Technical Notes on LIFT used in FLEx

Ken Zook

August 15, 2023

Contents

1 Introduction

LIFT (Lexicon Interchange FormaT) is an XML format for lexical information (dictionaries). It was designed by SIL to provide a standard for transferring lexical data between programs. FLEx, WeSay, Lexique Pro, and Dictionary App Builder are some of the programs that use or export/import LIFT data. See https://github.com/sillsdev/lift-standard for details of the LIFT standard. Although version 0.15 has been defined, at this point programs are all using version 0.13. The description for this version is at https://github.com/sillsdev/lift-standard/blob/master/lift_13.pdf. Note Lexique Pro does not match these specifications for variant and subentries, so if you try to import into FLEx, you'll need to fix the links first.

In its current state, LIFT supports much of what's in the FLEx lexicon, but it doesn't cover everything, and some things do not have enough detail to accurately keep track of data during Send/Receive with WeSay, which currently uses LIFT. It's unlikely that SIL will pursue further development of LIFT.

LIFT export produces a simplified XML format of the lexical data in LIFT that may be useful for further processing. LIFT import provides a way to import lexical data into FLEx from a LIFT file. It can add new entries as well as modify existing entries. The format is more complex than Standard Format (SFM) data, but it has the advantage of providing more accurate structure of the FLEx data as well as a way to modify existing entries.

The LIFT standard is fairly generic. FLEx has a specific way to store the FLEx model of a lexicon into the LIFT standard. This document describes the specific way FLEx data is represented in LIFT as of FieldWorks 9.1.5.

Here is a minimal LIFT file that would add a new entry for French 'homme' to FLEx via File…Import…LIFT Lexicon.

```
<?xml version="1.0"?>
<lift version="0.13">
<entry id="e1">
<lexical-unit>
<form lang="fr"><text>homme</text></form>
</lexical-unit>
</entry>
</lift>
```

All data in a LIFT file is stored in UTF-8 encoding using NFC normalization.

A LIFT file should always be stored in a folder that contains

- the .lift file containing lexical data,
- a .lift-ranges file containing various lists that are used by the data,
- a WritingSystems folder containing writing system .ldml files used in the LIFT file,
- a "pictures" folder containing pictures used in the LIFT file, and
- an "audio" folder containing audio and/or video files used in the LIFT file.

Everything other than the LIFT file are optional.

2 Header

The header element is optional in a LIFT file It provides a place to define fields that are not built into the basic LIFT structure. It also allows a definition of lists that can be referenced. A complete LIFT file for FLEx normally has these sections:

```
<?xml version="1.0"?>
<lift version="0.13">
<header>
<ranges>
…range elements
</ranges>
<fields>
…field elements
</fields
</header>
…entry elements
</lift>
```

2.1 Ranges

The ranges element contains a list of range elements. Each range element has an id attribute that lists the range name, and an href attribute that lists a full path to the lift-ranges file that holds the range definition. Although full paths are specified, the file will still open and work in other locations as well.

```
<range id="status" href="file://C:/Users/zook/Desktop/TLP/TLP.lift-ranges"/>
```

Range definitions can be stored directly in the LIFT file, but FLEx stores the definitions in a separate .lift-ranges file. The file contains multiple list ranges and range-elements (or list items). The range element has an id attribute that matches the id in the range elements in the LIFT file.

Range-element has an id attribute which is the name of the item, and an optional guid attribute which is the guid for the item. Range-element can have these elements, each with one or more writing systems.:

- label: the name of the item.
- abbrev: the abbreviation of the item.
- description: the description of the item.

This is an abbreviated sample for a .lift-ranges file:

```
<?xml version="1.0" encoding="UTF-8"?>
<lift-ranges>
<range id="semantic-domain-ddp4">
<range-element id="1 Universe, creation" guid="63403699-07c1-43f3-a47c-069d6e4316e5">
<label>
<form lang="en"><text>Universe, creation</text></form>
</label>
<abbrev>
<form lang="en"><text>1</text></form>
</abbrev>
<description>
<form lang="en"><text>Use this domain for general words referring to the physical universe.
Some languages may not have a single word for the universe and may have to use a phrase
such as 'rain, soil, and things of the sky' or 'sky, land, and water' or a descriptive phrase such
as 'everything you can see' or 'everything that exists'.</text></form>
</description>
</range-element>
… other range-elements
</range>
… other ranges
</lift-ranges>
```

Note that on import into FLEx, any references to range elements in senses and entries will try to find an existing item in the FLEx list. If not found, a new item will be added to the FLEx list, and the Import Log file will list the fact that a new item has been added to the list.

2.2 Fields

The fields element contains a sequence of field elements, or names, that are not covered by the basic LIFT elements. The purpose of the fields element is to document fields that are not in the basic LIFT standard. The fields element is not used during import, except for FLEx custom fields. Without a field definition for a custom field, the import will create a custom field in the target project, but it defaults to a MultiUnicode field with kwsAnalVerns as the selector. With a field element for the custom field, it will add missing custom fields with the desired types.

All field elements have a tag attribute that is the field name, and a form element that gives information about the field in one or more writing systems.

```
<field tag="comment">
<form lang="en"><text>This records a comment (note) in a LexEtymology in
FieldWorks.</text></form>
</field>
```

FLEx custom fields add a special qaa-x-spec ‘writing system’ that includes the following information in a text to specify the type of custom field being used.

- Class: The FLEx class name owning this field.
- Type” The FLEx basic object type (e.g., String, MultiUnicode, etc.) See Section 2.7 of

https://downloads.languagetechnology.org/fieldworks/Documentation/FieldWorks_7_XML_model.pdf for a list of possible types.

- WsSelector: The FLEx default writing system classification
  kwsAnal: The first analysis writing system
  kwsVern: The first vernacular writing system
  kwsAnals: All analysis writing systems
  kwsVerns: All vernacular writing systems
  kwsAnalVerns: All analysis then all vernacular writing systems
  kwsVernAnals: All vernacular then all analysis writing systems

```
<field tag="Custom Field">
<form lang="en"><text>This is a custom single line field</text></form>
<form lang="qaa-x-spec"><text>Class=LexEntry; Type=String;
WsSelector=kwsAnal</text></form>
</field>
```

3 Basic elements

3.1 Id, Guid

Entries and senses require id fields. There is an exception where the entry id is not needed if you have a guid attribute instead. Id fields can have any text as long as it is unique in the LIFT file. Guid fields must have a valid guid. For creating a new file for importing new entries, you could simply use e1, e2, … for entry ids and s1, s2,… for sense ids. Any non-guid id will be converted to a guid during import.

The WeSay entry id used the lexeme form and a guid as the ID, FLEx continued that tradition.

```
id="*hindoksa_016f2759-ed12-42a5-abcb-7fe3f53d05b0"
```

Entries also have a guid attribute which is used in FLEx as the primary ID.

```
guid="04758355-1e6c-49de-86f0-15d6347f25cd"
```

The sense id has always been a guid

```
id="1ac5aea9-038c-4e51-8655-02483a5e5f22"
```

References to entries and senses within the LIFT file use the id string.

3.2 Date and time

Date and time are stored in this format, YYYY-MM-DDTHH:MM:SSZ using the time in UTC.

```
dateCreated="2018-09-19T18:48:27Z"
```

Entries have a dateCreated attribute and a dateModified attribute.

3.3 Numbers

Numbers, such as homorgraph fields, etc., are stored as a decimal number.

```
order="1"
value="1971"
```

3.4 Strings

The minimal string has a lang attribute with the writing system identifier, and a text element containing the text.

```
<gloss lang="en"><text>English gloss</text></gloss>
```

Strings may contain embedded styles (defined in Format…Styles), writing systems (defined in Format…Set up Writing Systems), etc. Embeddings are handled with the span element embedded in the text element. This example shows an embedded span for the French language using the Strong style.

```
<form lang="en"><text>English with <span lang=”fr” class="Strong"> français</span>
embedded.</text></form>
```

A MultiUnicode field, such as gloss, that has multiple translations without embeddings is shown by repeating the field. Note that even though you could add spans to these types of fields, the text from the spans will be saved, but none of the attributes of spans will be saved since embeddings are not allowed in the FLEx model.

```
<gloss lang="en"><text>English gloss</text></gloss>
<gloss lang="fr"><text>Gloss anglais</text></gloss>
```

A MultiString field, such as definition, that has multiple translations that can have embeddings, is represented by a single field with multiple form elements. Note that within a MultiString field, each form must have a unique writing system.

```
<definition>
<form lang="en"><text>A small red fruit.</text></form>
<form lang="fr"><text> Un petit fruit rouge.</text></form>
</definition>
```

External references to URLs can be embedded in a string using Edit...Paste Hyperlink. This example can be used in Webonary to make a live link to an appendix in the webonary dictionary. The href attribute in the span gives the URL, and a Hyperlink style for the visible linked text.

```
<form lang="en"><text>(See Appendix 5: <span
href="https://www.webonary.org/tausug/language/appendices/plants/?lang=en"
class="Hyperlink">Plants</span>)</text></form>
```

Edit…Copy Location as Hyperlink provides a way to link to an entry or other location in another FLEx project. This uses a silfw: hyperlink which will open a project if not already open, and jump to the target location.

```
<form lang="en"><text>See <span
href="silfw://localhost/link?database%3dthis%24%26tool%3dlexiconEdit%26guid%3d47359
e87-527f-4105-8589-029ed0a08532%26tag%3d" class="Hyperlink">cow</span>
entry.</text></form>
```

Within FLEx, Shift+Enter forces a line break in a string by inserting a U+2028 LINE SEPARATOR in the string. In a LIFT file, this is represented by an actual CR/LF in the middle of a string. Any actual CR/LF in the middle of a string will be converted to a LINE SEPARATOR on import.

```
<form lang="en"><text>A note with a Shift+Return
in it.</text></form>
```

Note that in LIFT, you can embed things using spans in any string. However, actual FLEx strings are defined in the model to allow embedding (String) and not allow embedding (Unicode). If a LIFT file has embedding in a Unicode field, on import into FLEx, the text will be kept, but all span hierarchy will be lost including styles, writing systems, etc.

