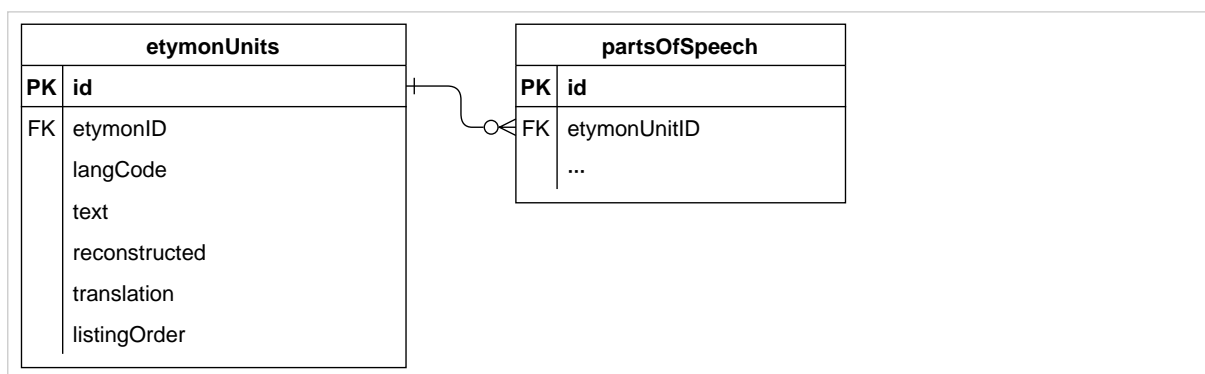
### Example 164. JSON

```
{
  "langCode": "...",
  "text": "...",
  "reconstructed": true,
  "partsOfSpeech": [...],
  "translation": "..."
}
```

### Example 165. RDF

```
<#etymonUnit>
  dmlex:langCode "...";
  dmlex:text "...";
  dmlex:reconstructed true;
  dmlex:partOfSpeech "...";
  dmlex:translation ...;
  dmlex:listingOrder 0 .
```

### Example 166. Relational database



## 4.5.5 Extensions to `lexicographicResource`

Extends the `lexicographicResource` object type from the [Core](#).

### Additional properties

- `etymonLanguage` OPTIONAL (zero or more)
- `etymonType` OPTIONAL (zero or more)

### Example 167. XML

```
<lexicographicResource>
  <etymonLanguage>...</etymonLanguage/>
  ...
</lexicographicResource>
```

### Example 168. JSON

```
{
  "etymonLanguages": [...],
  ...
}
```

### Example 169. RDF

```
<#lexicographicResource> dmlex:etymonLanguage ... .
```

## 4.5.6 `etymonType`

Represents one of possible values for the `type` of `etymon`.

### Property of

- `lexicographicResource`

### Properties

- `type` REQUIRED (exactly one) and UNIQUE. Non-empty string.
- `description` OPTIONAL (zero or one). Non-empty string. A human-readable explanation of this type.

#### Example 170. XML

```
<etymonType type="...">
  <description>...</description>
</etymonType>
```

#### Example 171. JSON

```
{
  "type": "...",
  "description": "..."
}
```

#### Example 172. RDF

```
<#lexicographicResource> dmlex:etymonType [
  dmlex:type "...";
  dmlex:description "..." ] .
```

#### Example 173. Relational database

etymonTypes	
PK	type
FK	lexicographicResourceID
	description

## 4.5.7 etymonLanguage

Represents one of several allowed values for the `language` property of `etymonUnit` objects.

### Property of

- [lexicographicResource](#)

### Properties

- `langCode` REQUIRED (exactly one) and UNIQUE. The IETF language code of the language.
- `displayName` OPTIONAL (zero or one). The name of the etymon language, in the language of the lexicographic resource.

## Note

Etymologies frequently refer to languages that are not covered by ISO standards. It may be necessary to avail of private-use subtags when composing IETF language tags for etymology, and to explain their meaning in the `displayName`.

### Example 174. XML

```
<etymonLanguage langCode="...">
  <displayName>...</displayName>
</etymonLanguage>
```

### Example 175. JSON

```
{
  "langCode": "...",
  "displayName": "..."
}
```

### Example 176. RDF

```
<#etymonLanguage> dmlex:langCode "...";
  dmlex:displayName "... " .
```

### Example 177. Relational database

etymonLanguages	
PK	langCode
FK	lexicographicResourceID displayName

## 4.5.8 Extensions to partOfSpeech

Extends the [partOfSpeech](#) object type from the [Core](#).

Can additionally be a property of

- [etymonUnit](#)

### Example 178. Relational database

partsOfSpeech	
PK	id
	...
FK	etymonUnitID

---

# 5 DMLex REQUIRED Serializations (Normative)

## 5.1 DMLex XML serialization (Normative)

This section specifies a serialization of the DMLex data model in XML (Extensible Markup Language) [XML].

For data exchange purposes, a valid DMLex XML document is one where either a `lexicographicResource` or an `entry` is the document element or one with an arbitrary document element other than `lexicographicResource` or `entry` containing a concatenation of one (and only one) of these two element types.

### 5.1.1 Design Principles (Informative)

The XML serialization of DMLex shown in this document follows these principles:

- The top-level `lexicographicResource` object is implemented as an XML element.
- All other objects are implemented as XML attributes of their parents, unless:
  - the object has an arity other than (0..1) and (1..1)
  - or the object can have child objects
  - or the object's value is human-readable text, such as a headword or a definition.

In such cases the object is implemented as a child XML element of its parent.

### 5.1.2 DMLex XML serialization elements (Normative)

#### 5.1.2.1 XML element: `<lexicographicResource>`

Implements the `lexicographicResource` object type.

##### *Attributes*

- `@title` OPTIONAL
- `@uri` REQUIRED
- `@langCode` REQUIRED

##### *Child elements*

- `<entry>` OPTIONAL (zero or more)

##### *Child elements if implementing the Crosslingual Module*

- `<translationLanguage>` REQUIRED (one or more)

##### *Child elements if implementing the Controlled Values Module*

- `<definitionTypeTag>` OPTIONAL (zero or more)
- `<inflectedFormTag>` OPTIONAL (zero or more)

- `<labelTag>` OPTIONAL (zero or more)
- `<labelTypeTag>` OPTIONAL (zero or more)
- `<partOfSpeechTag>` OPTIONAL (zero or more)
- `<sourceIdentityTag>` OPTIONAL (zero or more)

*Child elements if implementing the Linking Module*

- `<relation>` OPTIONAL (zero or more)
- `<relationType>` OPTIONAL (zero or more)

*Child elements if implementing the Etymology Module*

- `<etymonLanguage>` OPTIONAL (zero or more)
- `<etymonType>` OPTIONAL (zero or more)

### 5.1.2.2 XML element: `<entry>`

Implements the `entry` object type.

*Attributes*

- `@id` OPTIONAL
- `@homographNumber` OPTIONAL

*Child elements*

- `<headword>` REQUIRED (exactly one). If implementing the Annotation Module `<headword>` can contain a mixture of plain text and `<placeholderMarker>` elements.
- `<partOfSpeech>` OPTIONAL (zero or more)
- `<label>` OPTIONAL (zero or more)
- `<pronunciation>` OPTIONAL (zero or more)
- `<inflectedForm>` OPTIONAL (zero or more)
- `<sense>` OPTIONAL (zero or more)

*Child elements if implementing the Etymology Module*

- `<etymology>` OPTIONAL (zero or more)

### 5.1.2.3 XML element: `<partOfSpeech>`

Implements the `partOfSpeech` object type.

*Attributes*

- `@tag` REQUIRED

### 5.1.2.4 XML element: `<inflectedForm>`

Implements the `inflectedForm` object type.

### Attributes

- @tag OPTIONAL

### Child elements

- <text> REQUIRED (exactly one)
- <label> OPTIONAL (zero or more)
- <pronunciation> OPTIONAL (zero or more)

#### 5.1.2.5 XML element: <sense>

Implements the [sense](#) object type.

### Child elements

- @id OPTIONAL
- <indicator> OPTIONAL (zero or one)
- <label> OPTIONAL (zero or more)
- <definition> OPTIONAL (zero or more)
- <example> OPTIONAL (zero or more)

### Child elements if implementing the Crosslingual Module

- <headwordExplanation> OPTIONAL (zero or more)
- <headwordTranslation> OPTIONAL (zero or more)

#### 5.1.2.6 XML element: <definition>

Implements the [definition](#) object type.

### Attributes

- @definitionType OPTIONAL

### Text content

- REQUIRED, implements the `text` property. If implementing the Annotation Module <definition> can contain a mixture of plain text, <headwordMarker> elements and <collocateMarker> elements.

#### 5.1.2.7 XML element: <label>

Implements the [label](#) object type.

### Attributes

- @tag REQUIRED

#### 5.1.2.8 XML element: <pronunciation>

Implements the [pronunciation](#) object type.

### Attributes

- @soundFile OPTIONAL if transcription is present, REQUIRED otherwise

### Child elements

- <transcription> OPTIONAL (zero or more) if @soundFile is present, REQUIRED (one or more) otherwise
- <label> OPTIONAL (zero or more)

### 5.1.2.9 XML element: <transcription>

Implements the [transcription](#) object type.

### Attributes

- @scheme OPTIONAL

### Text content

- REQUIRED, implements the `text` property.

### 5.1.2.10 XML element: <example>

Implements the [example](#) object type.

### Attributes

- @sourceIdentity OPTIONAL
- @sourceElaboration OPTIONAL
- @soundFile OPTIONAL

### Child elements

- <text> REQUIRED (exactly one). If implementing the Annotation Module <text> can contain a mixture of plain text, <headwordMarker> elements and <collocateMarker> elements.
- <label> OPTIONAL (zero or more)

### Child elements if implementing the Crosslingual Module

- <exampleTranslation> OPTIONAL (zero or more)

### 5.1.2.11 XML element: <translationLanguage>

Implements the [translationLanguage](#) object type from the Crosslingual Module.

### Attributes

- @langCode REQUIRED

### 5.1.2.12 XML element: <headwordTranslation>

Implements the [headwordTranslation](#) object type from the Crosslingual Module.

### Attributes

- @langCode OPTIONAL if the ancestor <lexicographicResource> contains exactly one <translationLanguage>, REQUIRED otherwise



### *Child elements*

- `<text>` REQUIRED (exactly one). If implementing the Annotation Module `<text>` can contain a mixture of plain text and `<placeholderMarker>` elements.
- `<partOfSpeech>` OPTIONAL (zero or more)
- `<label>` OPTIONAL (zero or more)
- `<pronunciation>` OPTIONAL (zero or more)
- `<inflectedForm>` OPTIONAL (zero or more)

### **5.1.2.13 XML element: `<headwordExplanation>`**

Implements the `headwordExplanation` object type from the Crosslingual Module.

#### *Attributes*

- `@langCode` OPTIONAL if the ancestor `<lexicographicResource>` contains exactly one `<translationLanguage>`, REQUIRED otherwise

#### *Text content*

- REQUIRED, implements the `text` property.

### **5.1.2.14 XML element: `<exampleTranslation>`**

Implements the `exampleTranslation` object type from the Crosslingual Module.

#### *Attributes*

- `@langCode` OPTIONAL if the ancestor `<lexicographicResource>` contains exactly one `<translationLanguage>`, REQUIRED otherwise
- `@soundFile` OPTIONAL

#### *Child elements*

- `<text>` REQUIRED (exactly one). If implementing the Annotation Module `<text>` can contain a mixture of plain text, `<headwordMarker>` elements and `<collocateMarker>` elements.
- `<label>` OPTIONAL (zero or more)

### **5.1.2.15 XML element: `<partOfSpeechTag>`**

Implements the `partOfSpeechTag` object type from the Controlled Values Module.

#### *Attributes*

- `@tag` REQUIRED
- `@forHeadwords` OPTIONAL, true or false
- `@forTranslations` OPTIONAL, true or false
- `@forEtymology` OPTIONAL, true or false

#### *Child elements*

- `<description>` OPTIONAL (zero or one)

- `<forLanguage>` OPTIONAL (zero or more)
- `<sameAs>` OPTIONAL (zero or more)

#### 5.1.2.16 XML element: `<inflectedFormTag>`

Implements the `inflectedFormTag` object type from the Controlled Values Module.

##### *Attributes*

- `@tag` REQUIRED
- `@forHeadwords` OPTIONAL, true or false
- `@forTranslations` OPTIONAL, true or false

##### *Child elements*

- `<description>` OPTIONAL (zero or one)
- `<forLanguage>` OPTIONAL (zero or more)
- `<forPartOfSpeech>` OPTIONAL (zero or more)
- `<sameAs>` OPTIONAL (zero or more)

#### 5.1.2.17 XML element: `<definitionTypeTag>`

Implements the `definitionTypeTag` object type from the Controlled Values Module.

##### *Attributes*

- `@tag` REQUIRED

##### *Child elements*

- `<description>` OPTIONAL (zero or one)
- `<sameAs>` OPTIONAL (zero or more)

#### 5.1.2.18 XML element: `<labelTag>`

Implements the `labelTag` object type from the Controlled Values Module.

##### *Attributes*

- `@tag` REQUIRED
- `@typeTag` OPTIONAL
- `@forHeadwords` OPTIONAL, true or false
- `@forTranslations` OPTIONAL, true or false
- `@forCollocates` OPTIONAL, true or false

##### *Child elements*

- `<description>` OPTIONAL (zero or one)

- `<forLanguage>` OPTIONAL (zero or more)
- `<forPartOfSpeech>` OPTIONAL (zero or more)
- `<sameAs>` OPTIONAL (zero or more)

#### 5.1.2.19 XML element: `<labelTypeTag>`

Implements the `labelTypeTag` object type from the Controlled Values Module.

##### *Attributes*

- `@tag` REQUIRED

##### *Child elements*

- `<description>` OPTIONAL (zero or one)
- `<sameAs>` OPTIONAL (zero or more)

#### 5.1.2.20 XML element: `<sourceIdentityTag>`

Implements the `sourceIdentityTag` object type from the Controlled Values Module.

##### *Attributes*

- `@tag` REQUIRED

##### *Child elements*

- `<description>` OPTIONAL (zero or one)
- `<sameAs>` OPTIONAL (zero or more)

#### 5.1.2.21 XML element: `<transcriptionSchemeTag>`

Implements the `transcriptionSchemeTag` object type from the Controlled Values Module.

##### *Attributes*

- `@tag` REQUIRED
- `@forHeadwords` OPTIONAL, true or false
- `@forTranslations` OPTIONAL, true or false

##### *Child elements*

- `<description>` OPTIONAL (zero or one)
- `<forLanguage>` OPTIONAL (zero or more)

#### 5.1.2.22 XML element: `<forLanguage>`

Implements the `forLanguage` property of the `partOfSpeechTag`, `inflectedFormTag`, `transcriptionSchemeTag` and `labelTag` object types from the Controlled Values Module.

##### *Attributes*

- `@langCode` REQUIRED

### 5.1.2.23 XML element: <forPartOfSpeech>

Implements the `forPartOfSpeech` property of the `inflectedFormTag` and `labelTag` object types from the Controlled Values Module.

#### Attributes

- `@tag` REQUIRED

### 5.1.2.24 XML element: <sameAs>

Implements the `sameAs` object type from the Controlled Values Module.

#### Attributes

- `@uri` REQUIRED

### 5.1.2.25 XML element: <relation>

Implements the `relation` object type from the Linking Module.

#### Attributes

- `@type` REQUIRED

#### Child elements

- `<description>` OPTIONAL (zero or one)
- `<member>` REQUIRED (two or more)

### 5.1.2.26 XML element: <member>

Implements the `member` object type from the Linking Module.

#### Attributes

- `@memberID` REQUIRED
- `@role` OPTIONAL
- `@obverseListingOrder` REQUIRED

### 5.1.2.27 XML element: <relationType>

Implements the `relationType` object type from the Linking Module.

#### Attributes

- `@type` REQUIRED
- `@scopeRestriction` REQUIRED

#### Child elements

- `<description>` OPTIONAL (zero or one)
- `<memberType>` OPTIONAL (zero or more)
- `<sameAs>` OPTIONAL (zero or more)

### 5.1.2.28 XML element: <memberType>

Implements the [memberType](#) object type from the Linking Module.

#### *Attributes*

- @role REQUIRED
- @type REQUIRED
- @min OPTIONAL
- @max OPTIONAL
- @hint OPTIONAL

#### *Child elements*

- <description> OPTIONAL (zero or one)
- <sameAs> OPTIONAL (zero or more)

### 5.1.2.29 XML element: <placeholderMarker>

Implements the [placeholderMarker](#) object type from the Annotation module. Contains the text which it marks up.

### 5.1.2.30 XML element: <headwordMarker>

Implements the [headwordMarker](#) object type from the Annotation module. Contains the text which it marks up.

### 5.1.2.31 XML element: <collocateMarker>

Implements the [collocateMarker](#) object type from the Annotation module. Contains the text which it marks up, optionally followed by one or more <label> elements.

#### *Attributes*

- @id OPTIONAL
- @lemma OPTIONAL

#### *Child elements*

- <label> OPTIONAL (zero or more)

### 5.1.2.32 XML element: <etymology>

Implements the [etymology](#) object type from the Etymology Module.

#### *Child elements*

- <description> OPTIONAL (zero or one)
- <etymon> OPTIONAL (zero or more)

### 5.1.2.33 XML element: <etymon>

Implements the [etymon](#) object type from the Etymology Module.

### *Attributes*

- @when OPTIONAL
- @type OPTIONAL

### *Child elements*

- <note> OPTIONAL (zero or one)
- <etymonUnit> REQUIRED (one or more)
- <translation> OPTIONAL (zero or one)

### **5.1.2.34 XML element: <etymonUnit>**

Implements the [etymonUnit](#) object type from the Etymology Module.

### *Attributes*

- @langCode REQUIRED
- @reconstructed OPTIONAL

### *Child elements*

- <text> REQUIRED (exactly one)
- <partOfSpeech> OPTIONAL (zero or more)
- <translation> OPTIONAL (zero or one)

### **5.1.2.35 XML element: <etymonType>**

Implements the [etymonType](#) object type from the Etymology Module.

### *Attributes*

- @type REQUIRED

### *Child elements*

- <description> OPTIONAL (zero or one)

### **5.1.2.36 XML element: <etymonLanguage>**

Implements the [etymonLanguage](#) object type from the Etymology Module.

### *Attributes*

- @langCode REQUIRED

### *Child elements*

- <displayName> OPTIONAL (zero or one)

## **5.2 DMLex JSON serialization (Normative)**

This section specifies a serialization of the DMLex data model in JSON (JavaScript Object Notation) [JSON].

For data exchange purposes, a valid DMLex JSON file contains either a `lexicographicResource` object or an `entry` object or a concatenation of one (and only one) of these two object types in the JSON Lines format (i.e. separated by a system-specific newline delimiter).

## 5.2.1 Design principles

The JSON serialization of DMLex shown in this document follows these principles:

- The top-level `lexicographicResource` object is implemented as a JSON object: `{...}`.
- All other objects are implemented as JSON name-value pairs inside their parent JSON object: `{"name": ...}`.
- The values of objects are implemented:
  - If the object has an arity of `(0..1)` or `(1..1)`:
    - If the object cannot have any child objects: as a string or number.
    - If the object can have child objects: as a JSON object.
  - If the object has any other arity:
    - If the object cannot have any child objects: as an array of strings or numbers.
    - If the object can have child objects: as an array of JSON objects.

## 5.2.2 DMLex JSON serialization objects (Normative)

### 5.2.2.1 JSON object: `lexicographicResource`

Implements the `lexicographicResource` object type.

#### *Members*

- "title" OPTIONAL, string
- "uri" REQUIRED, string
- "langCode" REQUIRED, string
- "entries" OPTIONAL, array of zero or more `entry` instances

#### *Members if implementing the Crosslingual Module*

- "translationLanguages" REQUIRED, array of one or more strings implementing the `translationLanguage` object type

#### *Members if implementing the Controlled Values Module*

- "definitionTypeTags" OPTIONAL, array of zero or more `definitionTypeTag` instances
- "inflectedFormTags" OPTIONAL, array of zero or more `inflectedFormTag` instances
- "labelTags" OPTIONAL, array of zero or more `labelTag` instances
- "labelTypeTags" OPTIONAL, array of zero or more `labelTypeTag` instances
- "partOfSpeechTags" OPTIONAL, array of zero or more `partOfSpeechTag` instances
- "sourceIdentityTags" OPTIONAL, array of zero or more `sourceIdentityTag` instances

#### *Members if implementing the Linking Module*

- "relations" OPTIONAL, array of zero or more [relation](#) instances
- "relationTypes" OPTIONAL, array of zero or more [relationType](#) instances

#### *Members if implementing the Etymology Module*

- "etymonLanguage" OPTIONAL, array of zero or more [etymonLanguage](#) instances
- "etymonTypes" OPTIONAL, array of zero or more [etymonType](#) instances

### **5.2.2.2 JSON object: [entry](#)**

Implements the [entry](#) object type.

#### *Members*

- "id" OPTIONAL, string
- "headword" REQUIRED, string
- "homographNumber" OPTIONAL, string
- "partsOfSpeech" OPTIONAL, array of zero or more strings implementing the [partOfSpeech](#) object type
- "labels" OPTIONAL, array of zero or more strings implementing the [label](#) object type
- "pronunciations" OPTIONAL, array of zero or more [pronunciation](#) instances
- "inflectedForms" OPTIONAL, array of zero or more [inflectedForm](#) instances
- "senses" OPTIONAL, array of zero or more [sense](#) instances

#### *Members if implementing the Annotation Module*

- "placeholderMarkers" OPTIONAL, array of zero or more [placeholderMarker](#) instances

#### *Members if implementing the Etymology Module*

- "etymologies" OPTIONAL, array of zero or more [etymology](#) instances

### **5.2.2.3 JSON object: [inflectedForm](#)**

Implements the [inflectedForm](#) object type.

#### *Members*

- "tag" OPTIONAL, string
- "text" REQUIRED, string
- "labels" OPTIONAL, array of zero or more strings implementing the [label](#) object type
- "pronunciations" OPTIONAL, array of zero or more [pronunciation](#) instances

### **5.2.2.4 JSON object: [sense](#)**

Implements the [sense](#) object type.



