FLEx 9.1 Conceptual Model

Ken Zook

July 9, 2024

Contents

1 Introduction

FieldWorks Language Explorer (FLEx) is a program for developing dictionaries with any number of languages or writing systems, specifying grammar features, analyzing texts, parsing texts, and developing a research notebook.

This document describes the underlying format of the data in FieldWorks version 9.1.25, mainly stored in an XML .fwdata file in the project directory. It covers the basic conceptual model for processing different types of data. A corresponding FLEx project, “FLEx Conceptual Model sample project” demonstrates all of the items discussed in this document. This document can be downloaded from here, and the sample project can be downloaded from here. This document, the FLEx project, and three PowerPoint presentations from this document are available at https://github.com/sillsdev/FwDocumentation/tree/develop/HelpResourcesSources/TechnicalNotes.

Data for a language project is stored in a folder, normally under c:\ProgramData\SIL\FieldWorks\Projects. The data is stored in a single XML file with a .fwdata file extension. The project also has several subfolders as follows:

- WritingSystemStore directory that holds XML .ldml files for each writing system used in the project.
- ConfigurationSettings directory that stores various XML files containing various user settings for field visibility, sizing of windows, columns, filters, dictionary configuration, etc.
- LinkedFiles directory for storing pictures, media files, etc.
- SharedSettings directory that holds user settings for writing system fonts, keyboards, language name and abbreviation, etc.
- Temp directory that holds some caches for speeding up sorting.

Several settings files are also stored under %localappdata%\SIL
Some settings are in Windows environment variables and in the registry under
HKEY_LOCAL_MACHINE\SOFTWARE\SIL\FieldWorks, and
HKEY_CURRENT_USER\Software\SIL\FieldWorks.

2 Basic concepts

FieldWorks uses an object-oriented data structure. The Fieldworks versions before FW7.0 stored data in SQL Server. Fieldworks 7.0 switched to using an XML format with data stored in a fwdata file. At some point in the future we may use a different data storage. An fwdata file contains XML data in the following format.

```
<?xml version="1.0" encoding="utf-8"?>
<languageproject version="7000072">
<AdditionalFields>
<CustomField class="LexEntry" name="Phonetic" type="MultiUnicode" wsSelector="-4" />
</AdditionalFields>
<rt class="CmSemanticDomain" guid="00041516-72d1-4e56-9ed8-fe235a9b1a68"
ownerguid="00269021-e1c4-474d-9dba-341d296bdac7">
…
</rt>
more rt class (object) instances sorted by class guid
</languageproject>
```

A data model version number identifies the model for this file. When the model changes, this number is incremented, and when the file is opened in a newer version of FLEx with a higher version number, the data will be migrated to the newer number. Older programs cannot open a project that has been migrated to a higher version number. The 7000072 model number has not changed since FW9.0.1. Optional custom field definitions are stored in an AdditionalFields element. The rest of the file is a series of rt elements listing all of the class instances in the project. Each class instance gives the name of the class, a Globally Unique Identifier (guid), and an optional owner guid that specifies an ownership hierarchy. The rt elements are sorted by the guid field.

If you remove all CRLF from a fwdata file it will crash when you try to open it. I believe it's only the first two lines that must have CRLF at the end for it to open. Also, Send/Receive requires some field elements be in place, even though that's not an XML requirement. If you

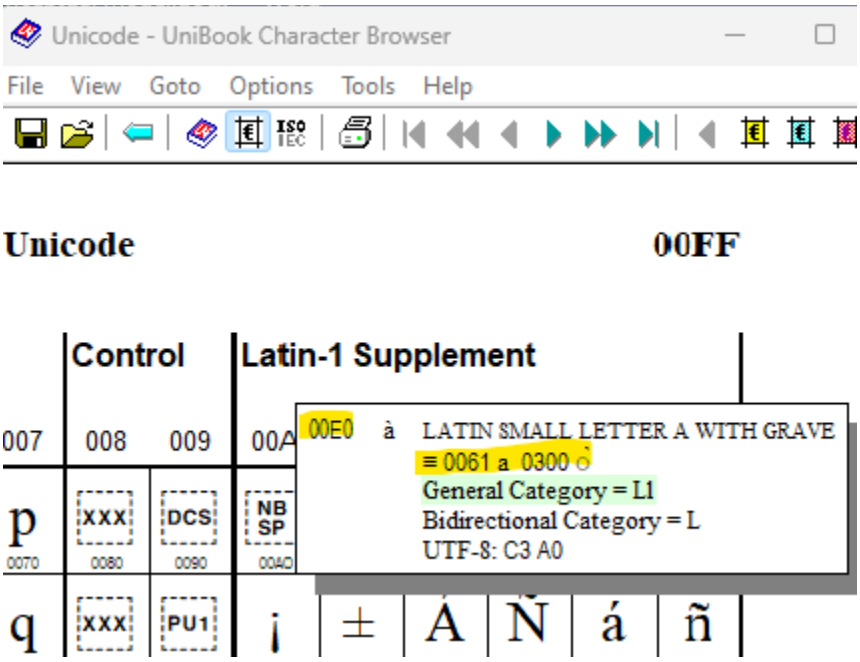
modify a fwdata file outside of FLEx, it's wise to open the project and choose Tools > Utilities > Write Everything. This makes sure it's formatted properly.

Note when processing fwdata that some rt elements may have no content, so they have this format.

```
<rt class="CmSemanticDomain" guid="00041516-72d1-4e56-9ed8-fe235a9b1a68"
ownerguid="00269021-e1c4-474d-9dba-341d296bdac7" />
```

Any time you modify the fwdata file outside of FLEx, you'd better make a backup of the file ahead of time in case your change makes it unloadable.

All data in FLEx should be valid Unicode data. When FLEx opens an fwdata file, it normalizes the data in memory to Normalization Form Decomposed (NFD). Anything that is imported, typed, or copied into Flex will be normalized to NFD. Anything that Flex writes out though the clipboard, exports, and saving to fwdata will be converted to Normalization Form Composed (NFC). A useful program that shows information on Unicode code points including normalization forms is UniBook (https://www.unicode.org/unibook/).



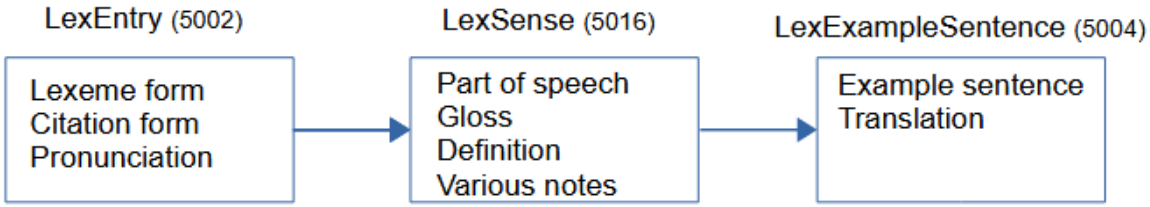
Any processes that work internally with FLEx such as bulk edit processes, keyboard definitions using context, and FlexTools must handle NFD data.

2.1 Classes and properties

In an object-oriented design, objects represent some object or concept we want to model. For example, an entry would be an object in a dictionary. An entry has various fields that are related at the entry level including a lexeme form. Fields that only occur once in an entry can be stored directly in the entry object. However, when something may occur more than once, then a different object needs to be used to define that part of an entry. Since an entry can have multiple senses, each with a part of speech, definition, etc., we need a sense object as part of an entry.

FieldWorks stores all data in objects with class names. Each class has a unique Class ID integer that is used in various places in code, often abbreviated clid for class ID. For instance, a lexical

entry is a LexEntry class with ID 5002. A sense within an entry is a LexSense class with ID 5016, and an example sentence within a sense is a LexExampleSentence class with ID 5004. Most classes are arranged in an ownership hierarchy.



In this case LexExampleSense is owned by LexSense, which is owned by LexEntry. If you delete the entry, it also deletes all objects in its ownership hierarchy. This is not true, however, if editing fwdata. Each object is an rt element in scattered locations in the file.

Each class has properties, or fields that hold the data for that class. Properties also have unique integer IDs, often abbreviated as flid for field ID in code. The flid includes the 4-digit class id followed by a 3 digit field number. For example, a Definition on LexSense has a flid of 5016005

The Classes in FieldWorks have an inheritance hierarchy starting with CmObject at the top of the hierarchy. Subclasses inherit properties from superclasses in the hierarchy. Since CmObject has a Guid property, and all classes are subclasses of CmObject, it means all classes will have a Guid property. CmMajorObject is a subclass of CmObject and defines a Name property. LexDb is a subclass of CmMajorObject, so it inherits Name from CmMajorObject and Guid from CmObject. The class hierarchy can be viewed using the LCM Browser discussed in section 3.3

Some classes are abstract, meaning they will never show up as an instance in the data. Only subclasses of the abstract class will show up in fwdata. An example of this is MoForm. The LexemeForm property of LexEntry holds a MoForm. What is actually in the data would be one of the subclasses: MoStemAllomorph, MoAffixAllomorph, or MoAffixForm.

2.2 Writing systems

Writing systems are required in a FieldWorks project. Almost every string identifies each range of characters by a writing system. The writing system identifies a language, and various other tags that further define aspects of the writing system in this language. For example, a language may use a non-Roman script, there may be a Romanized form of the string, and there could be an IPA form of the string. Writing system tags may identify dialects of the language. A special tag is used to identify a recording of the string. Example writing system tags:

- ar: Arabic language in Arabic script
- ar-latn: Arabic language in a Romanized script
- ar-fonipa: Arabic language in the International Phonetic Alphabet
- ar-Zxxx-x-audio: An audio recording of Arabic language (SIL user-defined)

Writing systems contain one or more collations for sorting (FLEx only uses one), a list of valid characters, possible fonts and keyboards, ways to display numbers, dates, time, currency, localized names for languages, countries, and scripts, etc. FLEx does not use most of this information, but passes it along.

Writing system information is stored using the Unicode Locale Data Markup Language specification in files with .ldml as the extension. Unicode Technical Standard #35 describes this

standard at http://unicode.org/reports/tr35/. The SIL Writing System Technology Department (WSTech) interfaces with the Unicode Consortium and has a special web server api to provide standard writing system codes for our use (e.g., this returns an English ldml file: https://ldml.api.sil.org/en). See https://downloads.languagetechnology.org/fieldworks/Documentation/FieldWorks_7_Writing_systems.pdf for more information on writing systems used in FieldWorks.

Each language has a default region and a default script. WsTech maintains a list of defaults. You can see this list in C:\ProgramData\SIL\SldrCache\langtags.json. An example is en-Latn-US for English using the Latin script and the US region. This is essentially identical to en, en-Latn, and en-US. Starting inFW9.0 we no longer allow a user to have both en and en-Latn in a project, and normally it is just the language code. The ldml identity element has these 1-4 elements:

```
<language type="en" />
<script type="Latn" />
<region type="US" />
<variant type="x-sci" />
```

The name of this writing system file is en-Latn-US-x-sci.ldml.

In order to avoid a model change to deal with merged writing systems from older versions (e.g., data in both en and en-Latn), FLEx will leave the en-Latn data in fwdata, and en.ldml and en-Latn.ldml, but the latter only has language type="en" /> without the script element. In the UI, in the Writing System Property dialog, it shows “en” for both writing system codes and a red outline appears around the writing systems and it will not allow you to click OK until you rename one of the writing systems. The region code is not restricted in this way.

FieldWorks maintains two lists of writing systems for use in a project. A FLEx project is assumed to have one vernacular language which can have any number of writing systems for that language. These writing systems can be seen in Format > Set up Vernacular Writing Systems. Glosses, definitions, and other languages describing this vernacular language are stored in analysis languages. FLEx can have any number of analysis languages with any number of writing systems for each one. These writing systems can be seen in Format > Set up Analysis Writing Systems.

In the Writing System Property dialog, The Code line at the top of the General tab corresponds to the writing system tag used in the ldml file names.

If FLEx opens an .ldml file it cannot parse, it changes the ldml extension to ldml.bad and then creates a new valid ldml file in order to continue. It also adds a badldml.log file explaining what happened.

There are several elements in the LangProject class of fwdata for recording lists of writing systems. VernWss and AnalysisWss list all of the writing systems that show in the Writing System Property dialog. The Cur elements list current ones that are checked. These will normally be displayed for all multilingual fields.

```
<VernWss>
<Uni>gur gur-Zxxx-x-audio</Uni>
</VernWss>

<AnalysisWss>
<Uni>fr en</Uni>
</AnalysisWss>
```

```
<CurAnalysisWss>
<Uni>fr en</Uni>
</CurAnalysisWss>

<CurPronunWss>
<Uni>gur</Uni>
</CurPronunWss>

<CurVernWss>
<Uni>gur gur-Zxxx-x-audio</Uni>
</CurVernWss>
```

When a writing system is hidden in the UI, it is just removed from the above lists, but the data remains in the fwdata file and can be shown again using the View Hidden Writing Systems option in the green + button in Writing System Properties. If you right-click a writing system and choose the Delete menu item, it will actually remove all data in the writing system from the fwdata file and remove the associated ldml file.

