DMLex (Draft)

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 straight-forwardly implementable in XML, JSON, as a relational database, and as a Semantic Web triplestore.

1.1 Pseudocode used in examples

In this document, examples of DMLex are given in pseudocode which represents dictionary entries, their headwords, their senses and other objects as a **hierarchical list of pairs of names and values**. Each object has a name, followed by a colon, followed by a value, followed by a line break, followed optionally by an indented list of **child objects**, each of which also has the same structure.

Example

```
lexicographicResource:
    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.
```

Line 1 declares a lexicograpic resource; the following lines define its content. Line 2 says that there exists an entry whose ID is `abandon-verb`. Lines 3 and 4 say what its headword and part of speech are. Lines 5 – 13 say that the entry has two senses, each with an ID and a definition, some with labels, and each with some example sentences, some of which also have labels.

1.2 Schema formalism

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>` an positive integer number
- `<id>` an 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.

Example

```
lexicographicResource: <empty>
    entry: (0..n)

entry: <id>
```

```
    headwword: (1..1) <string>
    partOfSpeech: (0..1) <string>
    sense: (0..n)

sense: <id>
    label: (0..n)
    definition: (0..1) <string>
    example: (0..n)

example: <string>
    label: (0..n)

label: <string>
```

This example gives the schema for the entry shown above such that the entry conforms to the schema.

- The first code block says that the value of a `lexicographicResource` is empty, and that it can contain:
  - any number of `entry` objects
- The second code block says that the value of an `entry` is its ID, and that it can contain:
  - exactly one `headword`, which must be a string
  - at most one `partOfSpeech`, which is also a string
  - any number of `sense` objects
- The third code block says that the value of a `sense` is its ID, and that it can contain:
  - at most one `definition`, which must be a string
  - any number of `example` objects
- The fourth code block says that the value of an `example` is a string, and that it can contain
  - any number of `label` objects
- The last code block says that the value of a `label` is a string, and that it cannot contain any child objects.

1.3 Implementing DMLex

DMLex is an abstract data model and makes no requirements on how it should be implemented. The following should offer some guidance on how to implement DMLex in XML, in JSON, and as a relational database.

1.3.1 Implementing DMLex in XML

We recommend that you follow these principles when implementing DMLex in XML.

- The top-level `lexicographicResource` object should be implemented as an XML element.
- All other objects should be 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 should be implemented as a child XML element of its parent.

Example

```
<lexicographicResource>
    <entry id="abandon-verb" partOfSpeech="verb">
        <headword>abandon</headword>
        <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>
```

1.3.2 Implementing DMLex in JSON

We recommend that you follow these principles when implementing DMLex in JSON.

- The top-level `lexicographicResource` object should be implemented as a JSON object: `{...}`.
- All other objects should be implemented as JSON name-value pairs inside their parent JSON object: `{"name": ...}`.
- The values of objects should be 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.

Example

```
{
    "entry": {
        "id": "abandon-verb",
        "headword": "abandon",
        "partOfSpeech": "verb",
        "senses": [{
            "id": "abandon-verb-1",
            "definition": "to suddenly leave a place or a person",
            "examples": [{
                "text": "I'm sorry I abandoned you like that."
            }, {
                "text": "Abandon ship!",
                "label": "idiom"
            }]
        }, {
            "id": "abandon-verb-2",
            "label": "mostly-passive",
            "definition": "to stop supporting an idea",
            "examples": [{
                "text": "That theory has been abandoned."
            }]
        }]
```

```
    }
}
```

1.3.3 Implementing DMLex as a relational database

- The `lexicographicResource` object should be implemented as table, or 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)` should be implemented as tables.
- The values of objects, and objects with an arity of `(0..1)` or `(1..1)` should be implemented as columns in those tables.
- The parent-child relation should be implemented as a one-to-many relation between tables.

An example can be seen in Figure 1.