### *Members*

- "id" OPTIONAL, string
- "indicator" OPTIONAL, string
- "labels" OPTIONAL, array of zero or more strings implementing the [label](#) object type
- "definitions" OPTIONAL, array of zero or more [definition](#) instances
- "examples" OPTIONAL, array of zero or more [example](#) instances

### *Members if implementing the Crosslingual Module*

- "headwordExplanations" OPTIONAL, array of zero or more [headwordExplanation](#) instances
- "headwordTranslations" OPTIONAL, array of zero or more [headwordTranslation](#) instances

## 5.2.2.5 JSON object: [definition](#)

Implements the [definition](#) object type.

### *Members*

- "definitionType" OPTIONAL, string
- "text" REQUIRED, string

### *Members if implementing the Annotation Module*

- "headwordMarkers" OPTIONAL, array of zero or more [headwordMarker](#) instances
- "collocateMarkers" OPTIONAL, array of zero or more [collocateMarker](#) instances

## 5.2.2.6 JSON object: [pronunciation](#)

Implements the [pronunciation](#) object type.

### *Members*

- "soundFile" OPTIONAL if "transcriptions" is present and non-empty, REQUIRED otherwise; string
- "transcriptions" OPTIONAL if "soundFile" is present, REQUIRED otherwise; array of zero or more [transcription](#) instances
- "labels" OPTIONAL, array of zero or more strings implementing the [label](#) object type

## 5.2.2.7 JSON object: [transcription](#)

Implements the [transcription](#) object type.

### *Members*

- "scheme" OPTIONAL, string
- "text" REQUIRED, string

### 5.2.2.8 JSON object: [example](#)

Implements the [example](#) object type.

#### *Members*

- "sourceIdentity" OPTIONAL, string
- "sourceElaboration" OPTIONAL, string
- "soundFile" OPTIONAL, string
- "text" REQUIRED, string
- "labels" OPTIONAL, array of zero or more strings implementing the [label](#) object type

#### *Members if implementing the Annotation Module*

- "headwordMarkers" OPTIONAL, array of zero or more [headwordMarker](#) instances
- "collocateMarkers" OPTIONAL, array of zero or more [collocateMarker](#) instances

#### *Members if implementing the Crosslingual Module*

- "exampleTranslations" OPTIONAL, array of zero or more [exampleTranslation](#) instances

### 5.2.2.9 JSON object: [headwordTranslation](#)

Implements the [headwordTranslation](#) object type from the Crosslingual Module.

#### *Members*

- "langCode" OPTIONAL if the `lexicographicResource` contains exactly one translation-language, REQUIRED otherwise; string
- "text" REQUIRED, string
- "partsOfSpeech" OPTIONAL, array of zero or more strings implementing the [partOfSpeech](#) object type
- "labels" OPTIONAL, array of zero or more strings implementing the [label](#) object type
- "pronunciations" OPTIONAL, array of zero or more [pronunciation](#) instances
- "inflectedForms" OPTIONAL, array of zero or more [inflectedForm](#) instances

#### *Members if implementing the Annotation Module*

- "placeholderMarkers" OPTIONAL, array of zero or more [placeholderMarker](#) instances

### 5.2.2.10 JSON object: [headwordExplanation](#)

Implements the [headwordExplanation](#) object type from the Crosslingual Module.

#### *Members*

- "langCode" OPTIONAL if the `lexicographicResource` contains exactly one translation-language, REQUIRED otherwise; string
- "text" REQUIRED, string

### 5.2.2.11 JSON object: `exampleTranslation`

Implements the `exampleTranslation` object type from the Crosslingual Module.

#### *Members*

- "langCode" OPTIONAL if the `lexicographicResource` contains exactly one `translationLanguage`, REQUIRED otherwise; string
- "text" REQUIRED, string
- "labels" OPTIONAL, array of zero or more strings implementing the `label` object type

#### *Members if implementing the Annotation Module*

- "headwordMarkers" OPTIONAL, array of zero or more `headwordMarker` instances
- "collocateMarkers" OPTIONAL, array of zero or more `collocateMarker` instances

### 5.2.2.12 JSON object: `partOfSpeechTag`

Implements the `partOfSpeechTag` object type from the Controlled Values Module.

#### *Members*

- "tag" REQUIRED, string
- "forHeadwords" OPTIONAL, boolean
- "forTranslations" OPTIONAL, boolean
- "forEtymology" OPTIONAL, boolean
- "description" OPTIONAL, string
- "forLanguages" OPTIONAL, array of zero or more strings implementing the `forLanguage` object type
- "sameAs" OPTIONAL, array of zero or more strings implementing the `sameAs` object type

### 5.2.2.13 JSON object: `inflectedFormTag`

Implements the `inflectedFormTag` object type from the Controlled Values Module.

#### *Members*

- "tag" REQUIRED, string
- "forHeadwords" OPTIONAL, boolean
- "forTranslations" OPTIONAL, boolean
- "description" OPTIONAL, string
- "forLanguages" OPTIONAL, array of zero or more strings implementing the `forLanguage` object type
- "forPartsOfSpeech" OPTIONAL, array of zero or more strings implementing the `forPartOfSpeech` object type
- "sameAs" OPTIONAL, array of zero or more strings implementing the `sameAs` object type

#### 5.2.2.14 JSON object: `definitionTypeTag`

Implements the `definitionTypeTag` object type from the Controlled Values Module.

##### *Members*

- "tag" REQUIRED, string
- "description" OPTIONAL, string
- "sameAs" OPTIONAL, array of zero or more strings implementing the `sameAs` object type

#### 5.2.2.15 JSON object: `labelTag`

Implements the `labelTag` object type from the Controlled Values Module.

##### *Members*

- "tag" REQUIRED, string
- "typeTag" OPTIONAL, string
- "forHeadwords" OPTIONAL, boolean
- "forTranslations" OPTIONAL, boolean
- "forCollocates" OPTIONAL, boolean
- "description" OPTIONAL, string
- "forLanguages" OPTIONAL, array of zero or more strings implementing the `forLanguage` object type
- "forPartsOfSpeech" OPTIONAL, array of zero or more strings implementing the `forPartOfSpeech` object type
- "sameAs" OPTIONAL, array of zero or more strings implementing the `sameAs` object type

#### 5.2.2.16 JSON object: `labelTypeTag`

Implements the `labelTypeTag` object type from the Controlled Values Module.

##### *Members*

- "tag" REQUIRED, string
- "description" OPTIONAL, string
- "sameAs" OPTIONAL, array of zero or more strings implementing the `sameAs` object type

#### 5.2.2.17 JSON object: `sourceIdentityTag`

Implements the `sourceIdentityTag` object type from the Controlled Values Module.

##### *Members*

- "tag" REQUIRED, string
- "description" OPTIONAL, string
- "sameAs" OPTIONAL, array of zero or more strings implementing the `sameAs` object type

### 5.2.2.18 JSON object: `transcriptionSchemeTag`

Implements the `transcriptionSchemeTag` object type from the Controlled Values Module.

#### *Members*

- "tag" REQUIRED, string
- "forHeadwords" OPTIONAL, boolean
- "forTranslations" OPTIONAL, boolean
- "description" OPTIONAL, string
- "forLanguages" OPTIONAL, array of zero or more strings implementing the `forLanguage` object type

### 5.2.2.19 JSON object: `relation`

Implements the `relation` object type from the Linking Module.

#### *Members*

- "type" REQUIRED, string
- "description" OPTIONAL, string
- "members" OPTIONAL, array of two or more `member` instances

### 5.2.2.20 JSON object: `member`

Implements the `member` object type from the Linking Module.

#### *Members*

- "memberID" REQUIRED, string
- "role" OPTIONAL, string
- "obverseListingOrder" REQUIRED, number

### 5.2.2.21 JSON object: `relationType`

Implements the `relationType` object type from the Linking Module.

#### *Members*

- "type" REQUIRED, string
- "scopeRestriction" OPTIONAL, string
- "description" OPTIONAL, string
- "memberTypes" OPTIONAL, array of zero or more `memberType` instances
- "sameAs" OPTIONAL, array of zero or more strings implementing the `sameAs` object type

### 5.2.2.22 JSON object: `memberType`

Implements the `memberType` object type from the Linking Module.

### *Members*

- "role" REQUIRED, string
- "type" REQUIRED, string
- "min" OPTIONAL, number
- "max" OPTIONAL, number
- "hint" OPTIONAL, string
- "description" OPTIONAL, string
- "sameAs" OPTIONAL, array of zero or more strings implementing the [sameAs](#) object type

#### **5.2.2.23 JSON object: [placeholderMarker](#)**

Implements the [placeholderMarker](#) object type from the Annotation Module.

### *Members*

- "startIndex" REQUIRED, number
- "endIndex" REQUIRED, number

#### **5.2.2.24 JSON object: [headwordMarker](#)**

Implements the [headwordMarker](#) object type from the Annotation Module.

### *Members*

- "startIndex" REQUIRED, number
- "endIndex" REQUIRED, number

#### **5.2.2.25 JSON object: [collocateMarker](#)**

Implements the [collocateMarker](#) object type from the Annotation Module.

### *Members*

- "startIndex" REQUIRED, number
- "endIndex" REQUIRED, number
- "id" OPTIONAL, string
- "lemma" OPTIONAL, string
- "labels" OPTIONAL, array of zero or more strings implementing the [label](#) object type

#### **5.2.2.26 JSON object: [etymology](#)**

Implements the [etymology](#) object type from the Etymology Module.

### *Members*

- "description" OPTIONAL, string
- "etymons" OPTIONAL, array of zero or more [etymon](#) instances

### 5.2.2.27 JSON object: `etymon`

Implements the `etymon` object type from the Etymology Module.

#### *Members*

- "when" OPTIONAL, string
- "type" OPTIONAL, string
- "note" OPTIONAL, string
- "etymonUnits" REQUIRED, array of one or more `etymonUnit` instances
- "translation" OPTIONAL, string

### 5.2.2.28 JSON object: `etymonUnit`

Implements the `etymonUnit` object type from the Etymology Module.

#### *Members*

- "langCode" REQUIRED, string
- "reconstructed" OPTIONAL, boolean
- "text" REQUIRED, string
- "partsOfSpeech" OPTIONAL, array of zero or more strings implementing the `partOfSpeech` object type
- "translation" OPTIONAL, string

### 5.2.2.29 JSON object: `etymonType`

Implements the `etymonType` object type from the Etymology Module.

#### *Members*

- "type" REQUIRED, string
- "description" OPTIONAL, string

### 5.2.2.30 JSON object: `etymonLanguage`

Implements the `etymonLanguage` object type from the Etymology Module.

#### *Members*

- "langCode" REQUIRED, string
- "displayName" OPTIONAL, string

## 5.3 DMLex RDF serialization (Normative)

### 5.3.1 Design principles

The RDF serialization used in this document follows the following principles

- The RDF serialization uses an vocabulary defined at the namespace <https://www.oasis-open.org/to-be-confirmed/dmlex>, all elements of the vocabulary are in this namespace
- The vocabulary file provides RDF Schema and OWL axioms which implement the restrictions described by the data model. It also provides a linking to the [OntoLex-Lemon](#) model. The ontology provides rules to infer an OntoLex-lemon model from DMLex data. This is achieved by means of subclass and subproperty axioms and so conversion can be performed by an OWL reasoner.
- All elements of the modules are modelled as individuals with an appropriate class definitions
- Each other object type is associated with a property (with a lowercase initial letter) and a class (with an uppercase initial letter. In all case the range of this property is the class
- Arity of properties is implemented by means of OWL class restrictions.
- Listing order can be specified with the property `dmlex:listingOrder` with values starting from 1 and ascending
- There is not a specific `id` property, instead the URI of the entity acts as the ID
- There is no property for [sameAs](#) in the controlled values module. Instead you can use OWL's `sameAs` property
- The implementation is not aware of which modules are in use, to enable publishing on the web. As such cardinality constraints that are only required in a module are not implemented.
- The domain of some properties is given as a `HasProperty` class when it can be used with multiple classes. As such, classes may be subclasses of several different `HasProperty` classes

## 5.3.2 DMLex RDF serialization elements

### 5.3.2.1 RDF element: `dmlex:LexicographicResource`

Represents a dictionary

#### *Superclasses*

- <http://www.w3.org/ns/lemon/lime#Lexicon>

#### *Properties*

- `dmlex:title` OPTIONAL (at most 1) of type <http://www.w3.org/2000/01/rdf-schema#Literal>
- `dmlex:uri` OPTIONAL (at most 1) of type <http://www.w3.org/2001/XMLSchema#anyURI>
- `dmlex:entry` OPTIONAL reference to [Entry](#) (subproperty of <http://www.w3.org/ns/lemon/lime#entry>)
- `dmlex:translationLanguage` OPTIONAL reference to [TranslationLanguage](#)
- `dmlex:definitionTypeTag` OPTIONAL reference to [DefinitionTypeTag](#)
- `dmlex:inflectedFormTag` OPTIONAL reference to [InflectedFormTag](#)
- `dmlex:labelTypeTag` OPTIONAL reference to [LabelTypeTag](#)
- `dmlex:partOfSpeechTag` OPTIONAL reference to [PartOfSpeechTag](#)



- `dmlex:sourceIdentityTag` OPTIONAL reference to [SourceIdentityTag](#)
- `dmlex:transcriptionSchemeTag` OPTIONAL reference to [TranscriptionSchemeTag](#)
- `dmlex:relation` OPTIONAL reference to [Relation](#)
- `dmlex:relationType` OPTIONAL reference to [RelationType](#)
- `dmlex:etymonType` OPTIONAL reference to [EtymonType](#)
- `dmlex:etymonLanguage` OPTIONAL reference to [EtymonLanguage](#)
- `dmlex:language` REQUIRED (exactly 1) of type `http://www.w3.org/2001/XMLSchema#language` (subproperty of `http://www.w3.org/ns/lemon/lime#language`)

### 5.3.2.2 RDF element: `dmlex:Entry`

Represents a dictionary entry

#### *Superclasses*

- `http://www.w3.org/ns/lemon/ontolex#LexicalEntry`

#### *Properties*

- `dmlex:headword` REQUIRED (exactly 1) of type `http://www.w3.org/2000/01/rdf-schema#Literal`
- `dmlex:homographNumber` OPTIONAL (at most 1) of type `http://www.w3.org/2001/XMLSchema#nonNegativeInteger`
- `dmlex:sense` OPTIONAL reference to [Sense](#) (subproperty of `http://www.w3.org/ns/lemon/ontolex#evokes`)
- `dmlex:etymology` OPTIONAL reference to [Etymology](#)
- `dmlex:partOfSpeech` OPTIONAL reference to [PartOfSpeech](#)
- `dmlex:label` OPTIONAL reference to [Label](#)
- `dmlex:pronunciation` OPTIONAL reference to [Pronunciation](#)
- `dmlex:inflectedForm` OPTIONAL reference to [InflectedForm](#)
- `dmlex:placeholderMarker` OPTIONAL reference to [PlaceholderMarker](#)

### 5.3.2.3 RDF element: `dmlex:PartOfSpeech`

Represents a part of speech label

#### *Properties*

- `dmlex:listingOrder` REQUIRED (exactly 1) of type `http://www.w3.org/2001/XMLSchema#nonNegativeInteger`
- `dmlex:tag` REQUIRED (exactly 1)

### 5.3.2.4 RDF element: `dmlex:InflectedForm`

Represents one (of possibly many) inflected forms of the headword

### *Superclasses*

- <http://www.w3.org/ns/lemon/ontolex#Form>

### *Properties*

- `dmlex:listingOrder` REQUIRED (exactly 1) of type <http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
- `dmlex:label` OPTIONAL reference to [Label](#)
- `dmlex:pronunciation` OPTIONAL reference to [Pronunciation](#)
- `dmlex:tag` OPTIONAL (at most 1)
- `dmlex:text` REQUIRED (exactly 1) of type <http://www.w3.org/2000/01/rdf-schema#Literal>

### **5.3.2.5 RDF element: `dmlex:Sense`**

Represents one of possibly many meanings (or meaning potentials) of the headword.

### *Superclasses*

- <http://www.w3.org/ns/lemon/ontolex#LexicalConcept>

### *Properties*

- `dmlex:listingOrder` REQUIRED (exactly 1) of type <http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
- `dmlex:label` OPTIONAL reference to [Label](#)
- `dmlex:indicator` OPTIONAL (at most 1) reference to [Indicator](#)
- `dmlex:definition` OPTIONAL reference to [Definition](#)
- `dmlex:example` OPTIONAL reference to [Example](#)
- `dmlex:headwordExplanation` OPTIONAL reference to [HeadwordExplanation](#)
- `dmlex:headwordTranslation` OPTIONAL reference to [HeadwordTranslation](#)

### **5.3.2.6 RDF element: `dmlex:Definition`**

Represents one of possibly several definitions of a sense.

### *Properties*

- `dmlex:listingOrder` REQUIRED (exactly 1) of type <http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
- `dmlex:text` REQUIRED (exactly 1) of type <http://www.w3.org/2000/01/rdf-schema#Literal>
- `dmlex:definitionType` OPTIONAL (at most 1) reference to [DefinitionType](#)
- `dmlex:headwordMarker` OPTIONAL reference to [HeadwordMarker](#)
- `dmlex:collocateMarker` OPTIONAL reference to [CollocateMarker](#)

### 5.3.2.7 RDF element: `dmlex:Label`

Represents a restriction on its parent such as temporal (old-fashioned, neologism), regional (dialect), register (formal, colloquial), domain (medicine, politics) or grammar (singular-only).

#### *Properties*

- `dmlex:listingOrder` REQUIRED (exactly 1) of type `http://www.w3.org/2001/XMLSchema#nonNegativeInteger`
- `dmlex:tag` REQUIRED (exactly 1)

### 5.3.2.8 RDF element: `dmlex:Pronunciation`

Represents the pronunciation of its parent. Pronunciations must have either a sound file or a transcription.

#### *Properties*

- `dmlex:listingOrder` REQUIRED (exactly 1) of type `http://www.w3.org/2001/XMLSchema#nonNegativeInteger`
- `dmlex:label` OPTIONAL reference to [Label](#)
- `dmlex:soundFile` OPTIONAL (at most 1) of type `http://www.w3.org/2001/XMLSchema#anyURI`
- `dmlex:transcription` OPTIONAL reference to [Transcription](#)

### 5.3.2.9 RDF element: `dmlex:Transcription`

Represents the transcription of a pronunciation in some notation such as IPA.

#### *Properties*

- `dmlex:listingOrder` REQUIRED (exactly 1) of type `http://www.w3.org/2001/XMLSchema#nonNegativeInteger`
- `dmlex:text` REQUIRED (exactly 1) of type `http://www.w3.org/2000/01/rdf-schema#Literal`
- `dmlex:scheme` OPTIONAL (at most 1) of type `http://www.w3.org/2001/XMLSchema#language`

### 5.3.2.10 RDF element: `dmlex:Example`

Represents a sentence or other text fragment which illustrates the headword being used.

#### *Properties*

- `dmlex:listingOrder` REQUIRED (exactly 1) of type `http://www.w3.org/2001/XMLSchema#nonNegativeInteger`
- `dmlex:label` OPTIONAL reference to [Label](#)
- `dmlex:text` REQUIRED (exactly 1) of type `http://www.w3.org/2000/01/rdf-schema#Literal`
- `dmlex:soundFile` OPTIONAL (at most 1) of type `http://www.w3.org/2001/XMLSchema#anyURI`

- `dmlex:sourceIdentity` OPTIONAL (at most 1) of type `http://www.w3.org/2000/01/rdf-schema#Literal`
- `dmlex:sourceElaboration` OPTIONAL (at most 1) of type `http://www.w3.org/2000/01/rdf-schema#Literal`
- `dmlex:exampleTranslation` OPTIONAL reference to [ExampleTranslation](#)
- `dmlex:headwordMarker` OPTIONAL reference to [HeadwordMarker](#)
- `dmlex:collocateMarker` OPTIONAL reference to [CollocateMarker](#)

### 5.3.2.11 RDF element: `dmlex:TranslationLanguage`

Represents one of the languages in which translations are given in this lexicographic resource

#### *Properties*

- `dmlex:listingOrder` REQUIRED (exactly 1) of type `http://www.w3.org/2001/XMLSchema#nonNegativeInteger`
- `dmlex:langCode` REQUIRED (exactly 1) of type `http://www.w3.org/2001/XMLSchema#language`

### 5.3.2.12 RDF element: `dmlex:HeadwordTranslation`

Represents one of possibly multiple translations of a headword

#### *Properties*

- `dmlex:listingOrder` REQUIRED (exactly 1) of type `http://www.w3.org/2001/XMLSchema#nonNegativeInteger`
- `dmlex:language` OPTIONAL (at most 1) of type `http://www.w3.org/2001/XMLSchema#language` (subproperty of `http://www.w3.org/ns/lemon/lime#language`)
- `dmlex:partOfSpeech` OPTIONAL reference to [PartOfSpeech](#)
- `dmlex:label` OPTIONAL reference to [Label](#)
- `dmlex:pronunciation` OPTIONAL reference to [Pronunciation](#)
- `dmlex:inflectedForm` OPTIONAL reference to [InflectedForm](#)
- `dmlex:text` REQUIRED (exactly 1) of type `http://www.w3.org/2000/01/rdf-schema#Literal`
- `dmlex:placeholderMarker` OPTIONAL reference to [PlaceholderMarker](#)

### 5.3.2.13 RDF element: `dmlex:HeadwordExplanation`

Represents a statement in the translation language which explains (but does not translate) the meaning of the headword.

#### *Properties*

- `dmlex:language` OPTIONAL (at most 1) of type `http://www.w3.org/2001/XMLSchema#language` (subproperty of `http://www.w3.org/ns/lemon/lime#language`)
- `dmlex:text` REQUIRED (exactly 1) of type `http://www.w3.org/2000/01/rdf-schema#Literal`

### 5.3.2.14 RDF element: `dmlex:ExampleTranslation`

Represents the translation of an example.

#### *Properties*

- `dmlex:listingOrder` OPTIONAL of type `http://www.w3.org/2001/XMLSchema#nonNegativeInteger`
- `dmlex:language` OPTIONAL (at most 1) of type `http://www.w3.org/2001/XMLSchema#language` (subproperty of `http://www.w3.org/ns/lemon/lime#language`)
- `dmlex:label` OPTIONAL reference to [Label](#)
- `dmlex:text` REQUIRED (exactly 1) of type `http://www.w3.org/2000/01/rdf-schema#Literal`
- `dmlex:soundFile` OPTIONAL of type `http://www.w3.org/2001/XMLSchema#anyURI`
- `dmlex:headwordMarker` OPTIONAL reference to [HeadwordMarker](#)
- `dmlex:collocateMarker` OPTIONAL reference to [CollocateMarker](#)

### 5.3.2.15 RDF element: `dmlex:DefinitionTypeTag`

Represents one (of many) possible values for `definitionType` of `definition`.

