DMLex: a data model for lexicography (draft)

OASIS LEXIDMA

Contents

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 if a monolingual lexicographic resource.
- **DMLex Crosslingual Module** extends DMLex Core to model bilingual and multilingual lexicographic resources.
- **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 Inline Markup 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.

1.2 Schema formalism

DMLex models a lexicographic resource as a **hierarchical list of objects**. Each object has a name, a value and an optional list of child objects, each of which can in turn also have a **name**, a **value** and an optional list of child objects.

The data model is defined in this standard through the means of a formalism which defines, for each object: (1) what its name is, (2) what its value is supposed to be (from a list of predefined primitive types) and (3) which child objects it may contain, with what arities.

The arities of child objects are indicated with the following codes:

- `(0..1)` zero or one
- `(0..n)` zero or one or more

- `(1..1)` exactly one
- `(1..n)` one or more
- `(2..n)` two or more

The primitive types of the values of objects are given with the following codes:

- `<string>` a non-empty string
- `<stringOrEmpty>` a string which may be empty
- `<number>` a positive integer number
- `<id>` an alphanumeric identifier
- `<idref>` a reference to something through its alphanumeric identifier
- `<uri>` a URI
- `<langCode>` an IETF language code
- `<empty>` nothing: the object serves only as a container for child objects
- `<symbol>` one of a specified finite number of values

When the primitive type of a child object is absent, this means that the schema for objects of that name is defined elsewhere in the code.

1.3 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, JSON and SQL. Examples of what such implementations look like with real-world data are given in Section 6.

1.3.1 Implementing DMLex in XML

The XML implementation 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.

1.3.2 Implementing DMLex in JSON

The XML implementation 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.

1.3.3 Implementing DMLex as a relational database

The SQL implementation 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.

2 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.

2.1 `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.

```
lexicographicResource: <id>
    title: (0..1) <string>
    uri: (0..1) <uri>
    language: (1..1) <langCode>
    entry: (0..n)
    tag: (0..n)
```

XML

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

JSON

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

SQL

```
create table lexicographicResources (
    id int primary key,
    title varchar(255),
    language varchar(10)
)
```

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 two object types that can optionally occur as children of a `lexicographicResource`, especially `tag`, are for lists of look-up values such as part-of-speech labels.

2.2 `entry`

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

```
entry: <id>
    headword: (1..1) <string>
    homographNumber: (0..1) <number>
    partOfSpeech: (0..n)
    label: (0..n)
    pronunciation: (0..n)
    inflectedForm: (0..n)
    sense: (0..n)
```

XML

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

JSON

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

SQL

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

Comments

- `headword` contains entry's headword. 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'. If you wish to have subentries (ie. entries inside entries) in your lexicographic resource you can use types from the Linking Module for that.

2.3 `partOfSpeech`

Represents a part-of-speech label.

```
partOfSpeech: <string>
    listingOrder: (1..1) <number>
```

XML

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

JSON

```
"..."
```

SQL

```
create table partsOfSpeech (
    entryID int foreign key references entries(id),
    value varchar(10),
    listingOrder int,
    id int primary key
)
```

Comments

- `partOfSpeech` is 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. You can use the `tag` datatype to explain the meaning of the part-of-speech tags, to constrain which part-of-speech tags are allowed to occur in your lexicographic resource, and to map them onto external inventories and ontologies.
- 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`.
- `listingOrder` is 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.

2.4 `sense`

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

```
sense: <id>
    listingOrder: (1..1) <number>
    indicator: (0..1) <string>
    label: (0..n)
```

```
    definition: (0..n)
    example: (0..n)
```

XML

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

JSON

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

SQL

```sql
create table senses (
    entryID int foreign key references entries(id),
    id int primary key,
    indicator nvarchar(50),
    listingOrder int
)
```

Comments

- `listingOrder` represents the position of this sense among other senses of the same entry. Can be implicit from the serialization.
- `indicator` is 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.

- `definition` is 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.

Note

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.

2.5 `definition`

Represents one of possibly several definitions of a sense.

```
definition: <string>
    definitionType: (0..1) <string>
    listingOrder: (1..1) <number>
```

XML

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

JSON

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

SQL

```
create table definitions (
    senseID int foreign key references sense(id),
    text nvarchar(255),
    definitionType nvarchar(10),
    listingOrder int,
    id int primary key
)
```

Comments

- If you have multiple definitions inside a single sense, you can use `definitionType` to indicate the difference between them, for example that they are intended for different audiences. Optionally, you can use the `tag` data type to constrain and/or explain the definition types that occur in your lexicographic resource.
- `listingOrder` is the position of this definition among other definitions of the same sense. This can be implicit from the serialization.

2.6 `inflectedForm`

Represents one (of possibly many) inflected forms of the headword. See Example 6.2.

```
inflectedForm: <string>
    inflectedTag: (0..1) <string>
    listingOrder: (1..1) <number>
    label: (0..n)
    pronunciation: (0..n)
```

XML

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

JSON

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

SQL

```
create table inflectedForms (
    entryID int foreign key references entries(id),
    inflectedTag varchar(10),
    text varchar(255),
    listingOrder int,
    id int primary key
)
```

Comments

- `inflectedTag` is 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. You can use the `tag` datatype to explain the meaning of the inflection tags, to constrain which inflection tags are allowed to occur in your lexicographic resource, and to map them onto external inventories and ontologies.
- The value of the `inflectedForm` object is the text of the inflected word itself.
- `listingOrder` is the position of this inflected form among other inflected forms of the same entry. This can be implicit from the serialization.
- 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.

2.7 `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).

```
label: <string>
    listingOrder: (1..1) <number>
```

XML

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

JSON

```
"..."
```

SQL

```
create table labels (
    entryID int foreign key references entries(id),
    senseID int foreign key references senses(id),
    inflectedFormID int foreign key references inflectedForms(id),
    pronunciationID int foreign key references pronunciations(id),
    exampleID int foreign key references examples(id),
    value varchar(10),
    listingOrder int,
    id int primary key
)
```

Comments

- The value of the label object is 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. You can use the `tag` datatype to explain the meaning of the label tags, to constrain which label tags are allowed to occur in your lexicographic resource, and to map them onto external inventories and ontologies.
- `listingOrder` is the position of this label among other labels of the same entry. This can be implicit from the serialization.
- 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 `inlectedForm`, 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).

