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 that 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)
- `tag` 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 occur 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.

Example 1. XML

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

Example 2. JSON

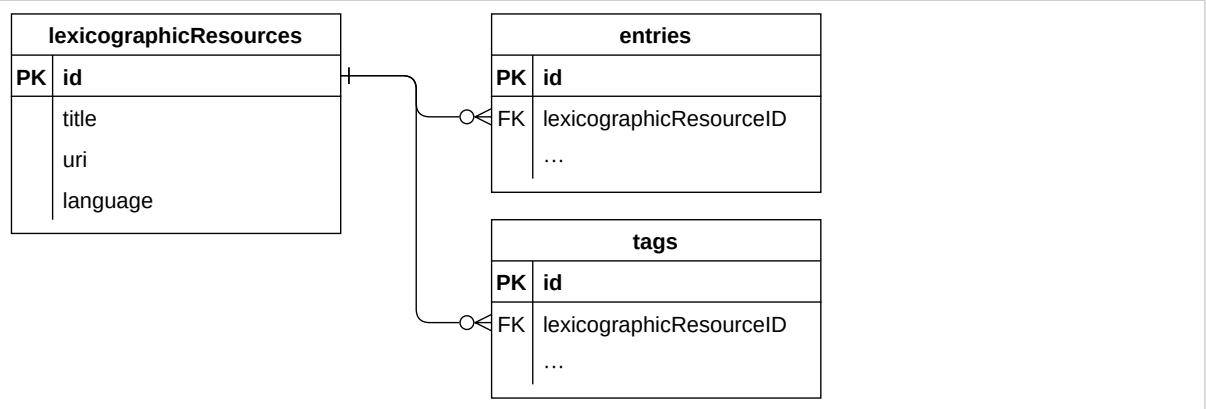
```
{
    "title": "...",
    "language": "...",
    "entries": [...],
    "tags": [...]
}
```

Example 3. RDF

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

<id> a dmlex:Lexicographic Resource ;
  dmlex:title "..." ;
  dmlex:uri "..." ;
  dmlex:language "..." ;
  dmlex:entry <entry1> , ... ;
  dmlex:tag ... .
```

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 for humans 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).

Comments

- 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.
- 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.
- DMLex Core does not have a concept of "subentry". To model subentries (ie. entries inside entries) in a lexicographic resource, you should use object types from the Linking Module for that.

Example 5. XML

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

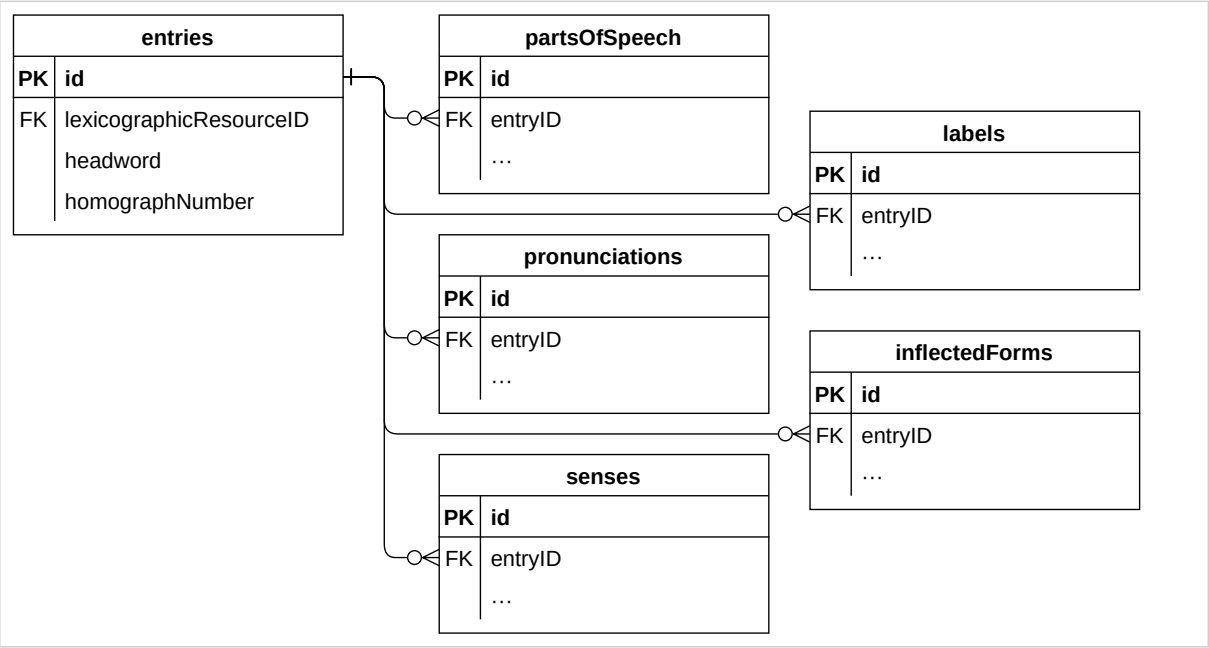
Example 6. JSON

