Data Model for Lexicography (DMLex), Version 1.0

Working Draft 01

17 July 2023

Specification URIs

This version:
http://docs.oasis-open.org/lexidma/dmlex/v1.0/wd01/dmlex-v1.0-wd01.html (Authoritative)
http://docs.oasis-open.org/lexidma/dmlex/v1.0/wd01/dmlex-v1.0-wd01.pdf
http://docs.oasis-open.org/lexidma/dmlex/v1.0/wd01/dmlex-v1.0-wd01.xml

Previous version:
http://docs.oasis-open.org/lexidma/dmlex/v1.0/N/A/dmlex-v1.0-N/A.html (Authoritative)
http://docs.oasis-open.org/lexidma/dmlex/v1.0/N/A/dmlex-v1.0-N/A.pdf
http://docs.oasis-open.org/lexidma/dmlex/v1.0/N/A/dmlex-v1.0-N/A.xml

Latest version:
http://docs.oasis-open.org/lexidma/dmlex/v1.0/dmlex-v1.0.html (Authoritative)
http://docs.oasis-open.org/lexidma/dmlex/v1.0/dmlex-v1.0.pdf
http://docs.oasis-open.org/lexidma/dmlex/v1.0/dmlex-v1.0.xml

Technical Committee:
OASIS Lexicographic Infrastructure Data Model and API (LEXIDMA) TC

Chair:
Tomaž Erjavec (tomaz.erjavec@ijs.si), Jozef Stefan Institute

Editors:
Michal M#chura (michmech@mail.muni.cz), Masaryk University
David Filip (david.filip@adaptcentre.ie), Trinity College Dublin (ADAPT)
Simon Krek (simon.krek@ijs.si), Jozef Stefan Institute

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 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 implementation 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.

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 M#chura, David Filip and Simon Krek. 17 July 2023. OASIS Working Draft 01. http://docs.oasis-open.org/lexidma/dmlex/v1.0/wd01/dmlex-v1.0-wd01.html. Latest version: http://docs.oasis-open.org/lexidma/dmlex/v1.0/dmlex-v1.0.html.

Notices

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.

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 Committee Specification or OASIS Standard, 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 specification.

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 specification 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 specification. 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 document 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 Committee Specification or OASIS Standard, 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.

Table of Contents

Appendixes

1 Introduction

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, as a relational database, and as a Semantic Web triplestore.

1.1 Modular structure of DMLex

The DMLex specification is divided into a 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 give recommended implementations in XML, in JSON, in RDF, and as a relational database.

- The XML and JSON implementations are intended as serializations for data exchange: for encoding lexicographic data while the data is in transit out of one software system into another. Examples of what the two serializations look like with real-world data are given in Section 9, “Examples”.
- The relational database implementation is intended as a representation for lexicographic data while the data is being edited and maintained inside a software system, such as a Dictionary Writing System (DWS).

2 Conformance

1. *DMLex Instances Conformance*

    a. Conformant DMLex Instances MUST be well formed and valid instances according to one of DMLex Serialization Specifications.

    b. Another Instance conformance clause.

    c. ...

    d. DMLex Instances MAY contain custom extensions, as defined in the Extension Mechanisms section. Extensions MUST be serialized in a way conformant 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 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 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.

Note

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

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 headwords, part-of-speech labels, senses, definitions and so on.

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 database.

Note

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

Contents

- `title` OPTIONAL (zero or one). Non-empty string. A human-readable title of the lexicographic resource.
- `uri` OPTIONAL (zero or one). The URI of the lexicographic resource, identifying it on the Web.
- `language` REQUIRED (exactly one). The IETF language code of the language that this lexicographic resource describes.
- `entry` OPTIONAL (zero, one or more)

Comments

- `language` 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 `language` child object of `lexicographicResource` informs potential users of the lexicographic resource which language that is.
- The main role of a lexicographic resource is to contain entries (`entry` objects). The other object type that can optionally occur inside a `lexicographicResource`, `tag`, is for lists of look-up values such as part-of-speech labels.
- Ideally, a lexicographic resouce should include at least one entry. However, DMLex specifies that `entry` is optional in `lexicographicResouce` to allow for the existence of lexicographic resources which are not yet complete.
- The `lexicographicResource` data type does not contain fields for detailed metadata about the lexicographic resource, such as author, editor, publisher, copyright status or publication year. Describing these properties of lexicographic resources is outside the scope of DMLex. DMLex is a formalism for modelling the internal structure of a lexicographic resource, not its metadata.

Example 1. XML

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

Example 2. JSON

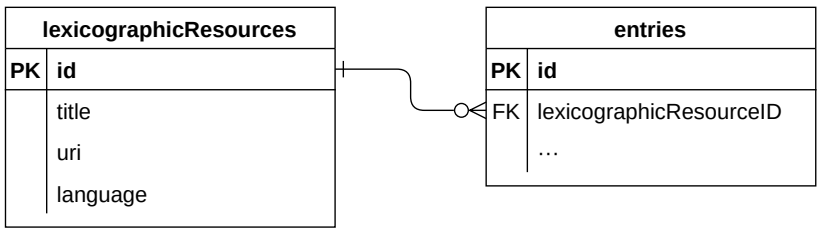
```
{
    "title": "...",
    "language": "...",
    "entries": [...]
}
```

Example 3. RDF

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

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

Example 4. Relational database



3.2 `entry`

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

Child of

- `lexicographicResource`

Contents

- `id` OPTIONAL (zero or one). An unique identifier of the entry. Entries which have idenfitiers are capable of being involved in relations created with the Linking module.
- `headword` REQUIRED (exactly one). Non-empty string. The entry's headword.
- `homographNumber` OPTIONAL (zero or one). The entry's homograph number, as a guide to distinguish entries with the same headword.
- `partOfSpeech` OPTIONAL (zero, one or more).
- `label` OPTIONAL (zero, one or more).
- `pronunciation` OPTIONAL (zero, one or more).

- `inflectedForm` OPTIONAL (zero, one or more).
- `sense` OPTIONAL (zero, one 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 like 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 9.16, “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 id="..." homographNumber="...">
    <headword>...</headword>
    <partOfSpeech.../>
    <label.../>
    <pronunciation.../>
    <inflectedForm.../>
    <sense.../>
</entry>
```

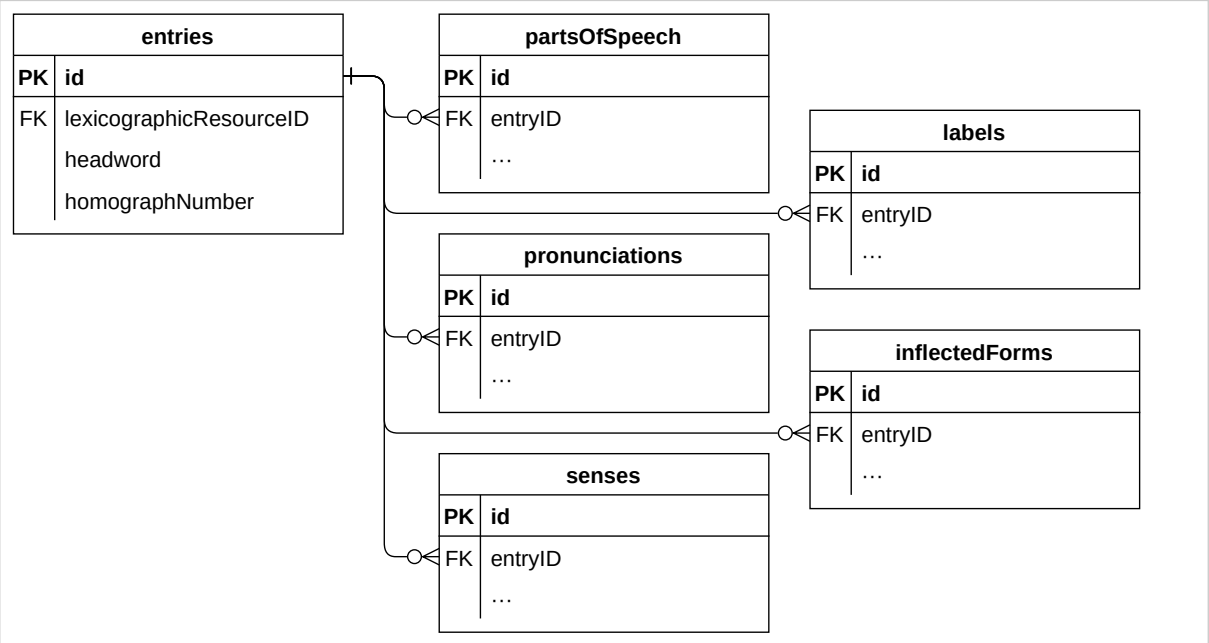
Example 6. JSON

```
{
    "id": "...",
    "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.

Child of

- `entry`

Contents

- `tag` REQUIRED (exactly one). 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 label, 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 9.2, “How to use `inflectedForm`”.