2.8 `pronunciation`

Represents the pronunciation of its parent. See Examples 6.3, 6.4, 6.5.

```
pronunciation: <empty>
    soundFile: (0..1) <uri>
    transcription: (0..n)
    listingOrder: (1..1) <number>
    label: (0..n)
```

XML

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

JSON

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

SQL

```
create table pronunciations (
    entryID int foreign key references entries(id),
    soundFile varchar(255),
    listingOrder int,
    id int primary key
)
```

Comments

- `transcription` is the transcription of the pronuciation in some notation, such as IPA. If more than transcription is present in a single pronuncuation object, then they must be different transcriptions (in different schemes) of the same pronunciation, eg. one in IPA and one in SAMPA.
- `soundFile` is a pointer to a file containing a sound recording of the pronunciation.

- `listingOrder` is the position of this `pronunciation` object among other `pronunciation` objects of the same parent. This can be implicit from the serialization.

2.9 `transcription`

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

```
transcription: <string>
    scheme: (0..1) <langCode>
    listingOrder: (1..1) <number>
```

XML

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

JSON

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

SQL

```
create table transcriptions (
    pronunciationID int foreign key references pronunciation(id),
    text varchar(255),
    scheme varchar(10),
    listingOrder int,
    id int primary key
)
```

Comments

- `scheme` object 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` is the position of this `transcription` object among other `transcription` objects of the same pronunciation. This can be implicit from the serialization.

2.10 `example`

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

```
example: <string>
    sourceIdentity: (0..1) <string>
    sourceElaboration: (0..1) <string>
    label: (0..n)
    soundFile: (0..1) <uri>
    listingOrder: (1..1) <number>
```

XML

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

JSON

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

SQL

```
create table examples (
    senseID int foreign key references senses(id),
    text varchar(255),
    sourceIdentity varchar(50),
    sourceElaboration varchar(255),
    soundFile varchar(255),
```

```
    id int primary key
)
```

Comments

- `sourceIdentity` is an abbreviation, a code or some other string of text which identifies the source. You can use the `tag` datatype to explain the meaning of the source identifiers and to constrain which source identifiers are allowed to occur in your lexicographic resource.
- `sourceElaboration` is 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.
- `soundFile` is a pointer to a file containing a sound recording of the example.
- `listingOrder` is the position of this example among other examples in the same sense. This can be implicit from the serialization.

2.11 `tag`

Represents one (of many) possible values for `partOfSpeech`, `inflectedTag`, `label`, and `source`. See Example 6.6.

```
tag: <string>
    description: (0..1) <string>
    target: (0..n) <symbol>
    partOfSpeechConstraint: (0..n) <string>
    sameAs: (0..n)
```

XML

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

JSON

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

SQL

```
create table tags (
    lexicographicResourceID int foreign key references lexicographicResource(id),
    value varchar(10),
    description varchar(255),
    targets varchar(255), --comma-separated list
    partOfSpeechConstraints varchar(255), --comma-separated list
    id int primary key
)
```

Comments

- The value is an abbreviation, a code or some other string of text which identifies the source. If you want, you can design your implementation to enforce referential integrity between `tag` values on the one hand and `partOfSpeech`, `inflectedTag` etc. objects on the other hand. In other words, you can make it so that the tags you define in `tag` objects are the only values allowed for `partOfSpeech`, `inflectedTag` etc. However, doing this is optional in DMLex. An implementation of DMLex is compliant regardless of whether it enforces referential integrity on `tag` values.
- `description` is a human-readable description of what the tag means.
- `target` tells us where exactly the tag is expected to be used. If omitted, then all four. The possible values are:
    - `partOfSpeech`: as the value of a `partOfSpeech` object
    - `inflectedTag`: as the value of an `inflectedTag` object
    - `sourceIdentity`: as the value of a `sourceIdentity` object
    - `label`: as the value of a `label` object
    - `definitionType`: as the value of a `definitionType` object
    - `collocateRole`: as the value of a `collocateRole` object

- `partOfSpeechConstraint`, if present, says that this tag is only intended to be used inside entries that are labelled with this part of speech. You can us this to constrain that, for example, only nouns and adjectives can have plurals but other parts of speech cannot.
- `target` and `partOfSpeechConstraint` allow you to specify constraints on which tags are expected to appear where throughout the lexicographic resource. Enforcing these constraints in your implementation is optional.

2.12 `sameAs`

Represents the fact that the parent object is equivalent to an item available from an external authority. See Example 6.7.

```
sameAs: <uri>
```

XML

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

JSON

```
"..."
```

SQL

```
create table sameAs (
    tagID int foreign key references tags(id),
    uri varchar(255),
    id int primary key
)
```

Comments

- The value is the URI of an item in an external inventory.

3 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).

3.1 Extensions to `lexicographicResource`

Additional children:

```
lexicographicResource: ...
    translationLanguage: (1...n)
```

XML

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

JSON

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

SQL

No changes needed.

3.2 `translationLanguage`

Represents one of the languages in which translations are given in this lexicographic resource. See Examples 6.8, 6.9.

```
translationLanguage: <langCode>
    listingOrder: (1..1) <number>
```

XML

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

JSON

```
"..."
```

SQL

```
create table translationLanguage (
    lexicographicResourceID int foreign key references lexicographicResources(id),
    langCode varchar(10) primary key,
    listingOrder int,
)
```

Comments

- `listingOrder` 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.

3.3 Extensions to `sense`

Additional children:

```
sense: ...
    headwordExplanation: (0..n)
    headwordTranslation: (0..n)
```

XML

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

JSON

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

SQL

No changes needed.