There is also a global writing system store in C:\ProgramData\SIL\WritingSystemRepository\3. When a user adds a new writing system to a project, if it is present in the global store, it will copy that writing system to the project WritingSystemStore directory. If it is not in the global store, and the Internet is available, it will request the writing system from https://ldml.api.sil.org and will store this in C:\ProgramData\SIL\SldrCache and in the global store in addition to the local project. If a writing system is modified in FLEx, the project ldml file is modified and then copied to the global store.

Fonts and keyboards for writing systems are selected independently by each user. This information is stored in the SharedSettings folder inside the project folder. This information is stored in a .ulsx file with the user name with a WritingSystem element

```
<WritingSystem id="tog-018-x-leya">
  <LocalKeyboard>en-US_US_US</LocalKeyboard>
  <DefaultFontName>Charis SIL</DefaultFontName>
</WritingSystem>
```

The writing system abbreviation, language name, spell checking id, and legacy mapping is stored in LexiconSetting.plsx in this directory.

```
<WritingSystem id="tog">
  <Abbreviation>Ton</Abbreviation>
  <LanguageName>Tonga (Nyasa) (tog-Latn-018-x-leya)</LanguageName>
  <SpellCheckingId>tog</SpellCheckingId>
  <LegacyMapping>Windows1252&amp;lt;&amp;gt;Unicode</LegacyMapping>
</WritingSystem>
```

The plsx file is send via Send/Receive, but the ulsx files are not sent. They need to be set by each user.

2.3 Basic properties

Classes in FLEx have basic properties that hold actual data such as strings, numbers, dates, etc. described in the following sections.

2.3.1 Strings

All strings in FieldWorks store Unicode data. In the fwdata file it is stored in NFC normalization. There are two basic types of strings in FieldWorks, which are poorly named since all strings are in Unicode.

- Unicode: This type of string is a sequence of Unicode characters which does not allow any embedding including styles and writing systems. In the FLEx UI these strings can be recognized because the writing system and style combo boxes are disabled in the toolbar.
- String: This type of string is a sequence of Unicode characters, but can also have substrings with styles and different writing systems, plus some other specialized attributes. In the FLEx UI these strings can be recognized because the writing system and style combo boxes are enabled in the toolbar.

If we started over, we would probably have a single kind of string, but when FLEx was first developed it was using SQL Server for storage, and it was much more efficient dealing with a simple type of string compared to added complexity storing strings with embeddings. There was originally a bigger variety of each string that could hold really long strings. When switching to XML storage in FW7, we dropped the big variety because it was irrelevant, but we maintained the distinction between strings with or without embedding.

Strings in Fieldworks should never have a CRLF or tab. FLEx tries to prevent this on import, typing, pasting, etc. Occasionally something slips through. In this case, when the field is displayed, all of the text following the CRLF or tab will not show. If you place your cursor at what appears to be the end of the string, and press Delete, it will delete the CRLF or tab and you'll see the rest of the string.

FLEx can force a line break in either type of string. In the UI, this happens when you enter Shift+Enter. This inserts a U+2028 LNE SEPARATOR character in the string. This is somewhat similar to having new paragraphs in a string, although there is no paragraph formatting applied in this case.

Both types of strings can be used by itself in a property, or a property can hold multiple sets of equivalent strings in different languages or writing systems. These multiple versions are called MultiUnicode and MultString. The multiple property always has a writing system associated with each alternative string. In a multiple property, every string in the property must have a unique writing system, which means you can't have two en alternatives. The strings in a multiple property are normally the same string written in a different language (e.g., English, French, German) or writing system (e.g., normal or IPA orthography), or any combination of these. The writing system attributes in strings should always have a matching ldml file in the WritingSystemStore directory.

2.3.1.1 Unicode

The simplest type of string is a FieldWorks Unicode string

```
<Name>
<Uni>Title Secondary</Uni>
</Name>
```

This is the Name property in StStyle. This type is used for file names and internal names for styles, etc. that are used in the program, but do not have an inherent language. There is a slight

problem with this. FieldWorks needs to use a writing system in order to determine which font and keyboard to use with the string. The FieldWorks display mechanism shows boxes for characters that are not in the current font rather than substituting other fonts as needed. FieldWorks generally chooses a writing system representing the system language on your computer, if the writing system exists, otherwise it defaults to English when displaying these strings.

As in any XML file, certain characters must use character entities:

- Ampersand − &
- Single quote − '
- Greater than − >
- Less than − <
- Double quote − "

2.3.1.2 String

A String property is represented with a Str element containing one or more Run elements, each with a specified writing system. All characters in a given Run share the same attributes. Whenever a set of attributes changes, it requires a new Run. The following shows the Contents of a StTxtPara object with a single Run in English.

```
<Contents>
<Str>
<Run ws="en">A contraction is a combination of two lexemes, each of which maintains its own meaning.</Run>
</Str>
</Contents>
```

The following shows the Contents of a StTxtPara that includes multiple Runs with changes of writing system and styles. Note that you can only have one namedStyle element in a Run. If you want strong emphasized, you would need to have a single style that combines these features.

```
<Contents>
<Str>
<Run ws="en">Some English with </Run>
<Run namedStyle="Emphasized Text" ws="en">emphasized</Run>
<Run ws="en"> and </Run>
<Run namedStyle="Strong" ws="en">bold</Run>
<Run ws="en"> text. (French: </Run>
<Run ws="fr">français</Run>
<Run ws="en"> [</Run>
<Run ws="fr-fonipa">fʁɑ̃sɛ</Run>
<Run ws="en">]</Run>
</Str>
</Contents>
```

In the FLEx UI, you switch writing systems or styles using the writing system and style combo boxes in the toolbar. Whenever a writing system is changed, FLEx automatically uses the keyboard and font associated with that writing system, or a style that overrides the default font from the Writing System Properties dialog. If the font does not have a glyph for a character in the string, FLEx shows an open box in the display instead of finding a glyph in some related font as is done in Microsoft Word. Note in document view, FLEx is using a web browser view which will substitute glyphs from other fonts, but the data entry fields will show boxes.

The following shows the Contents of a paragraph with a link to an external file named x.html.

```
<Contents>
<Str>
<Run ws="en">This is a link to file </Run>
<Run externalLink="Others\x.html" namedStyle="Hyperlink" ws="en">test</Run>
<Run ws="en">. </Run>
</Str>
</Contents>
```

The following is a hyperlink to a lexical entry in the current project, but it could be to a different project. The link URL includes the project name, tool, and object guid.

```
<Contents>
<Str>
<Run ws="en">This is a link to an entry </Run>
<Run
externalLink="silfw://localhost/link?database%3dthis%24%26tool%3dlexiconEdit%26guid%3de331fd3
8-d774-4023-bc56-75e5f4ab818e%26tag%3d" namedStyle="Hyperlink" ws="en">here</Run>
<Run ws="en">.</Run>
</Str>
</Contents>
```

2.3.1.3 MultiUnicode

Many properties (fields) in FieldWorks are designed to hold strings that will never have embedding, but may have translations in different languages or writing systems (e.g., lexeme form, gloss, and names and abbreviations for list items). Here is how these MultiUnicode properties are stored in fwdata.

```
<Name>
<AUni ws="en">adjunct</AUni>
<AUni ws="fr">accessoire</AUni>
<AUni ws="es">adjunto</AUni>
</Name>
```

This field is the Name for a CmPossibility object. In this case, there are three FieldWorks Unicode strings; English, French, and Spanish. Each FieldWorks Unicode string is stored in an AUni element. Even though the FieldWorks Unicode string itself does not have a writing system, FieldWorks knows the intended writing system by the “ws” attribute of the AUni element. It is an error to have more than one AUni element in a single property with the same writing system.

2.3.1.4 MultiString

Other properties (fields) need the ability to store equivalent translations, but also need to allow embedded writing systems and formatting (e.g., definition, example sentence, example translations, all note fields). Here is how these MultiString properties are encoded.

```
<Translation>
<AStr ws="en">
<Run ws="en">You may damage the computer.</Run>
</AStr>
<AStr ws="fr">
<Run ws="fr">Vous risquez d'endommager l'ordinateur.</Run>
</AStr>
</Translation>
```

This field is the Translation property on a CmTranslation object. This example has translations for English and French. This is a simple FieldWorks String in that it only contains one run, but any number is possible as described in section 2.3.1.2. Each FieldWorks String is stored in an AStr element which identifies the overall writing system for the FieldWorks String. Normally this is the same writing system as the first run, but it does not have to be. You could have an English string that starts with a French word. It is an error to have more than one AStr element in a single property with the same writing system.

2.3.2 Binary

Binary properties are represented as follows:

```
<Sid>
<Binary>6400000001000000</Binary>
</Sid>
```

Binary data is stored as a sequence of hexadecimal bytes. Our current model only has one Binary property: Sid on UserConfigAcct. But I don't think it is used anywhere.

2.3.3 Boolean

Boolean properties are represented as follows:

```
<IsSorted val="true"/>
```

This is the IsSorted property on CmPossibilityList. The value is stored in the "val" attribute as "true" or "false". The default value for missing properties is false.

2.3.4 GenDate

GenDate provides a way to store an approximate date without a time (e.g., before March 1200 AD). They are stored as an int (4 bytes). This is a decimal representation of a generic date without a time. It's up to the software to make sure the int is a valid date. The range is 21474 BC through 21474 AD. The format is:

```
[-]YYYYMMDDP
YYYY is the 1-5 digit year (negative is BC) 0000 is unknown.
MM is the 2 digit month (for BC months it is 13 - M) 00 is unknown
DD is the 2 digit day (for BC days it is 32 - D) 00 is unknown
P is one of the following:
  0 = Before (If GenDate = 0, it means nothing is entered)
  1 = Exact
  2 = Approximate
  3 = After
```

GenDate properties are represented as follows:

```
<DateOfEvent val="202404020" />
```

This is the DateOfEvent property on RnGenericRec representing Before April 2, 2024. The generic date is stored as a decimal number in the "val" attribute. In Notebook, GenDate is available on Date of Event and uses a dialog to specify the parts of the date. GenDate is also available on CmPerson for DateOfBirth and DateOfDeath, but it is not implemented in the current FLEx UI.

2.3.5 Guid

GUID (Globally Unique Identifier) properties are represented as follows:

```
<ListVersion val="b41ff27f-5caf-4eac-92de-0f92acb0caa3"/>
```

This is the ListVersion property on CmPossibilityList. The GUID value is stored in the “val” attribute in this string format using this hex format.

Internal FLEx code often refers to an object using an hvo (handle to a viewable object). This is a single integer that is assigned to each object when it is loaded. While in memory, it is a reliable and quick way to find and reference objects. However, the next time the project is loaded, especially if objects were added or deleted, it can affect some or all of the hvos the next time it is loaded. In debugging it’s often challenging to find a guid for an object when all you have is the hvo, and fwdata only has guids.

2.3.6 Integer

Integer properties hold positive or negative 64-bit integers and are represented as follows:

```
<HomographNumber val="12"/>
```

This is the HomographNumber property on LexEntry. The value of the integer is stored as a decimal “val” attribute. The default value for missing properties is 0.