```
{
    "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

- value 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 tag 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

- If you want to model other grammatical properties of the headword besides part of speech, such as gender (of nouns) or aspect (of verbs), the way to do that in DMLex is to conflate them to the part-of-speech label, for example `noun-masc` and `noun-fem`, or `v-perf` and `v-imperf`.

Example 9. XML

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

Example 10. JSON

```
"..."
```

Example 11. RDF

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

Example 12. Relational database

partsOfSpeech	
PK	**id**
FK	entryID
	value
	listingOrder

3.4 `inflectedForm`

Represents one (of possibly many) inflected forms of the headword. Example: Section 8.2, “How to use `inflectedForm`”.

Child of

- `entry`

Contents

- `inflectedTag` 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 `tag` 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 inflectedTag="...">
    <text>...</text>
    <label.../>
    <pronunciation.../>
</inflectedTag>
```

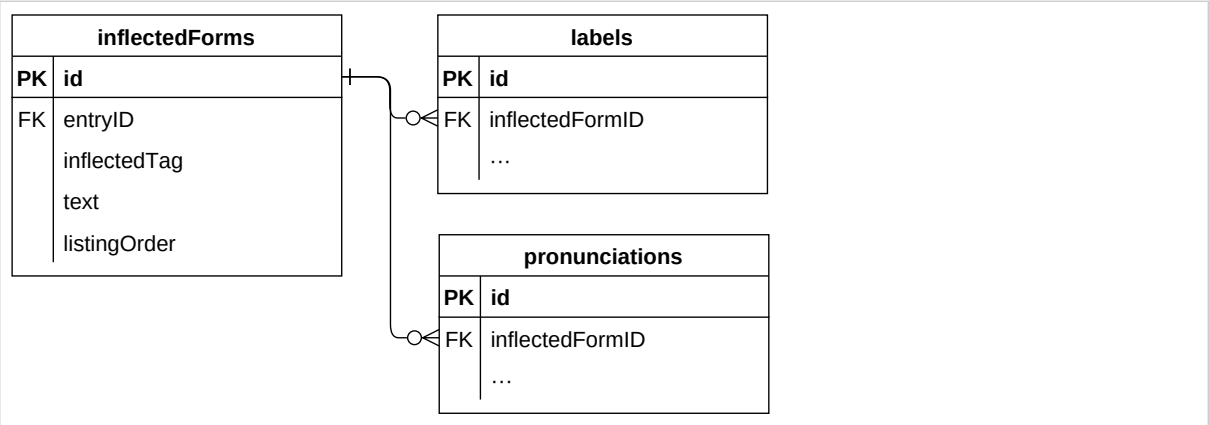
Example 14. JSON

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

Example 15. RDF

```
<entry> dmlex:inflectedForm [
  a dmlex:InflectedForm ;
  dmlex:inflectedTag "..." ;
  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 are capable of being 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

- An **entry** is a container for 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. If you have 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) then, in DMLex, you need to treat it as two entries (homographs), and you can use the Linking Module two link the two entries together and to make sure they are always shown together to human users.