3.4 `headwordTranslation`

Represents one of possibly multiple translations of a headword. See Examples 6.10, 6.11.

```
headwordTranslation: <string>
    language: (0..1) <langCode>
    listingOrder: (1..1) <number>
    partOfSpeech: (0..n) <string>
    label: (0..n)
    pronunciation: (0..n)
    inflectedForm: (0..n)
```

XML

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

JSON

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

SQL

```
create table headwordTranslations (
    senseID int foreign key references senses(id),
    language nvarchar(10) foreign key references translationLanguage(langCode),
    text nvarchar(255),
    listingOrder int,
    id int primary key
);
alter table partsOfSpeech (
    add headwordTranslationID int foreign key references headwordTranslations(id)
);
alter table labels (
    add headwordTranslationID int foreign key references headwordTranslations(id)
);
alter table pronunciations (
    add headwordTranslationID int foreign key references headwordTranslations(id)
);
alter table inflectedForms (
    add headwordTranslationID int foreign key references headwordTranslations(id)
)
```

Comments

- `language` indicates the language of this translation. You can use the `translationLanguage` datatype to explain the meaning of the language codes that appear here and/or to constrain which language codes are allowed.
- If ony one translation language exists in your lexicographic resource, then `language` can be left out.
- For more comments see comments under `headwordTranslation` in the Bilingual Module.

3.5 `headwordExplanation`

Represents a statement in the target language which explains (but does not translate) the meaning of the headword. See Example 6.12.

```
headwordExplanation: <string>
    language: (1..1) <langCode>
```

XML

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

JSON

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

SQL

```
create table headwordExplanations (
    senseID int foreign key references senses(id),
    language nvarchar(10) foreign key references translationLanguage(langCode),
    text nvarchar(255),
    id int primary key
)
```

Comments

- `language` indicates the language of this explanation. You can use the `translationLanguage` datatype to explain the meaning of the language codes that appear here and/or to constrain which language codes are allowed.
- If ony one translation language exists in your lexicographic resource, then `language` can be left out.
- It is assume that there will always be a maximum of one `headwordExplanation` per translation language.

3.6 Extensions to `example`

Additional children:

```
sense: ...
    exampleTranslation: (0..n)
```

XML

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

JSON

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

SQL

No changes needed.

3.7 `exampleTranslation`

Represents the translation of an example.

```
exampleTranslation: <string>
    language: (1..1) <langCode>
    soundFile: (0..1) <uri>
    listingOrder: (1..1) <number>
```

XML

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

JSON

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

SQL

```
create table exampleTranslations (
    exampleID int foreign key references examples(id),
    language varchar(10) foreign key references translationLanguage(langCode),
    text varchar(255),
    soundFile varchar(255),
    listingOrder int,
    id int primary key
);
alter table labels (
    add exampleTranslationID foreign key references exampleTranslations(id)
)
```

Comments

- `language` indicates the language of this translation. You can use the `translationLanguage` datatype to explain the meaning of the language codes that appear here and/or to constrain which language codes are allowed.
- If ony one translation language exists in your lexicographic resource, then `language` can be left out.
- For more comments see commens under `exampleTranslation` in the Bilingual Module.

3.8 Extensions to `tag`

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.

Additional child:

```
tag: ...
    translationLanguageConstraint: (0..n) <langCode>
```

XML

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

JSON

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

SQL

```
alter table tags (
    add translationLanguageConstraints varchar(255), --comma-separated list
)
```

Comments

- `translationLanguageConstraint`, if present, says that if this tag is being used inside a `headwordTranslation` or an `exampleTranslation`, then it is intended to be used only inside `headwordTranslation` and `exampleTranslation` objects labelled with this language.
- For a redefinition of `partOfSpeechConstraint`, see see `tag` in the Bilingual Module.

4 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.

See Examples ref{ex12}, 6.14, 6.15, 6.16, 6.17, 6.18, 6.19.

4.1 Extensions to `lexicographicResource`

Additional children:

```
lexicographicResource: ...
    relation: (0..n)
    relationType: (0..n)
```

XML

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

JSON

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

SQL

No changes needed.

4.2 `relation`

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

```
relation: <string>
    description: (0..1) <string>
    member: (2..n)
```

XML

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

JSON

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

SQL

```
create table relations (
    id int primary key,
    type varchar(10),
    description nvarchar(255)
)
```

Comments

- The value of a relation specifies what type of relation it is, for example a relation between synonyms or a relation between a sense and a subsense. Optionally, you can use `relationType` objects to explain those types and to constrain which types of relations are allowed to exist in your lexicographic resource.
- `description` is an optional human-readable explanation of this relation.

4.3 `member`

Represents the fact that an object is a member of a relation.

```
member: <idref>
    role: (0..1) <string>
    listingOrder: (1..1) <number>
    reverseListingOrder: (1..1) <number>
```

XML

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

JSON

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

SQL

```
create table members (
    lexicographicResourceID int foreign key references lexicographicResources(id),
    relationID int foreign key references relations(id),
    memberEntryID int foreign key references entries(id),
    memberSenseID int foreign key references senses(id),
    memberCollocateMarkerID int foreign key references collocateMarkers(id),
    role nvarchar(50),
    listingOrder int,
    reverseListingOrder int,
    id int primary key
)
```

Comments

- The value of `member` is the ID of an object, such as an entry or a sense.

- `role` is 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 `membershipRole` 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` is 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` is 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.

4.4 `relationType`

Represents one of possible values for `relation`.

```
relationType: <string>
    description: (0..1) <string>
    scope: (0..1) <symbol>
    sameAs: (0..n)
    memberRole: <0..n>
```

XML

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

JSON

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

SQL

```
create table relationTypes (
    lexicographicResourceID int foreign key references lexicographicResources(id),
    type varchar(10),
    scope varcar(50),
    id int primary key
);
alter table sameAs (
    add relationTypeID int foreign key references relationTypes(id)
)
```

Comments

- `description` is a human-readable explanation of this relation type.
- `scope` specifies restrictions on member of relations of this type. The possible values are:
  - `sameEntry`: members must be within of the same `entry`
  - `sameResource`: members must be within the same `lexicographicResource`
  - `any`: no restriction
- `memberRole` objects define roles for members of relations of this type.