2 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>
    transcriptionScheme: (0..1) <langCode>
    entry: (0..n)
    tag: (0..n)
    source: (0..n)
```

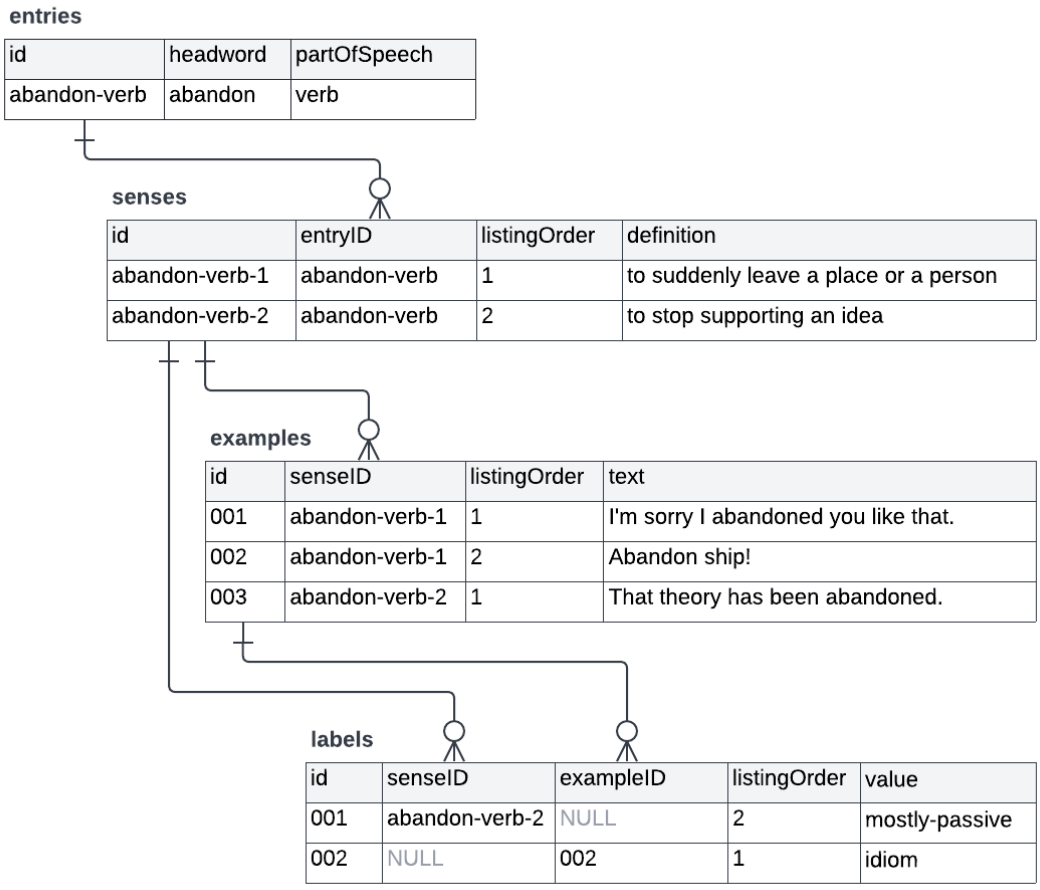

Figure 1: Example of DMLex as a relational database

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.
- `transcriptionScheme` object identifies the transcription scheme used in `pronunciation` objects. Example: `en-fonipa` for English IPA.
- 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`, namely `Ttg` and `source`, 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..1) <string>
    label: (0..n)
    pronunciation: (0..n)
    inflectedForm: (0..n)
    sense: (0..n)
```

Comments:

- `headword` contains entry's headword. The headword can be a single word, a multiword expression, or any expression in the source language which is being described by the entry.
- `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. ou 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`.

- 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 `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..1) <string>
    example: (0..n)
```

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 indicates the meaning of a sense and permits its differentiation from other senses in the entry. Indicators are sometimes used in dictionaries as "mini-definitions" instead of or in addition to regular definitions. They are short, one-word or two-word paraphrases of the sense. Their purpose is to allow the (human) user to find the desired sense quickly.
- `definition` is a long 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.
- 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 proproperties (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.

2.4 `inflectedForm`

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

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

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 that in DMLex is to create separate entries for the two words, and link them using the Linking Module.

Example

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.

```
entry: folúsghlantóir-n
    headword: folúsghlantóir
    partOfSpeech: n-masc
    inflectedForm: folúsghlantóra
        inflectedTag: sg-gen
    inflectedForm: folúsghlantóirí
        inflectedTag: pl
    sense: ...
```

2.5 `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>
```

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.
- 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).