Example 17. XML

```
<sense id="...">
    <indicator>...</indicator>
    <label.../>
    <definition.../>
    <example.../>
</sense>
```

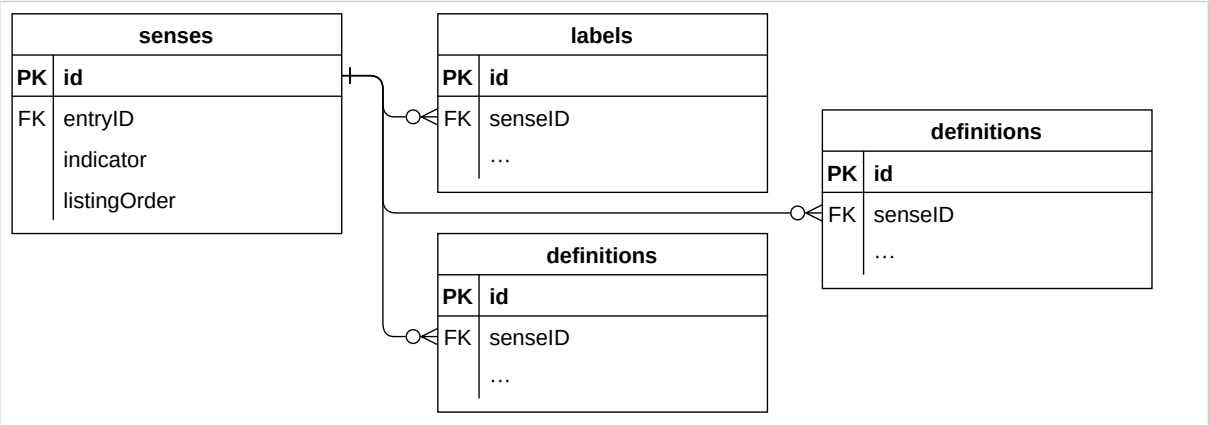
Example 18. JSON

```
{
    "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

- value 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 tag 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

```
<entry> dmlex:definition [
  a dmlex:Definition ;
  rdf:value "..." ;
  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

- `value` 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 `tag` 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 value="..."/>
```

Example 26. JSON

```
"..."
```

Example 27. RDF

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

Example 28. Relational database

labels	
PK	**id**
FK	entryID
FK	senseID
FK	inflectedFormID
FK	pronunciationID
FK	exampleID
	value
	listingOrder

3.8 pronunciation

Represents the pronunciation of its parent. Examples: Section 8.3, “Pronunciation given as transcription”, Section 8.4, “Pronunciation given as a sound file”, Section 8.5, “Pronunciation given both ways”.

Child of

- `entry`
- `inflectedForm`

Contents

- `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>
```

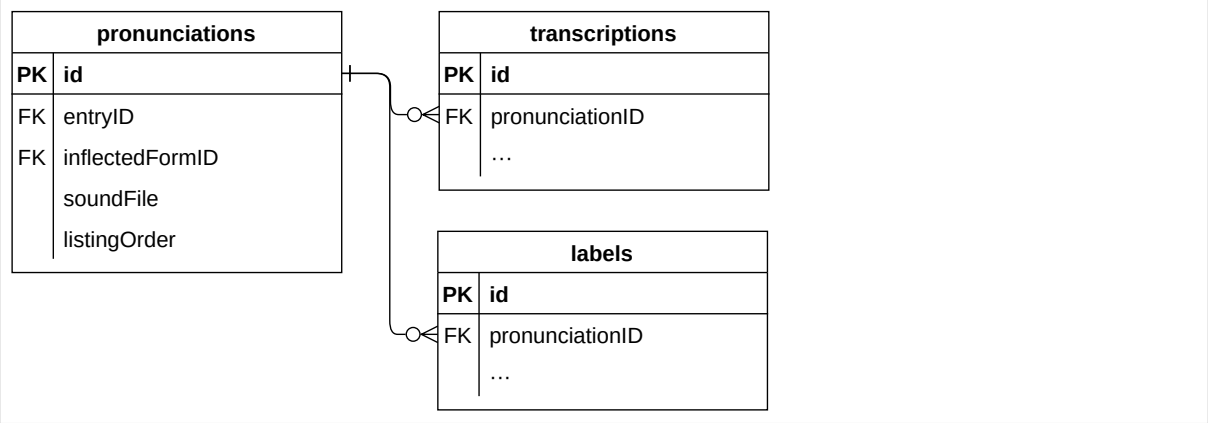
Example 30. JSON

```
{
    "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.
- `listingOrder` REQUIRED (exactly one). Number. The position of this transcription object among transcriptions of the same pronunciations. This can be implicit from the serialization.

Example 33. XML

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

Example 36. Relational database

transcriptions	
PK	**id**
FK	pronunciationID
	text
	scheme
	listingOrder

3.10 example

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

Child of

- sense

Contents

- `text` REQUIRED (exactly one). Non-empty string. The example itself.
- `sourceIdentity` OPTIONAL (zero or one). An abbreviation, a code or some other string of text which identifies the source. The `tag` object type can be used to explain the meaning of the source identifiers, to constrain which source identifiers are allowed to occur in the lexicographic resource, and to map them onto external inventories and ontologies.
- `sourceElaboration` OPTIONAL (zero or one). Non-empty string. A free-form statement about the source of the example. If `source` is present, then `sourceElaboration` can be used for information where in the source the example can be found: page number, chapter and so on. If `sourceIdentity` is absent then `sourceElaboration` can be used to fully name the source.
- `label` OPTIONAL (zero, 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>
```

