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.

`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.

Properties:

- `language` (optional, IETF language code) Identifies the language of headwords, definitions and examples in this dictionary.
- `transcriptionScheme` (optional, IETF language code) identifies the transcription scheme used in the `transcription` property of `Pronunciation` objects in this dictionary. Example: `en-fonipa` for English IPA.

Children:

- `Entry` (one or more)
- `Tag` (zero or more)
- `Source` (zero or more)

The main role of a lexicographic resource is to contain entries (`Entry` objects), and each lexicographic resource must contain at least one. The other two object types that can optionally occur as children of a `LexicographicResource`, namely `Tag` and `Source`, are for lists of look-up values such as part-of-speech labels and their meaning is explained later in this module.

Example of a simple lexicographic resource:

```
LexicographicResource: language="en"
    Entry: headword="aardvark"
        PartOfSpeech: tag="n"
        Sense: definition="a long-nosed animal..."
    Entry: headword="abacus"
        PartOfSpeech: tag="n"
```

```
        InflectedForm: tag="pl" text="abacuses"
        Sense: definition="a frame with balls..."
    Entry: headword="abandon"
        PartOfSpeech: tag="v"
        InflectedForm: tag="ing" text="abandoning"
        InflectedForm: tag="pres" text="abandons"
        InflectedForm: tag="past" text="abandoned"
        Sense: definition="to suddenly leave a place or a person"
            Example: text="I'm sorry I abandoned you like that"
        Sense: definition="to stop supporting an idea"
            Example: text="that theory has been abandoned"
```

Note on dictionary language: 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` property of `LexicographicResource` informs potential users of the lexicographic resource which language that is.

`Entry`

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

Child of:

- `LexicographicResource`

Properties:

- `headword` (obligatory, non-empty string) The 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.
- `homographNumber` (number, optional)

Children:

- `PartOfSpeech` (zero or more)
- `Label` (zero or more)
- `Pronunciation` (zero or more)
- `InflectedForm` (zero or more)
- `Sense` (zero or more)

Note on the listing order of entries: 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.

Note on entries with zero senses: TBD…

Note on subentries: 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 Linkling Module for that.

`Sense`

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

Child of:

- `Entry`

Properties:

- `listingOrder` (number, obligatory but can be implicit from the serialization) The position of this sense among other senses of the same entry.
- `indicator` (optional, non-empty string) 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` (optional, non-empty string) 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.

Children:

- `Label` (zero or more)
- `Example` (zero or more)

Note on subsenses: TBD…

Guidance: entries versus senses

TBD: 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.

TBD: No morphological info 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 you need treat it as two entries (homographs). Plus you can use the Linking Module two link the two entries together and to make sure they are always shown together to human users.

`PartOfSpeech`

Represents the headword's part of speech (or word class, or grammatical category).

Child of:

- `Entry`

Properties:

- `tag` (obligatory, string) An abbreviation, a code or some other string of text which identifies the part-of-speech label, for example `n` for noun, `v` for verb, `adj` for adjective.
- `listingOrder` (number, obligatory but can be implicit from the serialization) The position of this part-of-speech label among other part-of-speech labels of the same entry.

Note on part-of-speech tags: 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.

Note on complex grammatical categories: 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`.

Note on entries with more than one part-of-speech label: TBD…

`InflectedForm`

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

Child of:

- `Entry`

Properties:

- `tag` (obligatory, string) An abbreviation, a code or some other string of text which identifies the inflected form, for example `pl` for plural, `gs` for genitive singular, `com` for comparative.
- `text` (obligatory, string) The inflected form itself.
- `listingOrder` (number, obligatory but can be implicit from the serialization) The position of this inflected form among other inflected forms of the same entry.

Children:

- `Label` (zero or more)
- `Pronunciation` (zero or more)

Note on inflection tags: 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.

Note on inflectional morphology versus derivational morphology: TBD…

`Label`

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

Child of:

- `Entry`
- `Sense`
- `InflectedForm`
- `Pronunciation`

Properties:

- `tag` (obligatory, string) An abbreviation, a code or some other string of text which identifies the label, for example `neo` for neologism, `colloq` for colloquial, `polit` for politics.
- `listingOrder` (number, obligatory but can be implicit from the serialization) The position of this label among other labels of the same parent.

Note on label tags: 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.

Note on the applicability of labels: 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).

`Pronunciation`

Represents the pronunciation of its parent.

Child of:

- `Entry`
- `InflectedForm`

Properties:

- `transcription` (string, obligatory, but optional if `soundFile` is present) A transcription of the pronuciation in some notation, such as IPA.
- `soundFile` (string, obligatory, but optional if `transcription` is present) A pointer to a file containing a sound recording of the pronunciation.
- `listingOrder` (number, obligatory but can be implicit from the serialization) The position of this label among other labels of the same parent.

Children:

- `Label` (zero or more)

Example of pronunciation given as transcription:

```
Entry: headword="aardvark"
    Pronunciation: transcription="a:rdva:rk"
```

Example of pronunciation given as a sound file:

```
Entry: headword="aardvark"
    Pronunciation: recording="aardvark.mp3"
```

Example of pronunciation given both ways:

```
Entry: headword="aardvark"
    Pronunciation: transcription="a:rdva:rk", recording="aardvark.mp3"
```

Note on transcription scheme: 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` property of `LexicographicResource` tells potential users of the lexicographic resource which scheme that is.