2.6 `pronunciation`

Represents instructions on the pronunciation of its parent.

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

Comments:

- `transcription` is the transcription of the pronuciation in some notation, such as IPA. DMLex is based on the assumption that, if you do use any pronunciation transcriptions in your lexicographic resource, they all follow the same scheme. The `transcriptionScheme` child of `lexicographicResource` tells potential users of the lexicographic resource which scheme that is.
- `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.

Example of pronunciation given as transcription:

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

Example of pronunciation given as a sound file:

```
entry: aardvaark-noun
    headword: aardvark
    pronunciation:
        recording: aardvark.mp3
    sense: ...
```

Example of pronunciation given both ways:

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

2.7 `example`

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

```
example: <string>
    source: (0..1) <string>
    sourceElaboration: (0..1) <string>
    recording: (0..1) <uri>
    listingOrder: (1..1) <number>
```

Comments:

- `source` 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 `source` is absent then `sourceElaboration` can be used to fully name the source.
- `recording` 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.

2.8 `tag`

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

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

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`, `label`, `source` 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`, `label`, `source`. However, doing this is optional in DMLex. An implementation of DMLex is compliant regardless of whether it enfoces referential integrity on `tag` values.
- `description` is a human-readable description of what the tag means.
- `location` tells us where exactly the tag is expected to be used. If omitted, then everythere. The possible values are:
  - `partOfSpeech`
  - `inflectedTag`
  - `source`
  - `label`
  - `entry:label`
  - `sense:label`
  - `inflectedForm:label`
  - `pronunciation:label`
  - `example:label`

- `partOfSpeechConstraint`, if present, says that this tag is only intended to be used inside entries that are labelled with this part of speech.
- `scope` and `partOfSpeech` 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.

Example

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.

```
entry: folúsghlantóir-n
    headword: folúsghlantóir
    partOfSpeech: n-masc
    inflectedForm: folúsghlantóra
        inflectedTag: sg-gen
    inflectedForm: folúsghlantóirí
        inflectedTag: pl
    sense: ...

tag: n-masc
    description: noun, masculine
    location: partOfSpeech
tag: n-fem
    description: noun, feminine
    location: partOfSpeech

tag: sg-gen
    description: singular genitive
    location: inflectedTag
    partOfSpeechConstraint: n-masc
    partOfSpeechConstraint: n-fem
tag: pl
    description: plural
    location: inflectedTag
    partOfSpeechConstraint: n-masc
    partOfSpeechConstraint: n-fem
```

2.9 `sameAs`

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

```
sameAs: <uri>
```

Comments:

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

Example

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

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

3 Bilingual Module

DMLex's Bilingual Module extends the Core and turns a monolingual lexicographic resource into a bilingual one. A bilingual lexicographic resource is a lexicographic resource with two languages: the headwords and the examples are in one language (called the *headword language* in DMLex) and their translations are in another language (called the *translation language* in DMLex).

The Bilingual Module is what you should implement if you want to have only one translation language. If you want to have multiple translation languages in your lexicographic resource, then you should impement the Multilingual Module instead. The Bilingual Module and the Multilingual Module are mutually exclusive: you can implement one or the other but not both.

3.1 Extensions to `lexicographicResource`

Additional children:

```
lexicographicResource: ...
    translationLanguage: (1..1) <langCode>
    translationTranscriptionScheme: (0..1) <langCode>
```

Example

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.

```
lexicographicResource: deueng
    description: My German-English Dictionary
    language: de
    translationLanguage: en
    translationTranscriptionScheme: en-fonipa
```

3.2 Extensions to `sense`

Additional children:

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

3.3 Extensions to `example`

Additional children:

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

3.4 `headwordTranslation`

Represents one (of possibly multiple) translatins of a headword.

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

Comments:

- `listingOrder` is the position of this `headwordTranslation` object among other `headwordTranslation` objects and `headwordExplanation` objects in the same sense. This can be implicit from the serialization.
- `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. ou 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.

Example

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

```
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
```

3.5 `headwordExplanation`

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

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

Comments:

- `listingOrder` is the position of this `headwordTranslation` object among other `headwordExplanation` objects and `headwordTranslation` objects in the same sense. This can be implicit from the serialization.