2.3.7 TextPropBinary

TextPropBinary is binary data that represents text properties. They are used in Rules and StyleRules in StStyle. In the fwdata file they are stored in a Prop element with label and value pairs.

```
<Rules>
<Prop bulNumScheme="10" bulNumStartAt="1" firstIndent="-9000" leadingIndent="21000"></Prop>
</Rules>

<StyleRules>
<Prop namedStyle="Normal" /Prop>
</StyleRules>
```

This is a list of TextPropTypes from FwKernelTlb.idl in liblcm.

ktptWs = 1	ktptLineHeight = 24	ktptBorderTrailing = 141
ktptItalic = 2	ktptParaColor = 25	ktptBorderColor = 142
ktptBold = 3	ktptSpellCheck = 26	ktptBulNumScheme = 143
ktptSuperscript = 4	ktptMarginTop = 50	ktptBulNumStartAt = 144
ktptUnderline = 5	ktptFontFamily = 1	ktptBulNumTxtBef = 145
ktptFontSize = 6	ktptCharStyle = 2	ktptBulNumTxtAft = 146
ktptOffset = 7	ktptParaStyle = 3	ktptBulNumFontInfo = 147
ktptForeColor = 8	ktptTabList = 4	ktptKeepWithNext = 148
ktptBackColor = 9	ktptTags = 5	ktptKeepTogether = 149
ktptUnderColor = 10	ktptObjData = 6	ktptHyphenate = 150

ktptBaseWs = 16	ktptCustomBullet = 7	ktptMaxLines = 151
ktptAlign = 17	ktptRightToLeft = 128	ktptCellBorderWidth = 152
ktptFirstIndent = 18	ktptDirectionDepth = 129	ktptCellSpacing = 153
ktptLeadingIndent = 19	ktptFontVariations = 130	ktptCellPadding = 154
ktptMarginLeading = 19	ktptNamedStyle = 133	ktptEditable = 155
ktptTrailingIndent = 20	ktptPadLeading = 134	ktptWsStyle = 156
ktptMarginTrailing = 20	ktptPadTrailing = 135	ktptSetRowDefaults = 159
ktptSpaceBefore = 21	ktptPadTop = 136	ktptRelLineHeight = 160
ktptMswMarginTop = 21	ktptPadBottom = 137	ktptTableRule = 161
ktptSpaceAfter = 22	ktptBorderTop = 138	ktptWidowOrphanControl = 162
ktptMarginBottom = 22	ktptBorderBottom = 139	ktptFieldName = 9998
ktptTabDef = 23	ktptBorderLeading = 140	ktptMarkItem = 9999

2.3.8 Time

Time properties store a date and time and are represented as follows:

```
<DateModified val="2009-9-23 20:53:23.390"/>
```

This is the DateModified property on CmMajorObject. The time is stored in the “val” attribute in the format YYYY-MM-DD HH:MM:SS.TTT (TTT is thousandths of a second). This format should be used regardless of the locale of the computer. If the format is incorrect, usually Tools > Utilities > Write Everything will fix them.

2.4 Linking properties

Each instance of a class has a guid that identifies that object. There are several ways of connecting objects together to form entries, interlinear texts, etc. described in the following sections.

There are two types of linking:

- Owning – most objects are in an ownership hierarchy, normally starting from LangProject. An object can own any number of objects through owning properties, but each object can only have one owner.
- Reference – any object can reference other objects through reference properties.

In both cases, linking properties can be of three cardinalities:

- Atomic – links to zero or one other object.
- Collection – links to any number of objects where the order is unspecified at the conceptual model level. They may be sorted by the program during use.
- Sequence – links to any number of objects where the order is significant.

2.5 Owning relationships

Most objects in FLEx are owned by other objects that eventually lead to LangProject at the top level of the ownership hierarchy. For example, LangProject has a PartsOfSpeech property that

owns a CmPossibilityList object and the CmPossibilityList object has a Possibilities property that owns PartOfSpeech objects (a subclass of CmPossibility.

Ownership links always involve two links: one from the owner to the owned object, and the other from the owned object back to the owner. The owning class has an owning property that holds an objsur (object surrogate) element with a ‘t’ type attribute set to “o” specifying an owning relationship. The owned object has an ownerguid attribute in the rt header that holds the guid of the owning class. An object can only have one owning objsur reference from one owner since an object can never be owned in multiple places.

The following example illustrates how this works where the Discussion owning property in a CmPossibility owns an StText object which in turn owns one or more StTxtPara objects via the Paragraphs owning property. The actual text of the paragraph is in the StTxtPara object. In this case, the StTxtPara does not have a Contents property, so it doesn’t have any text.

```
<rt class="CmPossibility" guid="80a0dddb-8b4b-4454-b872-88adec6f2aba" ownerguid="d7f71649-e8cf-11d3-9764-00c04f186933">
...
<Discussion>
<objsur guid="3aeef2e2-9466-41d5-afa7-d569f667fc79" t="o" />
</Discussion>
</rt>

<rt class="StText" guid="3aeef2e2-9466-41d5-afa7-d569f667fc79" ownerguid="80a0dddb-8b4b-4454-b872-88adec6f2aba">
<Paragraphs>
<objsur guid="9555b62c-51f6-4b2a-ba51-944c5d69f4c1" t="o" />
</Paragraphs>
</rt>

<rt class="StTxtPara" guid="9555b62c-51f6-4b2a-ba51-944c5d69f4c1" ownerguid="3aeef2e2-9466-41d5-afa7-d569f667fc79">
</rt>
```

The yellow links show that the StText is owned by the CmPossibilty, but do not give an indication of the owning property. The green links show that the StText class is owned through the Discussion property of the CmPossibility, and show that the StTxtPara is owned by the StText. The blue links show that the StTxtPara is owned through the Paragraphs property of the StText. Using owning relationships, you can always trace up or down the ownership hierarchy using the guids. If you want to find the owning object for a guid in fwdata using a text editor, you can search for

" guid="<the guid>

If you just search for the guid, you may also get many hits for owning and reference links to the object.

Owning properties can be Atomic, Collection, or Sequence properties. Atomic owning elements have zero or one objsur element. Collection owning elements have any number of objsur elements and the order is unspecified at the conceptual model level. Sequence owning elements have any number of objsur elements and the order is significant.

Not all objects have ownerguid attributes. Certain objects such as LexEntry and WfiWordform are unowned. There is only one LexDb in a project, and all LexEntry objects are available through a virtual property, so we don’t need to store all of the ownerguids and objsurs. All WfiWordforms were originally owned by a WordformInventory, but since the

WordformInventory had no other purpose, it was removed from the model leaving the WfiWordforms unowned, but again available via a virtual property. The full list of classes that do not have owners is: LangProject, LexEntry, PunctuationForm, ScrRefSystem, Text, VirtualOrdering, and WfiWordform. CmPicture and CmPossibilityList may or may not have owners.

When you delete an object from FLEx, or via LCM code, it deletes the object and all objects in its ownership hierarchy. This will properly remove the objects plus all owning and reference links to all of the deleted objects. It's more challenging to delete objects straight from fwdata because you need to include all of the objects in the ownership hierarchy and delete them as well. If you succeed in getting all of the proper rt elements deleted, you can then run Tools > Utilities > Find and fix… which will clear out all of the orphaned owning and reference objsur links to those objects.

2.6 Reference relationships

Reference properties provide links to other objects without involving ownership hierarchy. Reference links are one-way in the model, although using virtual properties, it's possible to find objects referencing a given object. When you remove a reference to an object, both objects remain intact.

Reference properties are similar to owning properties in that both use objsur elements. The difference is that reference properties use the 't' attribute set to "r" for reference. The following example demonstrates how senses reference semantic domains through a SemanticDomains reference collection property.

```
<rt class="LexSense" guid="e79c8d25-eb3d-43ec-94b7-ccb4fead1d40" ownerguid="d6e17340-dc0f-458a-9fd6-c8de368918d5">
...
<SemanticDomains>
<objsur guid="ba06de9e-63e1-43e6-ae94-77bea498379a" t="r" />
<objsur guid="191ca5a5-0a67-426e-adfc-6fdf7c2aaa2c" t="r" />
</SemanticDomains>
</rt>

<rt guid="191ca5a5-0a67-426e-adfc-6fdf7c2aaa2c" class="CmSemanticDomain"
ownerguid="dc177f3c-d0fd-4232-adf1-a77b339cdbb2">
<Name>
<AUni ws="en">House</AUni>
</Name>
<Abbreviation>
<AUni ws="en">6.5.1.1</AUni>
</Abbreviation>
...
</rt>

<rt guid="ba06de9e-63e1-43e6-ae94-77bea498379a" class="CmSemanticDomain"
ownerguid="c924bfce-beed-4382-95e8-62b54951c83d">
<Name>
<AUni ws="en">Person</AUni>
</Name>
<Abbreviation>
<AUni ws="en">2</AUni>
</Abbreviation>
```

```
...
</rt>
```

The SemanticDomains element of the sense contains two objsur elements. Each one has a ‘t’ attribute set to ‘r’ to indicate it is a reference link. It also contains a guid attribute that matches the corresponding guid attribute of a CmSemanticDomain object. The yellow semantic domain link is to the ‘Person’ CmSemanticDomain and the green link is to the ‘House’ CmSemanticDomain.

Unlike an owning property, there is no corresponding link from the CmSemanticDomain back to the LexSense that references it. A reference link is a one-way link. If you want to find all of the senses that reference a particular CmSemanticDomain, you would need to search all SemanticDomains elements looking for ones that have an objsur with a matching guid. In code, CmSemanticDomain has a ReferringSenses virtual method to return these senses.

Reference collections can also be atomic, collection, or sequence. Atomic reference elements have zero or one objsur element. Collection reference elements have any number of objsur elements and the order is unspecified at the conceptual model level. Sequence reference elements have any number of objsur elements and the order is significant.

Whenever an object is deleted in code, any references to the deleted object are removed from the referring objects. If something goes wrong and a reference property points to a non-existent object, FLEx will likely crash at some point. The problem can be fixed easily by closing the project, opening another project and running Tools > Utilities > Find and fix.. and specify the desired project. This will remove any orphaned owning and reference links. In the fwdata file, you would remove any orphaned objsur links. If it’s the only link, remove the reference property (e.g., SemanticDonains).

2.7 Styles

Default paragraph and character styles are available in all projects. The master list is in C:\Program Files\SIL\FieldWorks 9\Language Explorer\FlexStyles.xml, but this is mainly used for upgrading styles when changes are needed. Users can add new styles as needed, or modify existing styles. Built-in styles cannot be deleted. Styles are hierarchical, so styles based on another style inherit styles from higher in the hierarchy. Paragraph styles are used in Dictionary Configuration for formatting elements that are treated as a paragraph which includes entries, and some other elements such as senses and examples. Character styles can be embedded in strings. Styles are stored in the Styles owning property of LangProject in StStyle classes.

This shows some of the elements in StStyle. BasedOn is used for hierarchy. The Name is Unicode, so it is not localizable and is the name that is used in Strings in namedStyle attributes. Type is 0 for Paragraph and 1 for Character styles.

```
<rt class="StStyle" guid="7a194e30-4c8c-4934-83ad-b12ecc47a768" ownerguid="af1d4e34-ea5e-11de-8ec5-0013722f8dec">
<BasedOn>
<objsur guid="7d2c80cd-af87-4d56-8337-cfd389b775bb" t="r" />
</BasedOn>
<IsBuiltIn val="True" />
<Name>
<Uni>Dictionary-Normal</Uni>
```

</Name>
<Rules>
<Prop firstIndent="-12000" leadingIndent="9000" spaceAfter="2000" spaceBefore="1000"></Prop>
</Rules>
<Type val="0" />
</rt>

Format > Styles is used to make changes to styles.

Styles may override the font that is defined in the Writing System Properties dialog. In the Normal style, you can specify a font and size for each writing system that affects how it looks in every location for all colleagues. When you do this, it hard-codes a font for everyone using the project. When this is not done, each user can assign a different font in Writing System Properties that will be used only in their version of the project. But if a font is specified in a style, every user will need to have the font available on their computer for the project to display properly, and the font sizes and other attributes will apply to all users of the project.

2.8 Possibility lists

Rather than letting users type (and mistype) fixed lists, FLEx stores many lists to allow users to choose options in various areas, including parts of speech, morpheme types, semantic domains, people, locations, types for complex forms and minor entries, lexical relations, etc. The class for these lists is CmPossibilityList, and items in the list can be flat or hierarchical. The base class for the items is CmPossibility. There are many properties on both of these classes including a MultiUnicode Name and MultiUnicode Abbreviation. A user can modify some of the properties of a list in the Lists area by going to Tools > Configure > List. Users can also create custom lists for special purposes.

List properties include sorting, whether hierarchy is wanted, and if so, how many levels, the writing system(s) used, whether duplicates are allowed, the class of items to include in the list, etc. List item properties include a MultiString description, properties for displaying items, etc.

Lists are used in various classes through reference relationships, which may allow a single reference or multiple references to guids of a CmPossibility.

Here are some key properties for a basic list with 2 nested items.