#### *Properties*

- `dmlex:tag` REQUIRED (exactly 1)
- `dmlex:description` OPTIONAL (at most 1) of type `http://www.w3.org/2000/01/rdf-schema#Literal`

### 5.3.2.16 RDF element: `dmlex:InflectedFormTag`

Represents one (of many) possible values for `tag` of `inflectedForm`

#### *Properties*

- `dmlex:inflectedForm` OPTIONAL reference to [InflectedForm](#)
- `dmlex:tag` REQUIRED (exactly 1)
- `dmlex:description` OPTIONAL (at most 1) of type `http://www.w3.org/2000/01/rdf-schema#Literal`
- `dmlex:forPartOfSpeech` OPTIONAL of type `http://www.w3.org/2000/01/rdf-schema#Literal`

### 5.3.2.17 RDF element: `dmlex:LabelTag`

Represents one (of many) possible values for `tag` of `label`.

#### *Properties*

- `dmlex:inflectedForm` OPTIONAL reference to [InflectedForm](#)
- `dmlex:tag` REQUIRED (exactly 1)

- `dmlex:description` OPTIONAL (at most 1) of type `http://www.w3.org/2000/01/rdf-schema#Literal`
- `dmlex:forPartOfSpeech` OPTIONAL of type `http://www.w3.org/2000/01/rdf-schema#Literal`
- `dmlex:forCollocates` OPTIONAL (at most 1) of type `http://www.w3.org/2001/XMLSchema#boolean`

### 5.3.2.18 RDF element: `dmlex:LabelTypeTag`

Represents one (of many) possible values for `typeTag` of `labelTag`.

#### *Properties*

- `dmlex:tag` REQUIRED (exactly 1)
- `dmlex:description` REQUIRED (exactly 1) of type `http://www.w3.org/2000/01/rdf-schema#Literal`

### 5.3.2.19 RDF element: `dmlex:PartOfSpeechTag`

Represents one (of many) possible values for `tag` of `partOfSpeech`.

#### *Properties*

- `dmlex:inflectedForm` OPTIONAL reference to [InflectedForm](#)
- `dmlex:tag` REQUIRED (exactly 1)
- `dmlex:description` OPTIONAL (at most 1) of type `http://www.w3.org/2000/01/rdf-schema#Literal`
- `dmlex:forEtymology` OPTIONAL (at most 1) of type `http://www.w3.org/2001/XMLSchema#boolean`

### 5.3.2.20 RDF element: `dmlex:SourceIdentityTag`

Represents one (of many) possible values for `sourceIdentity` of `example`.

#### *Properties*

- `dmlex:tag` REQUIRED (exactly 1)
- `dmlex:description` OPTIONAL (at most 1) of type `http://www.w3.org/2000/01/rdf-schema#Literal`

### 5.3.2.21 RDF element: `dmlex:TranscriptionSchemeTag`

Represents one (of many) possible values for `scheme` of `transcription`.

#### *Properties*

- `dmlex:inflectedForm` OPTIONAL reference to [InflectedForm](#)
- `dmlex:tag` REQUIRED (exactly 1)
- `dmlex:description` OPTIONAL (at most 1) of type `http://www.w3.org/2000/01/rdf-schema#Literal`

### 5.3.2.22 RDF element: `dmlex:Relation`

Represents the fact that a relation exists between two or more objects.

#### *Properties*

- `dmlex:description` OPTIONAL (at most 1) of type `http://www.w3.org/2000/01/rdf-schema#Literal`
- `dmlex:type` REQUIRED (exactly 1) of type `http://www.w3.org/2001/XMLSchema#string`
- `dmlex:member` REQUIRED (at least 2) reference to [Member](#)

### 5.3.2.23 RDF element: `dmlex:Member`

Represents the fact that an object, such as an entry or a sense, is a member of a relation.

#### *Properties*

- `dmlex:listingOrder` REQUIRED (exactly 1) of type `http://www.w3.org/2001/XMLSchema#nonNegativeInteger`
- `dmlex:memberID` REQUIRED (exactly 1) of type `http://www.w3.org/2001/XMLSchema#string`
- `dmlex:obverseListingOrder` REQUIRED (exactly 1) of type `http://www.w3.org/2001/XMLSchema#integer`
- `dmlex:role` OPTIONAL (at most 1) of type `http://www.w3.org/2001/XMLSchema#string`

### 5.3.2.24 RDF element: `dmlex:RelationType`

Represents one of possible values for the type of relation.

#### *Properties*

- `dmlex:description` OPTIONAL (at most 1) of type `http://www.w3.org/2000/01/rdf-schema#Literal`
- `dmlex:type` REQUIRED (exactly 1) of type `http://www.w3.org/2001/XMLSchema#string`
- `dmlex:scope` OPTIONAL (at most 1) of type `http://www.w3.org/2001/XMLSchema#string`
- `dmlex:memberType` OPTIONAL reference to [MemberType](#)

### 5.3.2.25 RDF element: `dmlex:MemberType`

Represents one of possible values for the role of member, as well as various restrictions on members having this role.

#### *Properties*

- `dmlex:description` OPTIONAL (at most 1) of type `http://www.w3.org/2000/01/rdf-schema#Literal`
- `dmlex:type` OPTIONAL of type `http://www.w3.org/2001/XMLSchema#string`
- `dmlex:role` REQUIRED (exactly 1) of type `http://www.w3.org/2001/XMLSchema#string`
- `dmlex:min` OPTIONAL (at most 1) of type `http://www.w3.org/2001/XMLSchema#nonNegativeInteger`

- `dmlex:max` OPTIONAL (at most 1) of type `http://www.w3.org/2001/XMLSchema#nonNegativeInteger`
- `dmlex:action` OPTIONAL (at most 1) of type `http://www.w3.org/2001/XMLSchema#string`

### 5.3.2.26 RDF element: `dmlex:PlaceholderMarker`

Marks up a substring inside a headword or inside a headword translation which is not part of the expression itself but stands for things that can take its place.

#### *Properties*

- `dmlex:startIndex` REQUIRED (exactly 1) of type `http://www.w3.org/2001/XMLSchema#string`
- `dmlex:endIndex` REQUIRED (exactly 1) of type `http://www.w3.org/2001/XMLSchema#string`

### 5.3.2.27 RDF element: `dmlex:HeadwordMarker`

Marks up a substring inside an example, inside an example translation or inside a definition which corresponds to the headword (or to a translation of the headword)

#### *Properties*

- `dmlex:startIndex` REQUIRED (exactly 1) of type `http://www.w3.org/2001/XMLSchema#string`
- `dmlex:endIndex` REQUIRED (exactly 1) of type `http://www.w3.org/2001/XMLSchema#string`

### 5.3.2.28 RDF element: `dmlex:CollocateMarker`

Marks up a substring other than the headword inside an example, inside an example translation or inside a definition.

#### *Properties*

- `dmlex:label` OPTIONAL reference to [Label](#)
- `dmlex:lemma` OPTIONAL (at most 1) of type `http://www.w3.org/2001/XMLSchema#string`

### 5.3.2.29 RDF element: `dmlex:Etymology`

Represents a chain of historical derivations of a word.

#### *Properties*

- `dmlex:listingOrder` REQUIRED (exactly 1) of type `http://www.w3.org/2001/XMLSchema#nonNegativeInteger`
- `dmlex:description` OPTIONAL (at most 1) of type `http://www.w3.org/2000/01/rdf-schema#Literal`
- `dmlex:etymon` OPTIONAL reference to [Etymon](#)

### 5.3.2.30 RDF element: `dmlex:Etymon`

Represents one stage (of possibly several) in the etymological history of the headword.



### *Properties*

- `dmlex:listingOrder` REQUIRED (exactly 1) of type `http://www.w3.org/2001/XMLSchema#nonNegativeInteger`
- `dmlex:type` OPTIONAL (at most 1) of type `http://www.w3.org/2001/XMLSchema#string`
- `dmlex:when` OPTIONAL (at most 1) of type `http://www.w3.org/2001/XMLSchema#string`
- `dmlex:note` OPTIONAL (at most 1) of type `http://www.w3.org/2001/XMLSchema#string`
- `dmlex:etymonUnit` REQUIRED (at least 1) reference to [EtymonUnit](#)
- `dmlex:translation` OPTIONAL (at most 1) of type `http://www.w3.org/2001/XMLSchema#string`

#### **5.3.2.31 RDF element: `dmlex:EtymonUnit`**

Represents a form (typically a word) which is the etymological origin of the headword, or another etymologically related form

### *Properties*

- `dmlex:listingOrder` REQUIRED (exactly 1) of type `http://www.w3.org/2001/XMLSchema#nonNegativeInteger`
- `dmlex:language` REQUIRED (exactly 1) of type `http://www.w3.org/2001/XMLSchema#language` (subproperty of `http://www.w3.org/ns/lemon/lime#language`)
- `dmlex:partOfSpeech` OPTIONAL reference to [PartOfSpeech](#)
- `dmlex:text` REQUIRED (exactly 1) of type `http://www.w3.org/2000/01/rdf-schema#Literal`
- `dmlex:translation` OPTIONAL (at most 1) of type `http://www.w3.org/2001/XMLSchema#string`
- `dmlex:reconstructed` OPTIONAL (at most 1) of type `http://www.w3.org/2001/XMLSchema#boolean`

#### **5.3.2.32 RDF element: `dmlex:EtymonType`**

Represents one of possible values for the type of etymon.

### *Properties*

- `dmlex:description` OPTIONAL (at most 1) of type `http://www.w3.org/2000/01/rdf-schema#Literal`
- `dmlex:type` REQUIRED (exactly 1) of type `http://www.w3.org/2001/XMLSchema#string`

#### **5.3.2.33 RDF element: `dmlex:EtymonLanguage`**

Represents one of several allowed values for the language property of etymonUnit objects.

### *Properties*

- `dmlex:langCode` REQUIRED (exactly 1) of type `http://www.w3.org/2001/XMLSchema#language`

- `dmlex:displayName` OPTIONAL (at most 1) of type `http://www.w3.org/2001/XMLSchema#string`

## 5.4 DMLex relational database serialization (Normative)

This section specifies a serialization of the DMLex data model as a relational database, using conventional database modelling concepts (tables with columns and rows, primary keys, relationships, foreign keys) and data types from the ISO SQL (Structured Query Language) Standard [SQL].

### 5.4.1 Design principles (Informative)

The relational database serialization of DMLex shown in this document follows these principles:

- The `lexicographicResource` object is implemented as table. (Alternatively, it can left unimplemented if the database is going to contain only one lexicographic resource.)
- Other objects with an arity other than  $(0..1)$  and  $(1..1)$  are implemented as tables.
- The values of objects, and objects with an arity of  $(0..1)$  or  $(1..1)$  are implemented as columns in those tables.
- The parent-child relation is implemented as a one-to-many relation between tables.

### 5.4.2 Database diagrams (Informative)

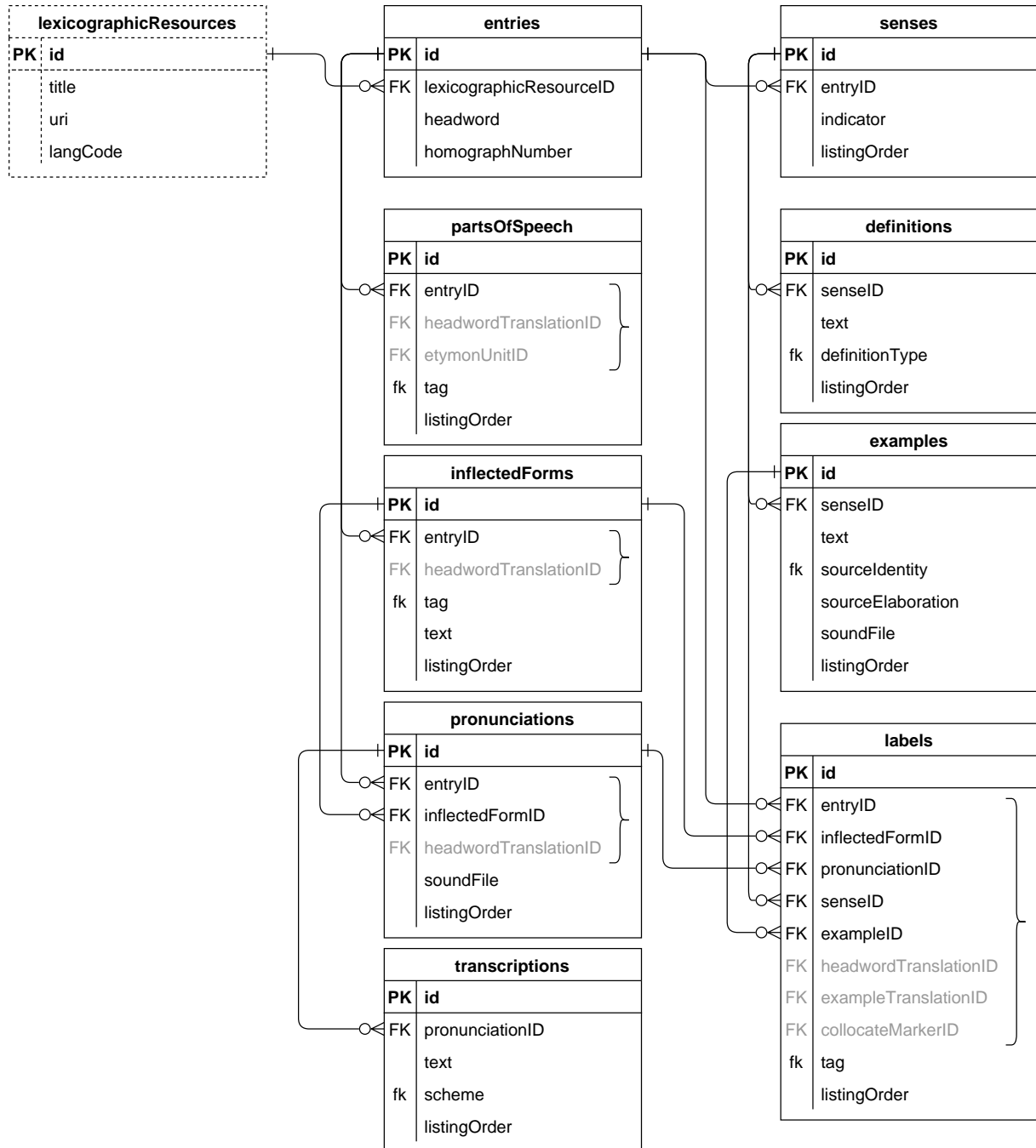
The following diagrams provide a high-level overview of the relational database serialization of DMLex. Each diagram presents a "view" of the database from the perspective of one of the modules or from the perspective of the DMLex Core. The diagrams shown here follow the usual conventions used in relational database diagrams:

- Each box represents a database table and contains a list of its column headings. The abbreviation PK next to a column name indicates that the column is a primary key. The abbreviation FK next to a column name indicates that the column is a foreign key.
- Lines between the boxes represent relationships between tables, with the conventional "crow's feet" symbols at the ends of the lines to indicate the relation's arity (one-to-many etc.).

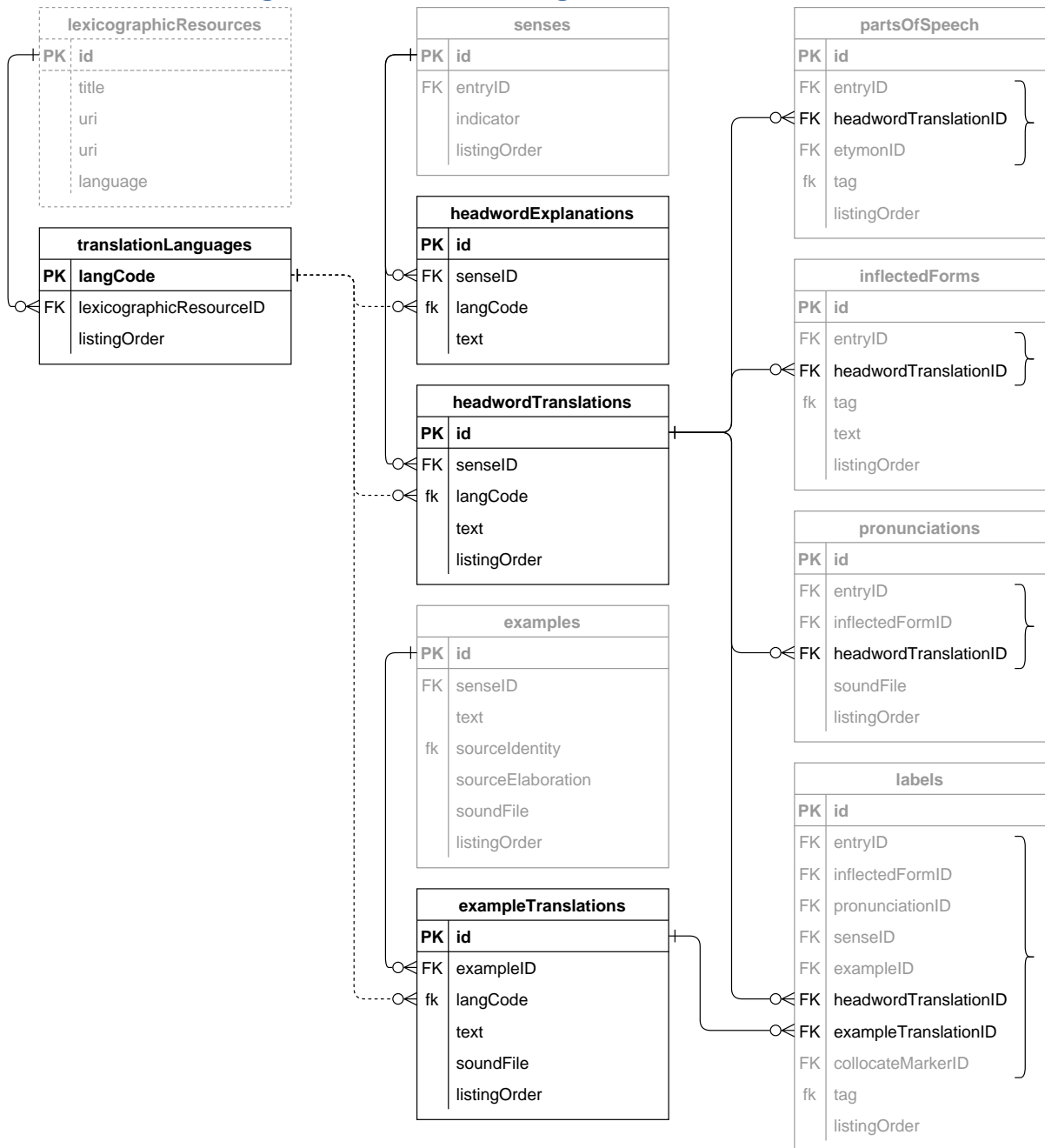
In addition to that, the following non-standard conventions are used in the diagrams:

- A curly bracket surrounding a group of foreign keys in a table indicates the constraint that exactly one of the foreign keys in this group must be non-empty while all the other ones must be null.
- Tables and columns shown in grey are tables are defined in another module (or in the core) and are shown only for information.
- The dashed outline on the table `lexicographicResources` indicates that the table is optional. If a database contains only one lexicographic resource, then this table, as well as any foreign-key columns in other tables that reference it, can be left unimplemented.
- Any relationships indicated by a dashed line and/or labelled with the lower-case abbreviation "fk" (as opposed to upper case "FK") are optional: they can be implemented as constraints (if the implementor wishes to have a high degree of referential integrity) or left unimplemented (for example if referential integrity cannot be assumed when importing data from an untrusted tree-structured serialization of DMLex).