Child of

- `entry`

Contents

- `tag` OPTIONAL (zero or one). Non-empty string. An abbreviation, a code or some other string of text which identifies the inflected form, for example `pl` 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.
- `text` REQUIRED (exactly one). Non-empty string. The text of the inflected form.
- `label` OPTIONAL (zero, one or more).
- `pronunciation` OPTIONAL (zero, one 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.../>
</inflectedTag>
```

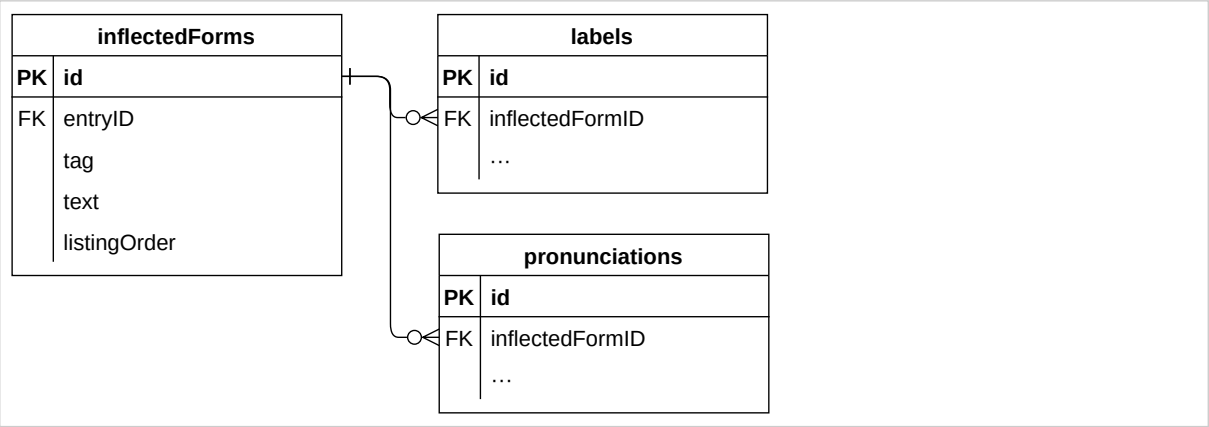
Example 14. JSON

```
{
    "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 is intended to model the **inflectional morphology** of a headword. To model derivational morphology, for example feminine forms of maculine 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.

Child of

- `entry`

Contents

- `id` OPTIONAL (zero or one). A unique identifier of the sense. Senses which have idenfitiers can be involved in relations created with the Linking module.
- `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). 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, one or more).
- `definition` OPTIONAL (zero, one or more).
- `example` OPTIONAL (zero, one or more).

Comments

- The contents 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 inm DMLex to model an entry where each sense has slightly different morphological properties (eg. a noun has a weak plural in one sense and a strong plural in another). Such phenomena need to be treatedas 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 id="...">
    <indicator>...</indicator>
    <label.../>
    <definition.../>
    <example.../>
</sense>
```

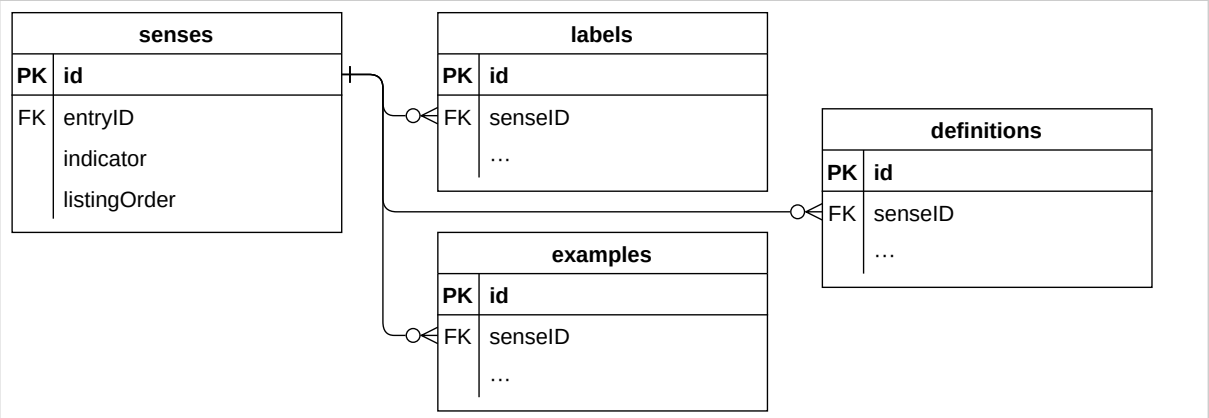
Example 18. JSON

```
{
    "id": "...",
    "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.

Child of

- `sense`

Contents

- `text` REQUIRED (exactly one). 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	
PK	**id**
FK	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).

Child of

- `entry`
- `sense`
- `inflectedForm`
- `pronunciation`
- `example`

Contents

- `tag` REQUIRED (exactly one). 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 pronuciation 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	
PK	**id**
FK	entryID
FK	senseID
FK	inflectedFormID
FK	pronunciationID
FK	exampleID
	tag
	listingOrder

3.8 pronunciation

Represents the pronunciation of its parent. Examples: Section 9.3, “Pronunciation given as transcription”, Section 9.4, “Pronunciation given as a sound file”, Section 9.5, “Pronunciation given both ways”.

Child of

- `entry`
- `inflectedForm`

Contents

- At least one of:
  - `soundFile` OPTIONAL (zero or one). A pointer to a file, such as a filename or a URI, containing a sound recording of the pronunciation
  - `transcription` OPTIONAL (zero, one or more).
- `listingOrder` REQUIRED (exactly one). Number. The position of this pronuciation object among other pronuciation objects of the same entry. This can be implicit from the serialization.
- `label` OPTIONAL (zero, one or more).

Example 29. XML

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

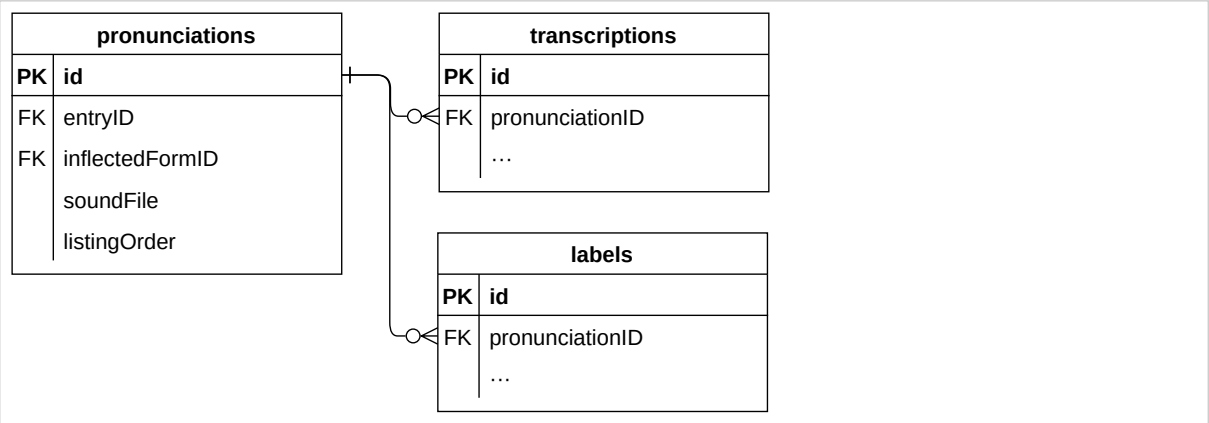
Example 30. JSON

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

Example 31. RDF

```
<entry> dmlex:pronunciation [
  a dmlex:Pronunicaton ;
  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.

Child of

- pronunciation

Contents

- `text` REQUIRED (exactly one). 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="...">...
```

- `label` OPTIONAL (zero, one 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>
```

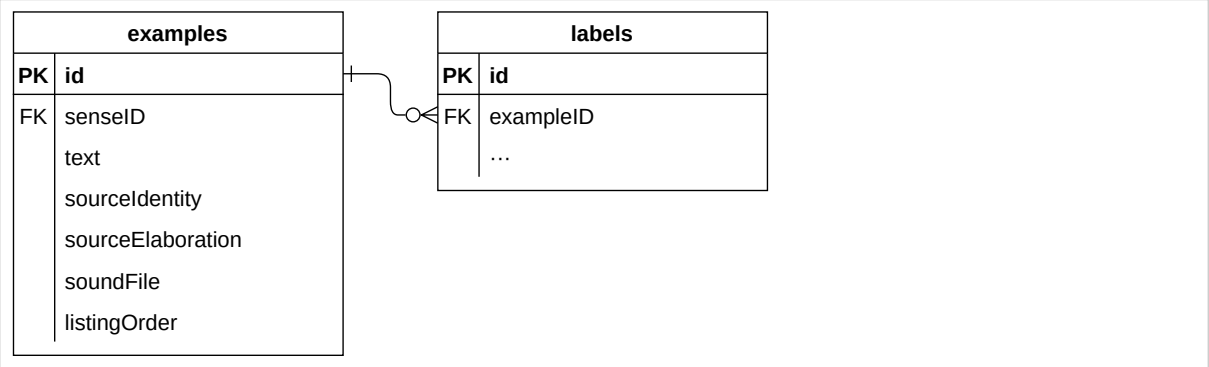
Example 38. JSON

```
{
    "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 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 "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 Extensions to `lexicographicResource`

Extends the `lexicographicResource` object type from the Core.

Additional contents

- `translationLanguage` REQUIRED (one or more)

Example 41. XML

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

Example 42. JSON

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

Example 43. RDF

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

4.2 `translationLanguage`

Represents one of the languages in which translations are given in this lexicographic resource. Examples: Section 9.8, “Defining a bilingual lexicographic resource”, Section 9.9, “Defining a multilingual lexicographic resource”.