<rt class="CmPossibilityList" guid="c06b099c-ea5e-11de-890c-0013722f8dec" ownerguid="f0065835-afbc-477d-9a7c-131f3fa25139">
<ItemClsid val="7" />
<Name>
<AUni ws="en">Genres</AUni>
</Name>
<Possibilities>
<objsur guid="c076f554-ea5e-11de-9793-0013722f8dec" t="o" />
</Possibilities>
<WsSelector val="-3" />
</rt>

<rt class="CmPossibility" guid="c076f554-ea5e-11de-9793-0013722f8dec" ownerguid="c06b099c-ea5e-11de-890c-0013722f8dec">
<Name>

<AUni ws="en">Monologue</AUni>
</Name>
<SubPossibilities>
<objsur guid="c082e102-ea5e-11de-92f2-0013722f8dec" t="o" />
</SubPossibilities>
</rt>

<rt class="CmPossibility" guid="c082e102-ea5e-11de-92f2-0013722f8dec" ownerguid="c076f554-ea5e-11de-9793-0013722f8dec">
<Name>
<AUni ws="en">Narrative</AUni>
</Name>
</rt>

In this Genres list, we show a Monologue item which has a nested Narrative item. There are two important properties in the CmPossibilityList

- ItemClsid: Gives the class ID for the items that can be included in this list. 7 in this case is the class ID for CmPossibility, so this list can only hold CmPossibility objects.
- WsSelector: This selects the writing systems to use when displaying list items for this list. In this case, we would show all checked analysis writing systems in Analysis Writing System Properties dialog. The possibilities for this property are:
  - -1 The first analysis writing system.
  - -2 The first vernacular writing system.
  - -3 All checked analysis writing systems.
  - -4 All checked vernacular writing systems.
  - -5 All checked analysis then all checked vernacular writing systems.
  - -6 All checked vernacular then all checked analysis writing systems.

There are 13 subclasses of CmPossibility, mostly to provide additional properties for certain classes. These are the subclasses with their class ids.

- 5125 ChkTerm (previously used by Translation Editor)
  - 35 CmAnnotationDefn (internal)
  - 26 CmAnthroItem (Anthropology Categories list)
  - 27 CmCustomItem (Custom lists)
  - 12 CmLocation (Locations list)
  - 13 CmPerson (People list)
  - 66 CmSemanticDomain (Semantic Domain list)
  - 5118 LexEntryType (Entry type list)
  - 5113 LexEntryInflType (Entry type list)
  - 5119 LexRefType (Lexical Relations list)
  - 5042 MoMorphType (Morpheme Types list)
  - 5049 PartOfSpeech (Grammatical Category list)
  - 5132 PhPhonRuleFeat (internal Phonological Rule Features list in Grammar, created dynamically based on exception features and inflection classes)

See ModelDocumentation.chm for more information on the subclasses.

There are some lists that are used internally in the program and do not show up in the Lists area of FLEx.

All lists, including the internal ones can be exported using File > Export > All lists XML. This produces one XML file containing all 36 lists, including the internally-used lists and custom lists.

The export includes common fields including guids and class ids. Instructions are provided in the Export dialog for converting this file into an htm file for easy viewing in a browser.

Three lists listed in the Lists area are unique

- Reversal Index Categories. Each reversal index has its own PartsOfSpeech list. This tool in the Lists area lets you pick one of the reversal indices from a chooser in the title bar, and it then displays the items in that list for editing.
- Feature Types. This is not really a CmPossibilityList. Instead, it is a list of FsFeatStrucType objects owned in the Types property of the FsFeatureSystem used in the Grammar area.
- Lexical Relations. This list defines cross references and lexical relations and holds data for these relations.

This table gives list name, owning object, and owning property for lists in the Lists and Grammar areas.

List Name	Owning Object	Owning Property
Academic Domains	LexDb	DomainTypes
Anthropology Categories	LangProject	AnthroList
Category (Parts Of Speech)	LangProject	PartsOfSpeech
Complex Form Types	LexDb	ComplexEntryTypes
Confidence Levels	LangProject	ConfidenceLevels
Dialect Labels	LexDb	DialectLabels
Education Levels	LangProject	Education
Extended Note Types	LexDb	ExtendedNoteTypes
Feature Types	FsFeatatureSystem	Types
Genres	LangProject	GenreList
Languages	LexDb	Languages
Lexical Relations	LexDb	References
Locations	LangProject	Locations
Morpheme Types	LexDb	MorphTypes
Notebook Record Types	RnResearchNbk	RecTypes
People	LangProject	People
Positions	LangProject	Positions
Publications	LexDb	PublicationTypes
Restrictions	LangProject	Restrictions
Reversal Index Categories	ReversalIndex	PartsOfSpeech
Roles	LangProject	Roles
Semantic Domains	LangProject	SemanticDomainsList
Sense Types	LexDb	SenseTypes
Status	LangProject	Status
Text Chart Markers	DsDiscourseData	ChartMarkers
Text Constituent Chart Templates	DsDiscourseData	ConstChartTempl
Text Markup Tags	LangProject	TextMarkupTags
Time of Day	LangProject	TimeOfDay
Translation Types	LangProject	TranslationTags
Usages	LexDb	UsageTypes
Variant Types	LexDb	VariantEntryTypes

2.9 Custom fields

FLEx allows users to create custom fields in Tools > Configure > Custom Fields for the following classes:

- LexEntry
- LexSense
- LexExampleSentence
- MoForm used in AllomorphIndex on LexDb shown in the Allomorphs section of an entry.
- Segment in interlinear text
- RnGenericRec in data notebook

When a user deletes a custom field, all the data will also be deleted.

When custom fields are defined, an AdditionalFields element is added to the languageproject element if not already there. AdditionalFields holds CustomField elements, each one defining a custom field defined by the user that is unique to this project. Here are some possibilities.

```
<AdditionalFields>
<CustomField name="Weather" class="RnGenericRec" destclass="7" type="RC" wsSelector="-3"
helpString="originally a standard part of Data Notebook records" listRoot="2bdd1159-f9b2-11d3-977b-
00c04f186933" />
<CustomField name=" My Complex String" class="LexEntry" type="String" wsSelector="-1" />
<CustomField name=" My Simple Strings" class="LexEntry" type="MultiUnicode" wsSelector="-3" />
<CustomField name=" My Paragraphs" class="LexEntry" destclass="14" type="OA" />
<CustomField name=" My List Items " class="LexEntry" destclass="7" type="RC" listRoot="d7f713a0-
e8cf-11d3-9764-00c04f186933" />
<CustomField name="My Date" class="LexEntry" type="GenDate" />
<CustomField name="My Number" class="LexEntry" type="Integer" />
</AdditionalFields>
```

Possible attributes of the CustomField element are

- name—Stores the original user-defined name of the custom field. A custom name is not allowed if there is already a built-in or custom field with that name.
- label—Added when a user modifies a custom field name. The label is shown in the UI, but data in fwdata still uses the original name.
- class—Specifies the class name to which the custom field applies.
- destclass—(optional) Specifies the class number of the object owned or referenced by this field.
- type—Specifies the kind of field. These are the options currently supported in the UI.
  - String—Single-line Text (first analysis or vernacular). This field stores a String allowing embedding (It actually stores MultiString, but the UI limits it to one writing system.)
  - MultiUnicode—Single-line Text (all analysis and/or vernacular). This field stores a MultiUnicode with no embedding.
  - OA—Multiparagraph Text. This field stores an owning atomic StText allowing multiple paragraphs. It works well for Data Notebook, but should probably not be used in the lexicon (use Shift+Enter instead).
  - GenDate—Date. The field holds a generic date stored as an integer. The format is described in section 2.3.4.
  - RC—List Reference (multiple items). This field holds a reference collection to multiple list items.

    - RA—List Reference (single item). This field holds an atomic reference to a single list item.
    - Integer—Number. The field holds an integer value
- wsSelector—(optional) This number determines the default writing system(s) for this field. Possibilities are
    - -1—The first analysis writing system.
    - -2—The first vernacular writing system.
    - -3—All checked analysis writing systems.
    - -4—All checked vernacular writing systems.
    - -5—All checked analysis then all checked vernacular writing systems.
    - -6—All checked vernacular then all checked analysis writing systems.
    - Note: Segment only offers Single-line Text with first analysis or vernacular.
- listRoot—For reference properties this holds the guid of the CmPossibilityList (including custom lists) that owns the items that can be held in this reference property.
- helpString—(optional) This is a user-defined description of how the field is used.

The following examples show how data is stored in custom fields. The name attribute matches the name attribute of the custom field (not the label that the user sees in the UI).

This is an example of a custom Single-line Text field set to First Analysis or First Vernacular writing system. The underlying type is a MultiString field, although the UI does not enable more than one writing system. The string itself can have embedded information.

```
<Custom name="My Complex String">
<AStr ws="en">
<Run ws="en">This is the field content.</Run>
</AStr>
</Custom>
```

This is an example of a custom Single-line Text field set to All ... writing systems. The underlying type is a MultiUnicode field, so it cannot have embedded information. The UI follows normal writing system rules for MultiUnicode fields.

```
<Custom name="My Simple Strings">
<AUni ws="en">house</AUni>
<AUni ws="fr">maison</AUni>
</Custom>
```

This is an example of a Multiparagraph Text custom field. This is always an owning atomic property that holds one StText. This example only shows the StText guid. The rest of the objects would be defined as normal <rt> elements.

```
<Custom name="My Paragraphs">
<objsur guid="c2ac5444-26cf-4318-9e08-1f12d64e173e" t="o" />
</Custom>
```

This is an example of a List Reference (multiple items) custom field. The underlying type is a reference collection field which holds references to objects. The (single item) type would be the same except the UI only allows it to hold one item.

```
<Custom name="My List Items">
<objsur guid="d7f713a5-e8cf-11d3-9764-00c04f186933" t="r" />
<objsur guid="d7f713a6-e8cf-11d3-9764-00c04f186933" t="r" />
</Custom>
```

This is an example of a date custom field. The underlying type is a GenDate field.

```
<Custom name="My Date" val="201011171" />
```

This is an example of a number custom field. The underlying type is an integer field.

```
<Custom name="Number" val="45" />
```

2.10 Send/Receive Data

Send/Receive allows users to collaborate in a single FLEx project. For full information on Send/Receive see https://downloads.languagetechnology.org/fieldworks/Documentation/Technical%20Notes%20on%20FieldWorks%20Send-Receive.pdf.

Although designed for Send/Receive, the format used in Send/Receive may prove more useful than fwdata for certain purposes. It's always possible to do a Send/Receive to USB to provide the Send/Receive format even if you are not normally using Send/Receive to collaborate with others. With appropriate mercurial commands, it's possible to make changes to the files and then load those changes into your FLEx project. This is one way to import a possibility list. See section 4.8 in the above linked file for details on this.

Before sending to a repository, the fwdata file is split into numerous files based on domains. This is done to reduce the size of files that need to be merged, and to provide appropriate merge logic for different kinds of files. In general, the format of the split files is similar to fwdata, however the significant difference is that instead of each object being a separate rt record as in fwdata, the files are based on certain objects, and owned objects within those objects.

The fwdata file gets split into these directories, with file extensions for the files in that directory.

Anthropology (.ntbk, .list)
General (.filter, .style, .langproj, .pictures, .orderings, .list)
Lexicon.fwstub
Lexicon.fwstub.ChorusNotes (Messages in entries)
Linguistics
    Discourse (.discourse, .list)
    Inventory (.inventory 10 files)
    Lexicon (.lexdb 10 files, .list)
    MorphologyAndSyntax (.agents, .morphdata, .feasys, .list)
    Phonology (.phondata, .featsys, .list)
    Reversals
        en (.reversal, .list)
        <separate directory for each index>

Here is a simple entry in FLEx with an example sentence and a translation.

maison *n* A house or dwelling. *Il a quitté la maison à 21 ans.* He left home when he was 21.

Heres how this entry would be stored in .fwdata with some extra fields removed. The rt objects are sorted by guid, so this is the order they would occur, with many other classes mixed between these classes. For links to possibilities, I've just added the string in {} rather than adding the actual possibility classes. I've also highlighted in green the actual text that appears in the entry display.