Example

These are two entries from a hypothetical Gerrman-English dictionary for English-speaking learners of German. In the first entry ("Fernweh") the lexicographer has decided to start with an explanation and then give two possible translations because this concept does not really have an accurate translation in English. In the second entry ("Treppenwitz") the lexicographer has decided to start with a translation, because an accurate translation of this concept does exist in Englsh, but it is rarely used and not very well known, so the lexicographer has decided to add an explanation as well.

```
entry: fernweh
    headword: Ferhweh
    partOfSpeech: n-neut
    sense: fernweh-1
        headwordExplanation: nostalgic desire to travel
        headwordTranslation: itchy feet
        headwordTranslation: wanderlust
entry: treppenwitz
    headword: Treppenwitz
    partOfSpeech: n-masc
    sense: treppenwitz-1
        headwordTranslation: staircase wit
        headwordExplanation: belated realisation of what one could have said
```

3.6 `exampleTranslation`

Represents the translation of an example.

```
exampleTranslation: <string>
    recording: (0..1) <uri>
    listingOrder: (1..1) <number>
```

Comments:

- `recording` is a pointer to a file containing a sound recording of the translation.
- `listingOrder` is the position of this translation among other translations of the same example. This can be implicit from the serialization.

3.7 Extensions to `tag`

Additional values for `location`:

- `entry:partOfSpeech`
- `headwordTranslation:partOfSpeech`
- `entry:inflectedTag`
- `headwordTranslation:inflectedTag`
- `headwordTranslation:label`
- `entry:inflectedForm:label`
- `headwordTranslation:inflectedForm:label`
- `entry:pronunciation:label`
- `headwordTranslation:pronunciation:label`
- `exampleTranslation:label`

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.

4 Multilingual Module

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

The Multilingual Module is what you should implement if you want to have more than one translation language. If you want to have only one translation language in your

lexicographic resource, then you should impement the Bilingual Module instead. The Multilingual Module and the Bilingual Module are mutually exclusive: you can implement one or the other but not both.

4.1 Extensions to `lexicographicResource`

Additional children:

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

4.2 `translationLanguage`

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

```
translationLanguage: <langCode>
    transcriptionScheme: (0..1) <langCode>
    listingOrder: (1..1) <number>
```

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.

Example

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

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

4.3 Extensions to `sense`

Additional children:

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

4.4 Extensions to `example`

Additional children:

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

4.5 `headwordTranslation`

Represents one (of possibly multiple) translations of a headword.

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

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.
- For more comments see commens under `headwordTranslation` in the Bilingual Module.

Example

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

```
entry: fómhar-n
    headword: fómhar
    partOfSpeech: n-masc
    inflectedForm: fómhair
        inflectedTag: genitive-case
    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
```

4.6 `headwordExplanation`

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

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

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.
- For more comments see commens under `headwordExplanation` in the Bilingual Module.

4.7 `exampleTranslation`

Represents the translation of an example.

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

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.
- For more comments see commens under `exampleTranslation` in the Bilingual Module.

4.8 Extensions to `tag`

Additional child:

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

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 additional values for `location`, see `tag` in the Bilingual Module.
- For a redefinition of `partOfSpeechConstraint`, see see `tag` in the Bilingual Module.

5 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 betwen objects inside the same lexicographic resource, or between objects residing in different lexicographic resources.

Relations themselves can be members of other relations.

5.1 Extensions to `lexicographicResource`

Additional children:

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

5.2 `relation`

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

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

Comments:

- Relations have IDs because relations can me members of other relations. For example, when a relation defines a set of synonyms, and this set of synonyms can themselves be a member of other relations, such as a relation that links it to something in another lexicographic resource.
- `type` specifies what type of relation it is, for example a relation between synonyms or a relation between a sense and a subsense. Further, you can `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.

5.3 `member`

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

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

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.

5.4 `relationType`

Represents one (of many) possible values for `type` of `relation`.

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

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.

5.5 `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)
```

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`.
  - `relation`: the object that has this role must be a `relation`.
  - `collocationMarker`: the object that has this role must be a `collocationMarker`.
- `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.

5.6 Examples

5.6.1 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”.