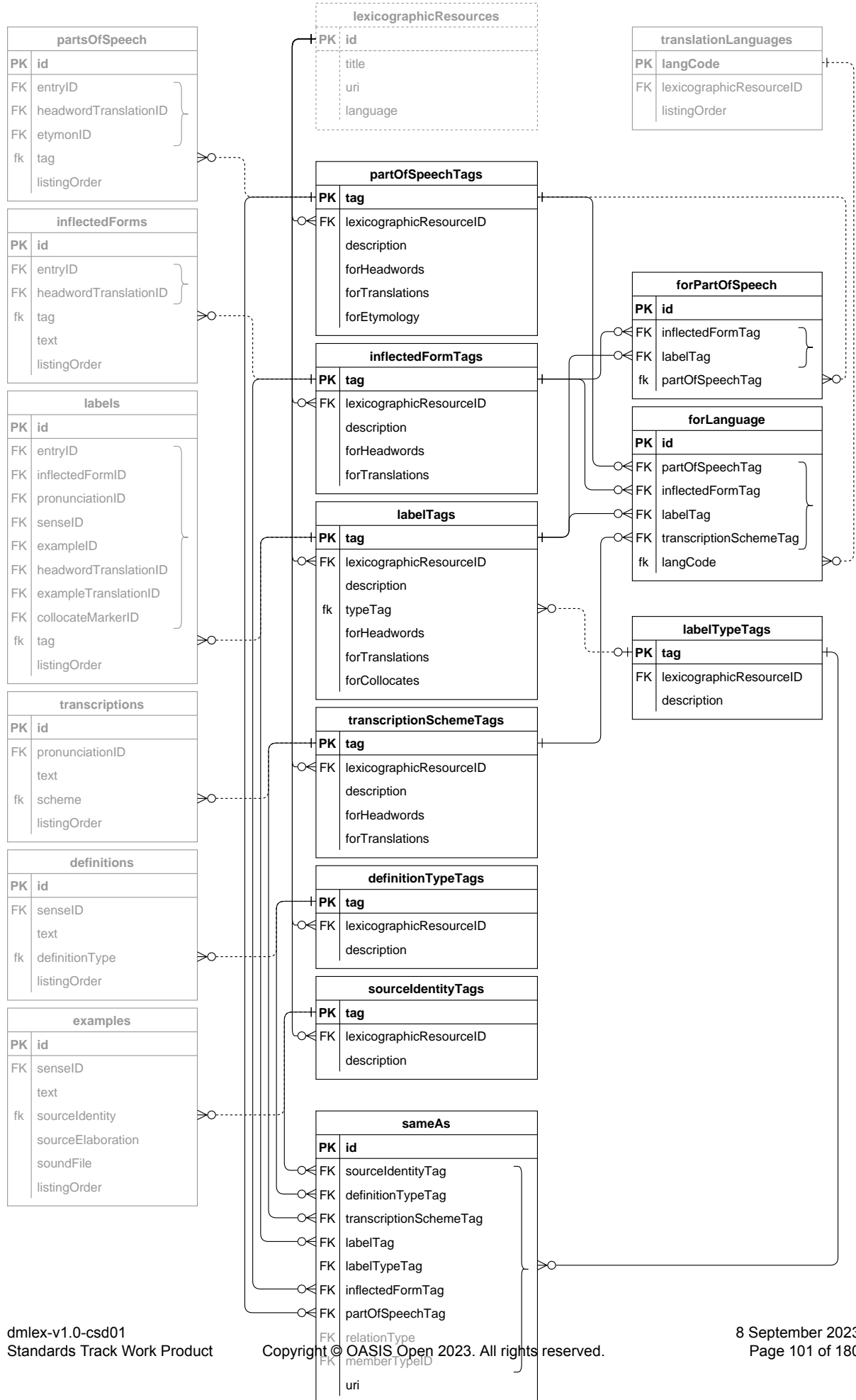
### 5.4.2.1 Database diagram: DMLex Core



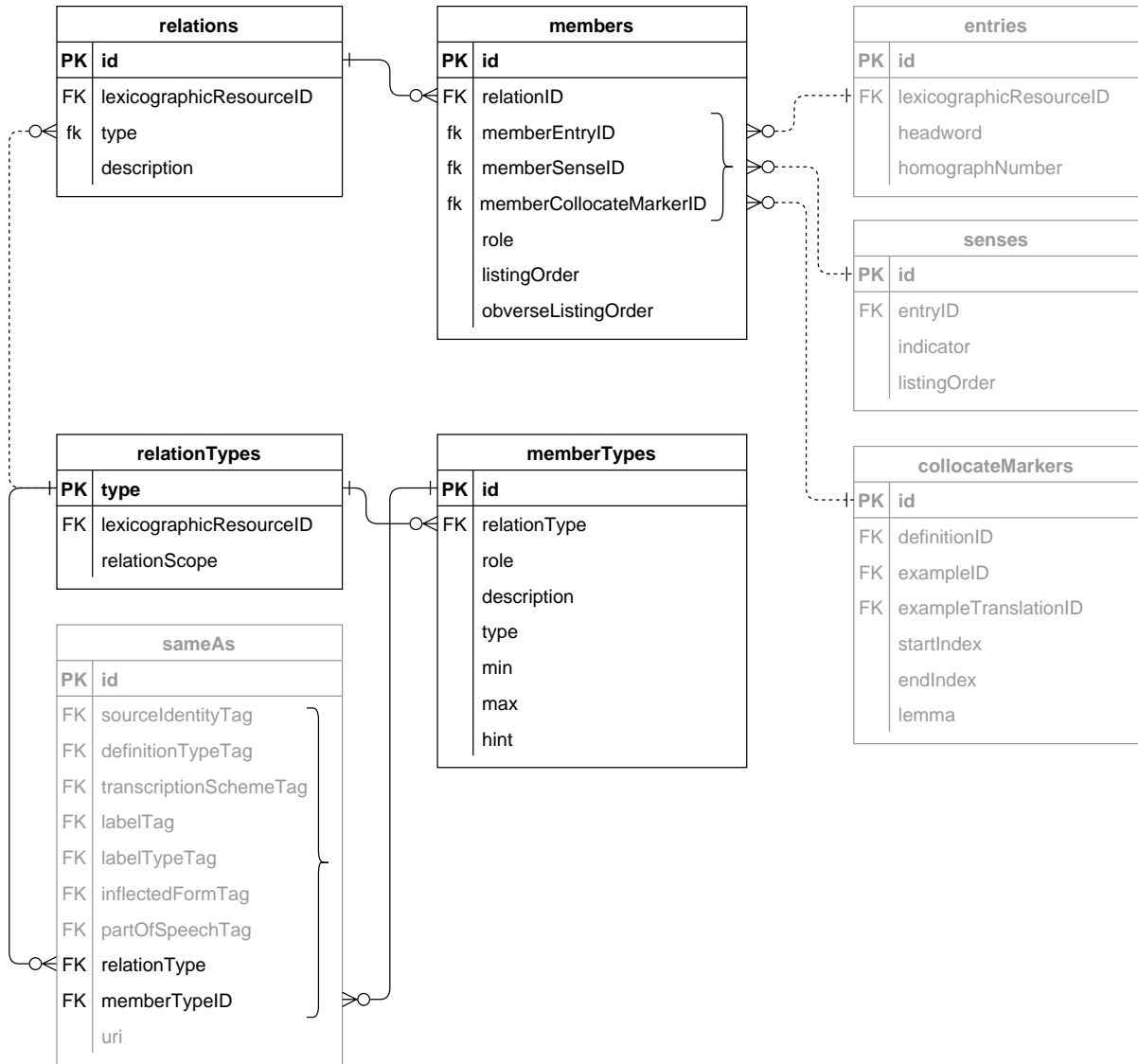
### 5.4.2.2 Database diagram: DMLex Crosslingual Module



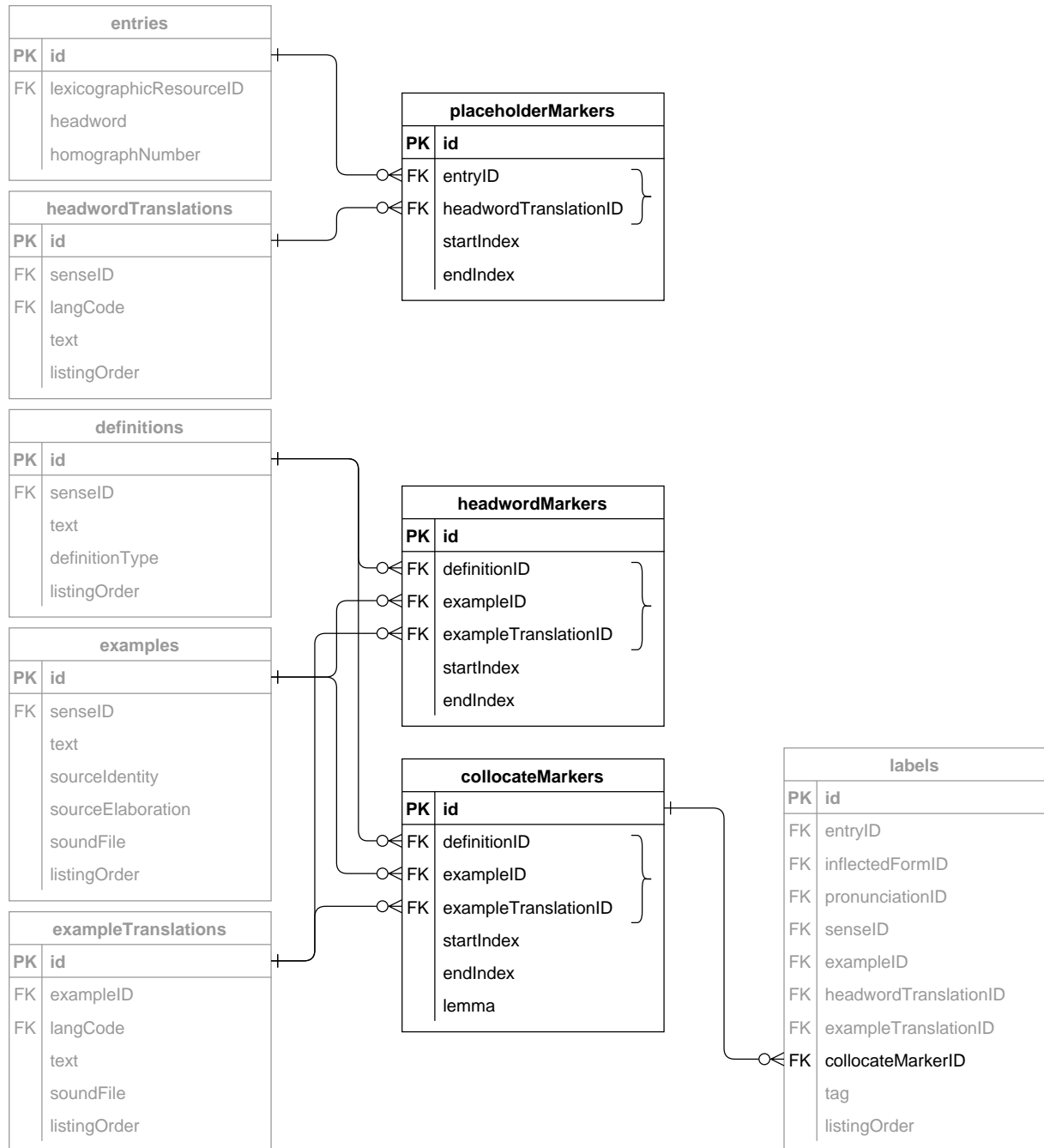
### 5.4.2.3 Database diagram: DMLex Controlled Values Module



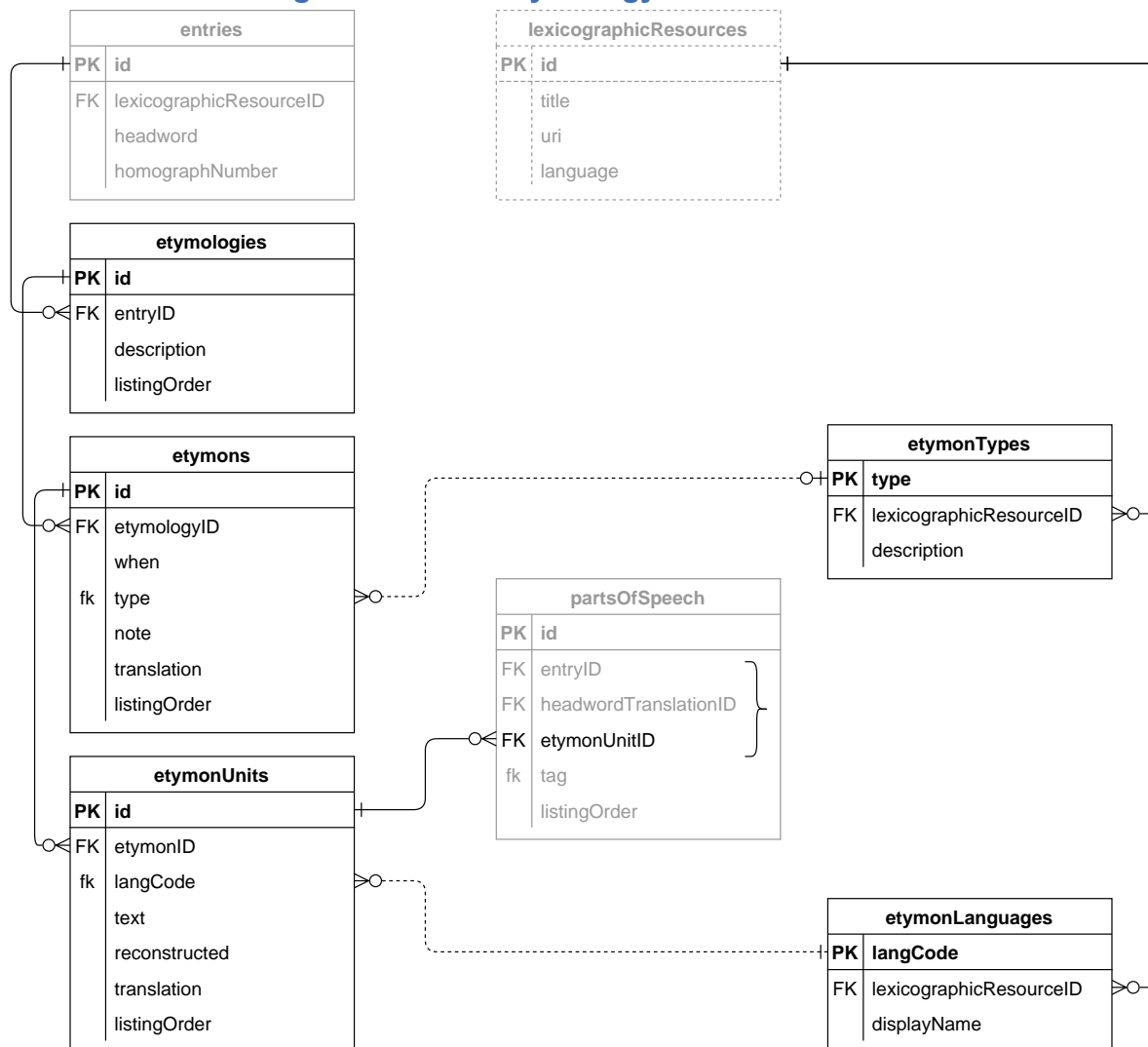
### 5.4.2.4 Database diagram: DMLex Linking Module



### 5.4.2.5 Database diagram: DMLex Annotation Module



### 5.4.2.6 Database diagram: DMLex Etymology Module



### 5.4.3 DMLex relational database serialization tables (Normative)

#### 5.4.3.1 Database table: lexicographicResources

Implements the `lexicographicResource` object type.

##### Columns

- `id`: int, primary key
- `title`: nvarchar
- `uri`: nvarchar
- `langCode`: varchar

##### Comment

- If the database is designed to contain only one lexicographic resource then this table, including all foreign keys in other tables that reference it, is not REQUIRED to be implemented.

#### 5.4.3.2 Database table: entries

Implements the `entry` object type.



### Columns

- `id`: int, primary key
- `lexicographicResourceID`: int, foreign key, references column `id` of table [lexicographicResources](#)
- `headword`: nvarchar
- `homographNumber`: int

### 5.4.3.3 Database table: `partsOfSpeech`

Implements the [partOfSpeech](#) object type.

### Columns

- `id`: int, primary key
- `entryID`: int, foreign key, references column `id` of table [entries](#)
- `tag`: nvarchar, optionally a foreign key referencing column `tag` of table [partOfSpeechTags](#)
- `listingOrder`: int

### Columns if implementing the Crosslingual Module

- `headwordTranslationID`: int, foreign key, references column `id` of table [headwordTranslations](#)

### Columns if implementing the Etymology Module

- `etymonUnitID`: int, foreign key, references column `id` of table [etymonUnits](#)

### Constraint

- Exactly one of the foreign keys `entryID`, `headwordTranslationID` and `etymonID` is REQUIRED to be non-empty.

### 5.4.3.4 Database table: `inflectedForms`

Implements the [inflectedForm](#) object type.

### Columns

- `id`: int, primary key
- `entryID`: int, foreign key, references column `id` of table [entries](#)
- `tag`: nvarchar, optionally a foreign key referencing column `tag` of table [inflectedFormTags](#)
- `text`: nvarchar
- `listingOrder`: int

### Columns if implementing the Crosslingual Module

- `headwordTranslationID`: int, foreign key, references column `id` of table [headwordTranslations](#)

### Constraint

- Exactly one of the foreign keys `entryID` and `headwordTranslationID` is REQUIRED to be non-empty.

### 5.4.3.5 Database table: *senses*

Implements the [sense](#) object type.

#### *Columns*

- `id`: `int`, primary key
- `entryID`: `int`, foreign key, references column `id` of table [entries](#)
- `indicator`: `nvarchar`
- `listingOrder`: `int`

### 5.4.3.6 Database table: *definitions*

Implements the [definition](#) object type.

#### *Columns*

- `id`: `int`, primary key
- `senseID`: `int`, foreign key, references column `id` of table [senses](#)
- `text`: `nvarchar`
- `definitionType`: `nvarchar`, optionally a foreign key referencing column `tag` of table [definitionTypeTags](#)
- `listingOrder`: `int`

### 5.4.3.7 Database table: *labels*

Implements the [label](#) object type.

#### *Columns*

- `id`: `int`, primary key
- `entryID`: `int`, foreign key, references column `id` of table [entries](#)
- `inflectedFormID`: `int`, foreign key, references column `id` of table [inflectedForms](#)
- `pronunciationID`: `int`, foreign key, references column `id` of table [pronunciations](#)
- `senseID`: `int`, foreign key, references column `id` of table [senses](#)
- `exampleID`: `int`, foreign key, references column `id` of table [examples](#)
- `tag`: `nvarchar`, optionally a foreign key referencing column `tag` of table [labelTags](#)
- `listingOrder`: `int`

#### *Columns if implementing the Crosslingual Module*

- `headwordTranslationID`: `int`, foreign key, references column `id` of table [headwordTranslations](#)
- `exampleTranslationID`: `int`, foreign key, references column `id` of table [exampleTranslations](#)

#### *Columns if implementing the Annotation Module*

- `collocateMarkerID`: `int`, foreign key, references column `id` of table [collocateMarkers](#)

### *Constraint*

- Exactly one of the foreign keys `entryID`, `inflectedFormID`, `pronunciationID`, `senseID`, `exampleID`, `headwordTranslationID`, `exampleTranslationID` and `collocateMarkerID` is REQUIRED to be non-empty.

### **5.4.3.8 Database table: pronunciations**

Implements the [pronunciation](#) object type.

#### *Columns*

- `id`: `int`, primary key
- `entryID`: `int`, foreign key, references column `id` of table [entries](#)
- `inflectedFormID`: `int`, foreign key, references column `id` of table [inflectedForms](#)
- `soundFile`: `nvarchar`
- `listingOrder`: `int`

#### *Columns if implementing the Crosslingual Module*

- `headwordTranslationID`: `int`, foreign key, references column `id` of table [headwordTranslations](#)

### *Constraint*

- Exactly one of the foreign keys `entryID`, `inflectedFormID` and `headwordTranslationID` is REQUIRED to be non-empty.

### **5.4.3.9 Database table: transcriptions**

Implements the [transcription](#) object type.

#### *Columns*

- `id`: `int`, primary key
- `pronunciationID`: `int`, foreign key, references column `id` of table [pronunciations](#)
- `text`: `nvarchar`
- `scheme`: `nvarchar`, optionally a foreign key referencing column `tag` of table [transcriptionSchemeTags](#)
- `listingOrder`: `int`

### **5.4.3.10 Database table: examples**

Implements the [example](#) object type.

#### *Columns*

- `id`: `int`, primary key
- `senseID`: `int`, foreign key, references column `id` of table [senses](#)
- `text`: `nvarchar`
- `sourceIdentity`: `nvarchar`, optionally a foreign key referencing column `tag` of table [sourceIdentityTags](#)

- sourceElaboration: nvarchar
- soundFile: nvarchar
- listingOrder: int

#### 5.4.3.11 Database table: translationLanguages

Implements the [translationLanguage](#) object type from the Crosslingual Module.

##### Columns

- langCode: varchar, primary key
- lexicographicResourceID: int, foreign key, references column id of table [lexicographicResources](#)
- listingOrder: int

#### 5.4.3.12 Database table: headwordTranslations

Implements the [headwordTranslation](#) object type from the Crosslingual Module.

##### Columns

- id: int, primary key
- senseID: int, foreign key, references column id of table [senses](#)
- langCode: varchar, optionally a foreign key referencing column langCode of table [translationLanguages](#)
- text: nvarchar
- listingOrder: int

#### 5.4.3.13 Database table: headwordExplanations

Implements the [headwordExplanation](#) object type from the Crosslingual Module.

##### Columns

- id: int, primary key
- senseID: int, foreign key, references column id of table [senses](#)
- langCode: varchar, optionally a foreign key referencing column langCode of table [translationLanguages](#)
- text: nvarchar

#### 5.4.3.14 Database table: exampleTranslations

Implements the [exampleTranslation](#) object type from the Crosslingual Module.

##### Columns

- id: int, primary key
- exampleID: int, foreign key, references column id of table [examples](#)
- langCode: varchar, foreign key, references column langCode of table [translationLanguages](#)

- text: nvarchar
- soundFile: nvarchar
- listingOrder: int

#### 5.4.3.15 Database table: definitionTypeTags

Implements the [definitionTypeTag](#) object type from the Controlled Values Module.

##### Columns

- tag: nvarchar, primary key
- lexicographicResourceID: int, foreign key, references column id of table [lexicographicResources](#)
- description: nvarchar

#### 5.4.3.16 Database table: inflectedFormTags

Implements the [inflectedFormTags](#) object type from the Controlled Values Module.

##### Columns

- tag: nvarchar, primary key
- lexicographicResourceID: int, foreign key, references column id of table [lexicographicResources](#)
- description: nvarchar
- forHeadwords: bit
- forTranslations: bit

#### 5.4.3.17 Database table: labelTags

Implements the [labelTag](#) object type from the Controlled Values Module.

##### Columns

- tag: nvarchar, primary key
- lexicographicResourceID: int, foreign key, references column id of table [lexicographicResources](#)
- description: nvarchar
- typeTag: nvarchar, optionally a foreign key referencing column tag of table [labelTypeTags](#)
- forHeadwords: bit
- forTranslations: bit
- forCollocates: bit

#### 5.4.3.18 Database table: labelTypeTags

Implements the [labelTypeTag](#) object type from the Controlled Values Module.

##### Columns

- tag: nvarchar, primary key

- `lexicographicResourceID`: int, foreign key, references column `id` of table [lexicographicResources](#)
- `description`: nvarchar

#### 5.4.3.19 Database table: `partOfSpeechTags`

Implements the [partOfSpeechTag](#) object type from the Controlled Values Module.

##### *Columns*

- `tag`: nvarchar, primary key
- `lexicographicResourceID`: int, foreign key, references column `id` of table [lexicographicResources](#)
- `description`: nvarchar
- `forHeadwords`: bit
- `forTranslations`: bit
- `forEtymology`: bit

#### 5.4.3.20 Database table: `sourceIdentityTags`

Implements the [sourceIdentityTag](#) object type from the Controlled Values Module.

##### *Columns*

- `tag`: nvarchar, primary key
- `lexicographicResourceID`: int, foreign key, references column `id` of table [lexicographicResources](#)
- `description`: nvarchar

#### 5.4.3.21 Database table: `transcriptionSchemeTags`

Implements the [transcriptionSchemeTag](#) object type from the Controlled Values Module.

##### *Columns*

- `tag`: nvarchar, primary key
- `lexicographicResourceID`: int, foreign key, references column `id` of table [lexicographicResources](#)
- `description`: nvarchar
- `forHeadwords`: bit
- `forTranslations`: bit

#### 5.4.3.22 Database table: `sameAs`

Implements the [sameAs](#) object type from the Controlled Values Module.