Note in XML strings you need to quote ampersand, less than, and greater than codes in the string
& = &
< = <
> = >

3.5 Custom fields

Field elements are used to define custom fields in the header, and to insert data in objects following the header. Without a fields element, custom fields will be added to the project on import, but will default to MultiUnicode with selector kwsAnalVerns. So if the target project is missing the needed custom fields, you should include a fields element in the header giving the desired specifications for the custom fields. See section 2.2 for more information on fields.

This is a custom field definition in the header.

```
<header>
<fields>
<field tag="Cust Single Line">
<form lang="en"><text>This is a custom single line field</text></form>
<form lang="qaa-x-spec"><text>Class=LexEntry; Type=String;
WsSelector=kwsAnal</text></form>
</field>
</fields>
</header>
```

This inserts data in an entry.

```
<field type=" Cust Single Line ">
<form lang="en"><text>Data in custom field.</text></form>
</field>
```

4 References to entries and senses

There are three types of references to entries and senses in FLEx that will be discussed in the sections below.

4.1 Lexical Relations

Lexical relations are defined in the Lexical Relations list in the Lists area of FLEx. Each relation contains a name, and some relations have a reverse name. The Reference set type specifies whether this reference is between senses, entries, or both, and whether the reference can have multiple references or not, and whether the order is important or not.

References to Entries show up in the Cross References section of the entry. References to senses show up in the Lexical Relations section of the sense. The examples given in this section are all sense to sense relations, so would show up under Lexical Relations in the sense.

Note: The ref attribute must have an existing target. LIFT import will not create any objects to fulfill a reference. Any ref targets that cannot be located are simply ignored without any indication that something went wrong. When ref targets are not found for variants or complex forms, FLEx may hang and require Task Manager to kill the program.

Note: If a relation type does not exist, a new type will be created in the Lexical Relation list and used during the import.

4.1.1 Collection

Collection relations (e.g., synonyms) are stored as a sequence of relation elements with a type attribute with the name of the lexical relation, and a ref attribute with the guid (for sense) or id (for entry) of the other senses or entries. Each entry or sense will reference the other items in the collection. These would be the links for a synonym set for *good*, *honorable*, and *respectable*.

Note: The import process will try to unify the relation sets. For example, if you have a ref b, and b ref c, the result will be a single relation set containing a, b, c. If collection references overlap in a way that cannot be unified, the import log will list multiple Combined Collections explaining what reference was added to some other set. If there are many of these, the result will probably be very bad. In these cases, you should use a Unidirectional type, or some other type other than Collection.

respectable (7c1d2aca-140c-4a2a-81f7-38a9061027c1) *syn:* **good, honorable**

```
<relation type="Synonyms" ref="6bb61874-125e-42e8-9f23-cbc119f39b91"/>
<relation type="Synonyms" ref="fcdf2612-4260-4042-af88-60432bc07bff"/>
```

good (6bb61874-125e-42e8-9f23-cbc119f39b91) *syn:* **honorable, respectable**

```
<relation type="Synonyms" ref="fcdf2612-4260-4042-af88-60432bc07bff"/>
<relation type="Synonyms" ref="7c1d2aca-140c-4a2a-81f7-38a9061027c1"/>
```

honorable (fcdf2612-4260-4042-af88-60432bc07bff) *syn:* **good, respectable**

```
<relation type="Synonyms" ref="6bb61874-125e-42e8-9f23-cbc119f39b91"/>
<relation type="Synonyms" ref="7c1d2aca-140c-4a2a-81f7-38a9061027c1"/>
```

4.1.2 Pair

Pair relations (e.g., antonyms) are stored as a relation element with a single type attribute with the name of the lexical relation, and a ref attribute with the guid (for sense) or id (for entry) of the other sense or entry. Each entry or sense will reference the other item in the set. These would be the links for an antonym set for *good*, and *bad*.

good (6bb61874-125e-42e8-9f23-cbc119f39b91) *ant:* **bad**

```
<relation type="Antonym" ref="4e31faea-0916-4b14-9ac9-9b5b1bc5d326"/>
```

bad (4e31faea-0916-4b14-9ac9-9b5b1bc5d326) *ant:* **good**

```
<relation type="Antonym" ref="6bb61874-125e-42e8-9f23-cbc119f39b91"/>
```

4.1.3 Pair – 2 relation names

Pair relations with two relation names (e.g., operator and operator of) are stored as a relation element with a single type attribute with the name of the lexical relation, and a ref attribute with the guid (for sense) or id (for entry) of the other sense or entry. Each entry or sense will reference the other item in the set. These would be the links for an operator set for *pilot* and *plane*.

pilot (8149fae6-22b6-42dd-9dd0-59245ffc9acb) *op of:* **plane**

```
<relation type="Operator of" ref="e64357e4-6f55-43bb-9643-beb69fec03e7"/>
```

plane (e64357e4-6f55-43bb-9643-beb69fec03e7) *op:* **pilot**

```
<relation type="Operator" ref="8149fae6-22b6-42dd-9dd0-59245ffc9acb"/>
```

4.1.4 Tree

Tree relations (e.g., part/whole) are stored as a sequence of relation elements with a type attribute with the name of the lexical relation, and a ref attribute with the guid (for sense) or id (for entry) of the other senses or entries. Each entry or sense will reference the other items in the set. This example demonstrates two part/whole relations where *house* has parts *roof* and *room*, and second, *room* has parts *floor* and *furniture*. A whole sense can have any number of parts, but a part can only have one whole. This forms a tree hierarchy of part/whole senses. Note that room is connected both as having parts floor and furniture, but it also has a whole relationship to house.

house (40fc11e6-efe8-4ca7-b154-83ad2ca49335) *pt:* **roof, room**

```
<relation type="Part" ref="cf0d9191-4e19-4df3-a9d2-dd2884320024"/>
<relation type="Part" ref="1dd34f2b-c839-4477-a472-986200e65b6a"/>
```

roof (cf0d9191-4e19-4df3-a9d2-dd2884320024) *wh:* **house**

```
<relation type="Whole" ref="40fc11e6-efe8-4ca7-b154-83ad2ca49335"/>
```

room (1dd34f2b-c839-4477-a472-986200e65b6a) *pt:* **floor, furniture;** *wh:* **house**

```
<relation type="Part" ref="ea52559a-b492-475c-a226-bafb0b8ffa0a"/>
<relation type="Part" ref="063af53f-7b7f-40c2-bad5-4ff2862495fb"/>
<relation type="Whole" ref="40fc11e6-efe8-4ca7-b154-83ad2ca49335"/>
```

floor (063af53f-7b7f-40c2-bad5-4ff2862495fb) *wh:* **room**

```
<relation type="Whole" ref="1dd34f2b-c839-4477-a472-986200e65b6a"/>
```

furniture (ea52559a-b492-475c-a226-bafb0b8ffa0a) *wh:* **room**

```
<relation type="Whole" ref="1dd34f2b-c839-4477-a472-986200e65b6a"/>
```

4.1.5 Sequence/Scale

Sequence or scale relations (e.g., calendar or days of week) are stored as a sequence of relation elements with a type attribute with the name of the lexical relation, and a ref

attribute with the guid (for sense) or id (for entry) of the other senses or entries. Because order is important in these elements, there is also an order attribute that starts at 1. Each entry or sense will reference all items in the collection, including the current sense. The difference from a collection is that these senses are always in a fixed order, and the current item is always shown as part of the list. This example shows the links for a calendar set of *Monday*, *Tuesday*, and *Wednesday*.

Monday (fd4b33d5-1be3-430b-b767-96caa9ffc8a4) *cal:* **Monday, Tuesday, Wednesday**

```
<relation type="Calendar" ref="fd4b33d5-1be3-430b-b767-96caa9ffc8a4" order="1"/>
<relation type="Calendar" ref="823397a9-9aef-4697-88b6-32a78b08d883" order="2"/>
<relation type="Calendar" ref="b537970e-5ba6-4f54-b4c9-24aeed08d27a" order="3"/>
```

Tuesday (823397a9-9aef-4697-88b6-32a78b08d883) *cal:* **Monday, Tuesday, Wednesday**

```
<relation type="Calendar" ref="fd4b33d5-1be3-430b-b767-96caa9ffc8a4" order="1"/>
<relation type="Calendar" ref="823397a9-9aef-4697-88b6-32a78b08d883" order="2"/>
<relation type="Calendar" ref="b537970e-5ba6-4f54-b4c9-24aeed08d27a" order="3"/>
```

Wednesday (b537970e-5ba6-4f54-b4c9-24aeed08d27a) *cal:* **Monday, Tuesday, Wednesday**

```
<relation type="Calendar" ref="fd4b33d5-1be3-430b-b767-96caa9ffc8a4" order="1"/>
<relation type="Calendar" ref="823397a9-9aef-4697-88b6-32a78b08d883" order="2"/>
<relation type="Calendar" ref="b537970e-5ba6-4f54-b4c9-24aeed08d27a" order="3"/>
```

4.1.6 Unidirectional

Unidirectional relations are a sequence of senses that are related in some way with the current sense. Unlike all of the other relations, there is no automatic reverse relation shown with this type. They are stored as a sequence of relation elements with a type attribute with the name of the lexical relation, and a ref attribute with the guid (for sense) or id (for entry) of the other senses or entries. As an example, a *baby* is related to a *mother*, *crib*, and *diaper* in some way. Also *mother* is related to a *baby*, but not directly to a *crib* or *diaper*. In this type, the reference from *mother* to *baby* requires a separate related set.

baby (a29f302c-1500-439c-bc36-8b1911ca04e2) *rel:* **mother, crib, diaper**

```
<relation type="Related" ref="9758fd2d-9133-4d7d-baab-dfd8c0866f4e"/>
<relation type="Related" ref="28bbec67-0f82-4ebc-86f4-a0d8bbf34699"/>
<relation type="Related" ref="4777eb1a-d437-424c-88e3-47095b052524"/>
```

mother (9758fd2d-9133-4d7d-baab-dfd8c0866f4e) *rel:* **baby**

```
<relation type="Related" ref="a29f302c-1500-439c-bc36-8b1911ca04e2"/>
```

crib (28bbec67-0f82-4ebc-86f4-a0d8bbf34699) <no reference displayed>

diaper (4777eb1a-d437-424c-88e3-47095b052524) <no reference displayed>

4.2 Variant entries

Variant entries in FLEx are full lexical entries, but are linked to the main entries.

Note the variant element in LIFT represents allomorphs, not a variant relationship between entries.