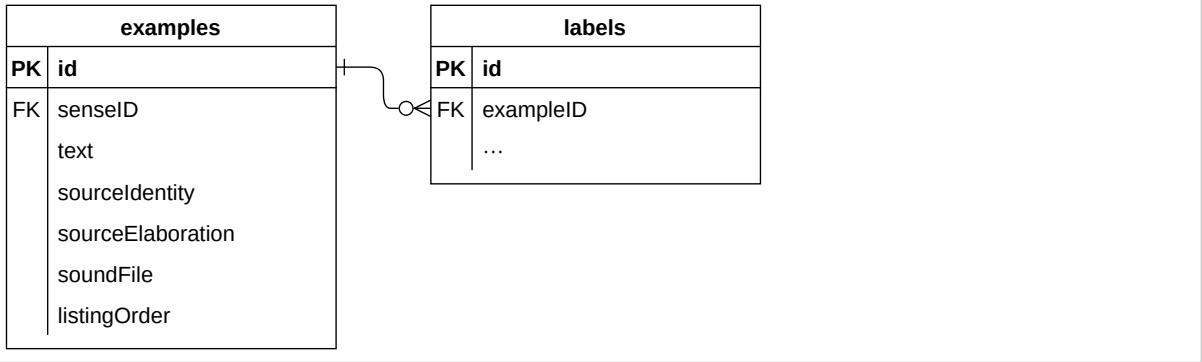
Example 38. JSON

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

Example 39. RDF

```
<entry> dmlex:example [
  a dmlex:Example ;
  rdf:value "..." ;
  dmlex:sourceIdentity "..." ;
  dmlex:sourceElaboration "..." ;
  dmlex:label ... ;
  dmlex:soundFile <...> ;
  dmlex:listingOrder 1 ] .
```

Example 40. Relational database



3.11 `tag`

Represents one (of many) possible values for `value` of `partOfSpeech`, `inflectedTag` of `inflectedForm`, `definitionType` of `definition`, `value` of `label` or `sourceIndentity` of `example`. Example: Section 8.6, “How to use `tag`”.

Child of

- `lexicographicResource`

Contents

- `value` 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.

- `target` OPTIONAL (zero, one or more). Indicates how the value of this tag can be used in this lexicographic resource. One of:
  - `partOfSpeech` The tag value can be used in `value` of `partOfSpeech`.
  - `inflectedTag` The tag value can be used in `inflectedTag` of `inflectedForm`.
  - `definitionType` The tag value can be used in `definitionType` of `definition`.
  - `label` The tag value can be used in `value` of `label`.
  - `sourceIndentity` The tag value can be used in `sourceIdentity` of `example`.

  If omitted, then all.
- `partOfSpeechConstraint` OPTIONAL (zero, one or more). If present, says that this tag is only intended to be used inside entries that are labelled with this part of speech. This can be used to constrain that, for example, only nouns and adjectives can have plurals but other parts of speech cannot.
- `sameAs` OPTIONAL (zero, one or more).

Comments

- In a lexicographic resource, `tag` objects can used to define controlled vocabularies (inventories of part-of-speech tags, labels etc.) and constrains on where they are expected to be used (e.g. part-of-speech tags are expected to be used as `value` of `partOfSpeech` objects).

  Enforcing these constraints in an implementation of DMLex, for example as business rules in a dictionary-writing system, is OPTIONAL: an implementation of DMLex is compliant regardless of whether it enforces the constraints defined by `tag` objects or not.