4.5 memberRole

```
memberRole: <stringOrEmpty>
    description: (1..1) <string>
    memberType: (1..1) <symbol>
    min: (0..1) <number>
    max: (0..1) <number>
    action: (1..1) <symbol>
    sameAs: (0..n)
```

XML

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

JSON

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

SQL

```
create table memberRoles (
    relationTypeID int foreign key references relationTypes(id),
    role varchar(50),
    description varchar(255),
    memberType varchar(50),
    min int,
    max int,
    action varchar(50)
);
alter table sameAs (
    add memberRoleID int foreign key references memberRoles(id)
)
```

Comments

- If the value is empty, then members having this role do not need to have a `role` property.
- `description` is a human-readable explanation of this member role.
- `memberType` is 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 a `itemMarker`.
- `min` is a number which 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` is a number which 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` 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.

5 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).

5.1 Extensions to `headword`

Additional children:

```
headword: ...
    placeholderMarker: (0..n)
```

XML

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

JSON

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

SQL

No changes needed.

5.2 Extensions to `headwordTranslation`

Additional children:

```
headwordTranslation: ...
    placeholderMarker: (0..n)
```

XML

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

JSON

```
{
    ...,
    "headwordTranslations": {
        <language>: [{
            "text": "...",
            "placeholderMarkers": [...],
            ...
        }],
        ...
    },
    ...
}
```

SQL

No changes needed.

5.3 Extensions to `example`

Additional children:

```
example: ...
    headwordMarker: (0..n)
    itemMarker: (0..n)
```

XML

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

JSON

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

SQL

No changes needed.

5.4 Extensions to `exampleTranslation`

Additional children:

```
exampleTranslation: ...
    headwordMarker: (0..n)
    itemMarker: (0..n)
```

XML

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

JSON

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

SQL

No changes needed.

5.5 Extensions to `definition`

Additional children:

```
definition: ...
    headwordMarker: (0..n)
    itemMarker: (0..n)
```

XML

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

```
    ...
</definition>
```

JSON

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

SQL

No changes needed.

5.6 `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. See Examples 6.20, 6.21.

`placeholderMarker: <string>`

XML

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

JSON

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

SQL

```
create table placeholderMarkers (
    entryID int foreign key references entries(id),
    startIndex int,
    endIndex int,
    id int primary key
)
```

5.7 `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. See Example 6.22.

```
headwordMarker: <string>
```

XML

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

JSON

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

SQL

```
create table headwordMarkers (
    entryID int foreign key references entries(id),
    headwordTranslationID int foreign key references headwordTranslations(id),
    definitionID int foreign key references definitions(id),
    startIndex int,
    endIndex int,
    id int primary key
)
```

5.8 `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. See Example 6.23.

```
itemMarker: <string>
    lemma: (0..1) <string>
    itemRole: (0..n) <string>
```

XML

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

JSON

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

SQL

```
create table itemMarkers (
    entryID int foreign key references entries(id),
    headwordTranslationID int foreign key references headwordTranslations(id),
    definitionID int foreign key references definitions(id),
    startIndex int,
    endIndex int,
    lemma varchar(50),
    id int primary key
);
create table itemMarkerRoles (
    itemMarkerID int foreign key references itemMarkers(id),
    role: "...",
    id int primary key
)
```

Comments

- `lemma` is 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` 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.) Optionally, you use the `tag` datatype to explain and/or constrain the item types that are allowed to appear in your lexicographic resource.

6 Examples

This section gives examples which show how to use DMLex to model lexicographic resources. The examples are shown in three formalisms: ### NVH {.unnumbered .unlisted}, ### XML {.unnumbered .unlisted} and ### JSON {.unnumbered .unlisted}.

Each example is shown in NVH first. NVH (Name-Value Hierarchy)[1] is a concise serialization language designed for lexicographic data. ### NVH {.unnumbered .unlisted} encodes data as a hierarchical list of names, values and children, which corresponds exactly to DMLex's own data model. We use NVH here in order to demonstrate the object model at an abstract level.

After that, each example is shown in ### XML {.unnumbered .unlisted} and ### JSON {.unnumbered .unlisted}, two popular serialization languages. The ### XML {.unnumbered .unlisted} and ### JSON {.unnumbered .unlisted} encoding shown here follows DMLex's own implementation guidance for ### XML {.unnumbered .unlisted} and ### JSON {.unnumbered .unlisted}.

6.1 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.

---

[1] https://www.namevaluehierarchy.org/

NVH

```
lexicographicResource: my-dictionary
    entry: abandon-verb
        headword: abandon
        partOfSpeech: verb
        sense: abandon-verb-1
            definition: to suddenly leave a place or a person
            example: I'm sorry I abandoned you like that.
            example: Abandon ship!
                label: idiom
        sense: abandon-verb-2
            label: mostly-passive
            definition: to stop supporting an idea
            example: That theory has been abandoned.
```

XML

```
<lexicographicResource id="my-dictionary">
    <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>
```

JSON

```
{
    "id": "my-dictionary",
    "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."
            }]
        }]
    }
}
```

6.2 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.

NVH

```
entry: folúsghlantóir-n
    headword: folúsghlantóir
    partOfSpeech: n-masc
    inflectedForm: folúsghlantóra
```

```
        inflectedTag: sg-gen
    inflectedForm: folúsghlantóirí
        inflectedTag: pl
    sense: ...
```

XML

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

JSON

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

6.3 Pronunciation given as transcription

NVH

```
entry: aardvark-noun
    headword: aardvark
    pronunciation:
        transcription: a:rdva:rk
    sense: ...
```

XML

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

JSON

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

6.4 Pronunciation given as a sound file

NVH

```
entry: aardvark-noun
    headword: aardvark
    pronunciation:
        soundFile: aardvark.mp3
    sense: ...
```

XML

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

JSON

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

6.5 Pronunciation given both ways

NVH

```
entry: aardvark-noun
    headword: aardvark
    pronunciation:
        transcription: a:rdva:rk
        soundFile: aardvark.mp3
    sense: ...
```

XML

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

JSON

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

```
    "senses": [...]
}
```

6.6 How to use `tag`

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 `tag` objects.