<rt class="MoStemMsa" guid="3619197f-1bb1-4fc3-93bb-207561a14137" ownerguid="acc2031e-4ee4-4532-b151-a0b3eae8ece3">
<PartOfSpeech>
<objsur guid="a8e41fd3-e343-4c7c-aa05-01ea3dd5cfb5" t="r" /> {n}
</PartOfSpeech>
</rt>

<rt class="LexSense" guid="5721a00b-2e17-451b-a004-ba02dfb0a761" ownerguid="acc2031e-4ee4-4532-b151-a0b3eae8ece3">
<Definition>
<AStr ws="en">
<Run ws="en">A house or dwelling.</Run>
</AStr>
</Definition>
<Examples>
<objsur guid="a389fd40-0cc0-440a-b25f-14f54df2c968" t="o" />
</Examples>
<MorphoSyntaxAnalysis>
<objsur guid="3619197f-1bb1-4fc3-93bb-207561a14137" t="r" />
</MorphoSyntaxAnalysis>
</rt>

<rt class="LexExampleSentence" guid="a389fd40-0cc0-440a-b25f-14f54df2c968" ownerguid="5721a00b-2e17-451b-a004-ba02dfb0a761">
<Example>
<AStr ws="fr">
<Run ws="fr">Il a quitté la maison à 21 ans.</Run>
</AStr>
</Example>
<Translations>
<objsur guid="f046a4dc-ae1f-4fbe-89cc-41e8d2ca31f6" t="o" />
</Translations>
</rt>

<rt class="LexEntry" guid="acc2031e-4ee4-4532-b151-a0b3eae8ece3">
<Custom name="Date" val="0" />
<LexemeForm>
<objsur guid="dd9a2fcf-28b9-451b-aad9-5ff92837549e" t="o" />
</LexemeForm>
<MorphoSyntaxAnalyses>
<objsur guid="3619197f-1bb1-4fc3-93bb-207561a14137" t="o" />
</MorphoSyntaxAnalyses>
<Senses>
<objsur guid="5721a00b-2e17-451b-a004-ba02dfb0a761" t="o" />
</Senses>
</rt>

<rt class="MoStemAllomorph" guid="dd9a2fcf-28b9-451b-aad9-5ff92837549e" ownerguid="acc2031e-4ee4-4532-b151-a0b3eae8ece3">
<Form>
<AUni ws="fr">maison</AUni>
</Form>
<MorphType>
<objsur guid="d7f713e8-e8cf-11d3-9764-00c04f186933" t="r" /> {stem}
</MorphType>
</rt>

```
<rt class="CmTranslation" guid="f046a4dc-ae1f-4fbe-89cc-41e8d2ca31f6" ownerguid="a389fd40-
0cc0-440a-b25f-14f54df2c968">
<Translation>
<AStr ws="en">
<Run ws="en">He left home when he was 21.</Run>
</AStr>
</Translation>
<Type>
<objsur guid="d7f7164a-e8cf-11d3-9764-00c04f186933" t="r" />                    { Free translation}
</Type>
</rt>
```

Here's the same sample in a .lexdb file used in Send/Receive. In this format, all of the owned objects for the entry and sense are included inside the owned objects rather than scattered around in fwdata, thus ownerguid is not needed. It's also indented to see the hierarchy. Deleting an entry is much easier because you just need to delete everything between <LexEntry and </LexEntry>.

```
<LexEntry
  guid="acc2031e-4ee4-4532-b151-a0b3eae8ece3">
  <LexemeForm>
    <MoStemAllomorph
      guid="dd9a2fcf-28b9-451b-aad9-5ff92837549e">
      <Form>
        <AUni
          ws="fr">maison</AUni>
      </Form>
      <MorphType>
        <objsur
          guid="d7f713e8-e8cf-11d3-9764-00c04f186933"              {stem}
          t="r" />
      </MorphType>
    </MoStemAllomorph>
  </LexemeForm>
  <MorphoSyntaxAnalyses>
    <MoStemMsa
      guid="3619197f-1bb1-4fc3-93bb-207561a14137">
      <PartOfSpeech>
        <objsur
          guid="a8e41fd3-e343-4c7c-aa05-01ea3dd5cfb5"                  {n}
          t="r" />
      </PartOfSpeech>
    </MoStemMsa>
  </MorphoSyntaxAnalyses>
  <Senses>
    <ownseq
      class="LexSense"
      guid="5721a00b-2e17-451b-a004-ba02dfb0a761">
      <Definition>
        <AStr
          ws="en">
          <Run
            ws="en">A house or dwelling.</Run>
        </AStr>
      </Definition>
      <Examples>
        <ownseq
```

```
            class="LexExampleSentence"
            guid="a389fd40-0cc0-440a-b25f-14f54df2c968">
            <Example>
              <AStr
                ws="fr">
                <Run
                  ws="fr">Il a quitté la maison à 21 ans.</Run>
              </AStr>
            </Example>
            <Translations>
              <CmTranslation
                guid="f046a4dc-ae1f-4fbe-89cc-41e8d2ca31f6">
                <Translation>
                  <AStr
                    ws="en">
                    <Run
                      ws="en">He left home when he was 21.</Run>
                  </AStr>
                </Translation>
                <Type>
                  <objsur
                    guid="d7f7164a-e8cf-11d3-9764-00c04f186933"          {Free translation}
                    t="r" />
                </Type>
              </CmTranslation>
            </Translations>
          </ownseq>
        </Examples>
        <MorphoSyntaxAnalysis>
          <objsur
            guid="3619197f-1bb1-4fc3-93bb-207561a14137"
            t="r" />
        </MorphoSyntaxAnalysis>
      </ownseq>
    </Senses>
  </LexEntry>
```

LIFT is not being discussed in this document, but for comparison, this is how this entry would look in LIFT. Documentation for LIFT is in https://downloads.languagetechnology.org/fieldworks/Documentation/Technical%20Notes%20on%20LIFT%20used%20in%20FLEx.pdf. FLEx can import and export LIFT format.

```
<entry dateCreated="2024-04-05T15:20:42Z" dateModified="2024-04-05T15:22:50Z"
id="maison_acc2031e-4ee4-4532-b151-a0b3eae8ece3" guid="acc2031e-4ee4-4532-b151-
a0b3eae8ece3">
<lexical-unit>
<form lang="fr"><text>maison</text></form>
</lexical-unit>
<trait  name="morph-type" value="stem"/>
<trait name="do-not-publish-in" value="Zook"/>
<sense id="5721a00b-2e17-451b-a004-ba02dfb0a761">
<grammatical-info value="Noun">
</grammatical-info>
<definition>
<form lang="en"><text>A house or dwelling.</text></form>
</definition>
```

```
<example>
<form lang="fr"><text>Il a quitté la maison à 21 ans.</text></form>
<translation type="Free translation">
<form lang="en"><text>He left home when he was 21.</text></form>
</translation>
</example>
</sense>
</entry>
```

3 Conceptual model resources

There are various tools and resources that can help with understanding the FLEx model and data. One way to see how something is modeled in FLEx is to search the fwdata file to see what is in the file for a particular object. You could type some unique string in the area you are interested in, and then search for that in fwdata. For an entry, you can Ctrl+right-click the entry in the dictionary or dictionary preview, and choose Inspect Entry. This opens a window with the entry guid at the top, and the xhtml data at the bottom.

In FLEx, Tools > Utilities > "Find and fix errors in a FieldWorks Data (XML)" will remove any links to non-existent objects. It will also delete objects that are missing owners. This can correct occasional problems that occur within the program, but is also useful for manual editing the fwdata file. If you delete a LexEntry rt object and then Run this utility, it will remove all of the ownership links, delete all objects that were owned by the entry, and remove all references to any of the deleted objects. To run the Find and Fix Utility, you need to close the target project, then open another project and run the utilities from that project specifying the target file for the utility.

The Write Everything utility will rewrite the current project fwdata in the standard way which will clean up manual changes that may be missing some null properties, improperly formatted dates, etc.

ZEdit.exe is a simple editor that is installed with FieldWorks and is available by typing Zedit into Windows Start or find it in c:\Program Files\SIL\FieldWorks 9. It memory-maps the data when a file is opened, so it can open large fwdata files very quickly and can be used to make changes. Just be sure to close the fwdata file before trying to do anything in FLEx that tries to update fwdata, or FLEx will be blocked. Notepad++ takes a lot longer to open a file, but it can remain open while FLEx is being used. One of its advantages is supporting regex for search and replace. In any text editor you can find the desired class by searching for
" guid=" followed by the guid.

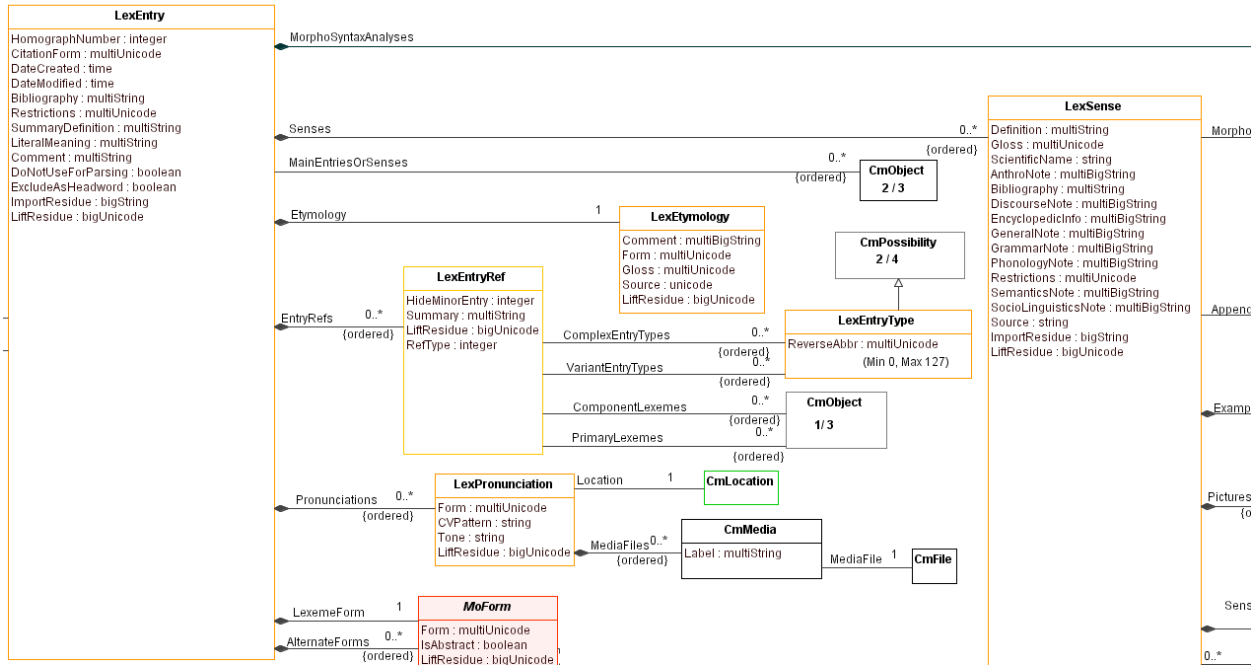
3.1 Model diagram chm

A useful model diagram can be downloaded from https://downloads.languagetechnology.org/fieldworks/Documentation/ModelDocumentation.chm. If you are using Windows 10 or 11, the file will be blocked by Windows after a download. To enable it, right-click the downloaded file and choose Properties, then check the "unblock" box.

Unfortunately, the program that was used to develop this disappeared many years ago, so we haven't been able to keep it up with the latest changes. However, for the most part it is still accurate and helpful. Interlinear text is not correct, and directional links for reversal indexes are wrong, and some newer things are missing. It also has all of the classes for Scripture used by

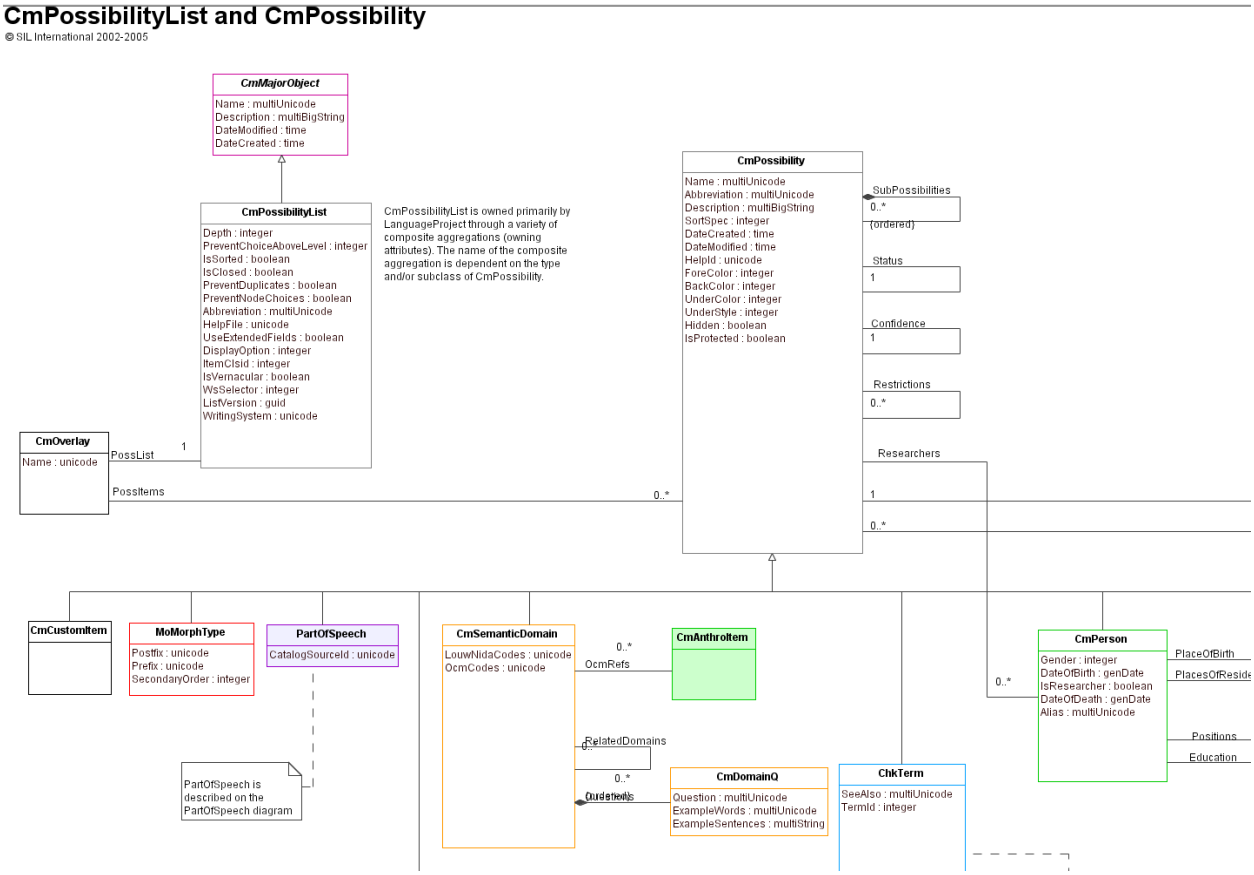
Translation Editor which no longer exists. Basic Scripture classes are still used when loading data from Paratext, but most of the other classes are unused.