`Example`

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

Child of:

- `Sense`

Properties:

- `text` (obligatory, non-empty string) The text of the example.

- `source` (optional, string) An abbreviation, a code or some other string of text which identifies the source.
- `sourceElaboration` (optional, string) A free-form statement about the source of the example.
- `soundFile` (string, optional) A pointer to a file containing a sound recording of the example.
- `listingOrder` (number, obligatory but can be implicit from the serialization) The position of this example among other examples in the same sense.

Children:

- `Label`

Note on the interplay between `source` and `sourceElaboration`: TBD…

Note on example sources: You can use the `Source` datatype to explain the meaning of the sourcre IDs and to constrain which source IDs are allowed to occur in your lexicographic resource.

`Source`

Represents one (of many) possible values for the `source` property of `Example` and explains what it means.

Child of:

- `LexicographicResource`

Properties:

- `id` (obligatory, string) An abbreviation, a code or some other string of text which identifies the source.
- `description` (optional, string) A human-readable description of the source, for example a book title.
- `url` (optional, string) A URL a human reader might click to navigate to the source or to information about the source.

A sourced example:

```
LexicographicResource: language="en"
    ExampleSource:
        id="wodehouse1967"
        description="P.G. Wodehose (1967) The World of Jeeves."
    Entry: headword="pack"
        Sense:
            Example:
```

```
            text="Have you started packing yet, Jeeves?"
            source="wodehouse1967"
```

A sourced example with additional elaboration of where in the source it comes from:

```
LexicographicResource: language="en"
    ExampleSource
        id="wodehouse1967"
        description="P.G. Wodehose (1967) The World of Jeeves."
    Entry: headword="pack"
        Sense:
            Example:
                text="Have you started packing yet, Jeeves?"
                source="wodehouse1967"
                sourceElaboration="p. 314 line 20"
```

An example with ad-hoc attribution:

```
LexicographicResource: language="en"
    Entry: headword="pack up"
        Sense:
            Example:
                text="Pack up and go!"
                sourcElaboration="overheard in Dublin"
```

Note about referential integrity: If you want, you can design your implementation to enforce referential integrity between `source` properties of `Example` and `Source` objects. In other words, you can make it so that the tags you define in `Source` objects are the only values allowed for `source` properties of `Example`. However, doing this is optional in DMLex. An implementation of DMLex is compliant regardless of whether it enfoces referential integrity between `Source` objects and `source` properties.

`Tag`

Represents one (of many) possible values for the `tag` property of `PartOfSpeech`, `InflectedForm` and `Label`, and explains what it means.

Child of:

- `LexicographicResource`

Properties:

- `tag` (obligatory, string) An abbreviation, a code or some other string of text which identifies the source.
- `description` (optional, string) A human-readable description of what the tag means

- `subtype` (optional, one of: `partOfSpeech`, `inflectedForm`, `label`) Whether this tag is intended for `PartOfSpeech` objects, `InflectedForm` objects or `Label` objects. If omitted, then all.
- `partOfSpeech` (optional, string) If present, then it says that this tag is only intended to be used inside entries that are labelled with this part of speech.

Children:

- `TagMapping` 

`TagMapping`

Represents the fact that a tag (defined by a `Tag` object) is equivalent to an item available from an external authority.

Child of:

- `Tag`

Properties:

- `sameAs` (string, obligatory) A URI of an item in an external inventory.

Guidance: working with tags

Explaining the meaning of tags

The `tag` property of `PartOfSpeech`, `InflectedForm` and `Label` is meant to be a (mainly) machine-redable code or abbreviation:

`TBD...`

Optionally, you can use `Tag` objects to explain what the codes/abbreviations mean:

`TBD...`

Note about referential integrity: If you want, you can design your implementation to enforce referential integrity between `tag` properties and `Tag` objects. In other words, you can make it so that the tags you define in `Tag` objects are the only values allowed for `tag` properties. However, doing this is optional in DMLex. An implementation of DMLex is compliant regardless of whether it enfoces referential integrity between `Tag` objects and `tag` properties.

Defining the subtype of tags

Additionally, you can use the `subtype` property of `Tag` objects to say which tags are intended to be used as part-of-speech labels (in `PartOfSpeech` objects), which as inflected form labels (in `InflectedForm` objects), and which as pragmatic labels (in `Label` objects).

```
TBD...
```

In this example, the tags `n`, `v` and `adj` are intended to be used in `PartOfSpeech` objects, the tags `pl`, `past` and `compar` in `InflectedForm` objects, and the tags `inf` and `colloq` in `Label` objects. A `Tag` object which does not have a `scope` property can be used for all three:

Note about subtype enforcement: If you want, you can design your implementation to enforce the constraints expressed by the `subtype` properties of `Tag` objects. However, doing this is optional in DMLex: an implementation of DMLex is compliant regardless of whether it does this or not.

Constraining the combinability of tags

Finally, you can use the `partOfSpeech` property to express the fact that certain tags can only be used with certain parts of speech:

```
TBD...
```

This example expresses the constraint an `InflectedForm tag="pl"` should only to occur inside an entry that also has a `PartOfSpeech tag="n"`. This following entry complies with that constraint:

```
TBD
```

Whereas the following entry violates it:

```
TBD
```

Note about constraint enforcement: If you want, you can design your implementation to enforce the constraints expressed by the `partOfSpeech` properties of `Tag` objects. However, doing this is optional in DMLex: an implementation of DMLex is compliant regardless of whether it does this or not.