##### *Columns*

- `id`: int, primary key

- `sourceIdentityTag`: `nvarchar`, foreign key, references column `tag` of table [sourceIdentityTags](#)
- `definitionTypeTag`: `nvarchar`, foreign key, references column `tag` of table [definitionTypeTags](#)
- `transcriptionSchemeTag`: `nvarchar`, foreign key, references column `tag` of table [transcriptionSchemeTags](#)
- `labelTag`: `nvarchar`, foreign key, references column `tag` of table [labelTags](#)
- `labelTypeTag`: `nvarchar`, foreign key, references column `tag` of table [labelTypeTags](#)
- `inflectedFormTag`: `nvarchar`, foreign key, references column `tag` of table [inflectedFormTags](#)
- `partOfSpeechTag`: `nvarchar`, foreign key, references column `tag` of table [partOfSpeechTags](#)
- `uri`: `nvarchar`

#### *Columns if implementing the Linking Module*

- `relationType`: `nvarchar`, foreign key, references column `type` of table [relationTypes](#)
- `memberTypeID`: `int`, foreign key, references column `id` of table [memberTypes](#)

#### *Constraint*

- Exactly one of the foreign keys `sourceIdentityTag`, `definitionTypeTag`, `transcriptionSchemeTag`, `labelTag`, `labelTypeTag`, `inflectedFormTag`, `partOfSpeechTag`, `relationType` and `memberTypeID` is REQUIRED to be non-empty.

### 5.4.3.23 Database table: [forPartOfSpeech](#)

Implements the `forPartOfSpeech` property of the [inflectedFormTag](#) and [labelTag](#) object types from the Controlled Values Module.

#### *Columns*

- `id`: `int`, primary key
- `inflectedFormTag`: `nvarchar`, foreign key, references column `tag` of table [inflectedFormTags](#)
- `labelTag`: `nvarchar`, foreign key, references column `tag` of table [labelTags](#)
- `partOfSpeechTag`: `nvarchar`, optionally a foreign key referencing column `tag` of table [partOfSpeechTags](#)

#### *Constraint*

- Exactly one of the foreign keys `inflectedFormTag` and `labelTag` is REQUIRED to be non-empty.

### 5.4.3.24 Database table: [forLanguage](#)

Implements the `forLanguage` property of the [partOfSpeechTag](#), [inflectedFormTag](#), [labelTag](#) and [transcriptionScheme](#) object types from the Controlled Values Module.

#### *Columns*

- `id`: `int`, primary key

- `partOfSpeechTag`: `nvarchar`, foreign key, references column `tag` of table [partOfSpeechTags](#)
- `inflectedFormTag`: `nvarchar`, foreign key, references column `tag` of table [inflectedFormTags](#)
- `labelTag`: `nvarchar`, foreign key, references column `tag` of table [labelTags](#)
- `transcriptionSchemeTag`: `nvarchar`, foreign key, references column `tag` of table [transcriptionSchemeTags](#)
- `langCode`: `nvarchar`, optionally a foreign key referencing column `langCode` of table [translationLanguages](#)

#### *Constraint*

- Exactly one of the foreign keys `partOfSpeechTag`, `inflectedFormTag`, `labelTag` and `transcriptionSchemeTag` is REQUIRED to be non-empty.

### 5.4.3.25 Database table: [relations](#)

Implements the [relation](#) object type from the Linking Module.

#### *Columns*

- `id`: `int`, primary key
- `lexicographicResourceID`: `int`, foreign key, references column `id` of table [lexicographicResources](#)
- `type`: `nvarchar`, optionally a foreign key referencing column `type` of table [relationTypes](#)
- `description`: `nvarchar`

### 5.4.3.26 Database table: [members](#)

Implements the [member](#) object type from the Linking Module.

#### *Columns*

- `id`: `int`, primary key
- `relationID`: `int`, foreign key, references column `id` of table [relations](#)
- `memberEntryID`: `int`, foreign key, references column `id` of table [entries](#)
- `memberSenseID`: `int`, foreign key, references column `id` of table [senses](#)
- `role`: `nvarchar`
- `listingOrder`: `int`
- `obverseListingOrder`: `int`

#### *Columns if implementing the Annotation Module*

- `memberCollocateMarkerID`: `int`, foreign key, references column `id` of table [entries](#)

#### *Constraint*

- Exactly one of the foreign keys `memberEntryID`, `memberSenseID` and `memberCollocateMarkerID` is REQUIRED to be non-empty.



### 5.4.3.27 Database table: `relationTypes`

Implements the `relationType` object type from the Linking Module.

#### *Columns*

- `type`: `nvarchar`, primary key
- `lexicographicResourceID`: `int`, foreign key, references column `id` of table `lexicographicResources`
- `relationScope`: `nvarchar`

### 5.4.3.28 Database table: `memberTypes`

Implements the `memberType` object type from the Linking Module.

#### *Columns*

- `id`: `int`, primary key
- `relationType`: `nvarchar`, foreign key, references column `type` of table `relationType`
- `role`: `nvarchar`
- `description`: `nvarchar`
- `type`: `nvarchar`
- `min`: `int`
- `max`: `int`
- `hint`: `varchar`

### 5.4.3.29 Database table: `placeholderMarkers`

Implements the `placeholderMarker` object type from the Annotationm Module.

#### *Columns*

- `id`: `int`, primary key
- `entryID`: `int`, foreign key, references column `id` of table `entries`
- `startIndex`: `int`
- `endIndex`: `int`

#### *Columns if implementing the Crosslingual Module*

- `headwordTranslationID`: `int`, foreign key, references column `id` of table `headwordTranslations`

#### *Constraint*

- Exactly one of the foreign keys `entryID` and `headwordTranslationID`, is REQUIRED to be non-empty.

### 5.4.3.30 Database table: `headwordMarkers`

Implements the `headwordMarker` object type from the Annotationm Module.

### *Columns*

- `id`: `int`, primary key
- `definitionID`: `int`, foreign key, references column `id` of table [definitions](#)
- `exampleID`: `int`, foreign key, references column `id` of table [examples](#)
- `startIndex`: `int`
- `endIndex`: `int`

### *Columns if implementing the Crosslingual Module*

- `exampleTranslationID`: `int`, foreign key, references column `id` of table [exampleTranslations](#)

### *Constraint*

- Exactly one of the foreign keys `definitionID`, `exampleID` and `exampleTranslationID`, is REQUIRED to be non-empty.

## **5.4.3.31 Database table: `collocateMarkers`**

Implements the [collocateMarker](#) object type from the Annotationm Module.

### *Columns*

- `id`: `int`, primary key
- `definitionID`: `int`, foreign key, references column `id` of table [definitions](#)
- `exampleID`: `int`, foreign key, references column `id` of table [examples](#)
- `startIndex`: `int`
- `endIndex`: `int`
- `lemma`: `nvarchar`

### *Columns if implementing the Crosslingual Module*

- `exampleTranslationID`: `int`, foreign key, references column `id` of table [exampleTranslations](#)

### *Constraint*

- Exactly one of the foreign keys `definitionID`, `exampleID` and `exampleTranslationID`, is REQUIRED to be non-empty.

## **5.4.3.32 Database table: `etymologies`**

Implements the [etymology](#) object type from the Etymology Module.

### *Columns*

- `id`: `int`, primary key
- `entryID`: `int`, foreign key, references column `id` of table [entries](#)
- `description`: `nvarchar`
- `listingOrder`: `int`

#### 5.4.3.33 Database table: [etymons](#)

Implements the [etymon](#) object type from the Etymology Module.

##### *Columns*

- `id`: `int`, primary key
- `etymologyID`: `int`, foreign key, references column `id` of table [etymologies](#)
- `when`: `nvarchar`
- `type`: `nvarchar`, optionally a foreign key referencing column `type` of table [etymonTypes](#)
- `note`: `nvarchar`
- `translation`: `nvarchar`
- `listingOrder`: `int`

#### 5.4.3.34 Database table: [etymonUnits](#)

Implements the [etymonUnit](#) object type from the Etymology Module.

##### *Columns*

- `id`: `int`, primary key
- `etymonID`: `int`, foreign key, references column `id` of table [etymons](#)
- `langCode`: `nvarchar`, optionally a foreign key referencing column `langCode` of table [etymon-Languages](#)
- `text`: `nvarchar`
- `reconstructed`: `bit`
- `translation`: `nvarchar`
- `listingOrder`: `int`

#### 5.4.3.35 Database table: [etymonTypes](#)

Implements the [etymonType](#) object type from the Etymology Module.

##### *Columns*

- `type`: `nvarchar`, primary key
- `lexicographicResourceID`: `int`, foreign key, references column `id` of table [lexicographicResources](#)
- `description`: `nvarchar`

#### 5.4.3.36 Database table: [etymonLanguages](#)

Implements the [etymonLanguage](#) object type from the Etymology Module.

##### *Columns*

- `langCode`: `nvarchar`, primary key

- `lexicographicResourceID`: int, foreign key, references column `id` of table `lexicographicResources`
- `displayName`: nvarchar

---

# Appendix A Informative material on serializations (Informative)

## A.1 Examples (Informative)

This section gives examples which show how to use DMLex to model lexicographic resources. Each example is shown in pseudocode first to demonstrate the object model at an abstract level. After that, each example is shown in XML and JSON. The XML and JSON encoding shown here follows DMLex's own implementation guidance for XML and JSON.

### A.1.1 A basic entry

This is a basic, beginner-level example of how to use DMLex to represent a simple monolingual lexicographic resource consisting of one entry with two senses. It demonstrates some of the basic features of DMLex Core: how to subdivide an entry into senses, how to attach various data such as definition, part-of-speech labels to entries and senses, and how to add labels to various objects such as senses and examples.

#### *Example A.1. Pseudocode*

```
- lexicographicResource (uri: http://example.com, langCode: en)
  - entry (id: abandon-verb, headword: abandon)
    - partOfSpeech (tag: verb)
    - sense (id: abandon-verb-1)
      - definition (text: to suddenly leave a place or a person)
      - example (text: I'm sorry I abandoned you like that.)
      - example (text: Abandon ship!)
        - label (tag: idiom)
    - sense (id: abandon-verb-2)
      - label (tag: mostly-passive)
      - definition (text: to stop supporting an idea)
      - example (text: That theory has been abandoned.)
```

## Example A.2. XML

```
<lexicographicResource uri="http://example.com" langCode="en">
  <entry id="abandon-verb">
    <headword>abandon</headword>
    <partOfSpeech value="verb"/>
    <sense id="abandon-verb-1">
      <definition>to suddenly leave a place or a person</definition>
      <example>
        <text>I'm sorry I abandoned you like that.</text>
      </example>
      <example>
        <text>Abandon ship!</text>
        <label value="idiom"/>
      </example>
    <sense id="abandon-verb-2">
      <label value="mostly-passive"/>
      <definition>to stop supporting an idea</definition>
      <example>
        <text>That theory has been abandoned.</text>
      </example>
    </sense>
  </entry>
</lexicographicResource>
```

### Example A.3. JSON

```
{
  "uri": "http://example.com",
  "langCode": "en",
  "entry": {
    "id": "abandon-verb",
    "headword": "abandon",
    "partsOfSpeech": ["verb"],
    "senses": [{
      "id": "abandon-verb-1",
      "definitions": [{
        "text": "to suddenly leave a place or a person"
      }],
      "examples": [{
        "text": "I'm sorry I abandoned you like that."
      }, {
        "text": "Abandon ship!",
        "labels": ["idiom"]
      }
    ]
  }, {
    "id": "abandon-verb-2",
    "labels": ["mostly-passive"],
    "definitions": ["to stop supporting an idea"],
    "examples": [{
      "text": "That theory has been abandoned."
    }
  ]
}
}
```

### Example A.4. RDF

```
<#my-dictionary> dmlex:entry <#abandon-verb> .

<abandon-verb> dmlex:headword "abandon" ;
  dmlex:partOfSpeech "verb" ;
  dmlex:sense <#abandon-verb-1"> , <#abandon-verb-2"> .

<#abandon-verb-1< dmlex:definition [
  dmlex:text "to suddenly leave a place or a person" ] ;
  dmlex:example [
  dmlex:text "I'm sorry I abandoned you like that."
  ] , [
  dmlex:text "Abandon ship!" ;
  dmlex:labels "idiom" ] .

<#abandon-verb-2">
  dmlex:label "mostly-passive" ;
  dmlex:definition [
  dmlex:text "to stop supporting an idea" ] ;
  dmlex:example [
  dmlex:text "That theory has been abandoned." ] .
```

## A.1.2 How to use inflectedForm

This is an entry from a hypothetical Irish dictionary for the headword "folúsghlantóir" ("vacuum cleaner") which gives its two inflected forms, the singular genitive and the plural.

### Example A.5. Pseudocode

```
- entry (id: folúsghlantóir-n, headword: folúsghlantóir)
  - partOfSpeech (tag: n-masc)
  - inflectedForm (tag: sg-gen, text: folúsghlantóra)
  - inflectedForm (tag: pl, text: folúsghlantóirí)
  - sense...
```

### Example A.6. XML

```
<entry id="folúsghlantóir-n">
  <headword>folúsghlantóir</headword>
  <partOfSpeech tag="n-masc"/>
  <inflectedForm tag="sg-gen">
    <text>folúsghlantóra</text>
  </inflectedForm>
  <inflectedForm tag="pl">
    <text>folúsghlantóirí</text>
  </inflectedForm>
  <sense>...</sense>
</entry>
```



### Example A.7. JSON

```
{
  "id": "folúsghlantóir-n",
  "headword": "folúsghlantóir",
  "partsOfSpeech": ["n-masc"],
  "inflectedForms": [{
    "tag": "sg-gen",
    "text": "folúsghlantóra"
  }, {
    "tag": "pl",
    "text": "folúsghlantóirí"
  }],
  "senses": [...]
}
```

## A.1.3 Pronunciation given as transcription

### Example A.8. Pseudocode

```
- entry (id: aardvark-noun, headword: aardvark)
  - pronunciation
    - transcription (text: a:rdva:rk)
  - sense...
```

### Example A.9. XML

```
<entry id="aardvark-noun">
  <headword>aardvark</headword>
  <pronunciation>
    <transcription>a:rdva:rk</transcription>
  </pronunciation>
  <sense>...</sense>
</entry>
```

### Example A.10. JSON

```
{
  "id": "aardvark-noun",
  "headword": "aardvark",
  "pronunciations": [{
    "transcriptions": [{"text": "a:rdva:rk"}]
  }],
  "senses": [...]
}
```

## A.1.4 Pronunciation given as a sound file

### Example A.11. Pseudocode

```
- entry (id: aardvark-noun, headword: aardvark)
  - pronunciation (soundFile: aardvark.mp3)
  - sense: ...
```

### Example A.12. XML

```
<entry id="aardvark-noun">
  <headword>aardvark</headword>
  <pronunciation soundFile="aardvark.mp3"/>
  <sense>...</sense>
</entry>
```

### Example A.13. JSON

```
{
  "id": "aardvark-noun",
  "headword": "aardvark",
  "pronunciations": [{
    "soundFile": "aardvark.mp3"
  }],
  "senses": [...]
}
```

## A.1.5 Pronunciation given both ways

### Example A.14. Pseudocode

```
- entry (id: aardvark-noun, headword: aardvark)
  - pronunciation (soundFile: aardvark.mp3)
    - transcription (text: a:rdva:rk)
  - sense: ...
```

### Example A.15. XML

```
<entry id="aardvark-noun">
  <headword>aardvark</headword>
  <pronunciation soundFile="aardvark.mp3">
    <transcription>a:rdva:rk</transcription>
  </pronunciation>
  <sense>...</sense>
</entry>
```

### Example A.16. JSON

```
{
  "id": "aardvark-noun",
  "headword": "aardvark",
  "pronunciations": [{
    "soundFile": "aardvark.mp3",
    "transcriptions": [{"text": "a:rdva:rk"}]
  }],
  "senses": [...]
}
```

## A.1.6 How to use `partOfSpeechTag` and `inflectedFormTag`

This is an entry from a hypothetical Irish dictionary for the headword "folúsghlantóir" ("vacuum cleaner"). The meaning of the various tags used in this entry is explained in the `partOfSpeechTag` and `inflectedFormTag` objects.

### Example A.17. Pseudocode

```
- lexicographicResource (uri: http://example.com, langCode: ga)
  - entry (id: folúsghlantóir-n, headword: folúsghlantóir)
    - partOfSpeech (tag: n-masc)
    - inflectedForm (tag: sg-gen, text: folúsghlantóra)
    - inflectedForm (tag: pl, text: folúsghlantóirí)
    - sense: ...
  - partOfSpeechTag (tag: n-masc, description: masculine noun)
  - partOfSpeechTag (tag: n-fem, description: feminine noun)
  - inflectedFormTag (tag: sg-gen, description: singular genitive)
    - forPartOfSpeech: n-masc
    - forPartOfSpeech: n-fem
  - inflectedFormTag (tag: pl, description: plural)
    - forPartOfSpeech: n-masc
    - forPartOfSpeech: n-fem
```

Example A.18. XML

```
<lexicographicResource uri="http://example.com" langCode="ga">
  <entry id="folúsghlantóir-n">
    <headword>folúsghlantóir</headword>
    <partOfSpeech value="n-masc"/>
    <inflectedForm tag="sg-gen">
      <text>folúsghlantóra</text>
    </inflectedForm>
    <inflectedForm tag="pl">
      <text>folúsghlantóirí</text>
    </inflectedForm>
    <sense>...</sense>
  </entry>
  <partOfSpeechTag tag="n-masc">
    <description>masculine noun</description>
  </partOfSpeechTag>
  <partOfSpeechTag tag="n-fem">
    <description>feminine noun</description>
  </partOfSpeechTag>
  <inflectedFormTag tag="sg-gen">
    <description>singular genitive</description>
    <forPartOfSpeech tag="n-masc"/>
    <forPartOfSpeech tag="n-fem"/>
  </tag>
  <tag value="pl">
    <description>plural</description>
    <target value="inflectedTag"/>
    <forPartOfSpeech tag="n-masc"/>
    <forPartOfSpeech tag="n-fem"/>
  </tag>
</lexicographicResource>
```

### Example A.19. JSON

```
{
  "uri": "http://example.com",
  "langCode": "ga",
  "entries": [{
    "id": "folúsghlantóir-n",
    "headword": "folúsghlantóir",
    "partsOfSpeech": ["n-masc"],
    "inflectedForms": [{
      "tag": "sg-gen",
      "text": "folúsghlantóra"
    }, {
      "tag": "pl",
      "text": "folúsghlantóirí"
    }],
    "senses": [...]
  }],
  "partOfSpeechTags": [{
    "tag": "n-masc",
    "description": "masculine noun"
  }, {
    "tag": "n-fem",
    "description": "feminine noun"
  }],
  "inflectedFormTags": [{
    "tag": "sg-gen",
    "description": "singular genitive",
    "forPartsOfSpeech": ["n-masc", "n-fem"]
  }, {
    "tag": "pl",
    "description": "plural",
    "forPartsOfSpeech": ["n-masc", "n-fem"]
  }],
}
```

## A.1.7 Mapping controlled values to external inventories

This shows how to map the value of a tag such as `n-masc` and `n-fem` to items in an external inventory such as LexInfo.

### Example A.20. Pseudocode

```
- partOfSpeechTag (tag: n-masc, description: masculine noun)
  - sameAs (uri: http://www.lexinfo.net/ontology/3.0/lexinfo#noun)
  - sameAs (uri: http://www.lexinfo.net/ontology/3.0/lexinfo#masculine)
- partOfSpeechTag (tag: n-fem, description: feminine noun)
  - sameAs (uri: http://www.lexinfo.net/ontology/3.0/lexinfo#noun)
  - sameAs (uri: http://www.lexinfo.net/ontology/3.0/lexinfo#feminine)
```

### Example A.21. XML

```
<partOfSpeechTag tag="n-masc">
  <description>masculine noun</description>
  <sameAs uri="http://www.lexinfo.net/ontology/3.0/lexinfo#noun"/>
  <sameAs uri="http://www.lexinfo.net/ontology/3.0/lexinfo#masculine"/>
</partOfSpeechTag>
<partOfSpeechTag tag="n-fem">
  <description>feminine noun</description>
  <sameAs uri="http://www.lexinfo.net/ontology/3.0/lexinfo#noun"/>
  <sameAs uri="http://www.lexinfo.net/ontology/3.0/lexinfo#feminine"/>
</partOfSpeechTag>
```

### Example A.22. JSON

```
{
  "partOfSpeechTags": [{
    "tag": "n-masc",
    "description": "masculine noun",
    "sameAs": [
      "http://www.lexinfo.net/ontology/3.0/lexinfo#noun",
      "http://www.lexinfo.net/ontology/3.0/lexinfo#masculine"
    ]
  }, {
    "tag": "n-fem",
    "description": "feminine noun",
    "sameAs": [
      "http://www.lexinfo.net/ontology/3.0/lexinfo#noun",
      "http://www.lexinfo.net/ontology/3.0/lexinfo#feminine"
    ]
  }
]
```

## A.1.8 Defining a bilingual lexicographic resource

This defines a lexicographic resource where the source language is German and the translation language is English and the English translations are going to come with pronunciation transcriptions in English IPA.

### Example A.23. Pseudocode

```
- lexicographicResource (title: My German-English Dictionary, uri: http://example.com,
  - translationLanguage (langCode: en)
```

### Example A.24. XML

```
<lexicographicResource uri="http://example.com" langCode="de">
  <title>My German-English Dictionary</title>
  <translationLanguage langCode="en"/>
  ...
</lexicographicResource>
```

### Example A.25. JSON

```
{
  "title": "My German-English Dictionary",
  "uri": "http://example.com",
  "langCode": "de",
  "translationLanguages": ["en"],
  ...
}
```

## A.1.9 Defining a multilingual lexicographic resource

This defines a lexicographic resource where the source language is Irish and the translation languages are English, German and Czech.

### Example A.26. Pseudocode

```
- lexicographicResource (title: My Irish-Multilingual Dictionary, uri: http://example.com
  - translationLanguage (langCode: en)
  - translationLanguage (langCode: de)
  - translationLanguage (langCode: cs)
```

### Example A.27. XML

```
<lexicographicResource uri="http://example.com" langCode="ga">
  <title>My Irish-Multilingual Dictionary</title>
  <translationLanguage langCode="en"/>
  <translationLanguage langCode="de"/>
  <translationLanguage langCode="cs"/>
  ...
</lexicographicResource>
```

### Example A.28. JSON

```
{
  "title": "My Irish-Multilingual Dictionary",
  "uri": "http://example.com",
  "langCode": "ga",
  "translationLanguages": ["en", "de", "cs"],
  ...
}
```

## A.1.10 How to use headwordTranslation in a bilingual lexicographic resource

This is an entry from a hypothetical English-German dictionary for English-speaking learners of German.