The chm file has a Diagrams link in the upper left pane that lists 25 diagrams for different areas of the model. Here's part of the Lexical Database diagram.



Properties inside the class box are basic properties. Links are shown with lines between boxes, starting with a diamond for owning relations and no diamond for reference properties. Somewhere along the link lines, usually at the destination, there is a number indicating how many objects can be linked with this property. 1 means it's atomic and 0..* means it's a sequence or a collection. A sequence property has (ordered) along the line. The property name is given above the line, usually at the source. Class hierarchy is shown sometimes, but not always.

Here is a diagram of lists that does show the class hierarchy



The vertical lines with an open arrow are class hierarchy lines with the arrow pointing to the superclass.

If you click a property line linking classes, it will show information for that property. The Association End information is no longer relevant.

View: Hide Browser | Browser on the left | Browser on the right | Dictionary
Report: General Info | Association Ends | Tagged Values

Association **LexemeForm**

The lexeme form is the form of the entry that is normally used by linguists in interlinear text. The lexeme form is either the default allomorph f this lexEntry (e.g. French 'march' as opposed to citation form 'marcher') or the abstract underlying representation of it. (e.g. -[C][V]^2) Wh it is an abstract representation, the IsAbstract property is set. The lexeme form is equivalent to the base form in LinguaLinks or the \lx field i MDF files.

General Info	
Name	LexemeForm
Abstract	false
Leaf	false
Root	false

Association Ends			
	Name	**Multiplicity**	**Element**
End A	<unnamed>		LexEntry
End B	<unnamed>	1	MoForm

Tagged Values		
Tag Definition/Tag Name	**Value**	**Documentation**
Tag Definition: num : String[1]	29	

If you click a class, it shows basic, owning, and reference attributes along with Back references. Back references are no longer relevant (that was in SQL Server) but the other fields

LexEntry

num:2	abbrev: ent	abstract: false	module: Ling(num:5)	base: C

Basic Attributes (sorted by attribute name)

Type	Name	Num	Signature	Other	
Basic	**Bibliography**	12	MultiString		
Basic	**CitationForm**	3	MultiUnicode		
Basic	**Comment**	25	MultiString		
Basic	**DateCreated**	5	Time		
Basic	**DateModified**	6	Time		
Basic	**DoNotUseForParsing**	26	Boolean		
Basic	**ExcludeAsHeadword**	27	Boolean		
Basic	**HomographNumber**	1	Integer	min: 0	max:255
Basic	**ImportResidue**	32	String-big		
Basic	**LiftResidue**	33	Unicode-big		
Basic	**LiteralMeaning**	18	MultiString		
Basic	**Restrictions**	14	MultiUnicode		
Basic	**SummaryDefinition**	17	MultiString		

Owning and reference attributes (sorted by type and then attribute name)

Type	Name	Num	Card	Sig
Refer	**MainEntriesOrSenses**	19	sequence	CmObject
Owning	**AlternateForms**	30	sequence	MoForm
Owning	**EntryRefs**	34	sequence	LexEntryRef
Owning	**Etymology**	13	atomic	LexEtymology
Owning	**LexemeForm**	29	atomic	MoForm
Owning	**MorphoSyntaxAnalyses**	9	collection	MoMorphSynAnalysis
Owning	**Pronunciations**	31	sequence	LexPronunciation
Owning	**Senses**	11	sequence	LexSense

“There is a Show All docs” and “Hide All docs” button at the top that shows or hides documentation for properties. Or if you click one of the right triangles at the left of the property it shows the documentation for that property. Note that class numbers are actually formed by a 1-digit module number and a 3-digit class number within the module. Module 5 is for Linguistics and the number for LexEntry is 2, so the class number for LexEntry is 5002.

3.2 Model class/property spreadsheet

If you download the source code liblcm (SIL Language & Culture Model), the MasterLCModel.xml file is the current master file with information on all classes and properties with some documentation. This is located in liblcm\SIL.LCModel\src. MasterLCModel.xsd

defines the XML structure in this file. You can get to this file directly from https://github.com/sillsdev/liblcm/blob/master/src/SIL.LCModel/MasterLCModel.xml.

The information for this model file has been put into a spreadsheet that can be downloaded from https://downloads.languagetechnology.org/fieldworks/Documentation/MasterFieldWorksModel%20classes%20and%20fields%207000072.xlsx. This spreadsheet is one way you can find accurate information for all classes and fields. This spreadsheet has two tabs, one for classes and the other for fields. The labels along the top all have dropdown menus that let you sort on that column, or you can use normal Data tab Sort and Filter options in Excel for custom filters or sorting.

The Classes tab has these columns that come from the master model file.

- Clsid; This is the class id (comes from the module and the class number).
- Class: The name of the class.
- Base: The base class for this class in the class hierarchy.
- Depth: The depth in the class hierarchy from CmObject.
- Abstract: True means the class cannot occur in data. Only subclasses can exist. False means this class can occur in data.
- Abbr: This is an abbreviation for the class. It's no longer used.
- Singleton: This is true if only one occurrence can occur in a project.
- Owner: None means there is never an owner for this class in the ownership hierarchy. Optional means there may or may not be an owner.
- GenCreate: When false, this class cannot be created in code with a Create() method. Check for required parameters in overrides in both the factory and the class overrides.
- Comment: Documentation for this class.

The Fields tab has these columns that come from the master model file.

- Flid The field id which includes the module and class numbers.
- Class: The class name.
- Field: The field name:
- Type: The type of field: bas for basic, own for owning, rel for reference relation.
- Card: Cardinality for owning and reference, atomic, col for collection, and seq for sequence.
- Sig: Signature of target.
  - Basic type: Binary, Boolean, Guid, Integer, MultiString, MultiUnicode, String, TextPropBinary, Time, Unicode
  - Owning and reference types: The class name of the object in this property. When a class is listed here, any subclass in the class hierarchy is valid for this property.
- Big: Unused since leaving SQL Server.
- Min: For Integer properties, this is the minimum value.
- Max: For Integer properties, this is the maximum value.
- Comment: Documentation for this field.

With this spreadsheet, you can easily find answers to common questions:

- What are all of the properties for a given class? In the Fields tab sort the class names and find the class(es) of interest.

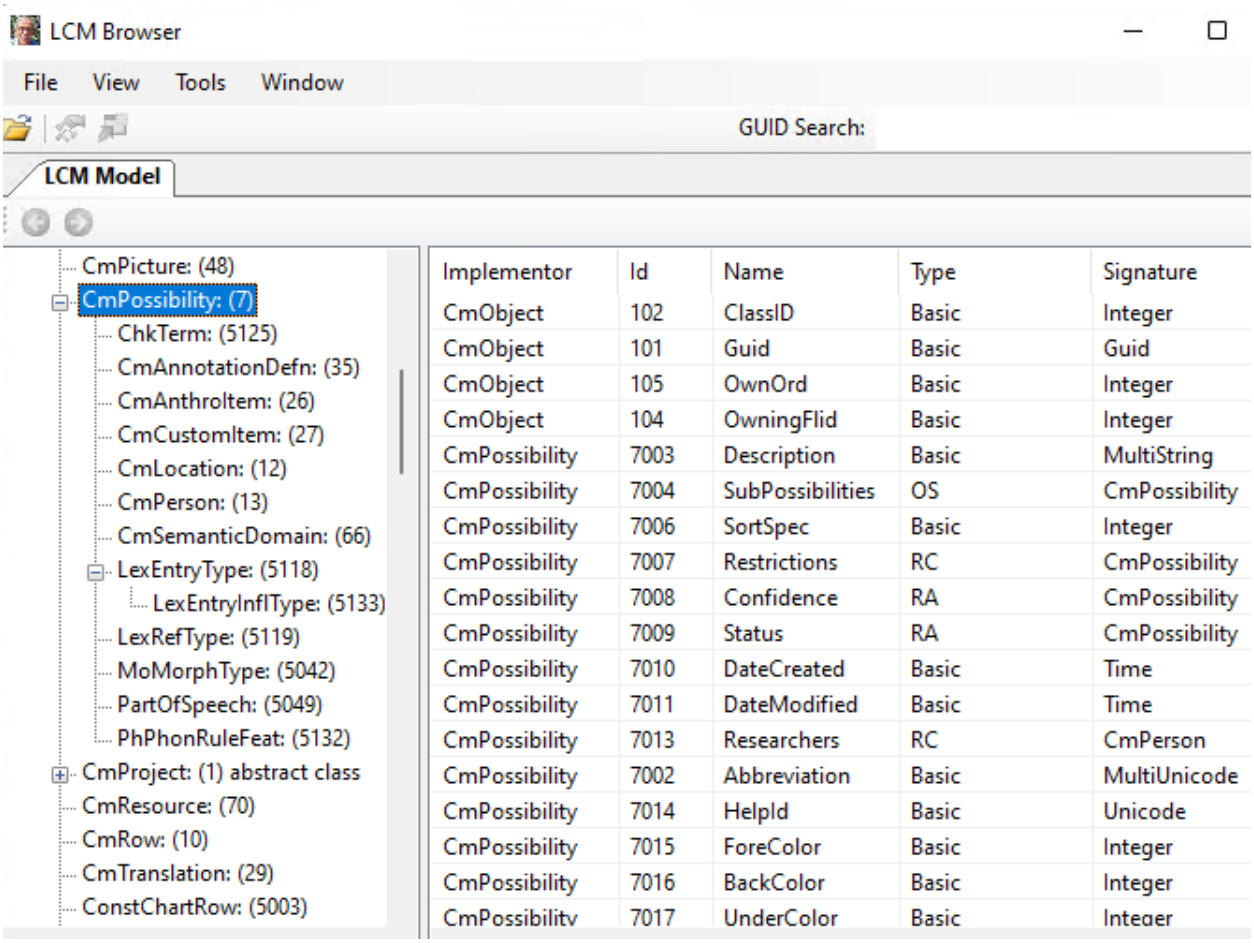
- What classes are subclasses of a given class? A possible answer can be found in the Classes tab by sorting on the Base. It's more complicated when there are multiple levels under a class.
- What are the class or field IDs? They are the first column of both tabs. If you have an ID and want to know what it's for, you can sort the ids and find it or search in the ID column to find it.
- What kind of strings are in given fields? This can be found in the Fields tab looking in the Sig column.

3.3 LCMBrowser

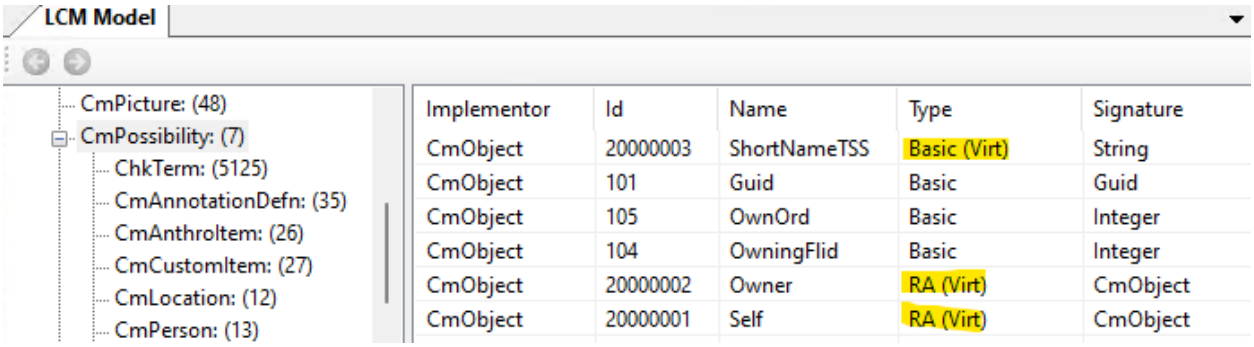
LCM is an abbreviation for SIL Language & Culture Model. LCMBrowser.exe is a program that comes with FieldWorks and can be started by typing into the Windows start menu or finding it in c:\Program Files\SIL\FieldWorks 9. The program has a tab for showing the current conceptual model. You can open a fwdata file and it lets you browse the data, and make limited modifications, such as deleting an object.