NVH

```
lexicographicResource: my-irish-dictionary
    language: ga
    entry: folúsghlantóir-n
        headword: folúsghlantóir
        partOfSpeech: n-masc
        inflectedForm: folúsghlantóra
            inflectedTag: sg-gen
        inflectedForm: folúsghlantóirí
            inflectedTag: pl
        sense: ...
    tag: n-masc
        description: noun, masculine
        target: partOfSpeech
    tag: n-fem
        description: noun, feminine
        target: partOfSpeech
    tag: sg-gen
        description: singular genitive
        target: inflectedTag
        partOfSpeechConstraint: n-masc
        partOfSpeechConstraint: n-fem
    tag: pl
        description: plural
        target: inflectedTag
        partOfSpeechConstraint: n-masc
        partOfSpeechConstraint: n-fem
```

XML

```
<lexicographicResource id="my-irish-dictionary" language="ga">
    <entry id="folúsghlantóir-n">
```

```
        <headword>folúsghlantóir</headword>
        <partOfSpeech value="n-masc"/>
        <inflectedForm inflectedTag="sg-gen">folúsghlantóra</inflectedForm>
        <inflectedForm inflectedTag="pl">folúsghlantóirí</inflectedForm>
        <sense>...</sense>
    </entry>
    <tag value="n-masc">
        <description>noun, masculine</description>
        <target value="partOfSPeech"/>
    </tag>
    <tag value="n-fem">
        <description>noun, feminine</description>
        <target value="partOfSPeech"/>
    </tag>
    <tag value="sg-gen">
        <description>singular genitive</description>
        <target value="inflectedTag"/>
        <partOfSpeechConstraint value="n-masc"/>
        <partOfSpeechConstraint value="n-fem"/>
    </tag>
    <tag value="pl">
        <description>plural</description>
        <target value="inflectedTag"/>
        <partOfSpeechConstraint value="n-masc"/>
        <partOfSpeechConstraint value="n-fem"/>
    </tag>
</lexicographicResource>
```

JSON

```
{
    "id": "my-irish-dictionary",
    "language": "ga",
    "entries": [{
        "id": "folúsghlantóir-n",
        "headword": "folúsghlantóir",
        "partsOfSpeech": ["n-masc"],
        "inflectedForms": [{
            "text": "folúsghlantóra",
            "inflectedTag": "sg-gen",
        }, {
            "text": "folúsghlantóirí",
            "inflectedTag": "pl",
```

```
        }],
        "senses": [...]
    }],
    "tags": [{
        "value": "n-masc",
        "description": "noun, masculine",
        "targets": ["partOfSpeech"]
    }, {
        "value": "n-fem",
        "description": "noun, feminine",
        "targets": ["partOfSpeech"]
    }, {
        "value": "sg-gen",
        "description": "singular genitive",
        "targets": ["inflectedTag"],
        "partOfSpeechConstraints": ["n-masc", "n-fem"]
    }, {
        "value": "pl",
        "description": "plural",
        "targets": ["inflectedTag"],
        "partOfSpeechConstraints": ["n-masc", "n-fem"]
    }]
}
```

6.7 Mapping `tag` 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.

NVH

```
tag: n-masc
    description: noun, masculine
    target: partOfSpeech
    sameAs: http://www.lexinfo.net/ontology/3.0/lexinfo#noun
    sameAs: http://www.lexinfo.net/ontology/3.0/lexinfo#masculine
tag: n-fem
    description: noun, feminine
    target: partOfSpeech
    sameAs: http://www.lexinfo.net/ontology/3.0/lexinfo#noun
    sameAs: http://www.lexinfo.net/ontology/3.0/lexinfo#feminine
```

XML

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

JSON

```json
{
    "tags": [{
        "value": "n-masc",
        "description": "noun, masculine",
        "targets": ["partOfSpeech"],
        "sameAs": [
            "http://www.lexinfo.net/ontology/3.0/lexinfo#noun",
            "http://www.lexinfo.net/ontology/3.0/lexinfo#masculine"
        ]
    }, {
        "value": "n-fem",
        "description": "noun, feminine",
        "targets": ["partOfSpeech"],
        "sameAs": [
            "http://www.lexinfo.net/ontology/3.0/lexinfo#noun",
            "http://www.lexinfo.net/ontology/3.0/lexinfo#feminine"
        ]
    }]
}
```

6.8 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.

NVH

```
lexicographicResource: deueng
    title: My German-English Dictionary
    language: de
    translationLanguage: en
```

XML

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

JSON

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

6.9 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.

NVH

```
lexicographicResource: irish-multilingual
    description: My Irish-Multilingual Dictionary
    language: ga
    translationLanguage: en
    translationLanguage: de
    translationLanguage: cs
```

XML

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

JSON

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

6.10 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.

NVH

```
entry: doctor-n
    headword: doctor
    sense: doctor-n-1
        indicator: medical doctor
        headwordTranslation: Arzt
            partOfSpeech: n-masc
        headwordTranslation: Ärztin
            partOfSpeech: n-fem
    sense: doctor-n-2
        indicator: academic title
        headwordTranslation: Doktor
            partOfSpeech: n-masc
        headwordTranslation: Doktorin
            partOfSpeech: n-fem
            label: rare
```

XML

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

JSON

```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"]
        }]
    }]
}
```

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

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

NVH

```
entry: fómhar-n
    headword: fómhar
    sense: fómhar-n-1
        headwordTranslation: autumn
            language: en
        headwordTranslation: fall
            language: en
        headwordTranslation: Herbst
            language: de
        headwordTranslation: podzim
            language: cs
    sense: fómhar-n-2
        headwordTranslation: harvest
            language: en
        headwordTranslation: Ernte
            language: de
        headwordTranslation: sklizeň
            language: cs
```

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>sklizeň</text>
        </headwordTranslation>
    </sense>
</entry>
```

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": "sklizeň"
        }]
    },]
}
```

6.12 How to use `headwordExplanation`

NVH

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

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>
```

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"
        }]
    }]
}
```

6.13 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”.