Example 41. XML

```
<tag value="...">
    <description>...</description>
    <target value="..."/>
    <partOfSpeechConstraint value="..."/>
    <sameAs.../>
</tag>
```

Example 42. JSON

```
{
    "value": "...",
    "description": "...",
    "targets": ["..."],
    "partOfSpeechConstraints": ["..."],
    "sameAs": [...]
}
```

Example 43. RDF

```
<entry> dmlex:tag [
  a dmlex:Tag ;
  dmlex:description "..." ;
  dmlex:target "..." ;
  dmlex:partOfSpeechConstraint "..." ;
  dmlex:sameAs ... ] .
```

Example 44. Relational database

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

3.12 `sameAs`

Represents the fact that the parent object is equivalent to an item available from an external authority. Example: Section 8.7, “Mapping `tag` to external inventories”.

Child of

- `tag`

Contents

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

Example 45. XML

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

Example 46. JSON

```
"..."
```

Example 47. Relational database

sameAs	
PK	**id**
FK	tagID
	uri

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 headword language 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 48. XML

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

Example 49. JSON

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

4.2 `translationLanguage`

Represents one of the languages in which translations are given in this lexicographic resource. Examples: Section 8.8, “Defining a bilingual lexicographic resource”, Section 8.9, “Defining a multilingual lexicographic resource”.

Child of

- `lexicographicResource`

Contents

- `value` 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 50. XML

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

Example 51. JSON

```
"..."
```

Example 52. 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 53. XML

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

Example 54. JSON

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

4.4 headwordTranslation

Represents one of possibly multiple translations of a headword. Examples: Section 8.10, “How to use headwordTranslation in a bilingual lexicographic resource”, Section 8.11, “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 thelexicographic resource, REQUIRED (one or more) 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).

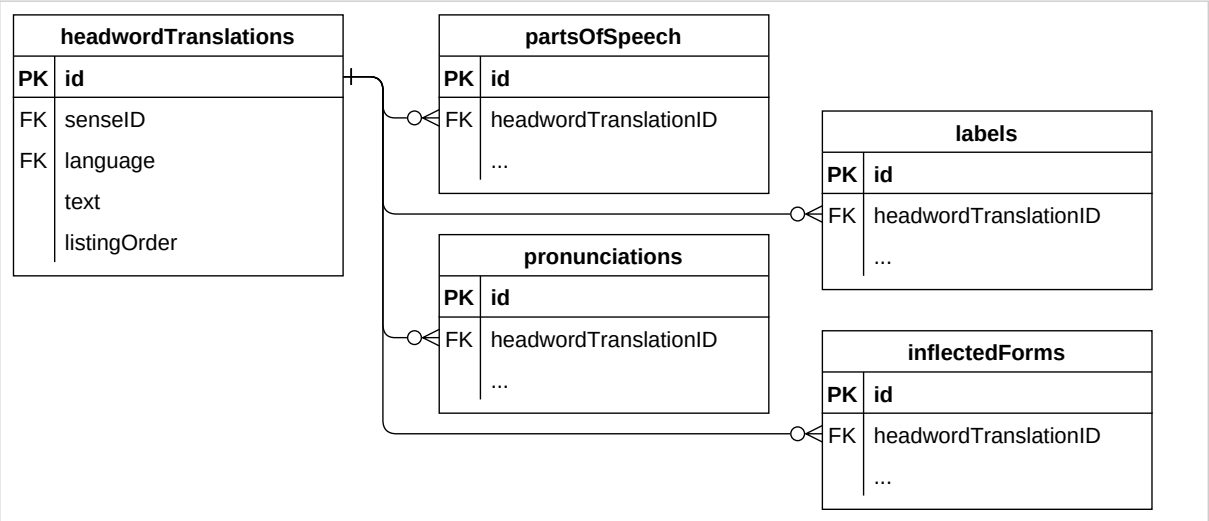
Example 55. XML

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

Example 56. JSON

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

Example 57. Relational database



4.5 headwordExplanation

Represents a statement in the target language which explains (but does not translate) the meaning of the headword. Example: Section 8.12, “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 thelexicographic resource, REQUIRED (one or more) 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 58. XML

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

Example 59. JSON

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

Example 60. 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 61. XML

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

Example 62. JSON

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

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 (one or more) 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.
- `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 63. XML