Child of

- `lexicographicResource`

Contents

- `langCode` REQUIRED (exactly one). 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

4.3 Extensions to sense

Extends the `sense` object type from the Core.

Additional contents

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

Example 48. XML

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

Example 49. JSON

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

Example 50. RDF

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

4.4 headwordTranslation

Represents one of possibly multiple translations of a headword. Examples: Section 9.10, “How to use headwordTranslation in a bilingual lexicographic resource”, Section 9.11, “How to use headwordTranslation in a multilingual lexicographic resource”.

Child of

- sense

Contents

- text REQUIRED (exactly one). Non-empty string.
- language OPTIONAL (zero or one) if only one translation language exists in the lexicographic resource, REQUIRED (exactly one) 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, one or more).
- label OPTIONAL (zero, one or more).
- pronunciation OPTIONAL (zero, one or more).
- inflectedForm OPTIONAL (zero, one or more).

Example 51. XML

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

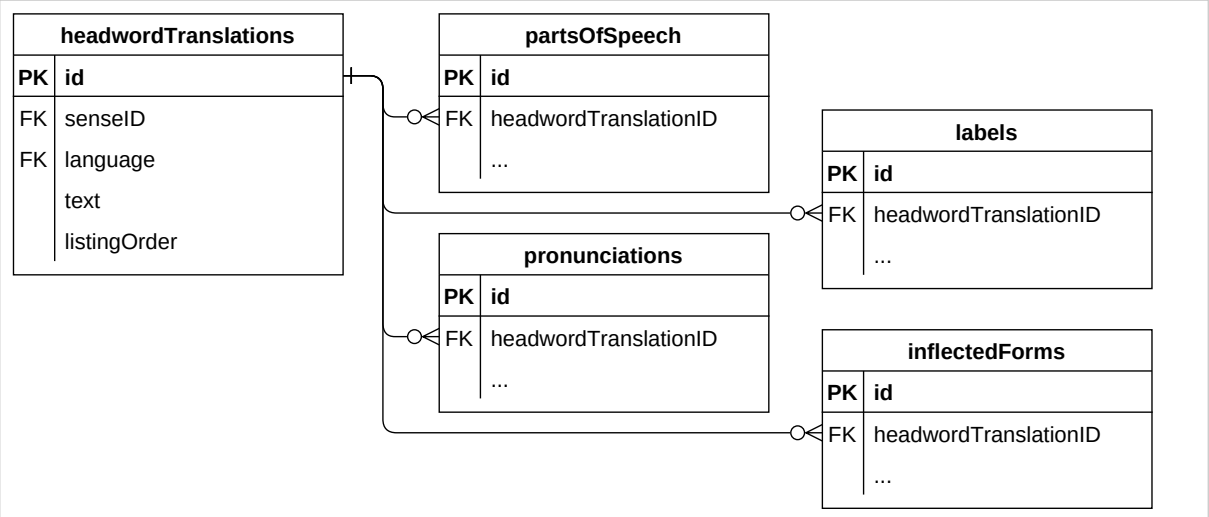
Example 52. JSON

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

Example 53. RDF

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

Example 54. Relational database



4.5 headwordExplanation

Represents a statement in the translation language which explains (but does not translate) the meaning of the headword. Example: Section 9.12, “How to use headwordExplanation”.

Child of

- sense

Contents

- text REQUIRED (exactly one). Non-empty string.
- language OPTIONAL (zero or one) if only one translation language exists in the lexicographic resource, REQUIRED (exactly one) otherwise. 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 listingOrder.

Example 55. XML

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

Example 56. JSON

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

Example 57. RDF

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

Example 58. Relational database

headwordExplanations	
PK	**id**
FK	senseID
FK	language
	text

4.6 Extensions to `example`

Extends the `example` object type from the Core.

Additional contents

- `exampleTranslation` OPTIONAL (zero, one or more)

Example 59. XML

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

Example 60. JSON

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

Example 61. RDF

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

4.7 `exampleTranslation`

Represents the translation of an example.

Child of

- `example`

Contents

- `text` REQUIRED (exactly one). Non-empty string.
- `language` OPTIONAL (zero or one) if only one translation language exists in thelexicographic resource, REQUIRED (exactly one) 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, one 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 language="..." soundFile="...">
    <text>...</text>
    <label.../>
</exampleTranslation>
```

Example 63. JSON

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

Example 64. RDF

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

Example 65. Relational database

4.8 Extensions to `partOfSpeech`

Extends the `partOfSpeech` object type from the Core.

Can additionally be a child of

- `headwordTranslation`

Example 66. Relational database

partsOfSpeech	
PK	id
FK	entryID
FK	headwordTranslationID
	tag
	listingOrder

4.9 Extensions to `label`

Extends the `label` object type from the Core.

Can additionally be a child 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

4.10 Extensions to `pronunciation`

Extends the `pronunciation` object type from the Core.

Can additionally be a child of

- `headwordTranslation`

Example 68. Relational database

pronunciations	
PK	id
FK	entryID
FK	inflectedFormID
FK	headwordTranslationID
	soundFile
	listingOrder

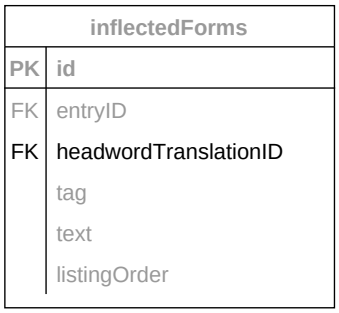
4.11 Extensions to `inflectedForm`

Extends the `inflectedForm` object type from the Core.

Can additionally be a child of

- `headwordTranslation`

Example 69. Relational database



5 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.

5.1 Extensions to `lexicographicResource`

Extends the `lexicographicResource` object type from the Core.

Additional contents

- `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)

Example 70. XML

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

Example 71. JSON

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

Example 72. RDF

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

5.2 partOfSpeechTag

Represents one (of many) possible values for `tag` of `partOfSpeech`. Example: Section 9.6, “How to use `partOfSpeechTag` and `inflectedFormTag`”.

Child of

- `lexicographicResource`

Contents

- `tag` REQUIRED (exactly one). 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 ore 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 ore 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 ore one). Boolean. If present, indicates whether this tag is intended to be used in etymolgy: as `partOfSpeech` of `etymon`.
- `forLanguage` OPTIONAL (zero, one 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, one or more).

Example 73. XML

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

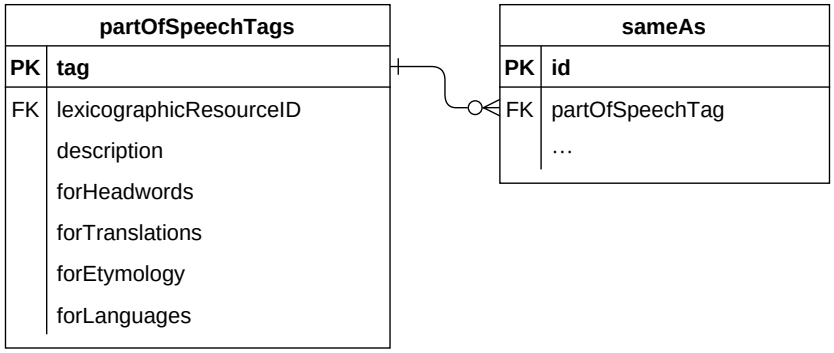
Example 74. JSON

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

Example 75. RDF

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

Example 76. Relational database



The `forLanguages` column is a comma-delimited list of language codes.

5.3 `inflectedFormTag`

Represents one (of many) possible values for `tag` of `inflectedForm`. Example: Section 9.6, “How to use `partOfSpeechTag` and `inflectedFormTag`”.

Child of

- `lexicographicResource`

Contents

- `tag` REQUIRED (exactly one). 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 ore 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 ore 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, one 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, one 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, one or more).

Example 77. XML

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

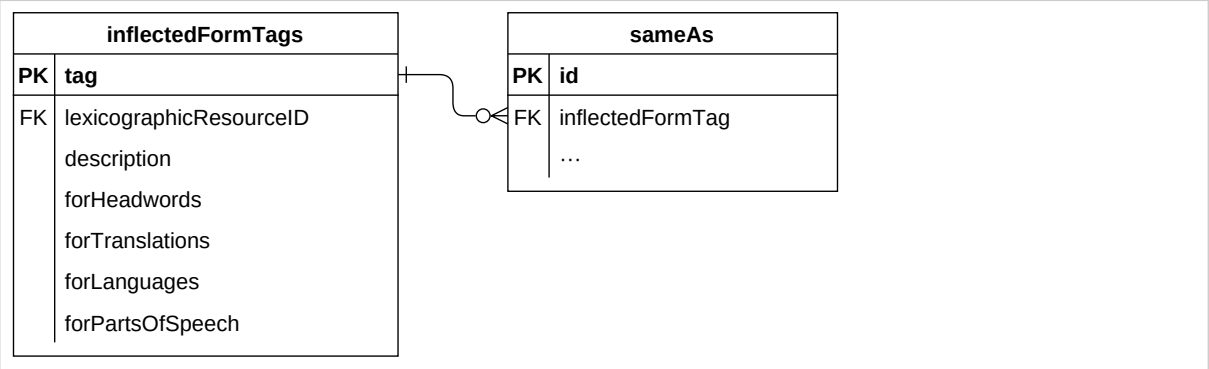
Example 78. JSON