NVH

```
lexicographicResource: my-dictionary
    language: en
    entry: glasses
        headword: glasses
        sense: glasses-1
            definition: an optical seeing aid
    entry: microscope
        headword: microscope
        sense: microscope-1
            definition: equipment for looking at very small things
    entry: lens
```

```
        headword: lens
        sense: lens-1
            definition: curved glass that makes things seem bigger
    relation: meronymy
        member: glasses-1
            role: whole
        member: lens-1
            role: part
    relation: meronymy
        member: microscrope-1
            role: whole
        member: lens-1
            role: part
    relationType: meronomy
        description: used for modelling part-whole relationships
        memberRole: whole
            description: the whole
            memberType: sense
            min: 1
            max: 1
            action: navigate
        memberRole: part
            description: the part
            memberType: sense
            min: 1
            max: 1
            action: navigate
```

XML

```
<lexicographicResource id="my-dictionary" 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 idref="glasses-1" role="whole"/>
        <member idref="lens-1" role="part"/>
    </relation>
    <relation type="meronymy">
        <member idref="microscrope-1" role="whole"/>
        <member idref="lens-1" role="part"/>
    </relation>
    <relationType type="meronomy">
        <description>used for modelling part-whole relationships</description>
        <memberRole role="whole" memberType="sense"  min="1" max="1" action="navigate">
            <description>the whole</description>
        </memberRole>
        <memberRole role="part" memberType="sense" min="1" max="1" action="navigate">
            <description>the part</description>
        </memberRole>
    </relationType>
</lexicographicResource>
```

JSON

```
{
    "id": "my-dictionary",
    "language": "en",
    "entries": [{
        "id": "glasses",
        "headword": "glasses",
        "senses": [{
            "id": "glasses-1",
            "definition": "an optical seeing aid"
        }, {
        "id": "microscope",
        "headword": "microscope",
        "senses": [{
            "id": "microscope-1",
            "definition": "equipment for looking at very small things"
        }, {
```

```
        "id": "lens",
        "headword": "lens",
        "senses": [{
            "id": "lens-1",
            "definition": "curved glass that makes things seem bigger"
        }]
    }],
    "relations": [{
        "type": "meronymy",
        "members": [{
            "idref": "glasses-1",
            "role": "whole"
        }, {
            "idref": "lens-1",
            "role": "part"
        }]
    }, {
        "type": "meronymy",
        "members": [{
            "idref": "microscope-1",
            "role": "whole"
        }, {
            "idref": "lens-1",
            "role": "part"
        }]
    }],
    "relationTypes": [{
        "type": "meronymy",
        "description": "used for modelling part-whole relationships",
        "memberRoles": [{
            "role": "whole",
            "description": "the whole",
            "memberType": "sense",
            "min": 1,
            "max": 1,
            "action": "navigate"
        }, {
            "role": "part",
            "description": "the part",
            "memberType": "sense",
            "min": 1,
            "max": 1,
            "action": "navigate"
        }]
```

```
    }]
}
```

Suggested rendering for human users

> **lens**
>
> - curved glass that makes things seem bigger
>   *things that contain lens:* **glasses**, **microscope**

6.14 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.

NVH

```
lexicographicResource: my-dictionary
    language: en
    entry: buy
        headword: buy
        sense: buy-1
            definition: get something by paying money for it
    entry: sell
        headword: sell
        sense: sell-1
            definition: exchange something for money
    relation: ants
        member: buy-1
        member: sell-1
    relationType: ants
        description: antonyms
        memberRole:
            memberType: sense
            min: 2
            max: 2
            action: navigate
```

XML

```xml
<lexicographicResource id="my-dictionary" 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="ants">
        <member idref="buy-1"/>
        <member idref="sell-1"/>
    </relation>
    <relationType type="ants">
        <description>antonyms</description>
        <memberRole memberType="sense" min="2" max="2" action="navigate"/>
    </relationType>
</lexicographicResource>
```

JSON

```json
{
    "id": "my-dictionary",
    "language": "en",
    "entries": [{
        "id": "buy",
        "headword": "buy",
        "senses": [{
            "id": "buy-1",
            "definition": "get something by paying money for it"
        }, {
        "id": "sell",
        "headword": "sell",
        "senses": [{
            "id": "sell-1",
            "definition": "exchange something for money"
        }]
```

```
    }],
    "relations": [{
        "type": "ants",
        "members": [
            {"idref": "buy-1"},
            {"idref": "sell-1"}
        ]
    }],
    "relationTypes": [{
        "type": "ants",
        "description": "antonyms",
        "memberRoles": [{
            "memberType": "sense",
            "min": 2,
            "max": 2,
            "action": "navigate"
        }]
    }]
}
```

Suggested rendering for human users

buy

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

6.15 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.

NVH

```
lexicographicResource: my-dictionary
    language: de
    translationLanguage: en
    entry: die-see
        headword: See
        partOfSpeech: n-fem
        sense: die-see-1
```

```
            headwordTranslation: sea
    entry: das-meer
        headword: Meer
        partOfSpeech: n-neut
        sense: das-meer-1
            headwordTranslation: sea
    entry: der-ozean
        headword: Ozean
        partOfSpeech: n-masc
        sense: der-ozean-1
            translation: ocean
    relation: syns
        description: words that mean sea and ocean
        member: die-see-1
        member: das-meer-1
        member: der-ozean-1
    relationType: syns
        description: synonyms and near synonyms
        memberRole:
            memberType: sense
            min: 2
            action: navigate
```

XML

```
<lexicographicResource id="my-dictionary" 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="syns">
        <description>words that mean sea and ocean</description>
        <member idref="die-see-1"/>
        <member idref="das-meer-1"/>
        <member idref="der-ozean-1"/>
    </relation>
    <relationType type="syns">
        <description>synonyms and near synonyms</description>
        <memberRole memberType="sense" min="2" action="navigate"/>
    </relationType>
</lexicographicResource>
```

JSON

```
{
    "id": "my-dictionary",
    "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": "syns",
        "description": "words that mean sea and ocean",
        "members": [
          {"idref": "die-see-1"},
          {"idref": "das-meer-1"},
          {"idref": "der-ozean-1"}
        ]
    }],
    "relationTypes": [{
        "type": "syns",
        "description": "synonyms and near synonyms",
        "memberRoles": [{
            "memberType": "sense",
            "min": 2,
            "action": "navigate"
        }]
    }]
}
```

Suggested rendering for human users

> **See** *feminine noun*
>
> - sea
>   *same or similar meaning:* **Meer**, **Ozean**

6.16 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.