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

Example 64. JSON

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

Example 65. Relational database

exampleTranslations	
PK	**id**
FK	exampleID
FK	language
	text
	soundFile
	listingOrder

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
	value
	listingOrder

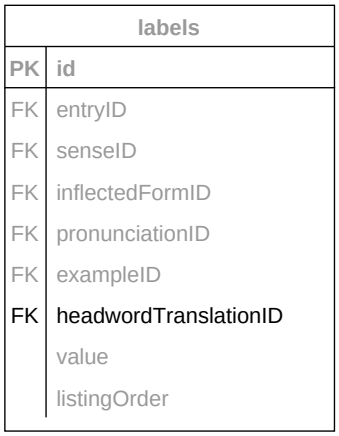
4.9 Extensions to `label`

Extends the `label` object type from the Core.

Can additionally be a child of

- `headwordTranslation`

Example 67. Relational database



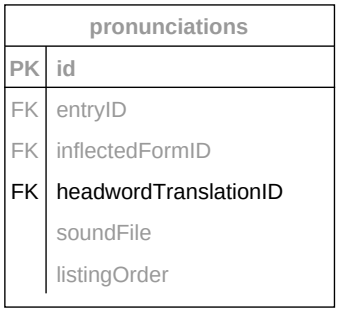
4.10 Extensions to `pronunciation`

Extends the `pronunciation` object type from the Core.

Can additionally be a child of

- `headwordTranslation`

Example 68. Relational database



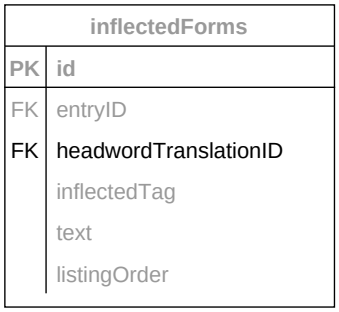
4.11 Extensions to `inflectedForm`

Extends the `inflectedForm` object type from the Core.

Can additionally be a child of

- `headwordTranslation`

Example 69. Relational database



4.12 Extensions to `tag`

Extends the `tag` object type from the Core.

Additional contents

- `sideConstraint` OPTIONAL (zero or one). If present, indicates whether this tag is intended to be used on the headword side or on the translation side of the lexicographic resource. One of:
  - `headwordSide` This tag is supposed to be used on the headword side of the lexicographic resource: anywhere except inside a `headwordTranslation` object.
  - `translationSide` This tag is supposed to be used on the translation side of the lexicographic resource: inside a `headwordTranslation` object only.
- `translationLanguageConstraint` 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.

Redefinition of `partOfSpeechConstraint`:

- 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.

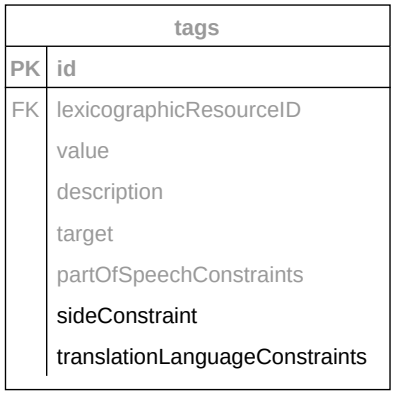
Example 70. XML

```
<tag ... sideConstraint="...">
    ...
    <translationLanguageConstraint langCode="..."/>
</tag>
```

Example 71. JSON

```
{
    ...,
    "sideConstraint": "...",
    "translationLanguageConstraint": ["..."]
}
```

Example 72. Relational database



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

5 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.

Relations themselves can be members of other relations.

Examples: Section 8.13, “Modelling parts and wholes”, Section 8.14, “Modelling antonyms”, Section 8.15, “Modelling synonyms”, Section 8.16, “Modelling variants”, Section 8.17, “Modelling subsenses”, Section 8.18, “Modelling subentries (at subsense level)”, Section 8.19, “Modelling subentries (at sense level)”.

5.1 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 73. XML

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

Example 74. JSON

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

5.2 `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. Is an optional human-readable explanation of this relation.
- `member` REQUIRED (two or more).