```
{
    "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 "..." ;
  dmlex:sameAs ... ] .
```

Example 80. Relational database



The `forPartsOfSpeech` column is a comma-delimited list of part-of-speech labels.

The `forLanguages` column is a comma-delimited list of language codes.

5.4 definitionTypeTag

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

Child of

- `lexicographicResource`

Contents

- `tag` REQUIRED (exactly one). 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, one or more).

Example 81. XML

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

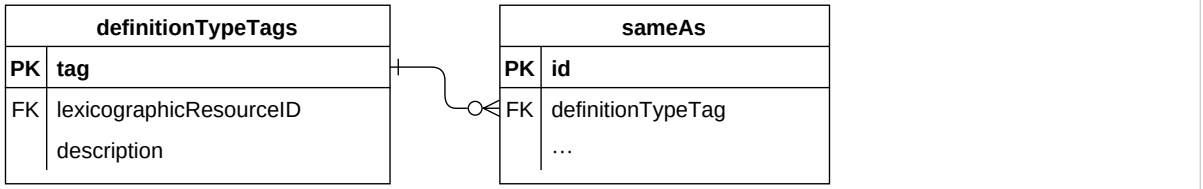
Example 82. JSON

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

Example 83. RDF

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

Example 84. Relational database



5.5 labelTag

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

Child of

- `lexicographicResource`

Contents

- `tag` REQUIRED (exactly one). 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.
- `forHeadwords` OPTIONAL (zero ore 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 ore 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 ore 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, one 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, one 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, one or more).

Example 85. XML

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

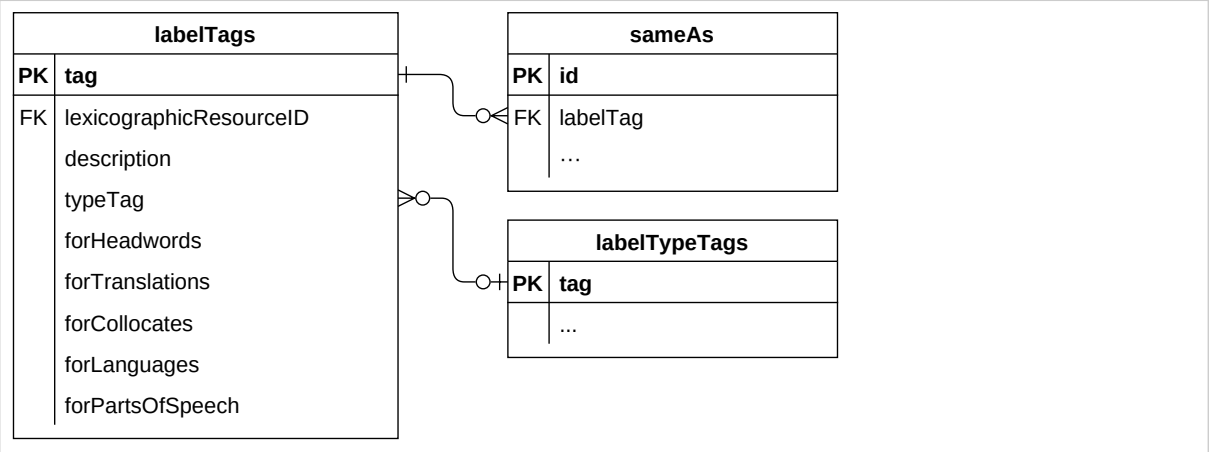
Example 86. JSON

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

Example 87. RDF

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

Example 88. Relational database



The `partOfSpeechConstraints` column is a comma-delimited list of part-of-speech labels.

The `translationLanguageConstraints` column is a comma-delimited list of language codes.

5.6 labelTypeTag

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

Child of

- `lexicographicResource`

Contents

- `tag` REQUIRED (exactly one). 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.

Example 89. XML

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

Example 90. JSON

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

Example 91. RDF

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

Example 92. Relational database

labelTypeTags	
PK	**tag**
FK	lexicographicResourceID
	description

5.7 sourceIdentityTag

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

Child of

- `lexicographicResource`

Contents

- `tag` REQUIRED (exactly one). 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, one or more).

Example 93. XML

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

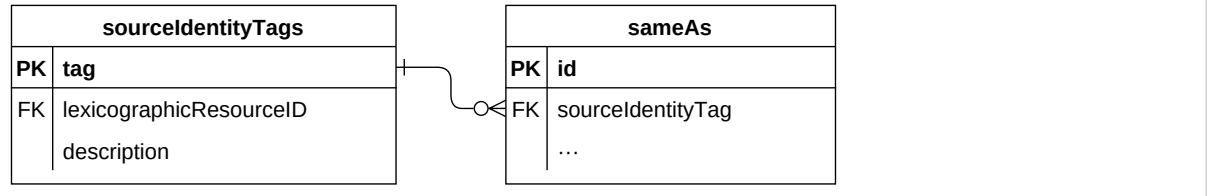
Example 94. JSON

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

Example 95. RDF

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

Example 96. Relational database



5.8 `transcriptionSchemeTag`

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

Child of

- `lexicographicResource`

Contents

- `tag` REQUIRED (exactly one). An IETF language tag.
- `description` OPTIONAL (zero or one). Non-empty string. A human-readable description of what the tag means.

- `forHeadwords` OPTIONAL (zero ore 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 ore 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, one 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="..."/>
</tag>
```

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

transcriptionSchemeTags	
PK	**tag**
FK	lexicographicResourceID
	description
	forHeadwords
	forTranslations
	forLanguages

The `forLanguages` column is a comma-delimited list of language codes.

5.9 sameAs

Represents the fact that the parent object is equivalent to an item available from an external authority. Example: Section 9.7, “Mapping controlled values to external inventories”.

Child of

- `definitionTypeTag`
- `inflectedFormTag`
- `labelTag`
- `partOfSpeechTag`
- `sourceIdentityTag`

Contents

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

Example 101. XML

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

Example 102. JSON

```
"..."
```

Example 103. Relational database

sameAs	
PK	**id**
FK	definitionTypeTag
FK	inflectedFormTag
FK	labelTag
FK	partOfSpeechTag
FK	sourceIdentityTag
	uri

6 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 9.13, “Modelling parts and wholes”, Section 9.14, “Modelling antonyms”, Section 9.15, “Modelling synonyms”, Section 9.16, “Modelling variants”, Section 9.17, “Modelling subsenses”, Section 9.18, “Modelling subentries (at subsense level)”, Section 9.19, “Modelling subentries (at sense level)”.

6.1 Extensions to `lexicographicResource`

Extends the `lexicographicResource` object type from the Core.

Additional contents

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

Example 104. XML

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

Example 105. JSON

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

Example 106. RDF

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

6.2 `relation`

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

Child of

- `lexicographicResource`

Contents

- `type` REQUIRED (exactly one). 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).