NVH

```
lexicographicResource: my-dictionary
    language: en
```

```
    entry: colour
        headword: colour
        partOfSpeech: n
        label: europeanSpelling
        sense: colour-1
            definition: red, blue, yellow etc.
            example: What is your favourite colour?
    entry: color
        headword: color
        partOfSpeech: n
        label: americanSpelling
    relation: vars
        member: colour
        member: color
    relationType: vars
        description: variants, words which differ only in spelling
        memberRole:
            memberType: entry
            min: 2
            action: navigate
```

XML

```
<lexicographicResource id="my-dictionary" language="en">
    <entry id="colour">
        <headword>colour</headword>
        <partOfSpeech value="n"/>
        <label value="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 value="n"/>
        <label value="americanSpelling"/>
    </entry>
    <relation type="vars">
        <member idref="colour"/>
        <member idref="color"/>
    </relation>
    <relationType type="vars">
```

```
        <description>variants, words which differ only in spelling</description>
        <memberRole memberType="entry" min="2" action="navigate"/>
    </relationType>
</lexicographicResource>
```

JSON

```
{
    "id": "my-dictionary",
    "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": "vars",
        "members": [
          {"idref": "colour"},
          {"idref": "color"}
        ]
    }],
    "relationTypes": [{
        "type": "vars",
        "description": "variants, words which differ only in spelling",
        "memberRoles": [{
            "memberType": "entry",
            "min": 2,
            "action": "navigate"
        }]
    }]
}
```

Suggested rendering for human users

colour *noun, European spelling*

- red, blue, yellow etc.
  What is your favourite colour?

see also: color

6.17 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.

NVH

```
lexicographicResource: my-dictionary
    language: en
    entry: colour
        headword: colour
            sense: colour-1
                definition: red, blue, yellow etc.
                example: What is your favourite colour?
            sense: colour-2
                definition: not being black and white
                example: Back then owning a colour TV meant you were rich.
            sense: colour-3
                definition: a sign of a person's race
                example: We welcome people of all creeds and colours.
            sense: colour-4
                definition: interest or excitement
                example: Examples add colour to your writing.
    relation: subsensing
        member: colour-1
            role: supersense
        member: colour-2
            role: subsense
    relation: subsensing
        member: colour-1
            role: supersense
        member: colour-3
            role: subsense
```

```
    relationType: subsensing
        description: expresses the fact that a sense is a subsense of another sense
        scope: sameEntry
        memberRole: supersense
            memberType: sense
            min: 1
            max: 1
            action: none
        memberRole: subsense
            memberType: sense
            min: 1
            max: 1
            action: embed
```

XML

```
<lexicographicResource id="my-dictionary" 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>Back then owning a colour TV meant you were rich.</text></exa
        </sense>
        <sense id="colour-3">
            <definition>a sign of a person's race</definition>
            <example><text>We welcome 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 idref="colour-1" role="supersense"/>
        <member idref="colour-2" role="subsense"/>
    </relation>
    <relation type="subsensing">
        <member idref="colour-1" role="supersense"/>
        <member idref="colour-3" role="subsense"/>
```

```
    </relation>
    <relationType type="subsensing" scope="sameEntry">
        <description>
            expresses the fact that a sense is a subsense of another sense
        </description>
        <memberRole role="supersense" memberType="sense" min="1" max="1"
                    action="none"/>
        <memberRole role="subsense" memberType="sense" min="1" max="1"
                    action="embed"/>
    </relationType>
</lexicographicResource>
```

JSON

```
{
    "id": "my-dictionary",
    "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": "Back then owning a colour TV meant you were rich."}]
        }, {
            "id": "colour-3",
            "definitions": [{"text": "a sign of a person's race"}],
            "examples": [{"text": "We welcome 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": "supersense", "idref": "colour-1"},
          {"role": "subsense", "idref": "colour-2"}
```

```
        ]
    }, {
        "type": "subsensing",
        "members": [
          {"role": "supersense", "idref": "colour-1"},
          {"role": "subsense", "idref": "colour-3"}
        ]
    }],
    "relationTypes": [{
        "type": "subsensing",
        "description": "expresses the fact that a sense is a subsense of another sense",
        "scope": "sameEntry",
        "memberRoles": [{
            "role": "supersense",
            "memberType": "sense",
            "min": 1,
            "max": 1,
            "action": "none"
        }, {
            "role": "subsense",
            "memberType": "sense",
            "min": 1,
            "max": 1,
            "action": "embed"
        }]
    }]
}
```

Suggested rendering for human users

colour

1. red, blue, yellow etc.
   What is your favourite colour?
   a. not being black and white
      Back then owning a colour TV meant you were rich.
   b. a sign of a person’s race
      We welcome people of all creeds and colours.
2. interest or excitement
   Examples add colour to your writing.

6.18 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.