The variant entry has one or more relation elements with a _component-lexeme type attribute. and a ref attribute with the guid (for sense) or id (for entry) of the main sense or entry. An order attribute also records the printed order of the given relation, starting at 0. There is also a trait element that has a variant-type name attribute and a value attribute that holds the name of the variant type from the Variant Types list in the Lists area.

Note the main entry does not have any indication that it has variants.

mom (mom_a42eba12-a001-4fec-b987-f6842670c4ed) fr. var. of **mother**

```
<relation type="_component-lexeme" ref="mother_9056dc3f-d0e9-4bbb-87e3-4932ce55796f"
order="0">
<trait  name="variant-type" value="Free Variant"/>
</relation>
```

mother (mother_9056dc3f-d0e9-4bbb-87e3-4932ce55796f) (fr. var. **mom**)

A variant entry can be a variant of multiple main entries. It's not clear why the *mainb* relation in this example has an is-primary trait. The variant can also include a comment as indicated in the second example below. This summary information goes into LexEntryRef_Summary and shows up in the Variant data entry section of main entries.

Note: Until LT-21075 is fixed, you will get error messages when finding main entries if the entry id contains most Unicode characters, such as U+0331 diacritic. In some cases this causes the import to simply hang without completion, requiring killing FLEx with Task Manager to get out of the problem.

maina (maina_966bbd95-108f-41e4-866e-ade49fb40755) (fr. var. **var**)

mainb (mainb_0c34c932-bb12-4b8e-af3f-9d2838333201) (fr. var. **var**)

var (var_bf62078c-8cb8-4f83-8e1c-8db7c649107b) **var** fr. var. of **maina, mainb**

```
<relation type="_component-lexeme" ref="maina_966bbd95-108f-41e4-866e-ade49fb40755"
order="0">
<trait  name="variant-type" value="Free Variant"/>
</relation>
<relation type="_component-lexeme" ref="mainb_0c34c932-bb12-4b8e-af3f-9d2838333201"
order="1">
<trait name="is-primary" value="true"/>
<trait  name="variant-type" value="Free Variant"/>
<field type="summary">
<form lang="en"><text>Comment on variant</text></form>
</field>
</relation>
```

In FLEx, if the Show Minor Entry checkbox is unchecked, then the corresponding relation has a trait element with a hide-minor-entry name attribute and 1 in the value attribute.

```
<trait name="hide-minor-entry" value="1"/>
```

4.3 Complex forms

Complex forms (frequently subentries) are full entries in FLEx. In the main entry, each component reference is stored in a relation element with a _component-lexeme type attribute, a ref attribute with the guid (for sense) or id (for entry) of the complex form sense or entry, and an order attribute starting at 0 indicating the order of the components in the display. It also contains a trait element with a complex-form-type name attribute, and a value attribute set to the name of the complex form type item in the Complex Form Types list in the Lists area. A trait element with an is-primary name attribute and a true value attribute is included if the subentry is supposed to show under the referenced headword in the relation element. In FLEx this is set in the Show Subentry under field.

Note: Until LT-21075 is fixed, you will get error messages when finding main entries if the entry id contains most Unicode characters, such as U+0331 diacritic. In some cases this causes the import to simply hang without completion, requiring killing FLEx with Task Manager to get out of the problem.

unkissable (unkissable_1cf9ae39-b10f-4a6d-b0df-da6c48bc0150) (der. of **un-**, **kiss**, **-able**, see under **kiss**)

```
<relation type="_component-lexeme" ref="un-_c5cc0bcc-61c5-4d4e-9a52-714c1d884f89"
order="0">
<trait  name="complex-form-type" value="Derivative"/>
</relation>
<relation type="_component-lexeme" ref="kiss_2f59b915-deff-49ef-a505-665baa1ecb3a"
order="1">
<trait name="is-primary" value="true"/>
<trait  name="complex-form-type" value="Derivative"/>
</relation>
<relation type="_component-lexeme" ref="-able_b818692a-259a-4535-be24-3580a3d40c80"
order="2">
<trait  name="complex-form-type" value="Derivative"/>
</relation>
```

un- (un-_c5cc0bcc-61c5-4d4e-9a52-714c1d884f89) der. **unkissable** (see under **kiss**)

kiss (kiss_2f59b915-deff-49ef-a505-665baa1ecb3a) **unkissable** der.

-able (-able_b818692a-259a-4535-be24-3580a3d40c80) der. **unkissable** (see under **kiss**)

Complex forms (subentries) can be part of more than one main entry.

maina (maina_966bbd95-108f-41e4-866e-ade49fb40755) **sub** der.

mainb (mainb_0c34c932-bb12-4b8e-af3f-9d2838333201) **sub** der.

sub (sub_74fb2464-ce1d-4176-acf7-6ad887eca2de) (der. of **maina, mainb**)

```
<relation type="_component-lexeme" ref="maina_966bbd95-108f-41e4-866e-ade49fb40755"
order="0">
<trait name="is-primary" value="true"/>
<trait  name="complex-form-type" value="Derivative"/>
</relation>
<relation type="_component-lexeme" ref="mainb_0c34c932-bb12-4b8e-af3f-9d2838333201"
order="1">
```

```
<trait name="is-primary" value="true"/>
<trait  name="complex-form-type" value="Derivative"/>
</relation>
```

In FLEx, if the Show Minor Entry checkbox is unchecked, then the corresponding relation has a trait element with a hide-minor-entry name attribute and 1 in the value attribute.

```
<trait name="hide-minor-entry" value="1"/>
```

4.4 Minor entries

Minor entries for variants or complex forms will show in the main dictionary if the Show Minor Entry checkbox is checked at the bottom of the entry. When unchecked, the corresponding relation element has a trait element with a hide-minor-entry name attribute and 1 in the value attribute.

```
<trait name="hide-minor-entry" value="1"/>
```

5 Entry elements

Entries in FLEx can be main entries, variant entries, or complex forms. Each of these are actual entries, but they are linked in special ways.

A lexical entry consists of the various entry fields, and any number of senses. This section focuses on the entry fields. The entry element holds all information for an entry and has the following start-tag attributes:

- dateCreated: date and time the entry was created
- dateModified: date and time the entry (including senses) was last modified
- dateDeleted: date and time the entry was deleted. If used, the entry must have a valid guid, and a date that parses correctly. On import into FLEx, the designated entry will be deleted.
- id: unique id used for referencing entries in the LIFT file. It normally consists of the lexeme form and guid for the entry.
- guid: a guid identifying the LexEntry object in FLEx
- order: a homograph number for the entry, starting at 1.

All of these attributes are optional except the id. This is a typical entry start-tag for an entry element.

```
<entry dateCreated="2021-03-24T15:51:45Z" dateModified="2021-03-24T15:52:11Z"
id="combine1_10bbb84e-7964-485e-bf94-7c618d2b07a9" guid="10bbb84e-7964-485e-bf94-
7c618d2b07a9" order="1">
```

This is a simple entry in LIFT that will add the entry and sense to the project on import.

```
<entry id="e1">
<lexical-unit>
<form lang="fr"><text>rouge</text></form>
</lexical-unit>
<sense id="s1">
<grammatical-info value="adjective"></grammatical-info>
<gloss lang="en"><text>red</text></gloss>
```

```
<definition>
<form lang="en"><text>The color red.</text></form>
</definition>
</sense>
</entry>
```

The following entry in a LIFT import will delete the entry with the specified guid.

```
<entry guid="5ddc014a-b5e4-400d-8ad3-a812c32c8c17" dateDeleted="2000-01-
01T00:00:00Z">
</entry>
```

The following entry fields are ordered as they appear in the FLEx data entry fields.

5.1 Lexeme Form

Lexeme forms are stored in a lexical-unit element that contains one or more form elements listing the form in each writing system. Audio or video files can be referenced using a special audio writing system. Only one file can be included in the audio writing system. The file path is relative to the LinkedFiles\AudioVisual FLEx project directory, so is typically just a file name. The audio writing system can handle .wav and .mp3 files.

```
<lexical-unit>
<form lang="krx"><text>wol</text></form>
<form lang="fr-Zxxx-x-audio"><text> 638277086107188840test.wav</text></form>
</lexical-unit>
```

5.2 Is Abstract Form

LIFT does not support Is Abstract Form.

5.3 Morph Type

The entry morpheme type is stored in a morph-type trait.

```
<trait  name="morph-type" value="stem"/>
```

5.4 Environments

The entry environments are stored in an environment trait.

```
<trait name ="environment" value="/ [C] _"/>
```

5.5 Stem Name

Stem name can be chosen when a category in the Grammatical Info Details have stem names defined. This applies both to the entry and allomorphs of the entry.

LIFT does not support Stem Name.

5.6 Citation Form

Citation form is in a citation element with different writing systems in form elements.

```
<citation>
<form lang="krx"><text> kaawol</text></form>
</citation>
```

5.7 Dialect Labels (Entry)

Each dialect label is stored as a dialect-labels trait using the name of the dialect in the Dialect Labels list in FLEx.

```
<trait  name="dialect-labels" value="City"/>
```

5.8 Complex Forms

In FLEx, a complex form (typically a subentry) is an entry that is linked to a main entry via a complex form type object. In LIFT, this linkage is made via a relation element that refers to the main entry or sense.

This is an example of *mondongani* that is a derivative of *dongan*. This is the complex form entry:

```
<entry id="mondongani">
<lexical-unit>
<form lang="blz"><text>mondongani</text></form>
</lexical-unit>
<trait  name="morph-type" value="stem"/>
<relation type="_component-lexeme" ref="dongan" order="0">
<trait name="is-primary" value="true"/>
<trait  name="complex-form-type" value="Derivative"/>
<trait name="hide-minor-entry" value="1"/>
</relation>
</entry>
```

The relation element has attributes:

- type: the type of relation. Complex forms and variants both use the type "_component-lexeme", which represents the LexEntryRef object in the FLEx model.
- ref: A link to another entry or sense id.
- order: order of the complex form when there is more than one for a given entry. The order starts at 0.

Other child elements of a _component-lexeme relation:

- trait name="is-primary”: If this is present, and set to “true”, then it will generate a subentry in a root-based view.
- trait name="complex-form-type; This identifies this relation as a complex form relation, and gives the name of the Complex Form Type (e.g., Derivative).
- trait name="hide-minor-entry: Complex forms (typically subentries) will generate a minor entry in the dictionary pointing to the main entry with the subentry unless this trait is present with a value of “1”, which then blocks this minor entry from showing.