### Example A.29. Pseudocode

```
- entry (id: doctor-n, headword: doctor)
  - sense (id: doctor-n-1, indicator: medical doctor)
    - headwordTranslation (text: Arzt)
      - partOfSpeech (tag: n-masc)
    - headwordTranslation (text: Ärztin)
      - partOfSpeech (tag: n-fem)
  - sense (id: doctor-n-2, indicator: academic title)
    - headwordTranslation (text: Doktor)
      - partOfSpeech (tag: n-masc)
    - headwordTranslation (text: Doktorin)
      - partOfSpeech (tag: n-fem)
```

### Example A.30. XML

```
<entry id="doctor-n">
  <headword>doctor</headword>
  <sense id="doctor-n-1">
    <indicator>medical doctor</indicator>
    <headwordTranslation>
      <text>Arzt</text>
      <partOfSpeech tag="n-masc"/>
    </headwordTranslation>
    <headwordTranslation>
      <text>Ärztin</text>
      <partOfSpeech tag="n-fem"/>
    </headwordTranslation>
  </sense>
  <sense id="doctor-n-2">
    <indicator>academic title</indicator>
    <headwordTranslation>
      <text>Doktor</text>
      <partOfSpeech tag="n-masc"/>
    </headwordTranslation>
    <headwordTranslation>
      <text>Doktorin</text>
      <partOfSpeech tag="n-fem"/>
    </headwordTranslation>
  </sense>
</entry>
```



### Example A.31. JSON

```
{
  "id": "doctor-n",
  "headword": "doctor",
  "senses": [{
    "id": "doctor-n-1",
    "indicator": "medical doctor",
    "headwordTranslations": [{
      "text": "Arzt",
      "partsOfSpeech": ["n-masc"]
    }, {
      "text": "Ärztin",
      "partsOfSpeech": ["n-fem"]
    }
  ]
}, {
  "id": "doctor-n-2",
  "indicator": "academic title",
  "headwordTranslations": [{
    "text": "Doktor",
    "partsOfSpeech": ["n-masc"]
  }, {
    "text": "Doktorin",
    "partsOfSpeech": ["n-fem"]
  }
  ]
}
]
```

## A.1.11 How to use `headwordTranslation` in a multilingual lexicographic resource

This is an entry from a hypothetical Irish-multilingual dictionary.

### Example A.32. Pseudocode

```
- entry (id: fómhar-n, headword: fómhar)
  - sense (id: fómhar-n-1)
    - headwordTranslation (langCode: en, text: autumn)
    - headwordTranslation (langCode: en, text: fall)
    - headwordTranslation (langCode: de, text: Herbst)
    - headwordTranslation (langCode: cs, text: podzim)
  - sense (id: fómhar-n-2)
    - headwordTranslation (langCode: en, text: harvest)
    - headwordTranslation (langCode: de, text: Ernte)
    - headwordTranslation (langCode: cs, text: sklize#)
```

Example A.33. XML

```
<entry id="fómhar-n">
  <headword>fómhar</headword>
  <sense id="fómhar-n-1">
    <headwordTranslation langCode="en">
      <text>autumn</text>
    </headwordTranslation>
    <headwordTranslation langCode="en">
      <text>fall</text>
    </headwordTranslation>
    <headwordTranslation langCode="de">
      <text>Herbst</text>
    </headwordTranslation>
    <headwordTranslation langCode="cs">
      <text>podzim</text>
    </headwordTranslation>
  </sense>
  <sense id="fómhar-n-2">
    <headwordTranslation langCode="en">
      <text>harvest</text>
    </headwordTranslation>
    <headwordTranslation langCode="de">
      <text>Ernte</text>
    </headwordTranslation>
    <headwordTranslation langCode="cs">
      <text>sklize#</text>
    </headwordTranslation>
  </sense>
</entry>
```

### Example A.34. JSON

```
{
  "id": "fómhar-n",
  "headword": "fómhar",
  "senses": [{
    "id": "fómhar-n-1",
    "headwordTranslations": [{
      "langCode": "en",
      "text": "autumn"
    }, {
      "langCode": "en",
      "text": "fall"
    }, {
      "langCode": "de",
      "text": "Herbst"
    }, {
      "langCode": "cs",
      "text": "podzim"
    }
  ]
}, {
  "id": "fómhar-n-2",
  "headwordTranslations": [{
    "langCode": "en",
    "text": "harvest"
  }, {
    "langCode": "de",
    "text": "Ernte"
  }, {
    "langCode": "cs",
    "text": "sklize#"
  }
  ]
},
]
```

## A.1.12 How to use headwordExplanation

### Example A.35. Pseudocode

```
- entry (id: treppenwitz, headword: Treppenwitz)
  - partOfSpeech (tag: n-masc)
  - sense (id: treppenwitz-1)
    - headwordExplanation (text: belated realisation of what one could have said)
    - headwordTranslation (text: staircase wit)
```

### Example A.36. XML

```
<entry id="treppenwitz">
  <headword>Treppenwitz</headword>
  <partOfSpeech value="n-masc"/>
  <sense id="treppenwitz-1">
    <headwordExplanation>
      belated realisation of what one could have said
    </headwordExplanation>
    <headwordTranslation>
      <text>staircase wit</text>
    </headwordTranslation>
  </sense>
```

### Example A.37. JSON

```
{
  "id": "treppenwitz",
  "headword": "Treppenwitz",
  "partsOfSpeech": ["n-masc"],
  "senses": [{
    "id": "treppenwitz-1",
    "headwordExplanations": [{
      "text": "belated realisation of what one could have said"
    }],
    "headwordTranslations": [{
      "text": "staircase wit"
    }]
  }]
}
```

## A.1.13 Modelling parts and wholes

We have three entries with one sense each: "glasses", "microscope" and "lens". We want to represent the fact that "lens" is a meronym of both "glasses" and "microscope", and simultaneously that "glasses" and "microscope" are both holonyms of "lens".

*Example A.38. Pseudocode*

```
- lexicographicResource (uri: http://example.com, langCode: en)
  - entry (id: glasses, headword: glasses)
    - sense (id: glasses-1)
      - definition (text: an optical seeing aid)
  - entry (id: microscope, headword: microscope)
    - sense (id: microscope-1)
      - definition (text: equipment for looking at very small things)
  - entry (id: lens, headword: lens)
    - sense (id: lens-1)
      - definition (text: curved glass that makes things seem bigger)
  - relation (type: meronymy)
    - member (memberID: glasses-1, role: whole)
    - member (memberID: lens-1, role: part)
  - relation (type: meronymy)
    - member (memberID: microscope-1, role: whole)
    - member (memberID: lens-1, role: part)
  - relationType (type: meronymy, description: part-whole relationship)
    - memberType (role: whole, type: sense, min: 1, max: 1, hint: navigate)
    - memberType (role: part, type: sense, min: 1, max: 1, hint: navigate)
```

Example A.39. XML

```
<lexicographicResource uri="http://example.com" langCode="en">
  <entry id="glasses">
    <headword>glasses</headword>
    <sense id="glasses-1">
      <definition>an optical seeing aid</definition>
    </sense>
  </entry>
  <entry id="microscope">
    <headword>microscope</headword>
    <sense id="microscope-1">
      <definition>equipment for looking at very small things</definition>
    </sense>
  </entry>
  <entry id="lens">
    <headword>lens</headword>
    <sense id="lens-1">
      <definition>curved glass that makes things seem bigger</definition>
    </sense>
  </entry>
  <relation type="meronymy">
    <member memberID="glasses-1" role="whole"/>
    <member memberID="lens-1" role="part"/>
  </relation>
  <relation type="meronymy">
    <member memberID="microscope-1" role="whole"/>
    <member memberID="lens-1" role="part"/>
  </relation>
  <relationType type="meronymy">
    <description>part-whole relationship</description>
    <memberType role="whole" type="sense" min="1" max="1" hint="navigate"/>
    <memberType role="part" type="sense" min="1" max="1" hint="navigate"/>
  </relationType>
</lexicographicResource>
```

```

{
  "uri": "http://example.com",
  "langCode": "en",
  "entries": [{
    "id": "glasses",
    "headword": "glasses",
    "senses": [{
      "id": "glasses-1",
      "definitions": [{"text": "an optical seeing aid"}]
    }, {
      "id": "microscope",
      "headword": "microscope",
      "senses": [{
        "id": "microscope-1",
        "definitions": [{"text": "equipment for looking at very small things"}]
      }, {
        "id": "lens",
        "headword": "lens",
        "senses": [{
          "id": "lens-1",
          "definitions": [{"text": "curved glass that makes things seem bigger"}]
        }
      ]
    }
  ]],
  "relations": [{
    "type": "meronymy",
    "members": [{
      "memberID": "glasses-1",
      "role": "whole"
    }, {
      "memberID": "lens-1",
      "role": "part"
    }
  ]],
  {
    "type": "meronymy",
    "members": [{
      "memberID": "microscope-1",
      "role": "whole"
    }, {
      "memberID": "lens-1",
      "role": "part"
    }
  ]
}],
  "relationTypes": [{
    "type": "meronymy",
    "description": "part-whole relationship",
    "memberTypes": [{
      "role": "whole",
      "type": "sense",
      "min": 1,
      "max": 1,
      "hint": "navigate"
    }, {
      "role": "part",
      "type": "sense",
      "min": 1,
      "max": 1,
      "hint": "navigate"
    }
  ]
}]
}

```

*Example A.41. Suggested rendering for human users*

```
glasses
- an optical seeing aid
  contains: lens

microscope
- equipment for looking at very small things
  contains: lens

lens
- curved glass that makes things seem bigger
  things that contain lens: glasses, microscope
```

### A.1.14 Modelling antonyms

We have two entries for the verbs "buy" and "sell" with one sense each. We want to express the fact that the senses are antonyms.

*Example A.42. Pseudocode*

```
- lexicographicResource (uri: http://example.com, langCode: en)
  - entry (id: buy, headword: buy)
    - sense (id: buy-1)
      - definition (text: get something by paying money for it)
  - entry (id: sell, headword: sell)
    - sense (id: sell-1)
      - definition (text: exchange something for money)
  - relation (type: antonyms)
    - member (memberID: buy-1)
    - member (memberID: sell-1)
  - relationType: (type: antonyms)
    - memberType (type: sense, min: 2, max: 2, hint: navigate)
```



Example A.43. XML

```
<lexicographicResource uri="http://example.com" langCode="en">
  <entry id="buy">
    <headword>buy</headword>
    <sense id="buy-1">
      <definition>get something by paying money for it</definition>
    </sense>
  </entry>
  <entry id="sell">
    <headword>sell</headword>
    <sense id="sell-1">
      <definition>exchange something for money</definition>
    </sense>
  </entry>
  <relation type="antonyms">
    <member memberID="buy-1"/>
    <member memberID="sell-1"/>
  </relation>
  <relationType type="antonyms">
    <memberType type="sense" min="2" max="2" hint="navigate"/>
  </relationType>
</lexicographicResource>
```

### Example A.44. JSON

```
{
  "uri": "http://example.com",
  "langCode": "en",
  "entries": [{
    "id": "buy",
    "headword": "buy",
    "senses": [{
      "id": "buy-1",
      "definitions": [{"text": "get something by paying money for it"}]
    }, {
      "id": "sell",
      "headword": "sell",
      "senses": [{
        "id": "sell-1",
        "definitions": [{"text": "exchange something for money"}]
      }
    ]
  }],
  "relations": [{
    "type": "antonyms",
    "members": [
      {"memberID": "buy-1"},
      {"memberID": "sell-1"}
    ]
  }],
  "relationTypes": [{
    "type": "antonyms",
    "memberTypes": [{
      "type": "sense",
      "min": 2,
      "max": 2,
      "hint": "navigate"
    }
  ]
}]
}
```

### Example A.45. Suggested rendering for human users

```
buy
- get something by paying money for it
  opposite meaning: sell

sell
- exchange something for money
  opposite meaning: buy
```

## A.1.15 Modelling synonyms

We have three German entries with one sense each, two which mean "sea" and one which means "ocean". We want to set up a relation which brings these three sense together as near-synonyms.

### Example A.46. Pseudocode

```
- lexicographicResource (uri: http://example.com, langCode: de)
  - translationLanguage (langCode: en)
  - entry (id: die-see, headword: See)
    - partOfSpeech (tag: n-fem)
    - sense (id: die-see-1)
      - headwordTranslation (text: sea)
  - entry (id: das-meer, headword: Meer)
    - partOfSpeech (tag: n-neut)
    - sense (id: das-meer-1)
      - headwordTranslation (text: sea)
  - entry (id: der-ozean, headword: Ozean)
    - partOfSpeech (tag: n-masc)
    - sense (id: der-ozean-1)
      - headwordTranslation (text: ocean)
  - relation (type: synonyms, description: words that mean sea and ocean)
    - member (memberID: die-see-1)
    - member (memberID: das-meer-1)
    - member (memberID: der-ozean-1)
  - relationType (type: synonyms, description: synonyms and near synonyms)
    memberType (type: sense, min: 2, hint: navigate)
```

Example A.47. XML

```
<lexicographicResource uri="http://example.com" langCode="en">
  <translationLanguage langCode="de"/>
  <entry id="die-see">
    <headword>See</headword>
    <partOfSpeech value="n-fem"/>
    <sense id="die-see-1">
      <headwordTranslation><text>sea</text></headwordTranslation>
    </sense>
  </entry>
  <entry id="das-meer">
    <headword>Meer</headword>
    <partOfSpeech value="n-neut"/>
    <sense id="das-meer-1">
      <headwordTranslation><text>sea</text></headwordTranslation>
    </sense>
  </entry>
  <entry id="der-ozean">
    <headword>Ozean</headword>
    <partOfSpeech value="n-masc"/>
    <sense id="der-ozean-1">
      <headwordTranslation><text>ocean</text></headwordTranslation>
    </sense>
  </entry>
  <relation type="synonyms">
    <description>words that mean sea and ocean</description>
    <member memberID="die-see-1"/>
    <member memberID="das-meer-1"/>
    <member memberID="der-ozean-1"/>
  </relation>
  <relationType type="synonyms">
    <description>synonyms and near synonyms</description>
    <memberType type="sense" min="2" hint="navigate"/>
  </relationType>
</lexicographicResource>
```

Example A.48. JSON

```
{
  "uri": "http://example.com",
  "langCode": "de",
  "translationLanguages": ["en"],
  "entries": [{
    "id": "die-see",
    "headword": "See",
    "partsOfSpeech": ["n-fem"],
    "senses": [{
      "id": "die-see-1",
      "headwordTranslations": [{"text": "sea"}]
    }]
  }, {
    "id": "das-meer",
    "headword": "Meer",
    "partsOfSpeech": ["n-neut"],
    "senses": [{
      "id": "das-meer-1",
      "headwordTranslations": [{"text": "sea"}]
    }]
  }, {
    "id": "der-ozean",
    "headword": "OZean",
    "partsOfSpeech": ["n-masc"],
    "senses": [{
      "id": "der-ozean-1",
      "headwordTranslations": [{"text": "ocean"}]
    }]
  }],
  "relations": [{
    "type": "synonyms",
    "description": "words that mean sea and ocean",
    "members": [
      {"memberID": "die-see-1"},
      {"memberID": "das-meer-1"},
      {"memberID": "der-ozean-1"}
    ]
  }],
  "relationTypes": [{
    "type": "synonyms",
    "description": "synonyms and near synonyms",
    "memberTypes": [{
      "type": "sense",
      "min": 2,
      "hint": "navigate"
    }]
  }]
}
```

*Example A.49. Suggested rendering for human users*

```
See, feminine noun
- see
  same or similar meaning: Meer, Ozean

Meer, neuter noun
- see
  same or similar meaning: See, Ozean

Ozean, masculine noun
- ocean
  same or similar meaning: See, Meer
```

## A.1.16 Modelling variants

We have two entries in our lexicographic resource, one for the headword "colour" and one for the headword "color". We want to create a relation to represent the fact that these are spelling variants. One of the entries is fully fleshed-out (has a sense with a definition, an example etc) while the other one is only skeletal: its only purpose is to serve as the origin of a navigable link to the other entry.

*Example A.50. Pseudocode*

```
- lexicographicResource (uri: http://example.com, langCode: en)
  - entry (id: colour, headword: colour)
    - partOfSpeech (tag: n)
    - label (tag: europeanSpelling)
    - sense (id: colour-1)
      - definition (text: red, blue, yellow etc.)
      - example (text: What is your favourite colour?)
  - entry (id: color, headword: color)
    - partOfSpeech (tag: n)
    - label (tag: americanSpelling)
  - relation (type: variants)
    - member (memberID: colour)
    - member (memberID: color)
  - relationType (type: variants, description: words which differ only in spelling)
    - memberType (type: entry, min: 2, hint: navigate)
```

Example A.51. XML

```
<lexicographicResource uri="http://example.com" langCode="en">
  <entry id="colour">
    <headword>colour</headword>
    <partOfSpeech tag="n"/>
    <label tag="europeanSpelling"/>
    <sense id="colour-1">
      <definition>red, blue, yellow etc.</definition>
      <example><text>What is your favourite colour?</text></example>
    </sense>
  </entry>
  <entry id="color">
    <headword>color</headword>
    <partOfSpeech tag="n"/>
    <label tag="americanSpelling"/>
  </entry>
  <relation type="variants">
    <member memberID="colour"/>
    <member memberID="color"/>
  </relation>
  <relationType type="variants">
    <description>words which differ only in spelling</description>
    <memberType type="entry" min="2" hint="navigate"/>
  </relationType>
</lexicographicResource>
```

### Example A.52. JSON

```
{
  "uri": "http://example.com",
  "langCode": "en",
  "entries": [{
    "id": "colour",
    "headword": "colour",
    "partsOfSpeech": ["n"],
    "labels": ["europeanSpelling"],
    "senses": [{
      "id": "colour-1",
      "definitions": [{"text": "red, blue, yellow etc."}],
      "examples": [{"text": "What is your favourite colour?"}]
    }
  ]
}, {
  "id": "color",
  "headword": "color",
  "partsOfSpeech": ["n"],
  "labels": ["americanSpelling"]
}],
"relations": [{
  "type": "variants",
  "members": [
    {"memberID": "colour"},
    {"memberID": "color"}
  ]
}],
"relationTypes": [{
  "type": "variants",
  "description": "words which differ only in spelling",
  "memberTypes": [{
    "type": "entry",
    "min": 2,
    "hint": "navigate"
  }]
}]
}]
```

### Example A.53. Suggested rendering for human users

```
colour (noun, European spelling), variant: color
- red, blue, yellow etc.
  "What is your favourite colour?"

color (noun, American spelling), see: colour
```

## A.1.17 Modelling subsenses

We have an entry for the noun "colour" with four senses. We want to express the fact that senses number two and three are subsenses of sense number one, and should be displayed as such to human users.



### Example A.54. Pseudocode

```
- lexicographicResource (uri: http://example.com, langCode: en)
  - entry (id: colour, headword: colour)
    - sense (id: colour-1)
      - definition (text: red, blue, yellow etc.)
      - example (text: What is your favourite colour?)
    - sense (id: colour-2)
      - definition (text: not being black and white)
      - example (text: Owning a colour TV meant you were rich.)
    - sense (id: colour-3)
      - definition (text: a sign of a person's race)
      - example (text: People of all creeds and colours.)
    - sense (id: colour-4)
      - definition (text: interest or excitement)
      - example (text: Examples add colour to your writing.)
  - relation (type: subsensings)
    - member (memberID: colour-1, role: super)
    - member (memberID: colour-2, role: sub)
  - relation (type: subsensings)
    - member (memberID: colour-1, role: super)
    - member (memberID: colour-3, role: sub)
  - relationType (type: subsensings, scopeRestriction: sameEntry)
    - memberType (role: super, type: sense, min: 1, max: 1, hint: none)
    - memberRole (role: sub, type: sense, min: 1, max: 1, hint: embed)
```

Example A.55. XML

```
<lexicographicResource uri="http://example.com" langCode="en">
  <entry id="colour">
    <headword>colour</headword>
    <sense id="colour-1">
      <definition>red, blue, yellow etc.</definition>
      <example><text>What is your favourite colour?</text></example>
    </sense>
    <sense id="colour-2">
      <definition>not being black and white</definition>
      <example><text>Owning a colour TV meant you were rich.</text></example>
    </sense>
    <sense id="colour-3">
      <definition>a sign of a person's race</definition>
      <example><text>People of all creeds and colours.</text></example>
    </sense>
    <sense id="colour-4">
      <definition>interest or excitement</definition>
      <example><text>Examples add colour to your writing.</text></example>
    </sense>
  </entry>
  <relation type="subsensing">
    <member memberID="colour-1" role="super"/>
    <member memberID="colour-2" role="sub"/>
  </relation>
  <relation type="subsensing">
    <member memberID="colour-1" role="super"/>
    <member memberID="colour-3" role="sub"/>
  </relation>
  <relationType type="subsensing" scopeRestriction="sameEntry">
    <memberType role="super" type="sense" min="1" max="1" hint="none"/>
    <memberType role="subs" type="sense" min="1" max="1" hint="embed"/>
  </relationType>
</lexicographicResource>
```

### Example A.56. JSON

```
{
  "uri": "http://example.com",
  "langCode": "en",
  "entries": [{
    "id": "colour",
    "headword": "colour",
    "senses": [{
      "id": "colour-1",
      "definitions": [{"text": "red, blue, yellow etc."}],
      "examples": [{"text": "What is your favourite colour?"}]
    }, {
      "id": "colour-2",
      "definitions": [{"text": "not being black and white"}],
      "examples": [{"text": "Owning a colour TV meant you were rich."}]
    }, {
      "id": "colour-3",
      "definitions": [{"text": "a sign of a person's race"}],
      "examples": [{"text": "People of all creeds and colours."}]
    }, {
      "id": "colour-4",
      "definitions": [{"text": "interest or excitement"}],
      "examples": [{"text": "Examples add colour to your writing."}]
    }
  ]
}],
  "relations": [{
    "type": "subsensing",
    "members": [
      {"role": "super", "memberID": "colour-1"},
      {"role": "sub", "memberID": "colour-2"}
    ]
  }, {
    "type": "subsensing",
    "members": [
      {"role": "super", "memberID": "colour-1"},
      {"role": "sub", "memberID": "colour-3"}
    ]
  }
],
  "relationTypes": [{
    "type": "subsensing",
    "scopeRestriction": "sameEntry",
    "memberTypes": [{
      "role": "super",
      "type": "sense",
      "min": 1,
      "max": 1,
      "hint": "none"
    }, {
      "role": "sub",
      "type": "sense",
      "min": 1,
      "max": 1,
      "hint": "embed"
    }
  ]
}
}]
}
```

### Example A.57. Suggested rendering for human users

colour

1. red, blue, yellow etc. "What is your favourite colour?"
  - (a) not being black and white "Owning a colour TV meant you were rich."
  - (b) a sign of a person's race "People of all creeds and colours.."
2. interest or excitement "Examples add colour to your writing."