Example 107. XML

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

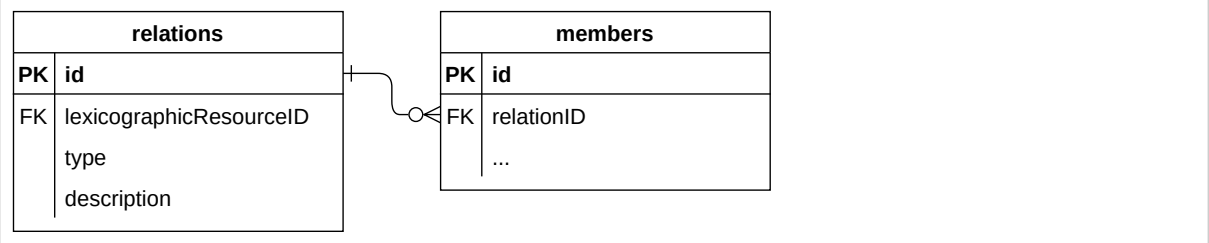
Example 108. JSON

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

Example 109. RDF

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

Example 110. Relational database



6.3 member

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

Child of

- relation

Contents

- memberID REQUIRED (exactly one). The ID of an object, such as an entry or a sense.
- role OPTIONAL (zero or one). 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="..."/>
```

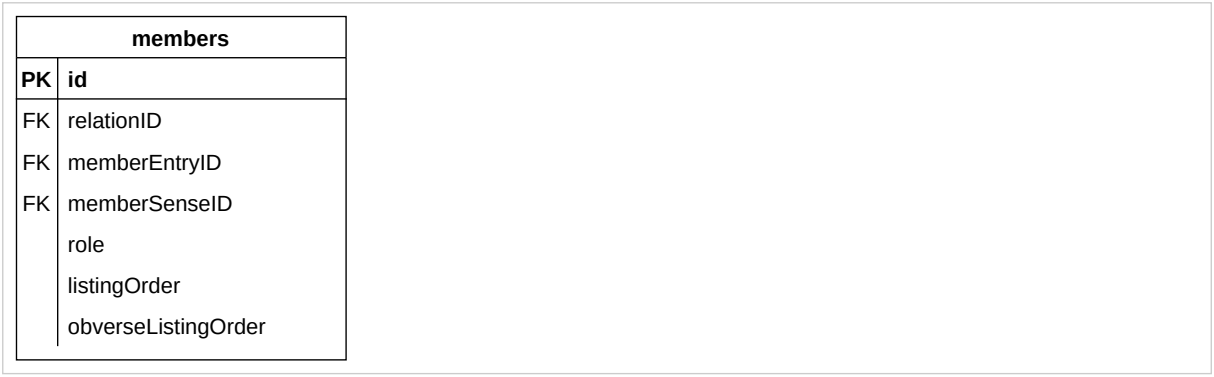
Example 112. JSON

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

Example 113. RDF

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

Example 114. Relational database



members	
PK	**id**
FK	relationID
FK	memberEntryID
FK	memberSenseID
	role
	listingOrder
	obverseListingOrder

6.4 `relationType`

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

Child of

- `lexicographicResource`

Contents

- `type` REQUIRED (exactly one). Non-empty string.
- `description` OPTIONAL (zero or one). Non-empty string. A human-readable explanation of this relation type.
- `scope` 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` (default): no restriction
- `memberType` OPTIONAL (zero or more).
- `sameAs` OPTIONAL (zero or more).

Example 115. XML

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

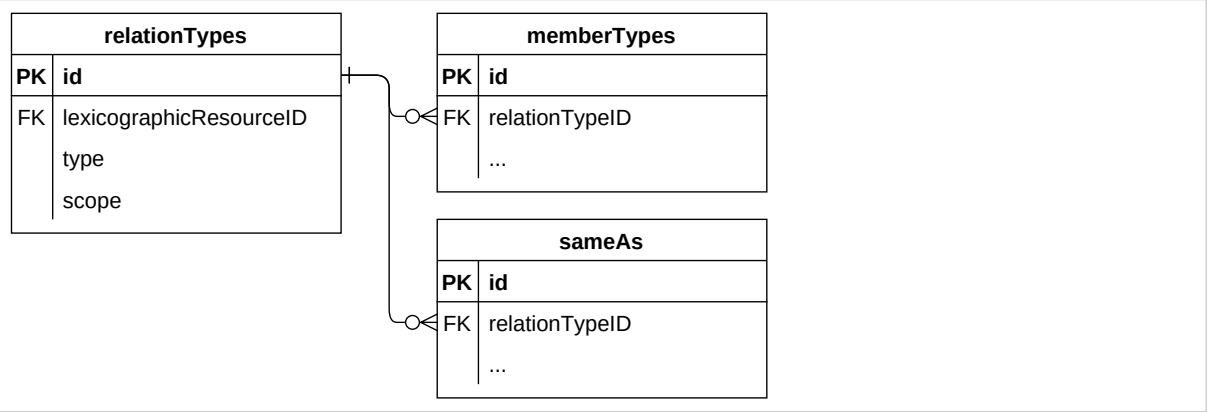
Example 116. JSON

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

Example 117. RDF

```
<#lexicographicResource> dmlex:relationType [
  dmlex:type "..." ;
  dmlex:scope "..." ;
  dmlex:memberType ... ;
  dmlex:sameAs ... ] .
```

Example 118. Relational database



6.5 `memberType`

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

Child of

- `relationType`

Contents

- `role` REQUIRED (exactly one). 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.
- `memberType` REQUIRED (exactly one). 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.
- `action` REQUIRED (exactly one). Non-empty string. Gives instructions on what machine agents should do when showing this relation to a human user (either on its own or in the context of one of its members). The possibe 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="..." memberType="..." min="..." max="..." action="...">
    <description></description>
    <sameAs.../>
</memberType>
```

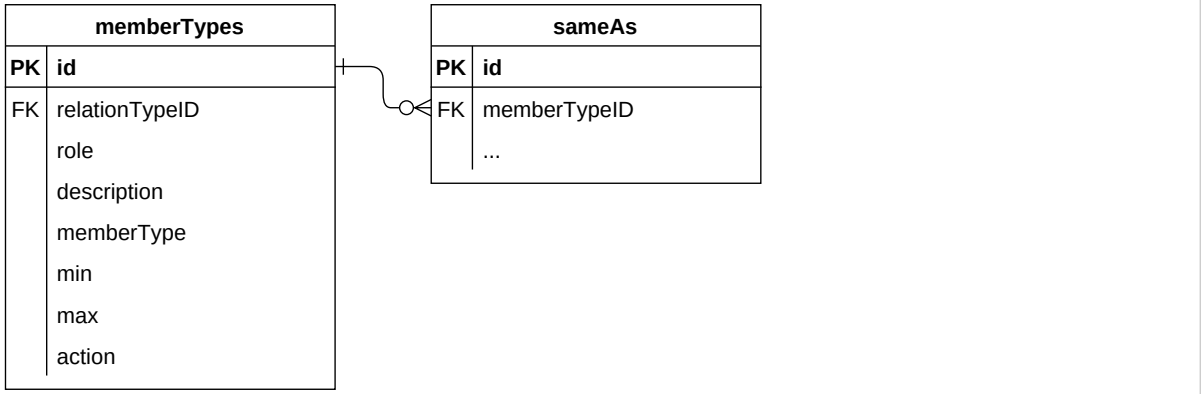
Example 120. JSON

```
{
    "role": "...",
    "description": "...",
    "memberType": "...",
    "min": "...",
    "max": "...",
    "action": "...",
    "sameAs": [...]
}
```

Example 121. RDF

```
<#relationType> dmlex:memberType [
  dmlex:role "..." ;
  dmlex:description "..." ;
  dmlex:memberType "..." ;
  dmlex:min 0 ;
  dmlex:max 0 ;
  dmlex:action "..." ;
  dmlex:sameAs ... ] .
```

Example 122. Relational database



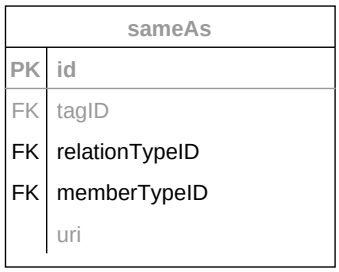
6.6 Extensions to sameAs

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

Can additionally be a child of

- `relationTypeTag`
- `memberRoleTag`

Example 123. Relational database



7 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).

7.1 Extensions to `entry`

Extends the `entry` object type from the Core.

Additional contents

- `placeholderMarker` OPTIONAL (zero, one 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 ... .
```

7.2 Extensions to `headwordTranslation`

Extends the `headwordTranslation` object type from the Crosslingual module.

Additional contents

- `placeholderMarker` OPTIONAL (zero, one 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 ... .
```

7.3 `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 9.20, “Using `placeholderMarker`”, Section 9.21, “Using `placeholderMarker` in a bilingual lexicographic resource”.

Child 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
	startIndex
	endIndex

7.4 Extensions to `definition`

Extends the `definition` object type from the Core.

Additional contents

- `headwordMarker` OPTIONAL (zero, one or more)
- `collocateMarker` OPTIONAL (zero, one 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 ... ] .
```

7.5 Extensions to `example`

Extends the `example` object type from the Core.

Additional contents

- `headwordMarker` OPTIONAL (zero, one or more)
- `collocateMarker` OPTIONAL (zero, one 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 ... ] .
```

7.6 Extensions to `exampleTranslation`

Extends the `exampleTranslation` object type from the Crosslingual module.

Additional contents

- `headwordMarker` OPTIONAL (zero, one or more)
- `collocateMarker` OPTIONAL (zero, one 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 ... ] .
```

7.7 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 occurence of the headword for human readers through formatting. Example: Section 9.22, “Using headwordMarker”.

Child 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

7.8 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 9.23, “Using collocateMarker”.

Child of

- `definition`
- `example`
- `exampleTranslation`

Contents

- `id` OPTIONAL (zero or one). A unique identifier of the marker. Markers which have idenfitiers are capable of being involved in relations created with the Linking module.
- `lemma` OPTIONAL (zero or one). 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, one 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>
```

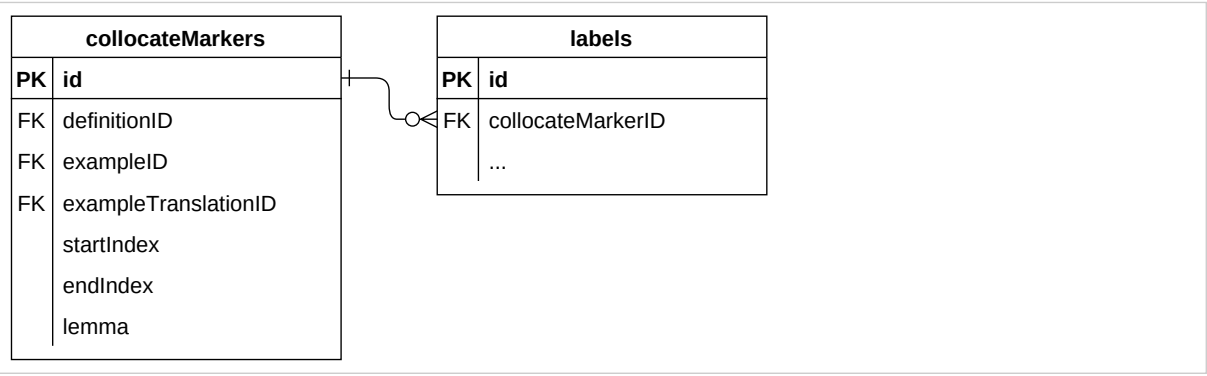
Example 148. JSON

```
{
    "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



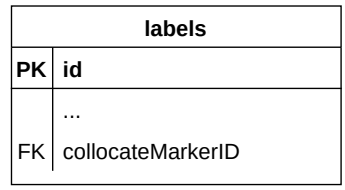
7.9 Extensions to `label`

Extends the `label` object type from the Core.

Can additionally be a child of

- `collocateMarker`

Example 151. Relational database



8 DMLex Etymology Module

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

Example: Section 9.24, “Modelling etymology”.

8.1 Extensions to `entry`

Extends the `entry` object type from the Core.

Additional contents

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

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 ... .
```

8.2 `etymology`

Represents a chain of historical derivations of a word.