This is the main entry. Note that this entry does not have any reference to its complex form entries. The links are only on the complex forms.

```
<entry id="dongan">
<lexical-unit>
<form lang="blz"><text>dongan</text></form>
</lexical-unit>
<trait  name="morph-type" value="stem"/>
</entry>
```

5.9 Components

The Components field lists main entries for complex forms. LIFT does not support this directly. It's a side-effect of the one-way Complex Form links.

5.10 Variant of

The Variant of field lists main entries for variant forms. LIFT does not support this directly. It's a side-effect of the one-wayVariant links.

5.11 Pronunciation

Each pronunciation is stored in a pronunciation object. It may contain:

- Pronunciation: one or more forms in form elements.
- Media File and Label: one or more media elements with the file name in a href attribute. The file name is relative to the project LinkedFiles\AudioVisual directory. It may also have a label for the media file in a label element with one or more form elements for the label string.
- CV Pattern: stored in a field element with a cv-pattern type attribute and form element for the string.
- Tone: stored in a field element with a tone type attribute and form element for the string.
- Location: A single location is stored in a location trait with the string in a value attribute. Locations are referenced to items in the Locations list.

LIFT does not support Publish Pronunciation In.

```
<pronunciation>
<form lang="krx"><text>Pronunciation</text></form>
<form lang="krx-fonipa"><text>pronIpa</text></form>
<media href="apple.wav">
<label>
<form lang="en"><text>Pron label</text></form>
</label>
</media>
<field type="cv-pattern">
<form lang="en"><text>CVVCV</text></form>
</field>
<field type="tone">
<form lang="en"><text>High</text></form>
</field>
<trait name="location" value="jungle"/>
</pronunciation>
```

5.12 Etymology

Each Etymology object is stored in an etymology element which may contain

- Preceding Annotation: stored in a field element with a preccomment type.
- Source Language: One or more languages from the Languages list with each one having a trait named languages with the value attribute holding the language name.

- Source Language Notes: stored in a field element with a languagenotes type with one or more form elements specifying the writing system for each one.
- Source Form: stored in one or more form elements specifying the writing system for each one.
- Gloss: stored in one or more gloss elements specifying the writing system for each one.
- Following Comment: stored in a field of type comment with one or more form elements specifying the writing system for each one.
- Bibliographic Source: stored in a field element with a bibliography type with one or more form elements specifying the writing system for each.
- Etymology Note: stored in a field element with a note type with one or more form elements specifying the writing system for each.

The etymology start-tag element may have obsolete type, and source attributes, but they are ignored in import and not exported. These were probably used prior to the etymology fields being expanded in FW8.3.

```
<etymology>
<form lang="krx"><text>source form</text></form>
<gloss lang="en"><text>English gloss</text></gloss>
<gloss lang="fr"><text> Gloss français</text></gloss>
<field type="comment">
<form lang="en"><text>English following comment</text></form>
<form lang="fr"><text> Commentaire suivant en français</text></form>
</field>
<field type="preccomment">
<form lang="en"><text>English preceding annotation</text></form>
</field>
<trait name="languages" value="German"/>
<trait name="languages" value="Swiss"/>
<field type="note">
<form lang="en"><text>English etymology note</text></form>
</field>
<field type="bibliography">
<form lang="en"><text>English bibliographic source</text></form>
</field>
<field type="languagenotes">
<form lang="en"><text>English source language notes</text></form>
</field>
</etymology>
```

5.13 Note

This is for a note on an entry. It is actually stored in the Comment property on LexEntry in fwdata, but has a Note label in data entry. This is stored in a note element with no attribute, with one or more form elements with the text in each writing system.

```
<note>
<form lang="en"><text>English note</text></form>
<form lang="fr"><text> Note française</text></form>
</note>
```

5.14 Literal Meaning

A literal meaning for an Entry can be given in multiple writing systems using the field element with literal-meaning type, and one or more form elements for text in each writing system.

```
<field type="literal-meaning">
<form lang="en"><text>English literal meaning</text></form>
</field>
```

5.15 Bibliography

A bibliography field for an Entry can be given in multiple writing systems using the note element with a bibliography type attribute, and one or more form elements for text in each writing system.

```
<note type=" bibliography ">
<form lang="en"><text>English bibliography</text></form>
</field>
```

5.16 Restrictions

Restrictions for an Entry can be given in multiple writing systems using the note element with a restrictions type attribute, and one or more form elements for text in each writing system.

```
<field type=" restrictions ">
<form lang="en"><text>English restrictions</text></form>
</field>
```

5.17 Summary Definition

A summary definition for an Entry can be given in multiple writing systems using the field element with summary-definition type, and one or more form elements for text in each writing system.

```
<field type=" summary-definition ">
<form lang="en"><text>English summary definition.</text></form>
</field>
```

5.18 Cross References

A cross reference is stored as a relation element with a type attribute specifying the name from the Lexical Relation list item, and a ref attribute storing the entry id of the target entry. See section 4.1 for more information.

```
<relation type="Compare" ref="aa2_c259b83a-fb7b-494a-982c-7cd0beb3fde3"/>
```

5.19 Custom fields

Custom fields on Entry are shown at this location. Custom fields are described in section 3.5

5.20 Import Residue

Import residue is stored in a field element with an import-residue type attribute. Only one form element is allowed which can have embedded writing systems. This field is mainly used in SFM import for fields that a user wants to keep, but there is no appropriate place in the FLEx model to store it.

```
<field type="import-residue">
<form lang="en"><text>Entry import residue</text></form>
</field>
```

5.21 Date Created

The date and time an entry is created is stored in the entry start-tag as a dateCreated attribute.

```
dateCreated="2021-03-24T15:51:45Z"
```

5.22 Date Modified

The last modification date and time an entry or owned senses was edited is stored in the entry start-tag as a dateModified attribute.

```
dateModified="2021-03-24T15:52:11Z"
```

5.23 Messages

Messages for Send/Receive communication are not stored in the fwdata file, and are not supported by LIFT.

5.24 Variants

In FLEx, a variant is an entry that is linked to a main entry via a variant type object. In LIFT, this linkage is made via a relation element that refers to the main entry or sense.

This is an example of *ebe-ebe* that is a dialect variant of *ebee*. This is the variant entry:

```
<entry id="ebe-ebe">
<lexical-unit>
<form lang="blz"><text>ebe-ebe</text></form>
</lexical-unit>
<trait  name="morph-type" value="stem"/>
<relation type="_component-lexeme" ref="ebee " order="0">
<trait  name="variant-type" value="Dialectal Variant"/>
<trait name="hide-minor-entry" value="1"/>
</relation>
</entry>
```

The relation element has attributes:

- type: the type of relation. Variants and complex forms both use the type "_component-lexeme", which represents the LexEntryRef object in the FLEx model.
- ref: A link to another entry or sense id.
- order: order of the variant when there is more than one for a given entry. The order starts at 0.

Other child elements of a _component-lexeme relation:

- trait name="variant-type; This identifies this relation as a variant relation, and gives the name of the Variant Type (e.g., Dialectal Variant).
- trait name="hide-minor-entry: Variant forms will generate a minor entry in the dictionary pointing to the main entry for the variant unless this trait is present with a value of "1", which then blocks this minor entry from showing.

This is the main entry. Note that this entry does not have any reference to its variant entries. The links are only on the variant entry.

```
<entry id="ebee">
<lexical-unit>
<form lang="blz"><text>ebee</text></form>
</lexical-unit>
<trait name="morph-type" value="stem"/>
</entry>
```

5.25 Allomorphs

Allomorphs, or alternate forms are stored in variant elements, one for each allomorph. The allomorph can have multiple writing systems, each in a form element It also supports an environment string in a trait named environment, and a morph type using a morph-type trait. Allomorphs allow user custom fields.

LIFT does not support Is Abstract Form and Stem Name fields.

```
<variant>
<form lang="krx"><text>kaawool</text></form>
<trait name ="environment" value="/ [V] _"/>
<trait name="morph-type" value="stem"/>
[optional custom fields]
</variant>
```

5.26 Grammatical Info Details

Grammatical Info Details on the entry level are not supported in LIFT. However, the Grammatical Information on sense does have traits for these.

5.27 Publish Entry in

Publish Entry information stored in FLEx is not supported in LIFT.

5.28 Show as Headword in

Show as headword information stored in FLEx is not supported in LIFT.

5.29 Subentries

This field on entries is not set directly in the LIFT file on the entry, but it comes from the is-primary attribute of the _component-lexeme relation stored on the complex form. See section 5.8.

5.30 Referenced Complex Forms

Referenced Complex Forms information stored in FLEX is not supported in LIFT.

6 Sense elements

Lexical entries may have any number of senses. Each sense is stored in a sense element. The start-tag has these attributes:

- id: This is a unique id within the project. For a lift import this can be any unique string, but once imported the id is always a guid.
- order: This is the sense order, starting at 0.

```
<sense id="c3569ffb-967b-4191-beaa-d92eb3a45166" order="0">
```

The following sense fields are ordered as they appear in the FLEx data entry fields.