## A.1.18 Modelling subentries (at subsense level)

We have an entry for the adjective "safe" with two senses, and an entry for the multi-word expression "better safe than sorry" with one sense. We want to express the fact that the multi-word entry should appear under the first sense of "safe" as a subentry.

### Example A.58. Pseudocode

```
- lexicographicResource (uri: http://example.com, langCode: en)
  - entry (id: safe, headword: safe)
    - sense (id: safe-1, indicator: protected from harm)
      - example (text: It isn't safe to park here.)
    - sense (id: safe-2, indicator: not likely to cause harm)
      - example (text: Is the ride safe for a small child?)
  - entry (id: better-safe, headword: better safe than sorry)
    - sense (id: better-safe-1)
      - definition (text: you should be careful even if it seems unnecessary)
  - relation (type: subentrying)
    member (memberID: safe-1, role: container)
    member (memberID: better-safe, role: subentry)
  - relationType (type: subentrying, scopeRestriction: sameResource)
    - memberType (role: container, type: sense, min: 1, max: 1, hint: navigate)
    - memberType (role: subentry, type: entry, min: 1, max: 1, hint: embed)
```

Example A.59. XML

```
<lexicographicResource uri="http://example.com" langCode="en">
  <entry id="safe">
    <headword>safe</headword>
    <sense id="safe-1">
      <indicator>protected from harm</indicator>
      <example><text>It isn't safe to park here.</text></example>
    </sense>
    <sense id="safe-2">
      <indicator>not likely to cause harm</indicator>
      <example><text>Is the ride safe for a small child?</text></example>
    </sense>
  </entry>
  <entry id="better-safe">
    <headword>better safe than sorry</headword>
    <sense id="better-safe-1">
      <definition>
        <text>you should be careful even if it seems unnecessary</text>
      </definition>
    </sense>
  </entry>
  <relation type="subentrying">
    <member memberID="safe-1" role="container"/>
    <member memberID="better-safe" role="subentry"/>
  </relation>
  <relationType type="subentrying" scopeRestriction="sameResource">
    <memberType role="container" type="sense" min="1" max="1" hint="navigate"/>
    <memberType role="subentry" type="entry" min="1" max="1" hint="embed"/>
  </relationType>
</lexicographicResource>
```

## Example A.60. JSON

```
{
  "uri": "http://example.com",
  "langCode": "en",
  "entries": [{
    "id": "safe",
    "headword": "safe",
    "senses": [{
      "id": "safe-1",
      "indicator": "protected from harm",
      "examples": [{"text": "It isn't safe to park here."}]
    }, {
      "id": "safe-2",
      "indicator": "not likely to cause harm",
      "examples": [{"text": "Is the ride safe for a small child?"}]
    }
  ], {
    "id": "better-safe",
    "headword": "better safe than sorry",
    "senses": [{
      "id": "better-safe-1",
      "definitions": [{
        "text": "you should be careful even if it seems unnecessary"
      }
    ]
  }
  ]],
  "relations": [{
    "type": "subentrying",
    "members": [
      {"role": "container", "memberID": "safe-1"},
      {"role": "subentry", "memberID": "better-safe"}
    ]
  }],
  "relationTypes": [{
    "type": "subentrying",
    "scopeRestriction": "sameResource",
    "memberTypes": [{
      "role": "container",
      "type": "sense",
      "min": 1,
      "max": 1,
      "hint": "navigate"
    }, {
      "role": "subentry",
      "type": "entry",
      "min": 1,
      "max": 1,
      "hint": "embed"
    }
  ]
  ]
}
```

### Example A.61. Suggested rendering for human users

```
safe
- protected from harm "It isn't safe to park here."
  better safe than sorry
  - you should be careful even if it seems unnecessary
- not likely to cause harm "Is the ride safe for a small child?"

better safe than sorry
- you should be careful even if it seems unnecessary
  see also: safe
```

## A.1.19 Modelling subentries (at sense level)

We have an entry for the word "bible" and another entry for the expression "the Bible". We want to make sure that, when a human user is viewing the entry for "bible", the entry for "the Bible" is shown as a subentry of it, as if it were its first sense.

### Example A.62. Pseudocode

```
- lexicographicResource (uri: http://example.com, langCode: en)
  - entry (id: the-bible, headword: the Bible)
    - sense (id: the-bible-1)
      - definition (text: the book considered holy by Christians)
  - entry (id: bible, headword: bible)
    - sense (id: bible-1)
    - sense (id: bible-2)
      - definition (text: a book considered important for a subject)
  - relation (type: subentrying)
    - member (memberID: bible-1, role: container)
    - member (memberID: the-bible, role: subentry)
  - relationType (type: subentrying, scopeRestriction: sameResource)
    - memberType (role: container, type: sense, min: 1, max: 1, hint: navigate)
    - memberRole (role: subentry, type: entry, min: 1, max: 1, hint: embed)
```

Example A.63. XML

```
<lexicographicResource uri="http://example.com" langCode="en">
  <entry id="the-bible">
    <headword>the Bible</headword>
    <sense id="the-bible-1">
      <definition>
        <text>the book considered holy by Christians</text>
      </definition>
    </sense>
  </entry>
  <entry id="bible">
    <headword>bible</headword>
    <sense id="bible-1"/>
    <sense id="bible-2">
      <definition>
        <text>a book considered important for a subject</text>
      </definition>
    </sense>
  </entry>
  <relation type="subentrying">
    <member memberID="bible-1" role="container"/>
    <member memberID="the-bible" role="subentry"/>
  </relation>
  <relationType type="subentrying" scopeRestriction="sameResource">
    <memberType role="container" type="sense" min="1" max="1" hint="navigate"/>
    <memberType role="subentry" type="entry" min="1" max="1" hint="embed"/>
  </relationType>
</lexicographicResource>
```



Example A.64. JSON

```
{
  "uri": "http://example.com",
  "langCode": "en",
  "entries": [{
    "id": "the-bible",
    "headword": "the Bible",
    "senses": [{
      "id": "the-bible-1",
      "definitions": [{"text": "the book considered holy by Christians"}]
    }
  ]}, {
    "id": "bible",
    "headword": "bible",
    "senses": [{
      "id": "bible-1",
      "definitions": [{"text": "a book considered important for a subject"}]
    }, {
      "id": "bible-2",
      "definitions": [{"text": "a book considered important for a subject"}]
    }
  ]},
  "relations": [{
    "type": "subentrying",
    "members": [
      {"role": "container", "memberID": "bible-1"},
      {"role": "subentry", "memberID": "the-bible"}
    ]
  }],
  "relationTypes": [{
    "type": "subentrying",
    "scopeRestriction": "sameResource",
    "memberTypes": [{
      "role": "container",
      "type": "sense",
      "min": 1,
      "max": 1,
      "hint": "navigate"
    }, {
      "role": "subentry",
      "type": "entry",
      "min": 1,
      "max": 1,
      "hint": "embed"
    }
  ]
}]
}
```

### Example A.65. Suggested rendering for human users

```
bible
- the Bible
  - the book considered holy by Christians
- a book considered important for a subject

the Bible
- the book considered holy by Christians
see also: bible
```

## A.1.20 Using placeholderMarker

### Example A.66. Pseudocode

```
- entry (id: continue-studies, headword: continue your studies, placeholderMarker: "you
  - sense ...
```

### Example A.67. XML

```
<entry id="continue-studies">
  <headword>
    continue <placeholderMarker>your</placeholderMarker> studies
  </headword>
  <sense.../>
</entry>
```

### Example A.68. JSON

```
{
  "id": "continue-studies",
  "headword": "continue your studies",
  "placeholderMarkers": [
    {"startIndex": 9, "endIndex": 13}
  ],
  "senses": [...]
}
```

## A.1.21 Using placeholderMarker in a bilingual lexicographic resource

### Example A.69. Pseudocode

```
- entry (id: beat-up, headword: beat sb. up, placeholderMarker: "sb.")
  - sense (id: beat-up-1)
    headwordTranslation (text: jemanden verprügeln, placeholderMarker: "jemanden")
```

### Example A.70. XML

```
<entry id="beat-up">
  <headword>
    beat <placeholderMarker>sb.</placeholderMarker> up
  </headword>
  <sense id="beat-up-1">
    <headwordTranslation>
      <text>
        <placeholderMarker>jemanden</placeholderMarker> verprügeln
      </text>
    </headwordTranslation>
  </sense>
</entry>
```

### Example A.71. JSON

```
{
  "id": "beat-up",
  "headword": "beat sb. up",
  "placeholderMarkers": [
    { "startIndex": 5, "endIndex": 8 }
  ],
  "senses": [
    {
      "id": "beat-up-1",
      "headwordTranslations": [
        {
          "text": "jemanden verprügeln",
          "placeholderMarkers": [
            { "startIndex": 0, "endIndex": 8 }
          ]
        }
      ]
    }
  ]
}
```

## A.1.22 Using headwordMarker

### Example A.72. Pseudocode

```
- entry (id: autopsy, headword: autopsy)
  - sense (id: autopsy-1)
    - headwordTranslation (text: pitva)
    - example (text: The coroner performed an autopsy., headwordMarker: "autopsy")
      - exampleTranslation (text: Koroner provedl pitvu., headwordMarker: "pitvu")
```

### Example A.73. XML

```
<entry id="autopsy">
  <headword>autopsy</headword>
  <sense id="autopsy-1">
    <headwordTranslation><text>pitva</text></headwordTranslation>
    <example>
      <text>
        The coroner performed an <headwordMarker>autopsy</headwordMarker>.
      </text>
      <exampleTranslation>
        <text>
          Koroner provedl <headwordMarker>pitvu</headwordMarker>.
        </text>
      </exampleTranslation>
    </example>
  </sense>
</entry>
```

### Example A.74. JSON

```
{
  "id": "autopsy",
  "headword": "autopsy",
  "senses": [{
    "id": "autopsy-1",
    "headwordTranslations": [{"text": "pitva"}],
    "examples": [{
      "text": "The coroner performed an autopsy.",
      "headwordMarkers": [
        {"startIndex": 25, "endIndex": 32}
      ],
      "exampleTranslations": [{
        "text": "Koroner provedl pitvu.",
        "headwordMarkers": [
          {"startIndex": 16, "endIndex": 21}
        ]
      }
    ]
  }
]}]
```

## A.1.23 Using collocateMarker

### Example A.75. Pseudocode

```
- entry (id: autopsy, headword: autopsy)
  - sense id: autopsy-1
    - headwordTranslation (text: pitva)
    - example (text: The coroner performed an autopsy.)
      - headwordMarker: "autopsy"
      - collocateMarker: "performed" (lemma: perform)
    - exampleTranslation (text: Koroner provedl pitvu.)
      - headwordMarker: "pitvu"
      - collocateMarker: "provedl" (lemma: provést)
```

### Example A.76. XML

```
<entry id="autopsy">
  <headword>autopsy</headword>
  <sense id="autopsy-1">
    <headwordTranslation><text>pitva</text></headwordTranslation>
    <example>
      <text>
        The coroner <collocateMarker lemma="perform">performed</collocateMarker>
        an <headwordMarker>autopsy</headwordMarker>.
      </text>
    <exampleTranslation>
      <text>
        Koroner <collocateMarker lemma="provést">provedl</collocateMarker>
        <headwordMarker>pitvu</headwordMarker>.
      </text>
    </exampleTranslation>
  </example>
</sense>
</entry>
```

### Example A.77. JSON

```
{
  "id": "autopsy",
  "headword": "autopsy",
  "senses": [{
    "id": "autopsy-1",
    "headwordTranslations": [{"text": "pitva"}],
    "examples": [{
      "text": "The coroner performed an autopsy.",
      "headwordMarkers": [
        {"startIndex": 25, "endIndex": 32}
      ],
      "collocateMarkers": [
        {"startIndex": 12, "endIndex": 21, "lemma": "perform"}
      ],
      "exampleTranslations": [{
        "text": "Koroner provedl pitvu.",
        "headwordMarkers": [
          {"startIndex": 16, "endIndex": 21}
        ],
        "collocateMarkers": [
          {"startIndex": 8, "endIndex": 15, "lemma": "provést"}
        ],
      }],
    }],
  }],
}
```

## A.1.24 Modelling etymology

### Example A.78. Pseudocode

```
- entry (id: cat-n, headword: cat)
  - sense ...
  - etymology
    - etymon
      - etymonUnit (langCode: enm, text: catte)
    - etymon
      - etymonUnit (langCode: ang, text: catt, translation: male cat)
      - etymonUnit (langCode: ang, text: catte, translation: female cat)
    - etymon
      - etymonUnit (langCode: gem-x-proto, text: kattuz, reconstructed: true)
```

### Example A.79. XML

```
<entry id="cat-n">
  <headword>cat</headword>
  <sense>...</sense>
  <etymology>
    <etymon>
      <etymonUnit langCode="enm">
        <text>catte</text>
      </etymonUnit>
    </etymon>
    <etymon>
      <etymonUnit langCode="ang">
        <text>catt</text>
        <translation>male cat</translation>
      </etymonUnit>
      <etymonUnit langCode="ang">
        <text>catte</text>
        <translation>female cat</translation>
      </etymonUnit>
    </etymon>
    <etymon>
      <etymonUnit langCode="gem-x-proto" reconstructed="true">
        <text>kattuz</text>
      </etymonUnit>
    </etymon>
  </etymology>
</entry>
```

### Example A.80. JSON

```
{
  "id": "cat-n",
  "headword": "cat",
  "senses": [...],
  "etymology": {
    "etymons" [{
      "etymonUnits": [
        { "langCode": "enm", "text": "catte" }
      ]
    }, {
      "etymonUnits": [
        { "langCode": "ang", "text": "catt", "translation": "male cat" },
        { "langCode": "ang", "text": "catte", "translation": "female cat" }
      ]
    }, {
      "etymonUnits": [
        { "langCode": "gem-x-proto", "text": "kattuz", "reconstructed": true }
      ]
    }
  ]
}
```

## A.2 DMLex NVH serialization (Informative)

This section specifies a serialization of the DMLex data model in NVH (Name-Value Hierarchy) [NVH].

For data exchange purposes, a valid DMLex NVH file is one where either a `lexicographicResource` or an `entry` is the top-level node or the file contains a concatenation of one (and only one) of these two nodes separated by a system-specific newline delimiter.

### A.2.1 Design Principles (Informative)

The NVH serialization of DMLex shown in this document follows these principles:

- The value of the node is the text property of the object, if defined, otherwise it is one of the required object properties of maximum arity 1.
- All other objects properties are implemented as NVH nodes.

See [namevaluehierarchy.org](http://namevaluehierarchy.org) for details about the NVH file format and processing libraries.

### A.2.2 DMLex NVH serialization nodes (Normative)

#### A.2.2.1 NVH node: `lexicographicResource`

Implements the `lexicographicResource` object type.

*Value*

- `title` OPTIONAL

*Child nodes*

- `entry` OPTIONAL (zero or one)
- `uri` REQUIRED (exactly one)
- `langCode` REQUIRED (exactly one)

*Child nodes if implementing the Crosslingual Module*

- `translationLanguage` REQUIRED (one or more)

*Child nodes if implementing the Controlled Values Module*

- `definitionTypeTag` OPTIONAL (zero or more)
- `inflectedFormTag` OPTIONAL (zero or more)
- `labelTag` OPTIONAL (zero or more)
- `labelTypeTag` OPTIONAL (zero or more)
- `partOfSpeechTag` OPTIONAL (zero or more)
- `sourceIdentityTag` OPTIONAL (zero or more)

*Child nodes if implementing the Linking Module*

- `relation` OPTIONAL (zero or more)
- `relationType` OPTIONAL (zero or more)



*Child nodes if implementing the Etymology Module*

- [etymonLanguage](#) OPTIONAL (zero or more)
- [originType](#) OPTIONAL (zero or more)

### **A.2.2.2 NVH node: [entry](#)**

Implements the [entry](#) object type.

*Value*

- [headword](#) REQUIRED

*Child nodes*

- [id](#) OPTIONAL (zero or one)
- [homographNumber](#) OPTIONAL (zero or one)
- [partOfSpeech](#) OPTIONAL (zero or more)
- [label](#) OPTIONAL (zero or more)
- [pronunciation](#) OPTIONAL (zero or more)
- [inflectedForm](#) OPTIONAL (zero or more)
- [sense](#) OPTIONAL (zero or more)

*Child nodes if implementing the Annotation Module*

- [placeholderMarker](#) OPTIONAL (zero or more)

*Child nodes if implementing the Etymology Module*

- [etymology](#) OPTIONAL (zero or more)

### **A.2.2.3 NVH node: [partOfSpeech](#)**

Implements the [partOfSpeech](#) object type.

*Value*

- [tag](#) REQUIRED

### **A.2.2.4 NVH node: [inflectedForm](#)**

Implements the [inflectedForm](#) object type.

*Value*

- [text](#) REQUIRED

*Child nodes*

- [tag](#) OPTIONAL (zero or one)
- [label](#) OPTIONAL (zero or more)
- [pronunciation](#) OPTIONAL (zero or more)

### A.2.2.5 NVH node: *sense*

Implements the *sense* object type.

#### *Value*

- *indicator* OPTIONAL (zero or one)

#### *Child nodes*

- *id* OPTIONAL (zero or one)
- *label* OPTIONAL (zero or more)
- *definition* OPTIONAL (zero or more)
- *example* OPTIONAL (zero or more)

#### *Child nodes if implementing the Crosslingual Module*

- *headwordExplanation* OPTIONAL (zero or more)
- *headwordTranslation* OPTIONAL (zero or more)

### A.2.2.6 NVH node: *definition*

Implements the *definition* object type.

#### *Value*

- REQUIRED, implements the *text* property.

#### *Child nodes*

- *definitionType* OPTIONAL (zero or one)

#### *Child nodes if implementing the Annotation Module*

- *headwordMarker* node OPTIONAL (zero or more)
- *collocateMarker* node OPTIONAL (zero or more)

### A.2.2.7 NVH node: *label*

Implements the *label* object type.

#### *Value*

- *tag* REQUIRED

### A.2.2.8 NVH node: *pronunciation*

Implements the *pronunciation* object type.

#### *Value*

- *soundFile* REQUIRED if *transcription* is present, OPTIONAL otherwise

#### *Child nodes*

- *transcription* OPTIONAL (zero or more) if *soundFile* is present, REQUIRED (one or more) otherwise

- `label` OPTIONAL (zero or more)

#### A.2.2.9 NVH node: `transcription`

Implements the `transcription` object type.

*Value*

- REQUIRED, implements the `text` property.

*Child nodes*

- `scheme` OPTIONAL (zero or one)

#### A.2.2.10 NVH node: `example`

Implements the `example` object type.

*Value*

- `text` REQUIRED

*Child nodes*

- `sourceIdentity` OPTIONAL (zero or one)
- `sourceElaboration` OPTIONAL (zero or one)
- `soundFile` OPTIONAL (zero or one)
- `label` OPTIONAL (zero or more)

*Child nodes if implementing the Annotation Module*

- `headwordMarker` OPTIONAL (zero or more)
- `collocateMarker` OPTIONAL (zero or more)

*Child nodes if implementing the Crosslingual Module*

- `exampleTranslation` OPTIONAL (zero or more)

#### A.2.2.11 NVH node: `translationLanguage`

Implements the `translationLanguage` object type from the Crosslingual Module.

*Value*

- `langCode` REQUIRED

#### A.2.2.12 NVH node: `headwordTranslation`

Implements the `headwordTranslation` object type from the Crosslingual Module.

*Value*

- REQUIRED, implements the `text` property.

*Child nodes*

- `langCode` OPTIONAL (zero or one) if the ancestor `lexicographicResource` contains exactly one `translationLanguage`, REQUIRED otherwise

- `partOfSpeech` OPTIONAL (zero or more)
- `label` OPTIONAL (zero or more)
- `pronunciation` OPTIONAL (zero or more)
- `inflectedForm` OPTIONAL (zero or more)

*Child nodes if implementing the Annotation Module*

- `placeholderMarker` OPTIONAL

### **A.2.2.13 NVH node: `headwordExplanation`**

Implements the `headwordExplanation` object type from the Crosslingual Module.

*Value*

- REQUIRED, implements the `text` property.

*Child nodes*

- `langCode` OPTIONAL (ZERO OR ONE) if the ancestor `lexicographicResource` contains exactly one `translationLanguage`, REQUIRED otherwise

### **A.2.2.14 NVH node: `exampleTranslation`**

Implements the `exampleTranslation` object type from the Crosslingual Module.

*Value*

- `text` REQUIRED

*Child nodes*

- `langCode` OPTIONAL (zero or one)
- `soundFile` OPTIONAL (zero or one) if the ancestor `lexicographicResource` contains exactly one `translationLanguage`, REQUIRED otherwise
- `label` OPTIONAL (zero or more)

*Child nodes if implementing the Annotation Module*

- `headwordMarker` OPTIONAL (zero or more)
- `collocateMarker` OPTIONAL (zero or more)

### **A.2.2.15 NVH node: `partOfSpeechTag`**

Implements the `partOfSpeechTag` object type from the Controlled Values Module.

*Value*

- `tag` REQUIRED

*Child nodes*

- `forHeadwords` OPTIONAL (zero or one), true or false
- `forTranslations` OPTIONAL (zero or one), true or false
- `forEtymology` OPTIONAL (zero or one), true or false

- `description` OPTIONAL (zero or one)
- `forLanguage` OPTIONAL (zero or more)
- `sameAs` OPTIONAL (zero or more)

#### **A.2.2.16 NVH node: `inflectedFormTag`**

Implements the `inflectedFormTag` object type from the Controlled Values Module.

*Value*

- `tag` REQUIRED

*Child nodes*

- `forHeadwords` OPTIONAL (zero or one), true or false
- `forTranslations` OPTIONAL (zero or one), true or false
- `description` OPTIONAL (zero or one)
- `forLanguage` OPTIONAL (zero or more)
- `forPartOfSpeech` OPTIONAL (zero or more)
- `sameAs` OPTIONAL (zero or more)

#### **A.2.2.17 NVH node: `definitionTypeTag`**

Implements the `definitionTypeTag` object type from the Controlled Values Module.