```
lexicographicResource:
    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:
        type: meronymy
        member: glasses-1
            role: super
        member: lens-1
            role: sub
    relation:
        type: meronymy
        member: microscrope-1
            role: super
        member: lens-1
            role: sub

    relationType: meronomy
        description: used for modelling part-whole relationships
        memberRole: super
            description: the whole
            memberType: sense
            min: 1
            max: 1
```

```
            action: navigate
        memberRole: sub
            description: the part
            memberType: sense
            min: 1
            max: 1
            action: navigate
```

Suggested rendering of the entry "lens" for human users:

lens

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

5.6.2 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.

```
lexicographicResource:
    language: en
    entry: buy
        headword: buy
        sense: buy-1
            definition: get something by paying money for it
    entry: sell
        headword: sell
        sense: see-1
            definition: exchange something for money

    relation:
        type: ants
        member: buy-1
        member: sell-1

    relationType: ants
        description: antonyms
        memberRole:
            memberType: sense
            min: 2
            max: 2
            action: navigate
```

Suggested rendering of the entry “buy” for human users:

buy

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

5.6.3 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.

```
lexicographicResource:
    language: de
    translationLanguage: en

    entry: die-see
        headword: See
        partOfSpeech: n-fem
        sense: die-see-1
            headwordTranslation: see

    entry: das-meer
        headword: Meer
        partOfSpeech: n-neut
        sense: das-meer-1
            headwordTranslation: see

    entry: der-ozean
        headword: Ozean
        partOfSpeech: n-masc
        sense: der-ozean-1
            translation: ocean

    relation: see-meer-ozean
        type: 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
```

Suggested rendering of the entry “See” for human users:

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

5.6.4 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.

```
lexicographicResource:
    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
```

Suggested rendering of the entry “colour” for human users:

colour *noun, European spelling*

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

see also: color

5.6.5 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.

```
lexicographicResource:
    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:
        type: subsensing
        member: colour-1
            role: supersense
        member: colour-2
            role: subsense
    relation:
        type: 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
```

Suggested rendering of the entry 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.

5.6.6 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.

```
lexicographicResource:
    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:
        definition: you should be careful even if it seems unnecessary

relation:
    type: subentrying
    membership: safe-2
        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
```

Suggested rendering of the entry “safe” 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?*

Suggeted rendering of the entry “better safe than sorry” for human users:

better safe than sorry

- you should be careful even if it seems unnecessary

see also: safe

5.6.7 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.

```
lexicographicResource:
    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:
        type: 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
```

Suggested rendering of the entry “bible” 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 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 in XML) or as *stand-off* markup (for example through start and end indexes).

6.1 Extensions to `headword`

Additional children:

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

6.2 Extensions to `headwordTranslation`

Additional children:

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

6.3 Extensions to `example`

Additional children:

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

6.4 Extensions to `exampleTranslation`

Additional children:

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

6.5 `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.

```
placeholderMarker: <string>
```

Example

```
lexicographicResource:
    language: en
    entry: continue-studies
        headword: continue your studies
            placeholderMarker: your
        sense: ...
```

Example

```
lexicographicResource:
    language: en
    translationLanguage: de
    entry: beat-up
        headword: beat sb. up
            placeholderMarker: sb.
        sense: beat-up-1
            headwordTranslation: jemanden verprügeln
                placeholderMarker: jemanden
```

6.6 headwordMarker

Marks up a substring inside an example (or inside an example translation) 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.

```
headwordMarker: <string>
```

Example

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

6.7 collocateMarker

Marks up a substring inside an example (or inside an example translation) where an important collocate of the headword (or of its translation) occurs. An application can use the inline markup to highlight the collocate for human readers through formatting.

```
collocateMarker: <string>
    lemma: (0..1) <string>
```

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.)

Example

```
lexicographicResource:
    language: en
    translationLanguage: cs
    entry: autopsy
        headword: autopsy
        sense: autopsy-1
            headwordTranslation: pitva
            example: The coroner performed an autopsy.
                headwordMarker: autopsy
                collocateMarker: performed
                    lemma: perform
                exampleTranslation: Koroner provedl pitvu.
                    headwordMarker: pitvu
                    collocateMarker: provedl
                        lemma: provést
```