6.1 Gloss

The gloss for a sense is stored in a gloss element with a lang writing system attribute. It can be repeated for any number of writing systems.

```
<gloss lang="en"><text>to.see</text></gloss>
<gloss lang="es"><text>comprender</text></gloss>
```

6.2 Reversal Entries

In FLEx, reversal entries are stored in a ReversalIndex with pointers to senses (since FW9.0.1). In LIFT, however, the reversal forms are stored in reversal elements in senses. A reversal element has a form with a single reversal string and writing system. In other words, you can't have "house; bungalow" in a single entry.

```
<reversal type="en"><form lang="en"><text>house</text></form>
</reversal>
<reversal type="en"><form lang="en"><text>bungalo</text></form>
</reversal>
```

Reversal entries can be hierarchal. In LIFT, this is accomplished by nesting main elements for owning elements up to the top level. This example has *Buick* under *American* under *car*.

```
<reversal type="en"><form lang="en"><text>Buick</text></form>
<main>
<form lang="en"><text>American</text></form>
<main>
<form lang="en"><text>car</text></form>
</main>
</main>
</reversal>
```

6.3 Definition

The definition is stored in a definition element with multiple form elements for each writing system.

```
<definition>
<form lang="en"><text>The English definition.</text></form>
<form lang="fr"><text> La définition anglaise.</text></form>
</definition>
```

6.4 Restrictions

A restrictions field is stored in a note element with a restrictions type attribute, followed by one or more form elements, one for each writing system.

```
<note type="restrictions">
<form lang="en"><text>These are English restrictions.</text></form>
<form lang="fr"><text> Ce sont des restrictions françaises.</text></form>
</note>
```

6.5 Grammatical Info.

Grammatical information (part of speech, etc.) is stored in a grammatical-info element with the value being the name of the category.

```
<grammatical-info value="noun">
</grammatical-info>
```

Grammatical information can also contain additional information on inflection class, features and exception features. This additional information is stored on trait elements with name and value. If the values for inflection-feature are not in the FLEx Inflection Features list in Grammar, it will be ignored and listed as invalid data in the Import Log. If a category is not present, it will be added and listed in the Import Log. If the inflection class is not present in the category, it will be added to the category and noted in the Import Log. If an exception-feature is not in the FLEx Exception “Features” list in Grammar, it will be added and noted in the Import Log.

```
<grammatical-info value="noun">
<trait name="noun-infl-class" value="2nd nominal declension"/>
<trait name="inflection-feature" value="{artAgr}[gen:m]"/>
<trait  name="exception-feature" value="Exception"/>
</grammatical-info>
```

6.6 Dialect Labels (Sense)

Each dialect label is stored as a dialect-labels trait using the name of the dialect in the value attribute.

```
<trait  name="dialect-labels" value="Country"/>
```

6.7 Complex Forms

In LIFT, complex forms on sense are identical to complex forms on entry (see section 5.8 for details), except the ref attribute of relation type="_component-lexeme" contains a sense Id instead of an entry Id. The relation element is still stored on the complex form entry. Nothing is stored in the sense.

6.8 Referenced Complex Forms

Referenced Complex Forms information stored in FLEx is not supported in LIFT.

6.9 Subentries

This field on sense is not set directly in the LIFT file on the sense, but it comes from the is-primary attribute of the _component-lexeme relation stored on the complex form. See section 5.8.

6.10 Variants of Sense

In LIFT, variants of sense are identical to variants on entry (see section 5.24 for details), except the ref attribute of relation type="_component-lexeme" contains a sense Id instead of an entry Id. The relation element is still stored on the variant entry. Nothing is stored in the sense.

6.11 Example

Examples are stored in an example element with an optional start-tag source attribute. If this is present, and there is no Reference field, the source content is stored in the Reference field on import. If source is included and there is a Reference field, on import, the Reference field is used.

An example element may have the following elements:

- Example: The vernacular example sentence is stored in a form element, one for each writing system.
- Translation: Example translations are stored in a translation element with a type attribute in the start-tag with the full type name from the Translation Types list in the Lists area, and one or more form elements, one for each writing system. Any number of translation elements can be present in an example element.
- Type: In FLEx this is a chooser to select an item from the Translation Types list in the Lists area. In LIFT this is stored in the type attribute in the start-tag.
- Reference: A reference for the example is stored in a note element with a reference type attribute, and form element for the reference string. Multiple form elements are not allowed in this field.
- Custom Example Fields: A custom field is stored in a field element with a type attribute with the custom field name, and then form elements with the field content, one for each writing system.
- Publish Example In is not supported in LIFT.

```
<example source="Example reference">
<form lang="krx"><text>Vernacular example.</text></form>
<translation type="Free translation">
<form lang="en"><text>English translation.</text></form>
<form lang="fr"><text>Traduction en français.</text></form>
</translation>
<note type="reference">
<form lang="en"><text>Example reference</text></form>
</note>
```

```
<field type="Cust Field">
<form lang="krx-fonipa"><text>Custom IPA example</text></form>
<form lang="krx"><text>Custom example</text></form>
</field>
</example>
```

6.12 Scientific Name

A scientific name is stored in a field element with a scientific-name type attribute, and a single form field with the content.

```
<field type="scientific-name">
<form lang="en"><text>Scientific name</text></form>
</field>
```

6.13 Anthropology Note

An anthropology note field is stored in a note element with an anthropology type attribute, followed by one or more form elements, one for each writing system.

```
<note type="anthropology">
<form lang="en"><text>These are English anthropology notes.</text></form>
<form lang="fr"><text> Ce sont des notes d'anthropologie française.</text></form>
</note>
```

6.14 Bibliography

A bibliography note field is stored in a note element with a bibliography type attribute, followed by one or more form elements, one for each writing system.

```
<note type="bibliography">
<form lang="en"><text>English bibliography information.</text></form>
<form lang="fr"><text> Informations bibliographiques en français.</text></form>
</note>
```

6.15 Discourse Note

A discourse note field is stored in a note element with a discourse type attribute, followed by one or more form elements, one for each writing system.

```
<note type="discourse">
<form lang="en"><text>These are English discourse notes.</text></form>
<form lang="fr"><text> Ce sont des notes de discours en français.</text></form>
</note>
```

6.16 Encyclopedic Info

An encyclopedic note field is stored in a note element with an encyclopedic type attribute, followed by one or more form elements, one for each writing system.

```
<note type="encyclopedic">
<form lang="en"><text>This is English encyclopedic information.</text></form>
<form lang="fr"><text> Il s'agit d'informations encyclopédiques françaises.</text></form>
</note>
```

6.17 General Note

A general note field is stored in a note element without a type attribute, followed by one or more form elements, one for each writing system.

```
<note>
<form lang="en"><text>These are English general notes.</text></form>
<form lang="fr"><text> Ce sont des notes générales françaises.</text></form>
</note>
```

6.18 Grammar Note

A grammar note field is stored in a note element with a grammar type attribute, followed by one or more form elements, one for each writing system.

```
<note type="grammar">
<form lang="en"><text>These are English grammer notes.</text></form>
<form lang="fr"><text> Ce sont des notes de grammaire française.</text></form>
</note>
```

6.19 Phonology Note

A phonology note field is stored in a note element with a phonology type attribute, followed by one or more form elements, one for each writing system.

```
<note type="phonology">
<form lang="en"><text>These are English phonology notes.</text></form>
<form lang="fr"><text> Ce sont des notes de phonologie françaises.</text></form>
</note>
```

6.20 Semantics Note

A semantics note field is stored in a note element with a semantics type attribute, followed by one or more form elements, one for each writing system.

```
<note type="semantics">
<form lang="en"><text>These are English semantic notes.</text></form>
<form lang="fr"><text> Ce sont des notes sémantiques anglaises.</text></form>
</note>
```

6.21 Sociolinguistics Note

A sociolinguistics note field is stored in a note element with a sociolinguistics type attribute, followed by one or more form elements, one for each writing system.

```
<note type="sociolinguistics">
<form lang="en"><text>These are English sociolinguistic notes.</text></form>
<form lang="fr"><text> Ce sont des notes sociolinguistiques française.</text></form>
</note>
```

6.22 Extended Note

Extended notes are not supported in LIFT.

6.23 Source

Source information for a sense is stored in a note element with a source type attribute and a single form element with the content.

```
<note type="source">
<form lang="en"><text>Source information</text></form>
</note>
```

6.24 Usages

A usages domain reference is stored in a trait element with a usage-type name attribute, and a value attribute with the usage name. These need to match items in the Usages list in the FLEx Lists area. There can be any number of usages in a sense.

```
<trait  name="usage-type" value="obsolete"/>
<trait  name="usage-type" value="old-fashioned"/>
```

6.25 Sense Type

A sense type reference is stored in a trait element with a sense-type name attribute, and a value attribute with the sense type name. The name needs to match the name in the Sense Types list in the FLEx Lists area. Only one is allowed.

```
<trait  name="sense-type" value="figurative"/>
```

6.26 Academic Domains

An academic domain reference is stored in a trait element with a domain-type name attribute, and a value attribute with the academic domain name. These need to match items in the Academic Domains list in the FLEx Lists area. There can be any number of domain references in a sense.

```
<trait  name="domain-type" value="language learning"/>
<trait  name="domain-type" value="linguistics"/>
```

6.27 Semantic Domains

A semantic domain reference is stored in a trait element with a semantic-domain-ddp4 name attribute, and a value attribute with the semantic domain abbreviation followed by the name. These need to match items in the Semantic Domains list in the FLEx Lists area. There can be any number of domain references in a sense.

```
<trait name ="semantic-domain-ddp4" value="1 Universe, creation"/>
<trait name ="semantic-domain-ddp4" value=" 9.1.2.5 Make"/>
```

6.28 Anthropology Categories

An anthropology category reference is stored in a trait element with an anthro-code name attribute, and a value attribute with the anthropology category abbreviation. These need to match items in the Anthropology Categories list in the FLEx Lists area. There can be any number of anthropology category references in a sense.

```
<trait  name="anthro-code" value="231"/>
<trait  name="anthro-code" value="240"/>
```

6.29 Status

A status reference is stored in a trait element with a status name attribute, and a value attribute with the status name. The name needs to match the name in the Status list in the FLEx Lists area. Only one is allowed.

```
<trait  name="status" value="Tentative"/>
```

6.30 Lexical Relations

A lexical relation is stored as a relation element with a type attribute specifying the name from the Lexical Relation list item, and a ref attribute storing the sense id of the target sense. See section 4.1 for more information.

```
<relation type="Antonym" ref="c259b83a-fb7b-494a-982c-7cd0beb3fde3"/>
```

6.31 Custom Fields

Custom fields on Sense are shown in this location. Custom fields are described in section 3.5

6.32 Import Residue

Import residue is stored in a field element with an import-residue type attribute. Only one form element is allowed which can have embedded writing systems.

```
<field type="import-residue">
<form lang="en"><text>Sense import residue</text></form>
</field>
```

6.33 Publish Sense In

Publish Sense In fields are not supported in LIFT.