*Value*

- `tag` REQUIRED

*Child nodes*

- `description` OPTIONAL (zero or one)
- `sameAs` OPTIONAL (zero or more)

#### **A.2.2.18 NVH node: `labelTag`**

Implements the `labelTag` object type from the Controlled Values Module.

*Value*

- `tag` REQUIRED

*Child nodes*

- `description` OPTIONAL (zero or one)
- `typeTag` OPTIONAL (zero or one)
- `forHeadwords` OPTIONAL (zero or one), true or false
- `forTranslations` OPTIONAL (zero or one), true or false
- `forCollocates` OPTIONAL (zero or one), true or false
- `forLanguage` OPTIONAL (zero or more)

- [forPartOfSpeech](#) OPTIONAL (zero or more)
- [sameAs](#) OPTIONAL (zero or more)

#### A.2.2.19 NVH node: [labelTypeTag](#)

Implements the [labelTypeTag](#) object type from the Controlled Values Module.

*Value*

- `tag` REQUIRED

*Child nodes*

- `description` OPTIONAL (zero or one)
- [sameAs](#) OPTIONAL (zero or more)

#### A.2.2.20 NVH node: [sourceIdentityTag](#)

Implements the [sourceIdentityTag](#) object type from the Controlled Values Module.

*Value*

- `tag` REQUIRED

*Child nodes*

- `description` OPTIONAL (zero or one)
- [sameAs](#) OPTIONAL (zero or more)

#### A.2.2.21 NVH node: [transcriptionSchemeTag](#)

Implements the [transcriptionSchemeTag](#) object type from the Controlled Values Module.

*Value*

- `tag` REQUIRED

*Child nodes*

- `forHeadwords` OPTIONAL (zero or one), true or false
- `forTranslations` OPTIONAL (zero or one), true or false
- `forHeadwords` OPTIONAL (zero or one), true or false
- `forTranslations` OPTIONAL (zero or one), true or false
- `description` OPTIONAL (zero or one)
- [forLanguage](#) OPTIONAL (zero or more)

#### A.2.2.22 NVH node: [forLanguage](#)

Implements the `forLanguage` property of the [partOfSpeechTag](#), [inflectedFormTag](#), [transcriptionSchemeTag](#) and [labelTag](#) object types from the Controlled Values Module.

*Value*

- `langCode` REQUIRED

#### A.2.2.23 NVH node: `forPartOfSpeech`

Implements the `forPartOfSpeech` property of the `inflectedFormTag` and `labelTag` object types from the Controlled Values Module.

##### *Value*

- `tag` REQUIRED

#### A.2.2.24 NVH node: `sameAs`

Implements the `sameAs` object type from the Controlled Values Module.

##### *Value*

- `uri` REQUIRED

#### A.2.2.25 NVH node: `relation`

Implements the `relation` object type from the Linking Module.

##### *Value*

- `type` REQUIRED

##### *Child nodes*

- `description` OPTIONAL (zero or one)
- `member` REQUIRED (two or more)

#### A.2.2.26 NVH node: `member`

Implements the `member` object type from the Linking Module.

##### *Value*

- `memberID` REQUIRED

##### *Child nodes*

- `role` OPTIONAL (zero or one)
- `obverseListingOrder` REQUIRED (exactly one)

#### A.2.2.27 NVH node: `relationType`

Implements the `relationType` object type from the Linking Module.

##### *Value*

- `type` REQUIRED

##### *Child nodes*

- `scopeRestriction` REQUIRED (exactly one)
- `description` OPTIONAL (zero or one)
- `memberType` OPTIONAL (zero or more)
- `sameAs` OPTIONAL (zero or more)

### A.2.2.28 NVH node: `memberType`

Implements the `memberType` object type from the Linking Module.

*Value*

- `type` REQUIRED

*Child nodes*

- `role` REQUIRED (exactly one)
- `min` OPTIONAL (zero or one)
- `max` OPTIONAL (zero or one)
- `hint` OPTIONAL (zero or one)
- `description` OPTIONAL (zero or one)
- `sameAs` OPTIONAL (zero or more)

### A.2.2.29 NVH node: `placeholderMarker`

Implements the `placeholderMarker` object type from the Annotation module. Contains the text which it marks up.

*Value*

- REQUIRED, implements the `text` property.

### A.2.2.30 NVH node: `headwordMarker`

Implements the `headwordMarker` object type from the Annotation module. Contains the text which it marks up.

*Value*

- REQUIRED, implements the `text` property.

### A.2.2.31 NVH node: `collocateMarker`

Implements the `collocateMarker` object type from the Annotation module. Contains the text which it marks up and optional child nodes.

*Value*

- REQUIRED, implements the `text` property.

*Child nodes*

- `id` OPTIONAL (zero or one)
- `lemma` OPTIONAL (zero or one)
- `label` OPTIONAL (zero or more)

### A.2.2.32 NVH node: `etymology`

Implements the `etymology` object type from the Etymology Module.

*Value*

- `description` OPTIONAL (zero or one)



#### *Child nodes*

- [etymon](#) OPTIONAL (zero or more)

#### **A.2.2.33 NVH node: [etymon](#)**

Implements the [etymon](#) object type from the Etymology Module.

#### *Child nodes*

- `when` OPTIONAL (zero or one)
- `type` OPTIONAL (zero or one)
- `note` OPTIONAL (zero or one)
- [etymonUnit](#) REQUIRED (one or more)
- `translation` OPTIONAL (zero or one)

#### **A.2.2.34 NVH node: [etymonUnit](#)**

Implements the [etymonUnit](#) object type from the Etymology Module.

#### *Value*

- `text` REQUIRED

#### *Child nodes*

- `langCode` REQUIRED (exactly one)
- `reconstructed` OPTIONAL (zero or one)
- [partOfSpeech](#) OPTIONAL (zero or more)
- `translation` OPTIONAL (zero or one)

#### **A.2.2.35 NVH node: [etymonType](#)**

Implements the [etymonType](#) object type from the Etymology Module.

#### *Value*

- `type` REQUIRED

#### *Child nodes*

- `description` OPTIONAL (zero or one)

#### **A.2.2.36 NVH node: [etymonLanguage](#)**

Implements the [etymonLanguage](#) object type from the Etymology Module.

#### *Value*

- `langCode` REQUIRED

#### *Child nodes*

- `displayName` OPTIONAL (zero or one)

---

## Appendix B References (Normative)

This appendix contains the normative and informative references that are used in this document. Normative references are specific (identified by date of publication and/or edition number or Version number) and Informative references are either specific or non-specific.

While any hyperlinks included in this appendix were valid at the time of publication, OASIS cannot guarantee their long-term validity.

### B.1 Normative references (Normative)

[BCP 14] is a concatenation of [RFC 2119] and [RFC 8174]

[RFC 2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, <https://www.ietf.org/rfc/rfc2119.txt> IETF (Internet Engineering Task Force) RFC 2119, March 1997.

[RFC 8174] B. Leiba, *Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words*, <https://www.ietf.org/rfc/rfc8174.txt> IETF (Internet Engineering Task Force) RFC 8174, May 2017.

[BCP 47] M. Davis, *Tags for Identifying Languages*, <http://tools.ietf.org/html/bcp47> IETF (Internet Engineering Task Force).

[ITS] David Filip, Shaun McCance, Dave Lewis, Christian Lieske, Arle Lommel, Jirka Kosek, Felix Sasaki, Yves Savourel *Internationalization Tag Set (ITS) Version 2.0*, <http://www.w3.org/TR/its20/> W3C Recommendation 29 October 2013.

[JSON] *The JavaScript Object Notation (JSON) Data Interchange Format*, <https://tools.ietf.org/html/rfc8259> IETF RFC 8259 December 2017.

[RFC 3987] M. Duerst and M. Suignard, *Internationalized Resource Identifiers (IRIs)*, <https://www.ietf.org/rfc/rfc3987.txt> IETF (Internet Engineering Task Force) RFC 3987, January 2005.

[RFC 7303] H. Thompson and C. Lilley, *XML Media Types*, <https://www.tools.ietf.org/html/rfc7303> IETF (Internet Engineering Task Force) RFC 7303, July 2014.

[Schematron] International Standards Organization, *ISO/IEC 19757-3, Information Technology - Document Schema Definition Languages (DSDL) - Part 3: Rule-Based Validation — Schematron (Second Edition)*, [http://standards.iso.org/ittf/PubliclyAvailableStandards/c055982\\_ISO\\_IEC\\_19757-3\\_2016.zip](http://standards.iso.org/ittf/PubliclyAvailableStandards/c055982_ISO_IEC_19757-3_2016.zip) ISO, January 15, 2016.

[SQL] International Standards Organization, *ISO/IEC 9075, Information technology – Database languages – SQL*, <https://www.iso.org/standard/76583.html> ISO, 2023.

[UAX #9] M. Davis, A. Lanin, A. Glass, *UNICODE BIDIRECTIONAL ALGORITHM*, <http://www.unicode.org/reports/tr9/tr9-35.html> Unicode Bidirectional Algorithm, May 18, 2016.

[UAX #15] M. Davis, K. Whistler, *UNICODE NORMALIZATION FORMS*, <http://www.unicode.org/reports/tr15/tr15-44.html> Unicode Normalization Forms, February 24, 2016.

[Unicode] The Unicode Consortium, *The Unicode Standard*, <http://www.unicode.org/versions/Unicode9.0.0/> Mountain View, CA: The Unicode Consortium, June 21, 2016.

[XLIFF 2.1] David Filip, Tom Comerford, Soroush Saadatfar, Felix Sasaki, and Yves Savourel, eds. *XLIFF Version 2.0*, <http://docs.oasis-open.org/xliff/xliff-core/v2.1/os/xliff-core-v2.1-os.html> OASIS Standard 13 February 2018

[XML] W3C, *Extensible Markup Language (XML) 1.0*, <http://www.w3.org/TR/xml/> (Fifth Edition) W3C Recommendation 26 November 2008.

[**XML namespace**] W3C, *Schema document for namespace* <http://www.w3.org/XML/1998/namespace> <http://www.w3.org/2001/xml.xsd> [<http://www.w3.org/2009/01/xml.xsd>]. at <https://docs.oasis-open.org/lexidma/dmlex/v1.0/csd01/schemas/informativeCopiesOf3rdPartySchemas/w3c/xml.xsd> in this distribution

[**XML Catalogs**] Norman Walsh, *XML Catalogs*, <https://www.oasis-open.org/committees/download.php/14809/xml-catalogs.html> OASIS Standard V1.1, 07 October 2005.

[**XML Schema**] W3C, *XML Schema*, refers to the two part standard comprising [[XML Schema Structures](#)] and [[XML Schema Datatypes](#)] (Second Editions) W3C Recommendations 28 October 2004.

[**XML Schema Datatypes**] W3C, *XML Schema Part 2: Datatypes*, <http://www.w3.org/TR/xmlschema-2/> (Second Edition) W3C Recommendation 28 October 2004.

[**XML Schema Structures**] W3C, *XML Schema Part 1: Structures*, <https://www.w3.org/TR/xmlschema-1/> (Second Edition) W3C Recommendation 28 October 2004.

[**DCMI Metadata Terms**] DCMI Usage Board, *DCMI Metadata Terms*, <https://www.dublincore.org/specifications/dublin-core/dcmi-terms/> Date issued: 2020-01-20.

## B.2 Informative references (Informative)

[**NVH**] *Name-Value Hierarchy*, <https://www.namevaluehierarchy.org/> .

[**LDML**] *Unicode Locale Data Markup Language* <http://unicode.org/reports/tr35/>

[**UAX #29**] M. Davis, *UNICODE TEXT SEGMENTATION*, <http://www.unicode.org/reports/tr29/> Unicode text Segmentation.

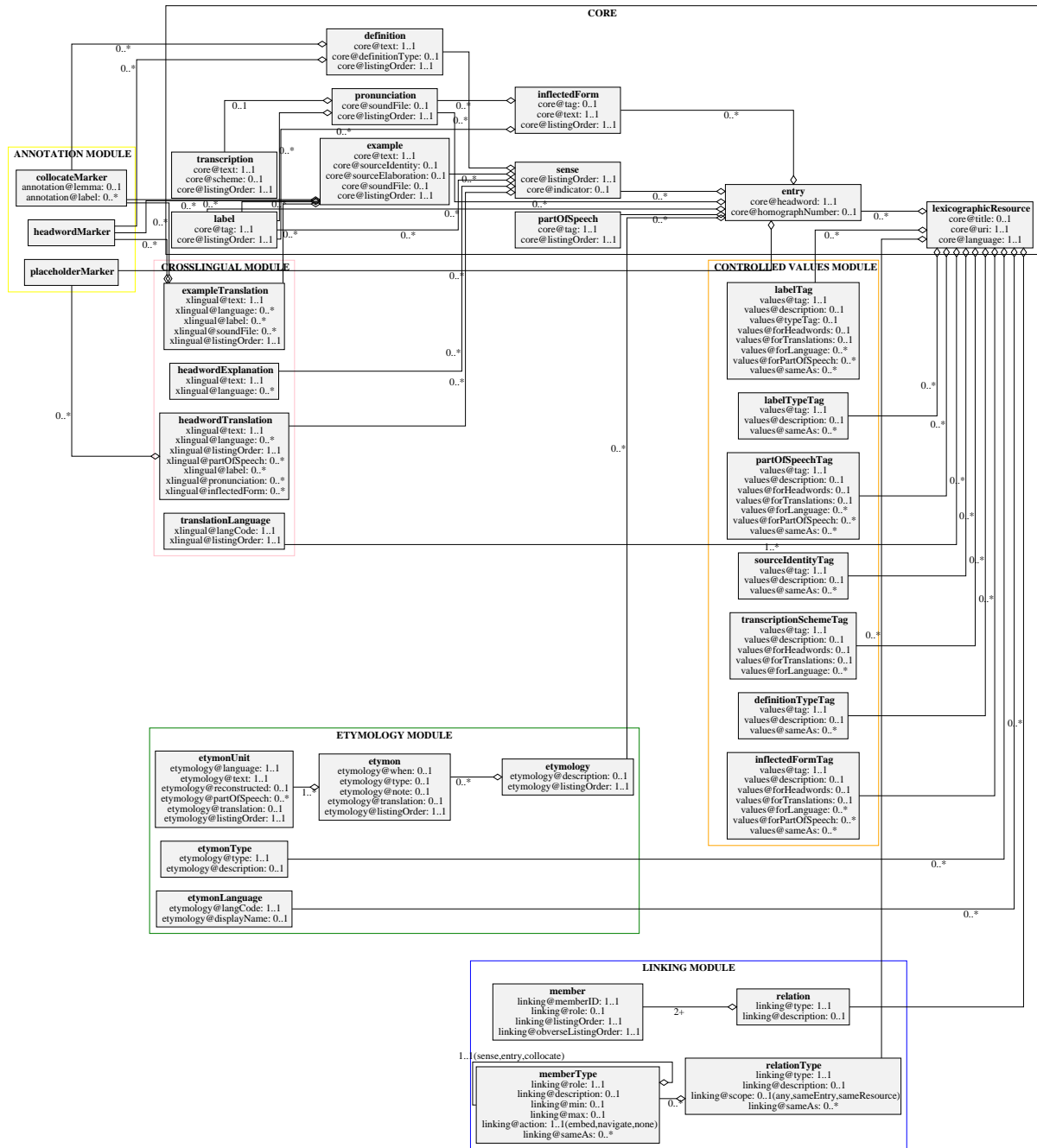
---

## Appendix C Machine Readable Validation Artifacts (Informative)

CURRENTLY NO VALIDATION ARTIFACTS FORESEEN FOR THE OM.. JUST FOR SERIALIZATIONS

MAY LIST CONFORMANT ARTIFACTS FOR SPECIFIC SERIALIZATIONS AT A LATER STAGE

# Appendix D DMLex UML diagram (Normative)



---

## Appendix E DMLex plain text description (Informative)

This description uses the [NVH schema notation style](#) and the convention that all object types and their properties are prefixed by either `core@` or `MODULENAME@` strings. Their arity is denoted by a `*` (zero or more), `+` (one or more), `?` (zero or one) or `N+` (`N` or more). If no arity is given, it indicates the arity of 1, i.e. being required and occurring exactly once.

```
core@lexicographicResource:
  core@title: ?
  core@uri:
  core@language:
  core@entry: *
  xlingual@translationLanguage: +
  values@definitionTypeTag: *
  values@inflectedFormTag: *
  values@labelTag: *
  values@labelTypeTag: *
  values@partOfSpeechTag: *
  values@sourceIdentityTag: *
  values@transcriptionSchemeTag: *
  linking@relation: *
  linking@relationType: *
  etymology@etymonLanguage: *
  etymology@etymonType: *
core@entry:
  core@headword:
  core@homographNumber: ?
  core@partOfSpeech: *
  core@label: *
  core@pronunciation: *
  core@inflectedForm: *
  core@sense: *
  annotation@placeholderMarker: *
  etymology@etymology: *
core@partOfSpeech:
  core@tag:
  core@listingOrder:
core@inflectedForm:
  core@tag: ?
  core@text:
  core@label: *
  core@pronunciation: *
  core@listingOrder:
core@sense:
  core@listingOrder:
  core@indicator: ?
  core@label: *
  core@definition: *
  core@example: *
  xlingual@headwordExplanation: *
```

```

    xlingual@headwordTranslation: *
core@definition:
  core@text:
  core@definitionType: ?
  core@listingOrder:
  annotation@headwordMarker: *
  annotation@collocateMarker: *
core@label:
  core@tag:
  core@listingOrder:
core@pronunciation:
  core@soundFile: ?
  core@transcription: ?
  core@listingOrder:
  core@label: *
core@transcription:
  core@text:
  core@scheme: ?
  core@listingOrder:
core@example:
  core@text:
  core@sourceIdentity: ?
  core@sourceElaboration: ?
  core@label: *
  core@soundFile: ?
  core@listingOrder:
  xlingual@exampleTranslation: *
  annotation@headwordMarker: *
  annotation@collocateMarker: *
xlingual@translationLanguage:
  xlingual@langCode:
  xlingual@listingOrder:
xlingual@headwordExplanation:
  xlingual@text:
  xlingual@language: *
xlingual@headwordTranslation:
  xlingual@text:
  xlingual@language: *
  xlingual@listingOrder:
  xlingual@partOfSpeech: *
  xlingual@label: *
  xlingual@pronunciation: *
  xlingual@inflectedForm: *
  annotation@placeholderMarker: *
xlingual@exampleTranslation:
  xlingual@text:
  xlingual@language: *
  xlingual@label: *
  xlingual@soundFile: *
  xlingual@listingOrder:
  annotation@headwordMarker: *
  annotation@collocateMarker: *
values@definitionTypeTag:
  values@tag:
  values@description: ?
  values@sameAs: *
values@inflectedFormTag:
  values@tag:

```

```

values@description: ?
values@forHeadwords: ?
values@forTranslations: ?
values@forLanguage: *
values@forPartOfSpeech: *
values@sameAs: *
values@labelTag:
  values@tag:
  values@description: ?
  values@typeTag: ?
  values@forHeadwords: ?
  values@forTranslations: ?
  values@forLanguage: *
  values@forPartOfSpeech: *
  values@sameAs: *
values@labelTypeTag:
  values@tag:
  values@description: ?
  values@sameAs: *
values@partOfSpeechTag:
  values@tag:
  values@description: ?
  values@forHeadwords: ?
  values@forTranslations: ?
  values@forLanguage: *
  values@forPartOfSpeech: *
  values@sameAs: *
values@sourceIdentityTag:
  values@tag:
  values@description: ?
  values@sameAs: *
values@transcriptionSchemeTag:
  values@tag:
  values@description: ?
  values@forHeadwords: ?
  values@forTranslations: ?
  values@forLanguage: *
linking@relation:
  linking@type:
  linking@description: ?
  linking@member: 2+
linking@member:
  linking@memberID:
  linking@role: ?
  linking@listingOrder:
  linking@obverseListingOrder:
linking@relationType:
  linking@type:
  linking@description: ?
  linking@scope: ?(any,sameEntry,sameResource)
  linking@memberType: *
  linking@sameAs: *
linking@memberType:
  linking@role:
  linking@description: ?
  linking@memberType: (sense,entry,collocate)
  linking@min: ?
  linking@max: ?

```



```
    linking@action: (embed,navigate,none)
    linking@sameAs: *
annotation@placeholderMarker:
annotation@headwordMarker:
annotation@collocateMarker:
    annotation@lemma: ?
    annotation@label: *
etymology@etymology:
    etymology@description: ?
    etymology@etymon: *
    etymology@listingOrder:
etymology@etymon:
    etymology@when: ?
    etymology@type: ?
    etymology@note: ?
    etymology@etymonUnit: +
    etymology@translation: ?
    etymology@listingOrder:
etymology@etymonUnit:
    etymology@language:
    etymology@text:
    etymology@reconstructed: ?
    etymology@partOfSpeech: *
    etymology@translation: ?
    etymology@listingOrder:
etymology@etymonLanguage:
    etymology@langCode:
    etymology@displayName: ?
etymology@etymonType:
    etymology@type:
    etymology@description: ?
```

---

## Appendix F Specification Change Tracking (Informative)

This appendix will contain tracked changes after the csd01 phase will have been reached.

---

## Appendix G Acknowledgements (Informative)

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

- Erjavec, Tomaž - Jožef Stefan Institute
- Filip, David - Masaryk University (previously TCD, ADAPT Centre)
- Jakubíček, Miloš - Lexical Computing
- Kallas, Jelena - Institute of the Estonian Language
- Kernerman, Ilan - K Dictionaries
- Kokol, Marko - Jožef Stefan Institute
- Kosem, Iztok - Jožef Stefan Institute
- Krek, Simon - Jožef Stefan Institute
- McCrae, John - University of Galway
- Měchura, Michal - Masaryk University
- Tiberius, Carole - Dutch Language Institute

---

## Appendix H Notices (Informative)

Copyright © OASIS Open 2023. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full [Policy](#) may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

As stated in the OASIS IPR Policy, the following three paragraphs in brackets apply to OASIS Standards Final Deliverable documents (Committee Specification, OASIS Standard, or Approved Errata).

[OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Standards Final Deliverable, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this deliverable.]

[OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this OASIS Standards Final Deliverable by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this OASIS Standards Final Deliverable. OASIS may include such claims on its website, but disclaims any obligation to do so.]

[OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this OASIS Standards Final Deliverable or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Standards Final Deliverable, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.]

The name "OASIS" is a trademark of [OASIS](#), the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see <https://www.oasis-open.org/policies-guidelines/trademark/> for above guidance.