3.3.1 LCM model

When LCMBrowser opens, it has an LCM Model tab. This tab displays the class hierarchy of all classes in the FieldWorks model. The class number follows each class name in parentheses. In the left pane you can open classes with a + to show subclasses. The following display shows all of the subclasses of CmPossibility. The right pane shows the properties (fields) of the selected class, which is CmPossibility in this case. It shows that the first 4 properties are inherited from CmObject, and the remaining ones are defined on CmPossibility. The columns show the class that defines the property, the field ID (flid) of the property, the name of the property, the type, and the Signature. Types are O (owing) or R (reference) followed by A (atomic), C (collection), or S (sequence).



The View menu has a toggle option for “Display Virtual Properties” which is turned off in the above graphic, so you are seeing actual fields in the model that may contain data. When turned on, it also shows virtual properties. These are methods on the class that will return some result from FLEx code. They include (Virt) after the type. For example, there is an Owner property on every object that will return the object that owns the given object.



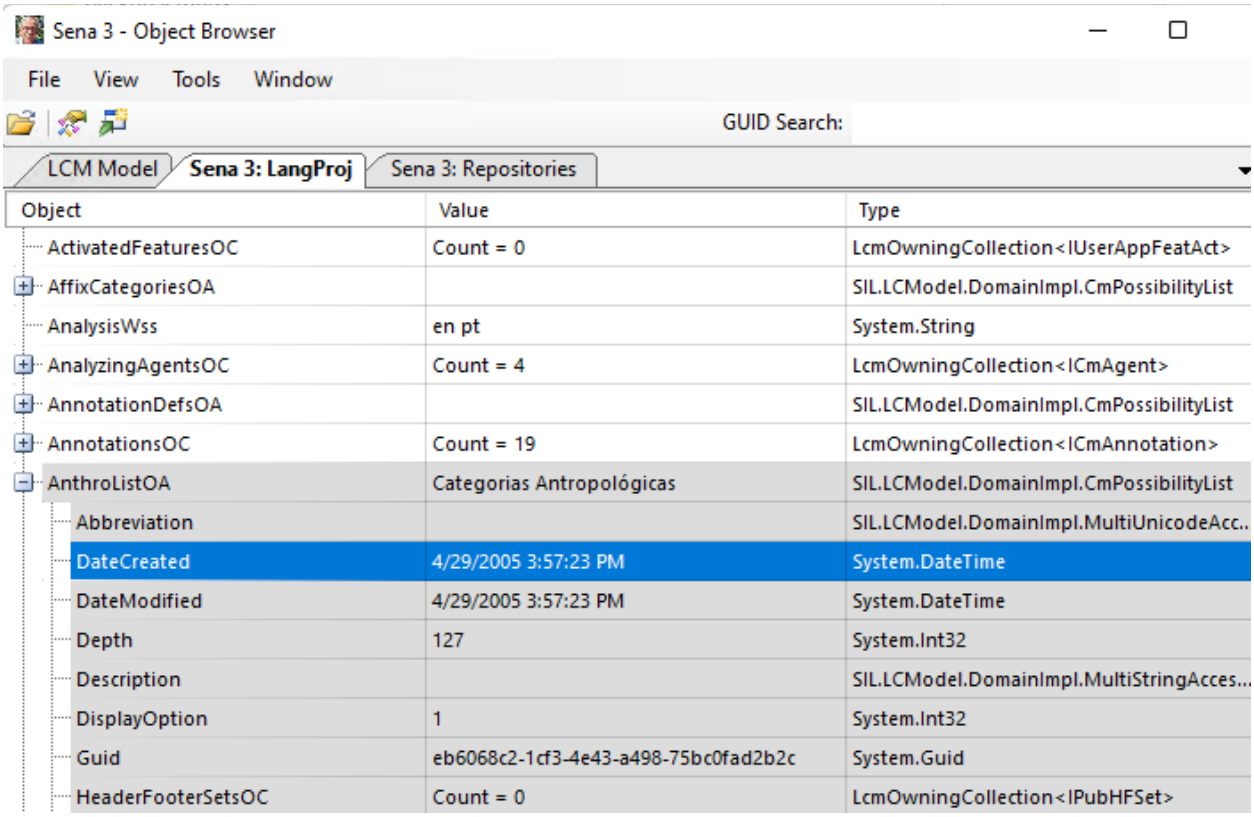
3.3.2 LCM data

To see the data in an actual project, choose File > Open Language Project, and navigate to an .fwdata file to open. Note, you cannot open a project that is open in FLEx, or one that has an .fwdata.lock file in the directory. The program will bring up an unhandled exception dialog saying you can open a project that is open. When a project is open in LCMBrowser, the lock file

is used to block FLEx from opening the project, and any changes you make to the data will be saved automatically just as in FLEx.

LCMBrowser has some problems with custom fields.

When a project is open, two additional tabs are available: <Project>: LangProj, and <Project>: Repositories.

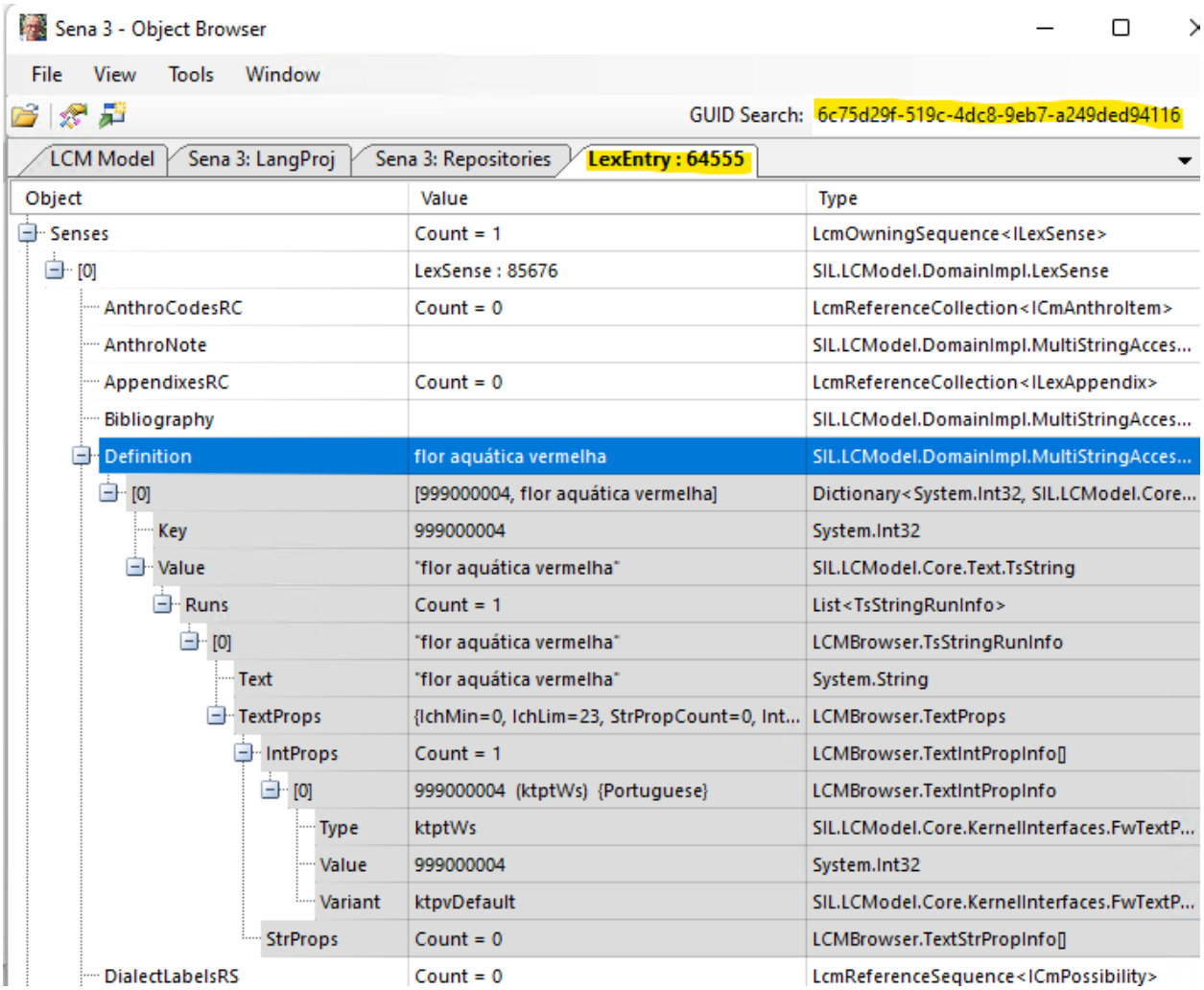


The LangProj tab shows the class ownership hierarchy starting at LangProj, + expansion boxes can be clicked to open further into the ownership hierarchy. Ownership and Reference class names have markers appended to indicate the type of property: O (Owning) or R (Reference) followed by A (Atomic), C (Collection), or S (Sequence). You can find LexEntry objects by opening LexDbOA and then open Entries to see the entries. Since the model does not have an Entries owning property, you have to have “Display Virtual Properties” checked in the View menu in order to see the Entries property. The middle column shows the Value, or a count objects in the property for owning and reference attributes. The right column shows the Type of object being viewed.

The Repositories tab lists all of the classes in the model with a count of the number of objects using this class. A + expansion box to the left can be opened to see a list of the objects in that class.

The only searching capability in LCMBrowser is a GUID search capability. If you enter a GUID in this search box it will open a new window on the object with this GUID. To get a GUID for lexical entries in FLEx, you can Ctrl+Right-click the entry in a dictionary or dictionary preview and choose Inspect Entry. This opens a dialog with the GUID at the top and then the xhtml view below that.

In the view of all objects, you can continue to drill down into the structure until you get to basic objects. Here's an example that gets down into a Definition MultiString



It is possible to delete objects in LCMBrowser, which will delete the object along with all owned objects and any references to those objects. There is no way to delete scripture objects in FLEx, but it can be done in LCMBrowser. This can be especially helpful if you are having trouble importing a book from Paratext. Usually deleting the book will then allow it to import correctly. To do this in the LangProj tab, open TranslatedScriptureOA, then open ScriptureBooksOS. This will list the ScrBook objects, if there are any. The book names will be shown in the Value column. Find the book you want to delete and right-click it, then choose "Delete the selected object".

3.4 Fieldworks source code

FieldWorks is an open source project, so all source code is available to anyone that wants it. You can open sources directly from github in a browser, or you can download the sources using Git (https://git-scm.com/). With the source code downloaded you can search all files for class names,

methods, etc. to try to understand how certain things are used in code. The program sources are in https://github.com/sillsdev/.

The following repositories are needed for FieldWorks code

- https://github.com/sillsdev/FieldWorks: The main FieldWorks code
- https://github.com/sillsdev/chorus: A library for using Send/Receive
- https://github.com/sillsdev/liblcm: A library with the basic FieldWorks model code. Most code that is relevant for FlexTools is in this repo.
- https://github.com/sillsdev/libpalaso: A library with various things FLEx uses.
- https://github.com/sillsdev/flexbridge: The source code for FLEx Bridge for doing S/R.

To get the source code using a Git Bash window, use this command:
```
git clone https://github.com/sillsdev/FieldWorks fw
```
where the appropriate repo is used from the above list, and the final argument is the target directory to create for the downloaded repo. The master branch is usually the appropriate branch to use in the repos.

FieldWorks can be built on Windows using a free Visual Studio Community Edition. For more information about building FieldWorks, see
https://github.com/sillsdev/FwDocumentation/wiki

Debugging can also be done without installing any program using dnSpy.

3.5 dnSpy debugger

If interested in exploring FLEx using a debugger, there is a free .NET debugger that can be used on any machine without installing any software. You just unzip a download into a directory and run the debugger from the directory.
To setup dnSpy
1. Download dnSpy-net-win64.zip from https://github.com/dnSpy/dnSpy/releases.
2. Unzip the file into a directory.
3. Run dnSpy.exe from the unzipped directory.
4. Drag C:\Program Files\SIL\FieldWorks 9\FieldWorks.exe into the left Assembly Explorer pane in dnSpy.