6.34 Nested senses

Senses may be nested inside of senses to any number of levels (within reason). Nested senses are stored in subsense elements. All elements in senses are also available in subsenses. The following example demonstrates this hierarchy

```
sense 1
sense 2
   sense 2.1
      sense 2.1.1
   sense 2.2
sense 3
```

```
<sense id="826f0312-53a2-42a6-b8ee-217b540d7b6d" order="0">
<grammatical-info value="verb">
</grammatical-info>
<gloss lang="en"><text>main sense 1</text></gloss>
<trait  name="dialect-labels" value="City"/>
</sense>
<sense id="8b91a4cd-6e85-41d8-bc52-327afae7db79" order="1">
<grammatical-info value="verb">
```

```
</grammatical-info>
<gloss lang="en"><text>main sense 2</text></gloss>
<subsense id="2485960d-4b79-45d5-ab9a-641c790b7bc6">
<grammatical-info value="verb">
</grammatical-info>
<gloss lang="en"><text>subsense 2.1</text></gloss>
<subsense id="a84eede2-f10f-45dd-bea2-98eb0fab2e75">
<grammatical-info value="verb">
</grammatical-info>
<gloss lang="en"><text>subsubsense 2.1.1</text></gloss>
</subsense>
</subsense>
<subsense id="3343777d-02e0-4372-b767-b4ca8615aeb6">
<grammatical-info value="verb">
</grammatical-info>
<gloss lang="en"><text>subsense 2.2</text></gloss>
</subsense>
</sense>
<sense id="f2b314f2-9844-4bcf-9ca4-ef451d3231c4" order="2">
<grammatical-info value="verb">
</grammatical-info>
<gloss lang="en"><text>main sense 3</text></gloss>
</sense>
```

6.35 Pictures

Picture information is stored in an illustration element in a LIFT sense element. You can have any number of illustration elements in a sense. The name of the picture file is stored in an href attribute on illustration. This is the relative path to the picture from LinkedFiles\Pictures, so it is normally just a file name. The Illustration element can also hold a picture caption in any number of languages in a label element. Each language (writing system) is stored in a separate form element with the writing system in a lang attribute and the string content in a text element which may have embedded spans.

```
<illustration href="Picture 1293.jpg">
<label>
<form lang="en"><text>English caption with <span lang="fr">French </span>
embedding.</text></form>
<form lang="fr"><text>French caption</text></form>
</label>
</illustration>
```

If the LIFT directory contains a “pictures” directory with a file specified in an illustration href attribute, the file will be copied to the LinkedFiles\Pictures directory in FLEx so that the picture will be displayed in FLEx. If a file specified in an illustration href is not in the pictures directory, the file will not be copied to the LinkedFiles\Pictures directory. However, the file path is still imported into a CmFile element and a CmPicture elelment will be added to the LexSense that references the CmFile element. Thus, if you copy the picture to the LinkedFiles\Pictures directory after the import, the picture will then show in FLEx.

If a file name is not specified in an href element, or it is a null string (e.g., href= ""), the import will add a CmPicture element with the caption, but a CmFile element will not be created. This is an unusual situation in FLEx in that it cannot be reproduced in the FLEx UI, but it is a possible situation that can happen via SFM or LIFT imports.
Note: If a later LIFT import includes an illustration with a file specified in href, the import will result in a new CmPicture and a CmFile object being created in FLEx resulting in the LexSense owning two CmPicture elements. Pictures cannot be deleted in bulk edit. If you plan to do this kind of import where FLEx already has a CmPicture element with captions, make sure your LIFT import includes the captions in the form element, and then delete the CmPicture element(s) from FLEx fwdata prior to the LIFT import. This has the effect of adding a file to existing picture captions.

If you have a picture in FLEx and export that to LIFT, then replace the picture file name in the LIFT file and import, the modified file with the option to “Import the conflicting data and overwrite the current data”, the import will not change the existing CmPicture and CmFile, but will add a new CmPicture and CmFile so that you will now have two pictures for this sense.

Note that copyright and licensing information entered in FLEx 9.1.22 or later stores this information as metadata in the picture file rather than in FLEx data. LIFT cannot specify this information separately. Also, note that Publish Picture information in FLEx is not supported in LIFT.

7 Exporting LIFT data

To export LIFT data, there are two options available under File…Export, 1) Filtered Lexicon LIFT 0.13 XML, and 2) Full Lexicon LIFT 0.13 XML. Neither option allows you to select which fields you want to export, or which writing systems. It always exports all writing systems, and it always exports all fields in entries and senses. The only difference between the two is that the filtered version will only export entries that are in the active filter, while the full version exports all entries.

When you choose a LIFT option from File…Export, it brings up an Export dialog. In this dialog you can choose to have the LIFT folder opened after the export completes. It also allows you to choose whether to copy picture and media files to the exported folder.

When you click Export, it brings up a Browse For Folder dialog. A LIFT file should **always** be in a directory just for that file. You should **never** select a directory that contains other files or folders in this dialog. It allows you to create a new folder for the export, which is usually a good thing to do. The export will include other files and folders inside the folder you choose. If you pick a previous LIFT folder, it will ask if you want to overwrite the data in that folder.

8 Importing LIFT data

One of the big advantages to LIFT import is the ability to merge LIFT data with an existing FLEx project. Unlike SFM import, which imports SFM data without regard to existing data, if prepared properly, LIFT import can add new entries or senses, but can also merge changes into existing entries and senses. To merge into existing objects, the LIFT file needs to use the guids of entries and senses that are already present in the FLEx

project. If the guids match, it will merge the LIFT data into the existing data. Otherwise, it will add new entries or senses. When merging into an entry or sense,

- Any fields in the LIFT file that are not already present in the object will be added to the existing object.
- Any data in the LIFT file that matches existing data already in the object will be unchanged.
- Any fields already in the existing object but not in the LIFT file will remain unchanged.

When importing into existing data, conflicts are possible. If a given entry has one definition, and that entry in the LIFT data has a different definition, this will be a conflict since the merge capability cannot merge two strings into one string in a field. There are three options for resolving conflicts during a LIFT import. You must make one choice for resolving all conflicts for the entire import.

1. Do not import conflicting data. With this option, the import would ignore the definition from the LIFT file.
2. Import the conflicting data and overwrite current data. With this option, the import would replace the existing definition with the one from the LIFT file.
3. Import the conflicting data into a new entry or sense. With this option, the import would keep the current sense with its definition, but add a new sense to the entry with the data from the LIFT file. When new entries or senses are created during this mode, the Import Log file will list the new entries/senses that were created due to the conflict with a report similar to this:

   The following conflicts resulted in duplicated entries or senses:

Type	Conflicting field	Original	New duplicate
Sense	Definition	ke =	ke =

   It contains live links that open the project in FLEx to the link chosen (when LT-20790 is fixed).

Merge conflicts only occur where the field can only have one result. For a property allowing multiple list items, such as semantic domains, you can't have a conflict. If an item exists in the list, it would not be changed. If an item from the LIFT file does not match an existing item, the LIFT item is added to the field. Thus, a LIFT import cannot remove existing items in lists or owned objects. There is a way for deleting an entire entry (using the dateDeleted attribute on entry, see section 5), but nothing at a lower level.

LiftResidue. LIFT imports attempt to store any data from the LIFT file that cannot be stored in normal FieldWorks objects so that a LIFT export will return the unused data. This is an attempt to never lose additional information that other programs need when using LIFT. FLEx does this by storing LiftResidue fields that will never get deleted via the user interface.

An import of a WeSay LIFT file will normally have a ListResidue field for every entry and sense similar to this for entry:

```
<LiftResidue>
<Uni>&lt;lift-residue id="chalèèc_00020a7a-6240-47a7-9dec-94ec76ce0a17"
dateCreated="2014-09-17T04:58:49Z" dateModified="2015-12-09T14:06:32Z"&gt;&lt;/lift-
residue&gt;</Uni>
</LiftResidue>
```

and this for sense:

```
<LiftResidue>
<Uni>&lt;lift-residue id="000520be-2da6-4251-8b32-ce68b25d96ab"&gt;&lt;/lift-
residue&gt;</Uni>
</LiftResidue>
```

If you don't plan do other LIFT imports or exports on a project, you can reduce clutter by deleting all of the LiftResidue elements. It can be done easily using regex, CC, etc. It is always 3 lines with one Uni element encoding the unused information.

8.1 Preparing LIFT data

Caution! Before importing a LIFT file, it should always be in a separate folder, which may include folders for writing systems (WritingSystems), pictures (pictures), and audio/visual files (audio). It may also include a .lift-ranges file that specifies the range lists. The import process before FW9.1.10 copies all of the files in the current directory to a temp directory, so it can waste a lot of time copying unnecessary files and directories, and can give strange error messages saying the path or filename is too long if you try to import a LIFT file directly from Documents, or some other heavily used directory. In FW9.1.10 the import changed to just copy the files it needs (LT-20954). But you should still keep all files related to a project in a LIFT directory for that project.

Caution! Any writing systems used in the LIFT file that are not present in the current project will be added to the current project. Also, any list references that are missing from the current project will be added to the appropriate list in the project. If the LIFT file uses a custom field that is not defined in the current project, it will add the custom field to the current project. Note if the LIFT file does not include a field definition for the custom field, the import will still add a custom field, but the type of field may be incorrect. The Import Log file will list any new writing systems and list items that are added during the import.

When you import data without guids, the LIFT import will create new entries and senses without modifying existing objects. All of the ids must be unique in a given file, but you are free to choose whatever ids you want. They will only be saved in LiftResidue. This example would add a new entry with a sense.

```
<entry id="e1">
<lexical-unit>
<form lang="fr"><text>rouge</text></form>
</lexical-unit>
<sense id="s1">
<grammatical-info value="adjective"></grammatical-info>
<gloss lang="en"><text>red</text></gloss>
<definition>
<form lang="en"><text>The color red.</text></form>
```

```
</definition>
</sense>
</entry>
```

If you want to merge into an existing entry or sense, you need to use the guid attribute rather than the id attribute to match the existing entry and sense. In this case, the entry id field is not needed. LIFT import is an additive process. You can add new information or modify some existing fields, but you can't remove fields that are already in the FLEx project. (LIFT import via Send/Receive can remove fields, but this option is not available during File…Import.) This example would add a definition to the sense with the specified id in the entry with the specified id.

```
<?xml version="1.0"?>
<lift version="0.13">
<entry guid="3c99c376-e6dc-45a5-aaa4-bde6808acdcb">
<sense id="85352e90-319f-4b63-98a3-ba3d9a094c52">
<definition>
<form lang="en"><text>female feline</text></form>
</definition>
</sense>
</entry>
</lift>
```

Caution: LIFT does not contain as much detail as a FLEx data file. As a result, some things could produce undesirable alterations to your existing data (e.g., duplicate examples, pronunciations, and undesirable changes to cross references and lexical relations). So be cautious with LIFT imports into FLEx.