Child of

- `entry`

Contents

- `description` OPTIONAL (zero or one). 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.

- `origin` OPTIONAL (zero, one or more).
- `listingOrder` REQUIRED (exactly one). Number. The position of this etymology among other etymologiers of the same entry. Can be implicit from the serialization.

Example 155. XML

```
<etymology>
  <description>...</description>
  <origin>...</origin>
  <origin>...</origin>
  <origin>...</origin>
  ...
</etymology>
```

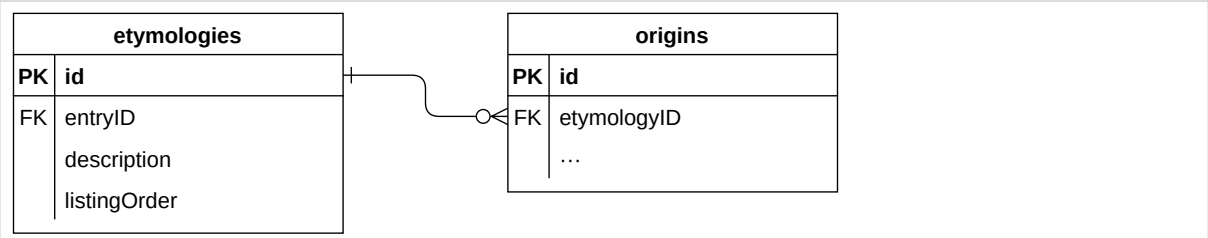
Example 156. JSON

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

Example 157. RDF

```
<entry> dmlex:etymology [
  dmlex:description "..." ;
  dmlex:origin ... ;
  dmlex:listingOrder: ...] .
```

Example 158. Relational database



8.3 `origin`

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

Child of

- `etymology`

Contents

- `when` OPTIONAL (zero or one). 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 occured at this stage of the headword's etymological history such as derivation, cognate, borrowing. The values can be explained and constrained using the `originType` object type.
- `note` OPTIONAL (zero or one). Any additional information about this stage of the headword's etymological history.
- `etymon` REQUIRED (one or more).
- `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

```
<origin when="..." type="...">
  <note>...</note>
  <etymon>...</etymon>
  <etymon>...</etymon>
  ...
</etymon>
```

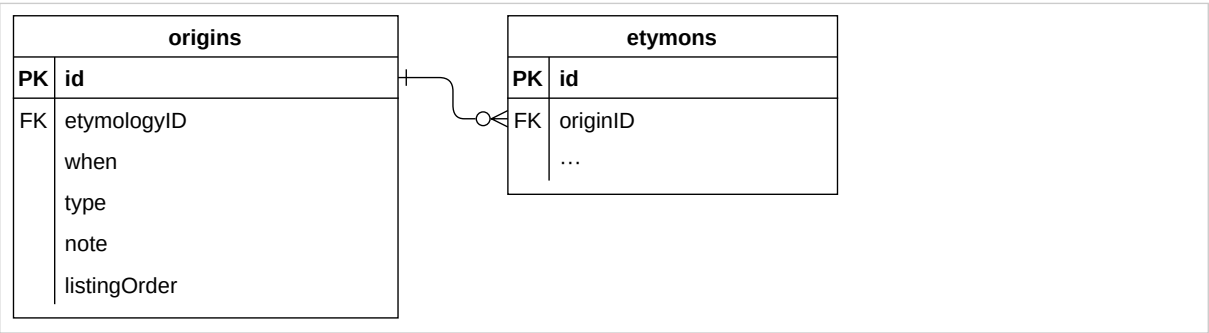
Example 160. JSON

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

Example 161. RDF

```
<origin>
  dmlex:when "..." ;
  dmlex:type "..." ;
  dmlex:note "..." ;
  dmlex:etymon ... ;
  dmlex:listingOrder 0 .
```

Example 162. Relational database



8.4 etymon

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

Child of

- origin

Contents

- language REQUIRED (exactly one). An IETF tag. The tags can be explained and constrained using the etymonLanguage object type.
- text REQUIRED (exactly one). 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

```
<etymon language="..." reconstructed="true">
  <text>...</text>
  <partOfSpeech tag="..."/>
  <translation>...</translation>
</etymon>
```

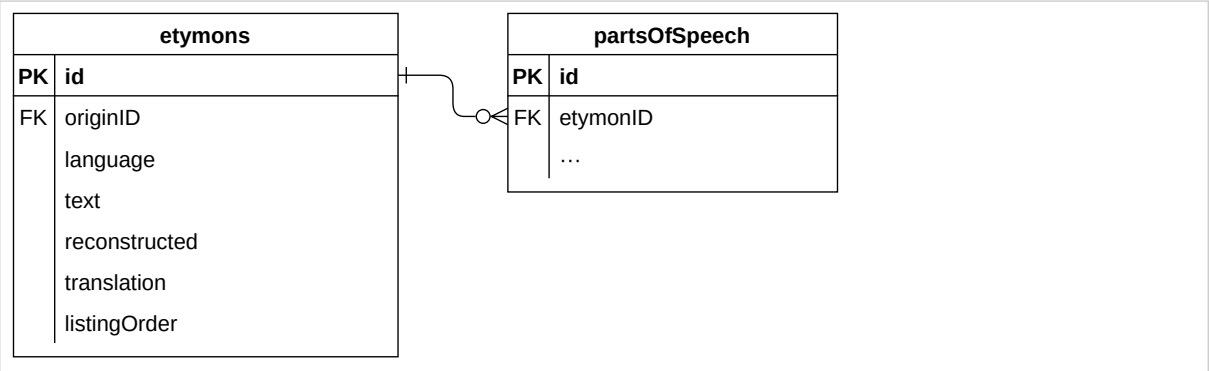
Example 164. JSON

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

Example 165. RDF

```
<etymon>
  dmlex:language "..." ;
  dmlex:text "..." ;
  dmlex:reconstructed true ;
  dmlex:partOfSpeech "..." ;
  dmlex:translation ... ;
  dmlex:listingOrder 0 .
```

Example 166. Relational database



8.5 Extensions to `lexicographicResource`

Extends the `lexicographicResource` object type from the Core.

Additional contents

- `etymonLanguage` OPTIONAL (zero, one or more)
- `originType` OPTIONAL (zero, one or more)

Example 167. XML

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

Example 168. JSON

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

Example 169. RDF

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

8.6 `originType`

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

Child of

- `lexicographicResource`

Contents

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

Example 170. XML

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

Example 171. JSON

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

Example 172. RDF

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

Example 173. Relational database

originTypes	
PK	**id**
FK	lexicographicResourceID
	type
	description

8.7 `etymonLanguage`

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

Child of

- `lexicographicResource`

Contents

- `langCode` REQUIRED (exactly one). 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

8.8 Extensions to `partOfSpeech`

Extends the `partOfSpeech` object type from the Core.

Can additionally be a child of

- `etymon`

Example 178. Relational database

partsOfSpeech	
PK	**id**
	...
FK	etymonID

9 Examples

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.

9.1 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 a entry into senses, how 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 179. Pseudocode

```
- lexicographicResource
    - 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 180. XML

```
<lexicographicResource>
    <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 181. JSON

```
{
    "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 182. 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." ] .
```

9.2 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 183. Pseudocode

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

Example 184. 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 185. JSON

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

9.3 Pronunciation given as transcription

Example 186. Pseudocode

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

Example 187. XML

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

Example 188. JSON

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

9.4 Pronunciation given as a sound file

Example 189. Pseudocode

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

Example 190. XML

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

Example 191. JSON

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

9.5 Pronunciation given both ways

Example 192. Pseudocode

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

Example 193. XML

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

Example 194. JSON

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

9.6 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 195. Pseudocode

```
- lexicographicResource (language: 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 196. XML

```
<lexicographicResource language="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 197. JSON

```
{
    "language": "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"]
    }]
}
```

9.7 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 198. 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 199. 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 200. 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"
        ]
    }]
}
```

9.8 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 201. Pseudocode

```
- lexicographicResource (title: My German-English Dictionary,  language: de)
    - translationLanguage (langCode: en)
```

Example 202. XML

```
<lexicographicResource language="de">
    <title>My German-English Dictionary</title>
    <translationLanguage langCode="en"/>
    ...
</lexicographicResource>
```

Example 203. JSON

```
{
    "title": "My German-English Dictionary",
    "language": "de",
    "translationLanguages": ["en"],
    ...
}
```

9.9 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 204. Pseudocode

```
- lexicographicResource (title: My Irish-Multilingual Dictionary, language: ga)
    - translationLanguage (langCode: en)
    - translationLanguage (langCode: de)
    - translationLanguage (langCode: cs)
```

Example 205. XML

```
<lexicographicResource language="ga">
    <title>My Irish-Multilingual Dictionary</title>
    <translationLanguage langCode="en"/>
    <translationLanguage langCode="de"/>
    <translationLanguage langCode="cs"/>
    ...
</lexicographicResource>
```

Example 206. JSON

```
{
    "title": "My Irish-Multilingual Dictionary",
    "language": "ga",
    "translationLanguages": ["en", "de", "cs"],
    ...
}
```

9.10 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 207. 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 208. 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 209. 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"]
        }]
    }]
}
```