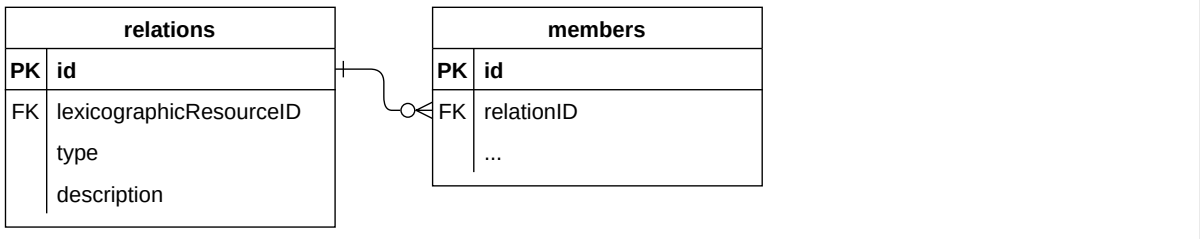
Example 75. XML

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

Example 76. JSON

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

Example 77. Relational database



5.3 `member`

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

Child of

- `relation`

Contents

- `idref` 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 `memberRole` 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. It should be respected when showing members of the relation to human users. This can be implicit from the serialization.
- `reverseListingOrder` REQUIRED (exactly one). Number. The position of this relation among other relations this member is involved in. It should be respected when showing the relations of this member to a human user. This can be implicit from the serialization.

Example 78. XML

```
<member idref="..." role="..." reverseListingOrder="..."/>
```

Example 79. JSON

```
{
    "idref": "...",
    "role": "...",
    "reverseListingOrder": "..."
}
```

Example 80. Relational database

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

5.4 `relationType`

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

Child of

- `lexicographicResource`

Contents

- `value` 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

- `memberRole` OPTIONAL (zero or more).
- `sameAs` OPTIONAL (zero or more).

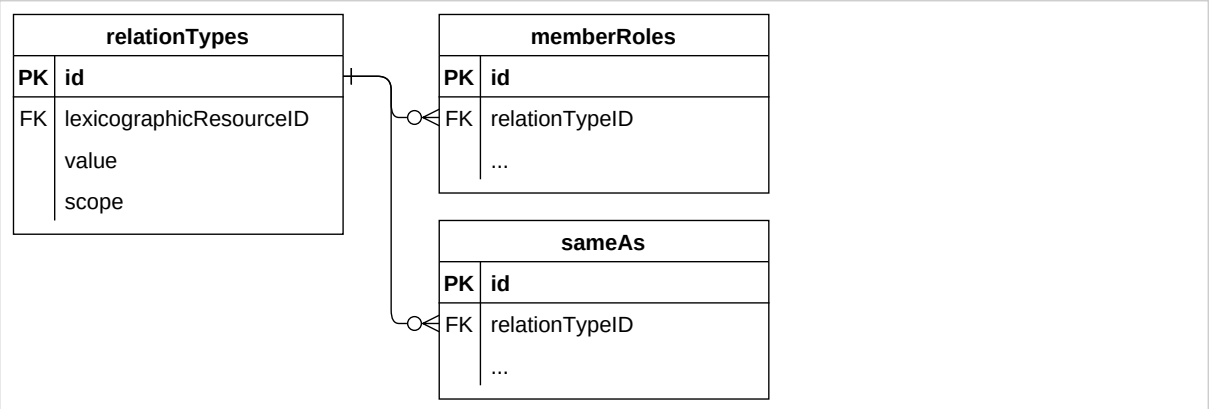
Example 81. XML

```
<relationType value="..." scope="...">
    <description>...</description>
    <memberRole.../>
    <sameAs.../>
</relationType>
```

Example 82. JSON

```
{
    "value": "...",
    "scope": "...",
    "memberRoles": [...],
    "sameAs": ["..."]
}
```

Example 83. Relational database



5.5 `memberRole`

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

Child of

- `relationType`

Contents

- `value` 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 restrictions 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`.
  - `itemMarker`: the object that has this role must be an `itemMarker` (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 the relation between entries and subentries, or senses and subsenses.
  - `navigate`: Members that have this role should not be shown in their entirety, but a navigable (e.g. clickable) link should be provided. This is suitable for the relation between synonyms, for example.
  - `none`: Members that have this role should not shown.
- `sameAs` OPTIONAL (zero or more).