- When importing example sentence objects, since LIFT does not specify guids, the only way an existing example can be matched is by the vernacular example string. If it matches the vernacular example string in the LIFT file, it will assume the same example. In this case, other fields for the example sentence (e.g., translations) can be added. However, with translations, again there is no guid in LIFT, so the same thing happens when adding a translation. If the translation string in the LIFT file matches an existing translation string in the example, it will use the existing translation. If the LIFT file has a different translation string, it will be added to the example as a new translation. If the vernacular example field in the LIFT file does not match an existing example sentence with the same vernacular example string, then the import will add the LIFT example sentence as a new example sentence. As a result of this ambiguity, you can never change an existing vernacular example or translation string using LIFT import, and any changes to these strings will result in new examples and translations being added to the sense.
- Pronunciation objects work in a similar way, since LIFT does not include guids for pronunciations. If the pronunciation form does not match an existing pronunciation, it will be added as a new pronunciation object.
- In FLEx, it's possible to have multiple lexical relation or cross reference sets of the same type on a sense. For example, you can have a synonym set holding a, b, c, and another one holding a and d. This prevents b and c being related to d. However, in LIFT this distinction is lost. So a LIFT import can mess up these types of relations

Note when merging into existing entries, DateCreated and DateModified will be set to the current time if not included in the LIFT file. That's generally desirable for DateModified, but not for DateCreated. So if you want to keep created dates, you'll need to add the date from the project into your LIFT file.

Note if you import just the citation form without a lexeme form, MoStemAllomorphs are not created, and bulk edit entries will not show these citation forms, changing a morph type is not possible, and some other oddities exist. If you do not want to import a lexeme form, you should import a morph type as in this example which will create the MoStemAllomorph to solve these potential problems.

```
<entry id="e1">
<citation>
<form lang="fr"><text>able</text></form>
</citation>
<trait  name="morph-type" value="root"/>
</entry>
```

At times you may find it useful to export a LIFT file from FLEx, then remove much of the data, especially examples, relations, and pronunciations so that the import will not mess these up. Then add whatever information you want to the LIFT file and then import that. This way you can add missing data to your FLEx project or alter existing data without possibly corrupting your data by doing a full LIFT import.

8.2 Importing LIFT data

It would be wise to make a backup of your FLEx project prior to an import in case the results are not what you intended. Undo will not undo a LIFT import.

To import a LIFT file, choose File…Import..LIFT Lexicon. This brings up an Import/Merge From LIFT file dialog. You need to select one of the three merge strategies. If you want to skip importing any LIFT entry where the current modification time matches the modification time in the LIFT file, then Check the Skip… checkbox. If the checkbox is unchecked, it will merge the entries regardless of the modification time. Finally, select the .lift file you want to import from a LIFT directory, then click OK. When the import is completed, it adds a <projectname>-ImportLog.htm file in the LIFT directory, and then opens it in your default browser to give you information on the import.

Prior to importing a LIFT file, FLEx validates the file and if it does not pass the validation process, it gives a yellow error message. (See section 10 for the Relax NG XML Schema file used to validate LIFT.) If you look at the details in the error message it tries to give you useful information on why it failed. Unfortunately, when you close this yellow dialog prior to LT-20792 being fixed, the import hangs FLEx and you have to kill FLEx with the Task Manager.

During an import, when a picture or audio\visual file name is specified in an illustration element for pictures, a media element for pronunciations, or an audio writing system for lexeme form, example sentence, etc., the file name will be imported into FLEx fwddata. If a corresponding file is included in the LIFT audio or pictures directory, the

corresponding file will be copied to the FLEx project LinkedFiles\Pictures directory for picture files and LinkedFile\AudioVisual directory for audio/visual files.

9 LIFT differences between FLEx and WeSay

WeSay uses LIFT for its data storage. FLEx and WeSay can collaborate using Send/Receive (S/R), but there are some limitations since FLEx uses a more robust data model (fwdata file). During S/R, FLEx exports the lexicon to LIFT, and uses this data to merge with a WeSay repository. After merging with the WeSay repository, it then imports the data back into FLEx which uses an option that is not available in the FLEx LIFT import dialog. It tries to make the FLEx project as close to the WeSay data as it can. Note the caution in section 8.1 with examples, pronunciations, and references which still applies in S/R.

Without using WeSay Send/Receive, FLEx can export a LIFT project and then WeSay can open this LIFT project. FLEx can also import the WeSay LIFT file directly from the WeSay project folder.

WeSay does not support many of the more advanced features of the FLEx dictionary model. For areas it doesn't support, it should not lose any LIFT data, but it will only allow you to edit parts it understands.

WeSay does not have the concept of vernacular and analysis writing systems. It just has a list of writing systems that can be assigned to fields as needed.

Some of the limitations are listed in section 8.1 above. Some other complications come from WeSay using Word for the Lexeme Form, and Meaning for the Definition. These issues are discussed in more detail in section 4.7 FLEx and WeSay compatibility issues in FLEx…Help…Resources…Technical Notes on FieldWorks Send-Receive. This document is also available from https://downloads.languagetechnology.org/fieldworks/Documentation/Technical_Notes_on_FieldWorks_Send_Receive.pdf.

There are other issues with custom fields, and list items which are handled in different ways in WeSay and FLEx. With care, and some manual work on each end, it's usually possible to work around these issues.