9.11 How to use `headwordTranslation` in a multilingual lexicographic resource

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

Example 210. Pseudocode

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

Example 211. XML

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

Example 212. JSON

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

9.12 How to use `headwordExplanation`

Example 213. 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 214. 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 215. 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"
        }]
    }]
}
```

9.13 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 simultanously that "glasses" and "microscope" are both holonyms of "lens".

Example 216. Pseudocode

```
- lexicographicResource (language: 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: microscrope-1, role: whole)
        - member (memberID: lens-1, role: part)
    - relationType (type: meronomy, description: part-whole relationship)
        - memberType (role: whole, memberType: sense, min: 1, max: 1, action: navigate)
        - memberType (role: part, memberType: sense, min: 1, max: 1, action: navigate)
```

Example 217. XML

```
<lexicographicResource language="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="microscrope-1" role="whole"/>
        <member memberID="lens-1" role="part"/>
    </relation>
    <relationType type="meronomy">
        <description>part-whole relationship</description>
        <memberType role="whole" memberType="sense" min="1" max="1" action="navigate"/>
        <memberType role="part" memberType="sense" min="1" max="1" action="navigate"/>
    </relationType
</lexicographicResource>
```

```
{
    "language": "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",
            "memberType": "sense",
            "min": 1,
            "max": 1,
            "action": "navigate"
        }, {
            "role": "part",
            "memberType": "sense",
            "min": 1,
            "max": 1,
            "action": "navigate"
        }]
    }]
}
```

Example 219. Suggested rendering for human users

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

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

lens
  - curved glass that makes things seem bigger
    thinks that contain lens: glasses, microscope
```

9.14 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 220. Pseudocode

```
- lexicographicResource (language: 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 (memberType: sense, min: 2, max: 2, action: navigate)
```

Example 221. XML

```
<lexicographicResource language="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 memberType="sense" min="2" max="2" action="navigate"/>
    </relationType>
</lexicographicResource>
```

Example 222. JSON

```
{
    "language": "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": [{
            "memberType": "sense",
            "min": 2,
            "max": 2,
            "action": "navigate"
        }]
    }]
}
```

Example 223. Suggested rendering for human users

```
buy
  - get something by paying money for it
    opposite meaning: sell

sell
  - exchange something for money
    opposite meaning: buy
```

9.15 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 224. Pseudocode

```
- lexicographicResource (language: 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 (memberType: sense, min: 2, action: navigate)
```

Example 225. XML

```
<lexicographicResource language="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 memberType="sense" min="2" action="navigate"/>
    </relationType>
</lexicographicResource>
```

Example 226. JSON

```
{
    "language": "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": [{
            "memberType": "sense",
            "min": 2,
            "action": "navigate"
        }]
    }]
}
```

Example 227. 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
```

9.16 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 os only skeletal: its only purpose is to serve as the origin of a navigable link to the other entry.

Example 228. Pseudocode

```
- lexicographicResource (language: 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 (memberType: entry, min: 2, action: navigate)
```

Example 229. XML

```
<lexicographicResource language="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 memberType="entry" min="2" action="navigate"/>
    </relationType>
</lexicographicResource>
```

Example 230. JSON

```
{
    "language": "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": [{
            "memberType": "entry",
            "min": 2,
            "action": "navigate"
        }]
    }]
}
```

Example 231. 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
```

9.17 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 232. Pseudocode

```
- lexicographicResource (language: 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: subsensing)
        - member (memberID: colour-1, role: super)
        - member (memberID: colour-2, role: sub)
    - relation (type: subsensing)
        - member (memberID: colour-1, role: super)
        - member (memberID: colour-3, role: sub)
    - relationType (type: subsensing, scope: sameEntry)
        - memberType (role: super, memberType: sense, min: 1, max: 1 action: none)
        - memberRole (role: sub, memberType: sense, min: 1, max: 1, action: embed)
```

Example 233. XML

```
<lexicographicResource language="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" scope="sameEntry">
        <memberType role="super" memberType="sense" min="1" max="1" action="none"/>
        <memberType role="subs" memberType="sense" min="1" max="1" action="embed"/>
    </relationType>
</lexicographicResource>
```

Example 234. JSON

```
{
    "language": "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",
        "scope": "sameEntry",
        "memberTypes": [{
            "role": "super",
            "memberType": "sense",
            "min": 1,
            "max": 1,
            "action": "none"
        }, {
            "role": "sub",
            "memberType": "sense",
            "min": 1,
            "max": 1,
            "action": "embed"
        }]
    }]
}
```

Example 235. 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."
```

9.18 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 236. Pseudocode

```
- lexicographicResource (language: 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, scope: sameResource)
        - memberType (role: container, memberType: sense, min: 1, max: 1, action: naviga
        - memberType (role: subentry, memberType: entry, min: 1, max: 1, action: embed)
```

Example 237. XML

```
<lexicographicResource language="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" scope="sameResource">
        <memberType role="container" memberType="sense" min="1" max="1" action="navigat
        <memberType role="subentry" memberType="entry" min="1" max="1" action="embed"/>
    </relationType>
</lexicographicResource>
```

Example 238. JSON

```
{
    "language": "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",
        "scope": "sameResource",
        "memberTypes": [{
            "role": "container",
            "memberType": "sense",
            "min": 1,
            "max": 1,
            "action": "navigate"
        }, {
            "role": "subentry",
            "memberType": "entry",
            "min": 1,
            "max": 1,
            "action": "embed"
        }]
    }]
}
```

Example 239. 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
```

9.19 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 240. Pseudocode

```
- lexicographicResource (language: 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, scope: sameResource)
        - memberType (role: container, memberType: sense, min: 1, max: 1, action: naviga
        - memberRole (role: subentry, memberType: entry, min: 1, max: 1, action: embed)
```

Example 241. XML

```
<lexicographicResource language="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" scope="sameResource">
        <memberType role="container" memberType="sense" min="1" max="1" action="navigat
        <memberType role="subentry" memberType="entry" min="1" max="1" action="embed"/>
    </relationType>
</lexicographicResource>
```

Example 242. JSON

```
{
    "language": "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"
        }, {
            "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",
        "scope": "sameResource",
        "memberTypes": [{
            "role": "container",
            "memberType": "sense",
            "min": 1,
            "max": 1,
            "action": "navigate"
        }, {
            "role": "subentry",
            "memberType": "entry",
            "min": 1,
            "max": 1,
            "action": "embed"
        }]
    }]
}
```

Example 243. 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
```

9.20 Using `placeholderMarker`

Example 244. Pseudocode

```
- entry (id: continue-studies, headword: continue your studies, placeholderMarker: "you
    - sense ...
```

Example 245. XML

```
<entry id="continue-studies">
    <headword>
        continue <placeholderMarker>your</placeholderMarker> studies
    </headword>
    <sense.../>
</entry>
```

Example 246. JSON

```
{
  "id": "continue-studies",
  "headword": "continue your studies",
  "placeholderMarkers": [
     {"startIndex": 9, "endIndex": 13}
  ],
  "senses": [...]
}
```

9.21 Using `placeholderMarker` in a bilingual lexicographic resource

Example 247. Pseudocode

```
- entry (id: beat-up, headword: beat sb. up, placeholderMarker: "sb.")
    - sense (id: beat-up-1)
        headwordTranslation (text: jemanden verprügeln, placeholderMarker: "jemanden")
```

Example 248. 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 249. 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}
      ],
    }]
  }]
}
```

9.22 Using `headwordMarker`

Example 250. 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 251. 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 252. 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}
        ]
      }]
    }]
  }]
}
```

9.23 Using `collocateMarker`

Example 253. 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 254. 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 255. 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"}
        ],
      }]
    }]
  }]
}
```

9.24 Modelling etymology

Example 256. Pseudocode

```
- entry (id: cat-n, headword: cat)
    - sense ...
    - etymology
        - origin
            - etymon (language: enm, text: catte)
        - origin
            - etymon (language: ang, text: catt, translation: male cat)
            - etymon (language: ang, text: catte, translation: female cat)
        - origin
            - etymon (language: gem-x-proto, text: kattuz, reconstructed: true)
```

Example 257. XML

```
<entry id="cat-n">
    <headword>cat</headword>
    <sense>...</sense>
    <etymology>
        <origin>
            <etymon language="enm">
                <text>catte</text>
            </etymon>
        </origin>
        <origin>
            <etymon language="ang">
                <text>catt</text>
                <translation>male cat</translation>
            </etymon>
            <etymon language="ang">
                <text>catte</text>
                <translation>female cat</translation>
            </etymon>
        </origin>
        <origin>
            <etymon language="gem-x-proto" reconstructed="true">
                <text>kattuz</text>
            </etymon>
        </origin>
    </etymology>
</entry>
```

Example 258. JSON

```
{
  "id": "cat-n",
  "headword": "cat",
  "senses": [...],
  "etymology": {
    "origins" [{
      "etymons": [
        {"language": "enm", "text": "catte"}
      ]
    }, {
      "etymons": [
        {"language": "ang", "text": "catt", "translation": "male cat"},
        {"language": "ang", "text": "catte", "translation": "female cat"}
      ]
    }, {
      "etymons": [
        {"language": "gem-x-proto", "text": "kattuz", "reconstructed": true}
      ]
    }]
  }
}
```

10 DMLex XML serialization

10.1 Design principles

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

- 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.

10.2 Element: `<lexicographicResource>`

Implements the lexicographicResource object type.