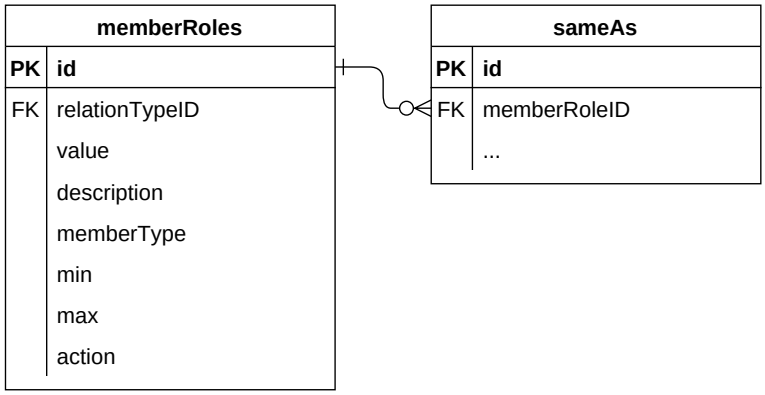
Example 84. XML

```
<memberRole value="..." memberType="..." min="..." max="..." action="...">
    <description></description>
    <sameAs.../>
</memberRole>
```

Example 85. JSON

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

Example 86. Relational database



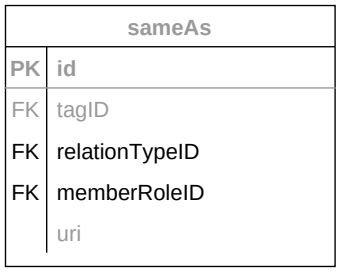
5.6 Extensions to sameAs

Extends the `sameAs` object type from the Core.

Can additionally be a child of

- `relationType`
- `memberRole`

Example 87. Relational database



6 DMLex Inline Markup 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 inline markup in an implementation of the DMLex Inline Markup module, whether in-place (as XML) or as stand-off markup (for example through start and end indexes).

6.1 Extensions to `headword`

Extends the `headword` object type from the Core.

Additional contents

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

Example 88. XML

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

Example 89. JSON

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

6.2 Extensions to `headwordTranslation`

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

Additional contents

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

Example 90. XML

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

Example 91. JSON

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

6.3 `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 8.20, “Using `placeholderMarker`”, Section 8.21, “Using `placeholderMarker` in a bilingual lexicographic resource”.

Child of

- `headword`
- `headwordTranslation`

Example 92. XML

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

Example 93. JSON

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

Example 94. Relational databases

placeholderMarkers	
PK	**id**
FK	entryID
	startIndex
	endIndex

6.4 Extensions to `definition`

Extends the `definition` object type from the Core.

Additional contents

- `headwordMarker` OPTIONAL (zero, one or more)
- `itemMarker` OPTIONAL (zero, one or more)

Example 95. XML

```
<definition...>
    ...
    <headwordMarker>...</headwordMarker>
    ...
    <itemMarker...>...</itemMarker>
    ...
</definition>
```

Example 96. JSON

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

6.5 Extensions to `example`

Extends the `example` object type from the Core.

Additional contents

- `headwordMarker` OPTIONAL (zero, one or more)
- `itemMarker` OPTIONAL (zero, one or more)

Example 97. XML

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

Example 98. JSON

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

6.6 Extensions to `exampleTranslation`

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

Additional contents

- `headwordMarker` OPTIONAL (zero, one or more)
- `itemMarker` OPTIONAL (zero, one or more)

Example 99. XML

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

Example 100. JSON

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

6.7 `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 8.22, “Using `headwordMarker`”.

Child of

- `definition`

- `example`
- `exampleTranslation`

Example 101. XML

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

Example 102. JSON

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

Example 103. Relational databases

headwordMarkers	
PK	**id**
FK	definitionID
FK	exampleID
FK	exampleTranslationID
	startIndex
	endIndex

6.8 `itemMarker`

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 8.23, “Using `itemMarker`”.

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.)

- `itemRole` OPTIONAL (zero, one or more). Non-empty string. Can be used to communicate facts about the role 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 `tag` object type can be used to explain and/or constrain the item types that are allowed to appear in the lexicographic resource.

Example 104. XML

```
<itemMarker id="..." lemma="...">
    ...
    <itemRole value="..."/>
</itemMarker>
```

Example 105. JSON

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

Example 106. SQL