10 Validation

The Relax NG XML Schema (lift-0.13.rng) file is embedded in SIL.Lift.dll in the FieldWorks program directory and is used for validating LIFT files. Here is the content of the .rng file, in case you want to use this for verification prior to import, or for other purposes.

```
<grammar datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"
                        xmlns="http://relaxng.org/ns/structure/1.0"
                        xmlns:sch="http://purl.oclc.org/dsdl/schematron">

 <!-- need to define the normalization or lack thereof in keys especially in regard to whitespace -->


 <!-- ================================================== date or dateTime -->
 <define name="date.or.dateTime">
          <choice>
```

```
            <data type="date"/>
            <data type="dateTime"/>
          </choice>
</define>

<!-- ================================================================ refid -->
<define name="refid">
          <attribute name="ref"/>
</define>

<!-- ========================================================== form-content -->
<define name="form-content">
          <attribute name="lang"/>
          <!-- rfc 4646 -->
          <ref name="form-no-lang-content"/>
</define>

<define name="form-no-lang-content">
          <interleave>
            <element name="text">
                    <ref name="span-content"/>
            </element>
            <zeroOrMore>
                    <element name="annotation">
                      <ref name="annotation-content"/>
                    </element>
            </zeroOrMore>
          </interleave>
</define>

<!-- ============================================================ span-content -->
<define name="span-content">
          <interleave>
            <text/>
            <zeroOrMore>
                    <element name="span">
                      <ref name="inner-span-content"/>
                    </element>
            </zeroOrMore>
          </interleave>
</define>

<!-- ================================================== inner-span-content -->
<define name="inner-span-content">
          <optional>
            <!-- rfc 4646 -->
            <attribute name="lang"/>
          </optional>
          <optional>
            <attribute name="href">
                    <data type="anyURI"/>
            </attribute>
          </optional>
          <optional>
            <attribute name="class"/>
          </optional>
          <ref name="span-content"/>
</define>

<!-- ================================================== multitext-content -->
<define name="multitext-content">
          <zeroOrMore>
```

```
            <element name="form">
                        <ref name="form-content"/>
                        <sch:rule context="form" >
                          <sch:assert test="not(preceding-sibling::form[@lang=current()/@lang])">
                                    Forms should be in different langs.
                                    There is only one form with a given lang allowed in any parent element.
                          </sch:assert>
                        </sch:rule>
            </element>
          </zeroOrMore>
</define>

<!-- ========================================================= URLRef-content -->
<define name="URLRef-content">
          <attribute name="href">
            <data type="anyURI"/>
          </attribute>
          <optional>
            <element name="label">
                        <ref name="multitext-content"/>
            </element>
          </optional>
</define>

<!-- ========================================================== field-content -->
<define name="field-content">
          <interleave>
          <ref name="multitext-content"/>
          <ref name="extensible-without-field-content"/>
          </interleave>
          <!-- problem should be same as extensible but without field -->
          <attribute name="type"/>
          <sch:rule context="field" >
            <sch:assert test="not(preceding-sibling::field[@type=current()/@type])">
                        Fields should have different types.
                        There is only one field with a given type allowed in any parent element.
            </sch:assert>
          </sch:rule>
</define>

<!-- ============================================================ trait-content -->
<define name="trait-content">
          <attribute name="name"/>
          <attribute name="value"/>
          <zeroOrMore>
            <!-- !!! documentation has this labeled as status-->
            <element name="annotation">
                        <ref name="annotation-content"/>
            </element>
          </zeroOrMore>
</define>

<!-- ================================================== annotation-content -->
<define name="annotation-content">
          <ref name="multitext-content"/>
          <attribute name="name"/>
          <optional>
            <attribute name="value"/>
          </optional>
          <optional>
            <attribute name="who"/>
          </optional>
```

```
            <optional>
              <attribute name="when">
                        <ref name="date.or.dateTime"/>
              </attribute>
            </optional>
</define>


<!-- ==================================== extensible-without-field-content -->
<define name="extensible-without-field-content">
            <interleave>
              <optional>
                        <attribute name="dateCreated">
                          <ref name="date.or.dateTime"/>
                        </attribute>
              </optional>
              <optional>
                        <attribute name="dateModified">
                          <ref name="date.or.dateTime"/>
                        </attribute>
              </optional>
              <zeroOrMore>
                        <element name="annotation">
                          <ref name="annotation-content"/>
                        </element>
              </zeroOrMore>
              <zeroOrMore>
                        <element name="trait">
                          <ref name="trait-content"/>
                        </element>
              </zeroOrMore>
            </interleave>
</define>

<!-- ================================================= extensible-content -->
<define name="extensible-content">
            <interleave>
              <ref name="extensible-without-field-content"/>
              <zeroOrMore>
                        <element name="field">
                          <ref name="field-content"/>
                        </element>
              </zeroOrMore>
            </interleave>
</define>

<!-- ========================================================= note-content -->
<define name="note-content">
            <optional>
              <attribute name="type"/>
            </optional>
            <sch:rule context="note" >
              <sch:assert test="not(preceding-sibling::note[@type=current()/@type])">
                        Notes should be of different types.
                        There is only one note with a given type allowed in any parent element.
              </sch:assert>
            </sch:rule>
            <interleave>
              <ref name="multitext-content"/>
              <ref name="extensible-content"/>
            </interleave>
</define>
```

```
<!-- ================================================ pronunciation-content -->
<define name="pronunciation-content">
        <interleave>
          <ref name="multitext-content"/>
          <ref name="extensible-content"/>
          <zeroOrMore>
                  <element name="media">
                    <ref name="URLRef-content"/>
                  </element>
          </zeroOrMore>
        </interleave>
</define>

<!-- ================================================== etymology-content -->
<define name="etymology-content">
        <attribute name="type"/>
        <attribute name="source"/>
        <interleave>
          <ref name="extensible-content"/>
          <zeroOrMore>
                  <element name="form">
                    <ref name="form-content"/>
                  </element>
          </zeroOrMore>
          <zeroOrMore>
                  <element name="gloss">
                    <ref name="form-content"/>
                  </element>
          </zeroOrMore>
        </interleave>
</define>

<!-- ============================================== grammatical-info-content -->
<define name="grammatical-info-content">
        <attribute name="value"/>
        <zeroOrMore>
          <element name="trait">
                  <ref name="trait-content"/>
          </element>
        </zeroOrMore>
</define>

<!-- ================================================== reversal-content -->
<define name="reversal-content">
        <optional>
          <attribute name="type"/>
        </optional>
        <interleave>
          <ref name="multitext-content"/>
          <optional>
                  <ref name="reversal-main"/>
          </optional>
          <optional>
                  <element name="grammatical-info">
                    <ref name="grammatical-info-content"/>
                  </element>
          </optional>
        </interleave>
</define>

<!-- ================================================== reversal-main -->
```

```
<define name="reversal-main">
		<element name="main">
		 <sch:rule context="main">
				<sch:assert test="parent::*/form">
				 A main should not exist without a parent form
				</sch:assert>
		 </sch:rule>
		 <interleave>
				<ref name="multitext-content"/>
				<optional>
				 <ref name="reversal-main"/>
				</optional>
				<optional>
				 <element name="grammatical-info">
						<ref name="grammatical-info-content"/>
				 </element>
				</optional>
		 </interleave>
		</element>
</define>

<!-- ================================================= translation-content -->
<define name="translation-content">
		<ref name="multitext-content"/>
		<optional>
		 <attribute name="type"/>
		 <!-- back | free | literal -->
		</optional>
		<sch:rule context="translation" >
		 <sch:assert test="not(preceding-sibling::translation[@type=current()/@type])">
				Translations should be of different types.
		 </sch:assert>
		</sch:rule>
</define>

<!-- ================================================== example-content -->
<define name="example-content">
		<optional>
		 <attribute name="source"/>
		 <!-- a key-->
		</optional>

		<interleave>
		 <ref name="multitext-content"/>
		 <ref name="extensible-content"/>
		 <zeroOrMore>
				<element name="translation">
				 <ref name="translation-content"/>
				</element>
		 </zeroOrMore>
		 <zeroOrMore>
				<element name="note">
				 <ref name="note-content"/>
				</element>
		 </zeroOrMore>
		</interleave>
</define>

<!-- ================================================== relation-content -->
<define name="relation-content">
		<!-- @ref is a @type relation of parent-->
		<attribute name="type"/>
```

```
            <ref name="refid"/>
            <optional>
              <attribute name="order">
                        <data type="integer"/>
              </attribute>
            </optional>

            <interleave>
              <ref name="extensible-content"/>

              <optional>
                        <element name="usage">
                          <ref name="multitext-content"/>
                        </element>
              </optional>
            </interleave>
</define>

<!-- ================================================== variant-content -->
<define name="variant-content">
            <optional>
              <ref name="refid"/>
            </optional>

            <interleave>
              <ref name="extensible-content"/>
              <ref name="multitext-content"/>

              <zeroOrMore>
                        <element name="pronunciation">
                          <ref name="pronunciation-content"/>
                        </element>
              </zeroOrMore>
              <zeroOrMore>
                        <element name="relation">
                          <ref name="relation-content"/>
                        </element>
              </zeroOrMore>
            </interleave>
</define>

<!-- ===================================================== sense-content -->
<define name="sense-content">
            <!--Handbook of Lexicography:
            a sense is a hypothesis that one meaning has derived from a previous meaning
            (i.e. the meanings are semantically related but have a distinct central meaning).-->
            <optional>
              <attribute name="id"/>
            </optional>
            <optional>
              <attribute name="order">
                        <data type="integer"/>
              </attribute>
            </optional>

            <interleave>
              <ref name="extensible-content"/>
              <optional>
                        <element name="grammatical-info">
                          <ref name="grammatical-info-content"/>
                        </element>
              </optional>
```

```
            <zeroOrMore>
                        <element name="gloss">
                          <ref name="form-content"/>
                        </element>
            </zeroOrMore>
            <optional>
                        <element name="definition">
                          <ref name="multitext-content"/>
                        </element>
            </optional>
            <zeroOrMore>
                        <element name="relation">
                          <ref name="relation-content"/>
                        </element>
            </zeroOrMore>
            <zeroOrMore>
                        <element name="note">
                          <ref name="note-content"/>
                        </element>
            </zeroOrMore>
            <zeroOrMore>
                        <element name="example">
                          <ref name="example-content"/>
                        </element>
            </zeroOrMore>
            <zeroOrMore>
                        <element name="reversal">
                          <ref name="reversal-content"/>
                        </element>
            </zeroOrMore>
            <zeroOrMore>
                        <element name="illustration">
                          <ref name="URLRef-content"/>
                        </element>
            </zeroOrMore>
            <zeroOrMore>
                        <element name="subsense">
                          <ref name="sense-content"/>
                        </element>
            </zeroOrMore>
          </interleave>
</define>

<!-- ================================================== entry-content -->
<define name="entry-content">
          <optional>
            <attribute name="id"/>
          </optional>
          <optional>
            <attribute name="guid"/>
          </optional>
          <optional>
            <attribute name="order">
                        <data type="integer"/>
            </attribute>
          </optional>
          <optional>
            <attribute name="dateDeleted">
                        <ref name="date.or.dateTime"/>
            </attribute>
          </optional>
```

```
            <interleave>
             <ref name="extensible-content"/>
             <optional>
                        <element name="lexical-unit">
                          <ref name="multitext-content"/>
                        </element>
             </optional>
             <optional>
                        <element name="citation">
                          <ref name="multitext-content"/>
                        </element>
             </optional>
             <zeroOrMore>
                        <element name="pronunciation">
                          <ref name="pronunciation-content"/>
                        </element>
             </zeroOrMore>
             <zeroOrMore>
                        <element name="variant">
                          <ref name="variant-content"/>
                        </element>
             </zeroOrMore>
             <zeroOrMore>
                        <element name="sense">
                          <ref name="sense-content"/>
                        </element>
             </zeroOrMore>
             <zeroOrMore>
                        <element name="note">
                          <ref name="note-content"/>
                        </element>
             </zeroOrMore>
             <zeroOrMore>
                        <element name="relation">
                          <ref name="relation-content"/>
                        </element>
             </zeroOrMore>
             <zeroOrMore>
                        <element name="etymology">
                          <ref name="etymology-content"/>
                        </element>
             </zeroOrMore>
            </interleave>
</define>

<!-- ================================================= field-defn-content -->
<define name="field-defn-content">
            <ref name="multitext-content"/>
            <attribute name="tag"/>
</define>

<!-- ================================================= field-defns-content -->
<define name="field-defns-content">
            <zeroOrMore>
              <element name="field">
                        <ref name="field-defn-content"/>
              </element>
            </zeroOrMore>
</define>

<!-- ================================================= range-element-content -->
<define name="range-element-content">
```

```
            <attribute name="id"/>
            <optional>
              <!-- refers to another range-element's id -->
              <attribute name="parent"/>
            </optional>
            <optional>
              <attribute name="guid"/>
            </optional>
            <interleave>
              <optional>
                        <element name="description">
                          <ref name="multitext-content"/>
                        </element>
              </optional>
              <optional>
                        <element name="label">
                          <ref name="multitext-content"/>
                        </element>
              </optional>
              <optional>
                        <element name="abbrev">
                          <ref name="multitext-content"/>
                        </element>
              </optional>
            </interleave>
</define>

<!-- ================================================== range-content -->
<define name="range-content">
            <attribute name="id"/>
            <optional>
              <attribute name="href">
                        <data type="anyURI"/>
              </attribute>
            </optional>
            <optional>
              <attribute name="guid"/>
            </optional>
            <interleave>
              <optional>
                        <element name="description">
                          <ref name="multitext-content"/>
                        </element>
              </optional>
              <optional>
                        <element name="label">
                          <ref name="multitext-content"/>
                        </element>
              </optional>
              <optional>
                        <element name="abbrev">
                          <ref name="multitext-content"/>
                        </element>
              </optional>
              <zeroOrMore>
                        <element name="range-element">
                          <ref name="range-element-content"/>
                        </element>
              </zeroOrMore>
            </interleave>
</define>
```

```
 <!-- ================================================== ranges-content -->
 <define name="ranges-content">
            <zeroOrMore>
              <element name="range">
                        <ref name="range-content"/>
              </element>
            </zeroOrMore>
 </define>

 <!-- ================================================== header-content -->
 <define name="header-content">
            <interleave>
              <optional>
                        <element name="description">
                          <ref name="multitext-content"/>
                        </element>
              </optional>
              <optional>
                        <element name="ranges">
                          <ref name="ranges-content"/>
                        </element>
              </optional>
              <optional>
                        <element name="fields">
                          <ref name="field-defns-content"/>
                        </element>
              </optional>
            </interleave>
 </define>

 <!-- ======================================================== lift-content -->
 <define name="lift-content">
            <attribute name="version">
              <value>0.13</value>
            </attribute>
            <optional>
              <attribute name="producer"/>
            </optional>
            <optional>
              <element name="header">
                        <ref name="header-content"/>
              </element>
            </optional>
            <zeroOrMore>
              <element name="entry">
                        <ref name="entry-content"/>
              </element>
            </zeroOrMore>
 </define>

 <!-- ================================== start ============================ -->
 <start>
            <element name="lift">
              <ref name="lift-content"/>
            </element>
 </start>
</grammar>
```