NVH

```
lexicographicResource: my-dictionary
    language: en
    entry: safe
        headword: safe
        sense: safe-1
            indicator: protected from harm
            example: It isn't safe to park here.
        sense: safe-2
            indicator: not likely to cause harm
            example: Is the ride safe for a small child?
    entry: better-safe
        headword: better safe than sorry
        sense: better-safe-1
            definition: you should be careful even if it seems unnecessary
    relation: subentrying
        membership: safe-1
            role: container
        membership: better-safe
            role: subentry
    relationType: subentrying
        scope: sameResource
        memberRole: container
            memberType: sense
            min: 1
            max: 1
            action: navigate
        memberRole: subentry
            memberType: entry
            min: 1
            max: 1
            action: embed
```

XML

```
<lexicographicResource id="my-dictionary" 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 idref="safe-1" role="container"/>
        <member idref="better-safe" role="subentry"/>
    </relation>
    <relationType type="subentrying" scope="sameResource">
        <memberRole role="container" memberType="sense" min="1" max="1"
                    action="navigate"/>
        <memberRole role="subentry" memberType="entry" min="1" max="1"
                    action="embed"/>
    </relationType>
</lexicographicResource>
```

JSON

```
{
    "id": "my-dictionary",
    "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", "idref": "safe-1"},
          {"role": "subentry", "idref": "better-safe"}
        ]
    }],
    "relationTypes": [{
        "type": "subentrying",
        "scope": "sameResource",
        "memberRoles": [{
            "role": "container",
            "memberType": "sense",
            "min": 1,
            "max": 1,
            "action": "navigate"
        }, {
            "role": "subentry",
            "memberType": "entry",
            "min": 1,
            "max": 1,
            "action": "embed"
        }]
    }]
}
```

Suggested rendering for human users

safe

1. protected from harm: *It isn't safe to park here.*
   - **better safe than sorry** you should be careful even if it seems unnecessary
2. 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

6.19 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.

NVH

```
lexicographicResource: my-dictionary
    language: en
    entry: the-bible
        headword: the Bible
        Sense: the-bible-1
            definition: the book considered holy by Christians
    entry: bible
        headword: bible
        sense: bible-1
        sense: bible-2
            definition: a book considered important for a subject
    relation: subentrying
        member: bible-1
            role: container
        member: the-bible
            role: subentry
    relationType: subentrying
        scope: sameResource
        memberRole: container
            memberType: sense
```

```
            min: 1
            max: 1
            action: navigate
        memberRole: subentry
            memberType: entry
            min: 1
            max: 1
            action: embed
```

XML

```
<lexicographicResource id="my-dictionary" 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 idref="bible-1" role="container"/>
        <member idref="the-bible" role="subentry"/>
    </relation>
    <relationType type="subentrying" scope="sameResource">
        <memberRole role="container" memberType="sense" min="1" max="1"
                    action="navigate"/>
        <memberRole role="subentry" memberType="entry" min="1" max="1"
                    action="embed"/>
    </relationType>
</lexicographicResource>
```

JSON

```
{
    "id": "my-dictionary",
    "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", "idref": "bible-1"},
          {"role": "subentry", "idref": "the-bible"}
        ]
    }],
    "relationTypes": [{
        "type": "subentrying",
        "scope": "sameResource",
        "memberRoles": [{
            "role": "container",
            "memberType": "sense",
            "min": 1,
            "max": 1,
            "action": "navigate"
        }, {
            "role": "subentry",
            "memberType": "entry",
            "min": 1,
            "max": 1,
```

```
                "action": "embed"
            }]
        }]
}
```

Suggested rendering for human users

> **bible**
>
> 1. **the Bible** the book considered holy by Christians
> 2. a book considered important for a subject

Suggeted rendering of the entry “the Bible” for human users:

> **the Bible**
>
> - the book considered holy by Christians
>
> see also: bible

6.20 Using `placeholderMarker`

NVH

```
entry: continue-studies
    headword: continue your studies
        placeholderMarker: your
    sense: ...
```

XML

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

JSON

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

6.21 Using `placeholderMarker` in a bilingual lexicographic resource

NVH

```
entry: beat-up
    headword: beat sb. up
        placeholderMarker: sb.
    sense: beat-up-1
        headwordTranslation: jemanden verprügeln
            placeholderMarker: jemanden
```

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>
```

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}
      ],
    }]
  }]
}
```

6.22 Using `headwordMarker`

NVH

```
entry: autopsy
    headword: autopsy
    sense: autopsy-1
        headwordTranslation: pitva
        example: The coroner performed an autopsy.
            headwordMarker: autopsy
            exampleTranslation: Koroner provedl pitvu.
                headwordMarker: pitvu
```

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>
```

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}
        ]
      }]
    }]
  }]
}
```

6.23 Using `itemMarker`

NVH

```
entry: autopsy
    headword: autopsy
    sense: autopsy-1
        headwordTranslation: pitva
        example: The coroner performed an autopsy.
            headwordMarker: autopsy
            itemMarker: performed
                lemma: perform
            exampleTranslation: Koroner provedl pitvu.
                headwordMarker: pitvu
```

```
                itemMarker: provedl
                    lemma: provést
```

XML

```
<entry id="autopsy">
    <headword>autopsy</headword>
    <sense id="autopsy-1">
        <headwordTranslation><text>pitva</text></headwordTranslation>
        <example>
            <text>
                The coroner <itemMarker lemma="perform">performed</itemMarker>
                an <headwordMarker>autopsy</headwordMarker>.
            </text>
            <exampleTranslation>
                <text>
                    Koroner <itemMarker lemma="provést">provedl</itemMarker>
                    <headwordMarker>pitvu</headwordMarker>.
                </text>
            </exampleTranslation>
        </example>
    </sense>
</entry>
```

JSON

```
{
  "id": "autopsy",
  "headword": "autopsy",
  "senses": [{
    "id": "autopsy-1",
    "headwordTranslations": [{"text": "pitva"}],
    "examples": [{
      "text": "The coroner performed an autopsy.",
      "headwordMarkers": [
        {"startIndex": 25, "endIndex": 32}
      ],
      "itemMarkers": [
        {"startIndex": 12, "endIndex": 21, "lemma": "perform"}
      ],
      "exampleTranslations": [{
        "text": "Koroner provedl pitvu.",
```

```
        "headwordMarkers": [
          {"startIndex": 16, "endIndex": 21}
        ],
        "itemMarkers": [
          {"startIndex": 8, "endIndex": 15, "lemma": "provést"}
        ],
      }]
    }]
  }]
}
```