Attributes

- `@title` OPTIONAL
- `@uri` OPTIONAL
- `@language` 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)

- <originType> OPTIONAL (zero or more)

10.3 Element: <entry>

Implements the entry object type.

Attributes

- @id REQUIRED
- @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)

10.4 Element: <partOfSpeech>

Implements the partOfSpeech object type.

Attributes

- @tag REQUIRED

10.5 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)

10.6 Element: <sense>

Implements the sense object type.

Attributes

- `@id` REQUIRED

Child elements

- `<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)

10.7 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.

10.8 Element: `<label>`

Implements the label object type.

Attributes

- `@tag` REQUIRED

10.9 Element: `<pronunciation>`

Implements the pronunciation object type.

Attributes

- `@soundFile` REQUIRED if `transcription` is present, OPTIONAL otherwise

Child elements

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

10.10 Element: `<transcription>`

Implements the transcription object type.

Attributes

- `@scheme` OPTIONAL

Text content

- REQUIRED, implements the `text` property.

10.11 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 (one or more)

10.12 Element: `<translationLanguage>`

Implements the translationLanguage object type from the Crosslingual Module.

Attributes

- `@langCode` REQUIRED

10.13 Element: `<headwordTranslation>`

Implements the headwordTranslation object type from the Crosslingual Module.

Attributes

- `@language` 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)

10.14 Element: <headwordExplanation>

Implements the headwordExplanation object type from the Crosslingual Module.

Attributes

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

Text content

- REQUIRED, implements the `text` property.

10.15 Element: <exampleTranslation>

Implements the exampleTranslation object type from the Crosslingual Module.

Attributes

- `@language` OPTIONAL
- `@soundFile` 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, `<headwordMarker>` elements and `<collocateMarker>` elements.
- `<label>` OPTIONAL (zero or more)

10.16 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)

10.17 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)

10.18 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)

10.19 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)

10.20 Element: `<labelTypeTag>`

Implements the labelTypeTag object type from the Controlled Values Module.

Attributes

- `@tag` REQUIRED

Child elements

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

10.21 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)

10.22 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)

10.23 Element: `<forLanguage>`

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

Attributes

- `@langCode` REQUIRED

10.24 Element: `<forPartOfSpeech>`

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

Attributes

- `@tag` REQUIRED

10.25 Element: `<sameAs>`

Implements the sameAs object type from the Controlled Values Module.

Attributes

- `@uri` REQUIRED

10.26 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)

10.27 Element: <member>

Implements the member object type from the Linking Module.

Attributes

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

10.28 Element: <relationType>

Implements the relationType object type from the Linking Module.

Attributes

- @type REQUIRED
- @scope REQUIRED

Child elements

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

10.29 Element: <memberType>

Implements the memberType object type from the Linking Module.

Attributes

- @role REQUIRED
- @memberType REQUIRED
- @min OPTIONAL
- @max OPTIONAL
- @action OPTIONAL

Child elements

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

10.30 Element: <placeholderMarker>

Implements the placeholderMarker object type from the Annotation module. Contains the text which it marks up.

10.31 Element: <headwordMarker>

Implements the headwordMarker object type from the Annotation module. Contains the text which it marks up.

10.32 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)

10.33 Element: <etymology>

Implements the etymology object type from the Etymology Module.

Child elements

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

10.34 Element: <origin>

Implements the origin object type from the Etymology Module.

Attributes

- `@when` OPTIONAL
- `@type` OPTIONAL

Child elements

- `<note>` OPTIONAL (zero or one)
- `<etymon>` REQUIRED (one or more)

10.35 Element: <etymon>

Implements the etymon object type from the Etymology Module.

Attributes

- `@language` REQUIRED
- `@reconstructed` OPTIONAL

Child elements

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

10.36 Element: <etymon>

Implements the etymon object type from the Etymology Module.

Attributes

- `@type` REQUIRED

Child elements

- `<description>` OPTIONAL (exactly one)

10.37 Element: <etymonLanguage>

Implements the etymonLanguage object type from the Etymology Module.

Attributes

- `@langCode` REQUIRED

Child elements

- `<displayName>` OPTIONAL (exactly one)

11 DMLex JSON serialization

11.1 Design principles

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

- 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.

11.2 Class: `lexicographicResource`

Implements the lexicographicResource object type.

Members

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

Members if implementing the Controlled Values Module

- `"tags"` OPTIONAL, array of zero or more `tag` objects

Members if implementing the Crosslingual Module

- `"translationLanguages"` REQUIRED, array of one or more `translationLanguage` objects

Members if implementing the Linking Module

- `"relations"` OPTIONAL, array of zero or more `relation` objects
- `"relationTypes"` OPTIONAL, array of zero or more `relationType` objects

11.3 Class: `entry`

Implements the entry data type.

Members

- `"id"` REQUIRED, string

- `"headword"` REQUIRED, string
- `"homographNumber"` OPTIONAL, string
- `"partsOfSpeech"` OPTIONAL, array of zero or more `partOfSpeech` objects
- `"labels"` OPTIONAL array of zero or more `label` objects
- `"pronunciations"` OPTIONAL, array of zero or more `pronunciation` objects
- `"inflectedForms"` OPTIONAL, array of zero or more `inflectedForm` objects
- `"senses"` OPTIONAL, array of zero or more `sense` objects

Members if implementing the Etymology Module

- `"etymologies"` OPTIONAL, array of zero or more `etymology` objects

12 DMLex RDF serialization

12.1 Design principles

The RDF serialization used in this document follows the following principles

- The `lexicographicResource` object is an individual of type `dmlex:LexicographicResource`
- All other objects are implemented as individuals
- Each other object type is associated with a property (with a lowercase intiial 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
- 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.

12.2 Interoperability with OntoLex-lemon

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.

13 DMLex relational database serialization

13.1 Design principles

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

- 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.

13.2 Table: `lexicographicResources`

Implements the lexicographicResource data type.

Columns

- `id int primary key`
- `title varchar(255)`
- `uri varchar(255)`
- `language varchar(10)`

Processing Requiremnts

- If the implementation contains only one lexicographic resource, then the lexicographicResource table is not REQUIRED to be unimplemented.

13.3 Table: `entries`

Implements the entry data type.

Columns

- `lexicographicResourceID int foreign key references lexicographicResource(id)`
- `id int`, primary key
- `headword varchar(255)`
- `homographNumber int`

Comment

If the implementation contains only one lexicographic resource, then the column `lexicographicResourceID` MAY remain unimplemented.

Appendix A References

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.

A.1 Normative references

[**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).

[**RFC 3552**] R. Escrola, B. Korver, *Guidelines for Writing RFC Text on Security Considerations*, *https://www.tools.ietf.org/rfc/rfc3552.txt* IETF (Internet Engineering Task Force) RFC 3552, July 2003.

[**EXAMPLE_ABBREV**] N. Surname, A. Surname, *Exampe Title*, *example.org/citetitleExample Citetitle*, Month dd, yyyy.

[**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.

[**NOTE-datetime**] M. Wolf, C. Wicksteed, *Date and Time Formats*, *http://www.w3.org/TR/NOTE-datetime* W3C Note, 15th September 1997.

[**NVDL**] International Standards Organization, *ISO/IEC 19757-4, Information Technology - Document Schema Definition Languages (DSDL) - Part 4: Namespace-based Validation Dispatching Language (NVDL)*, *http://standards.iso.org/ittf/PubliclyAvailableStandards/c038615_ISO_IEC_19757-4_2006(E).zip [http://standards.iso.org/ittf/PubliclyAvailableStandards/c038615_ISO_IEC_19757-4_2006(E).zip]* ISO, June 1, 2006.

[**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 [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.

[**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 http://docs.oasis-open.org/lexidma/dmlex/v1.0/wd01/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.

A.2 Informative references (Informative)

[**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 B Machine Readable Validation Artifacts (Informative)

CURRENTLY NO VALIDATION ARTIFACTS FORESEEN FOR THE OM.. JUST FOR SERIALIZATIONS

MAY LIST CONFORMANT ARTIFACTS FOR SPECIFIC SERILIZATIONS AT A LATER STAGE

Appendix C Security and privacy considerations

Note

OASIS strongly recommends that Technical Committees consider issues that might affect safety, security, privacy, and/or data protection in implementations of their work products and document these for implementers and adopters. For some purposes, you may find it required, e.g. if you apply for IANA registration.

While it may not be immediately obvious how your work product might make systems vulnerable to attack, most work products, because they involve communications between systems, message formats, or system settings, open potential channels for exploit. For example, IETF [RFC 3552] lists “eavesdropping, replay, message insertion, deletion, modification, and man-in-the-middle” as well as potential denial of service attacks as threats that must be considered and, if appropriate, addressed in IETF RFCs.

In addition to considering and describing foreseeable risks, this section should include guidance on how implementers and adopters can protect against these risks.

We encourage editors and TC members concerned with this subject to read Guidelines for Writing RFC Text on Security Considerations, IETF [RFC 3552], for more information.

Appendix D Specification Change Tracking (Informative)

This appendix will contain tracked changes after the csprd01 phase will have been reached.

Appendix E Acknowledgements (Informative)

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

- Erjavec, Tomaž - JSI
- Filip, David - TCD, ADAPT Centre
- Jakubí#ek, Miloš - Lexical Computing
- Kernerman, Ilan - K Dictionaries
- Kosem, Iztok - JSI
- Krek, Simon - JSI
- McCrae, John - National University of Ireland Galway
- M#chura, Michal - Masary kUniversity