To run the debugger.
1. Shut down Fieldworks if it is running.
2. Right-click FieldWorks in the Assembly Explorer.
3. Choose “Go to Entry Point”.
4. Click the Start button at the top, or press F5.
5. A dialog is opened for command line arguments. Typically, you can just click OK.
6. FieldWorks opens and you can start working. At any time you can break execution in the debugger to investigate where it is, look through the stack, set breakpoints, etc.

4 Lexicon model

Lexical data is owned by LangProject in the LexDb property. This property holds a single LexDb object that holds information relevant to the lexicon. It owns these Possibility Lists that are referenced from objects in the lexical database.

LexDb Property Name	Name in Lists area
SenseTypes	Sense Types
References	Lexical Relations
PublicationTypes	Publications
MorphTypes	Morph Types
Languages	Languages
UsageTypes	Usages
ExtendedNoteTypes	Extended Note Types
VariantEntryTypes	Variant Types
DialectLabels	Dialect Labels
ComplexEntryTypes	Complex Form Types
DomainTypes	Academic Domains

Entries do not have owners, but can be accessed from LexDb via a virtual Entries method or the ILexEntry repository..

FLEx uses reference links between different objects, and when information is displayed, it pulls information from each object as it is needed for the display. FLEx also removes any links to objects that get deleted. As a result, the data displayed is always current. If a headword changes, or the homograph numbers change, or sense numbers change, all references to that entry will automatically match the new state of the data.

4.1 Entries

LexEntry is the class for lexical entries. This class is used for main entries that show as headwords in the dictionary, and also for entries that are variants or complex forms of the main entries. Entries keep track of date/time created and modified.

The LexEntry class includes the following types of strings.

Name	Signature
CitationForm	MultiUnicode
Bibliography	MultiString
Restrictions	MultiString
SummaryDefinition	MultiString
LiteralMeaning	MultiString
Comment (Note)	MultiString
ImportResidue	String

Every LexEntry must have a reference to a MoMorphType item in the Morpheme Types list. There are 19 possibilities in this list. This list is fixed, meaning you can't add or delete any items because FLEx code depends on having each entry assigned to one of these. There are actually two types; stem and affix types.

Stem types	Affix types
bound root	circumfix
bound stem	infix
clitic	infixing interfix
discontiguous phrase	prefix

enclitic	prefixing interfix
particle	simulfix
phrase	suffix
proclitic	suffixing interfix
root	suprafix
stem	

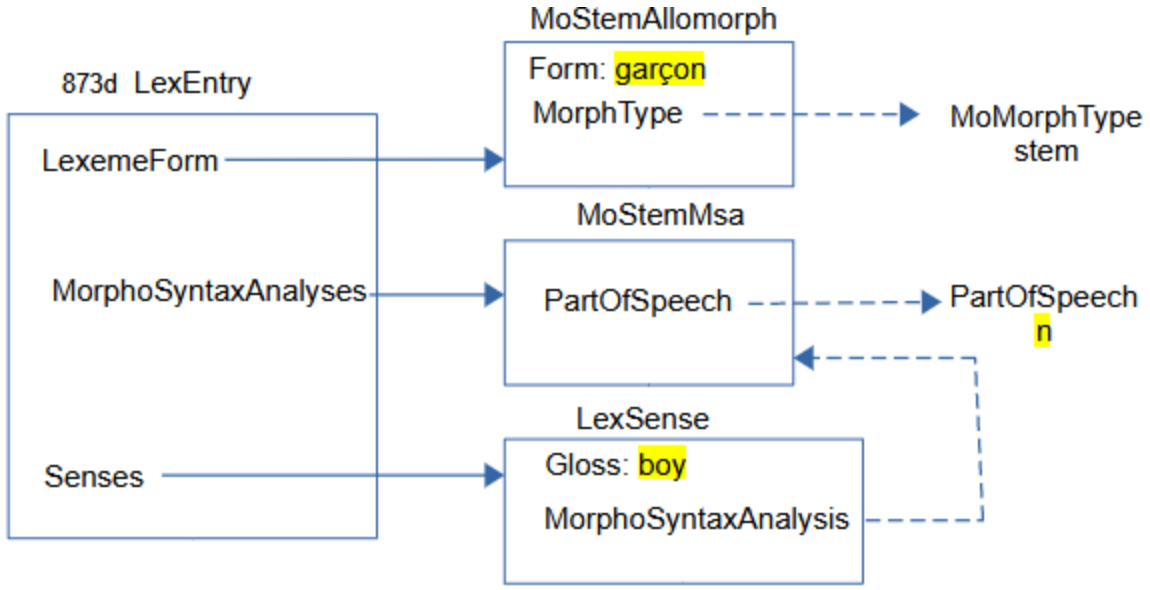
Every LexEntry must have a MoForm object in the LexemeForm property. This object actually holds the reference to a morph type in the MorphType reference property. But MoForm is actually abstract as well as MoAffixForm, so there are only 3 choices to use. For stem type morphemes, MoStemAllomorph is used. For affix types, MoAffixAllomorph is normally used, but MoAffixProcess is another option.

A second required part of every LexEntry, if it owns a sense, is owning at least one MoMorphSynAnalysis object in the MorphoSyntaxAnalyses collection property. MoMorphSynAnalysis is also an abstract class, so one of its subclasses must be used. For stem type morphemes, MoStemMsa is used. For affix types, MoDerivAffMsa, MoInfAffMsa, or MoUnclassifiedAffixMsa are used. These classes deal with grammatical information. Senses need to reference these objects to show grammatical information. If senses for the entry have different grammatical information, then additional MoMorphSynAnalysis objects will be added to the entry so they can be referenced by the senses. These items are removed when senses that use them are deleted.

In summary, both the MoForm and the MoMorphSynAnalysis objects depend on whether the entry is a stem type or an affix type. The Choose Morpheme Type dialog that comes up when you click the Morph Type button normally only shows morph types that are in the same group. When that happens, the change is minimal. If they click the “Show all types” checkbox, all morph types are shown, and if they choose one from the other set, the program will replace the MoForm and MoMorphSynAnalysis objects to the opposite type with new guids. If these two ever get out of sync, it causes problems.

Here’s a simple entry to demonstrate a stem-based entry which appears like this in FLEx.

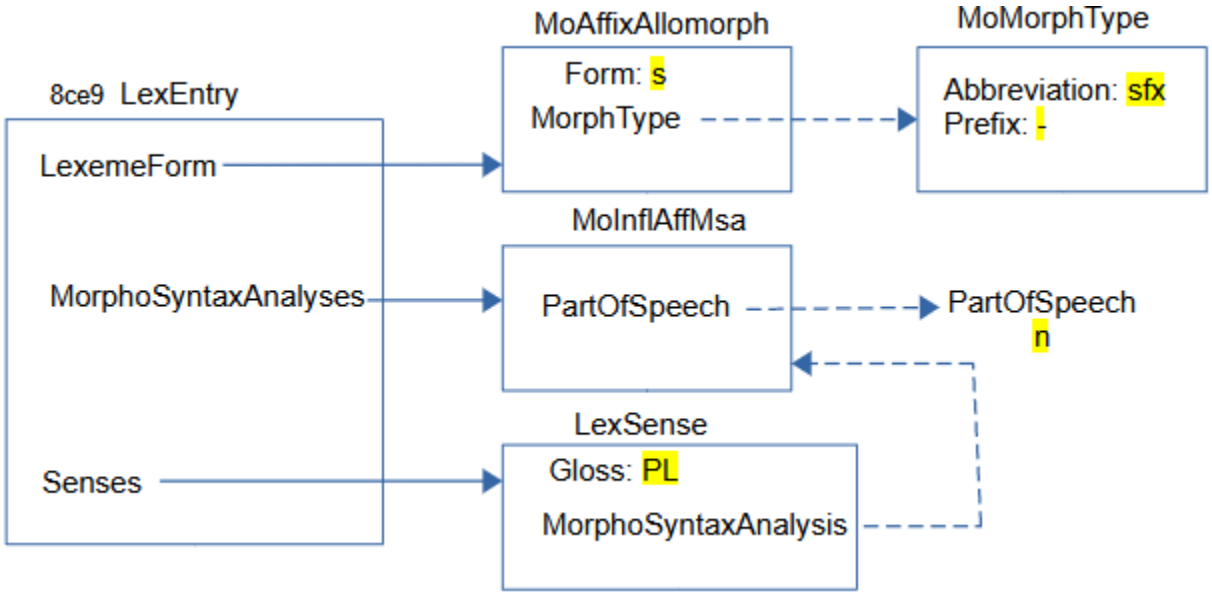
garçon *n* boy



The LexEntry (with guid starting with 873d in my sample project) owns a MoStemAllomorph that has a Form “garçon” and a MorphType reference to a MoMorphType with Name: “stem”. LexEntry owns a MoStemMsa that references a PartOfSpeech with Abbreviation: “n”. LexEntry also owns a LexSense with a Gloss: “boy”, and a MorphoSyntaxAnalysis reference to the MoStemMsa.

Here’s a simple entry to demonstrate an affix-based entry which appears like this in FLEx.

-s n sfx PL



The LexEntry (with guid starting with 8ce9) owns a MoAffixAllomorph that has a Form “s” and a MorphType reference to a MoMorphType with Abbreviation: “sfx” and Prefix: “-“. The Headword display inserts the Prefix string before the MoForm Form. LexEntry owns a MoInflAffMsa that references a PartOfSpeech with Abbreviation: “n”. LexEntry also owns a

LexSense with a Gloss: “PL”, and a MorphoSyntaxAnalysis reference to the MoInflAffMsa. The display for the inflectional affix displays the category followed by the morph type abbreviation.

All of the strings in the above examples are MultiUnicode, so they can have any number of vernacular or analysis writing systems. Most strings in LexEntry and LexSense are either MultiUnicode or MultiString.

4.1.1 Headwords

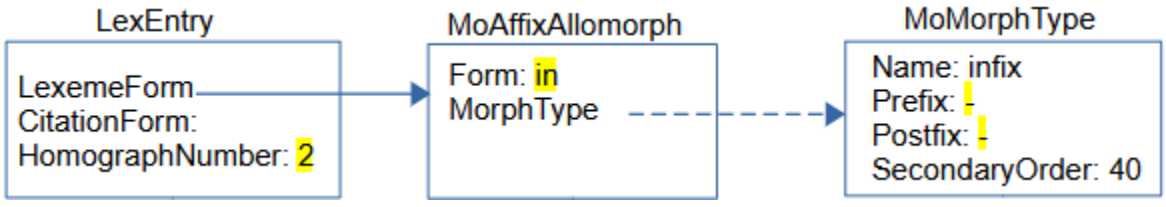
A headword is a word or phrase that is placed at the beginning of the entry to help the user quickly find the word they are looking for. Entries are also sorted by headwords. A headword in FLEx is a virtual property that joins several properties from several classes. Properties that are relevant to a headword are:

- LexemeForm owning property on LexEntry
- CitationForm MultiUnicode on LexEntry
- HomographNumber integer on LexEntry
- Form MultiUnicode on MoForm owned in the LexemeForm property of LexEntry
- Prefix Unicode on MoMorphType referenced from MoForm via MorphType (shown as ‘Leading Token’ in the Lists area)
- Postfix Unicode on MoMorphType referenced from MoForm via MorphType (shown as ‘Trailing Token’ in the Lists area)
- SecondaryOrder integer on MoMorphType referenced from MoForm via MorphType

During interlinearization, the lexeme form is used to identify the morphemes within a word (e.g., ‘cat’ and ‘-s’ for ‘cats’). Some languages prefer to use an inflected form of the stem instead of the raw stem form for the dictionary headword. For example, the French ‘mang’ lexeme form would use the infinitive form, ‘manger’, for the headword. In FLEx, the citation form is used for this, but it should only be used if the headword needs something different from the lexeme form.

Some languages prefer to only use citation forms without a lexeme form in their FLEx project. It’s not possible to create an entry in the FLEx UI without a lexeme form. This is because the MoForm in the LexemeForm field is required for other purposes. After the entry is created, you can remove the form in the lexeme form field and it will work OK. For SFM import you can import identical lexeme form and citation form fields, and then clear out all of the lexeme form fields after import.

Here is an example showing how infix **-in-**2 would be stored:



If there is a CitationForm, the headword will use that form. Otherwise it will use the LexemeForm (actually the Form from the MoForm owned in the LexemeForm). If there is no CitationForm, then the Prefix and Postfix markers are added to the LexemeForm. If there is a HomographNumber, that is appended at the end.

When sorting headwords, FLEx sorts by the form of the headword without affix markers followed by the SecondaryOrder from the MoMorphType, followed by the HomographNumber. SecondaryOrder allows you to order affixes by type, and uses a different homograph number.

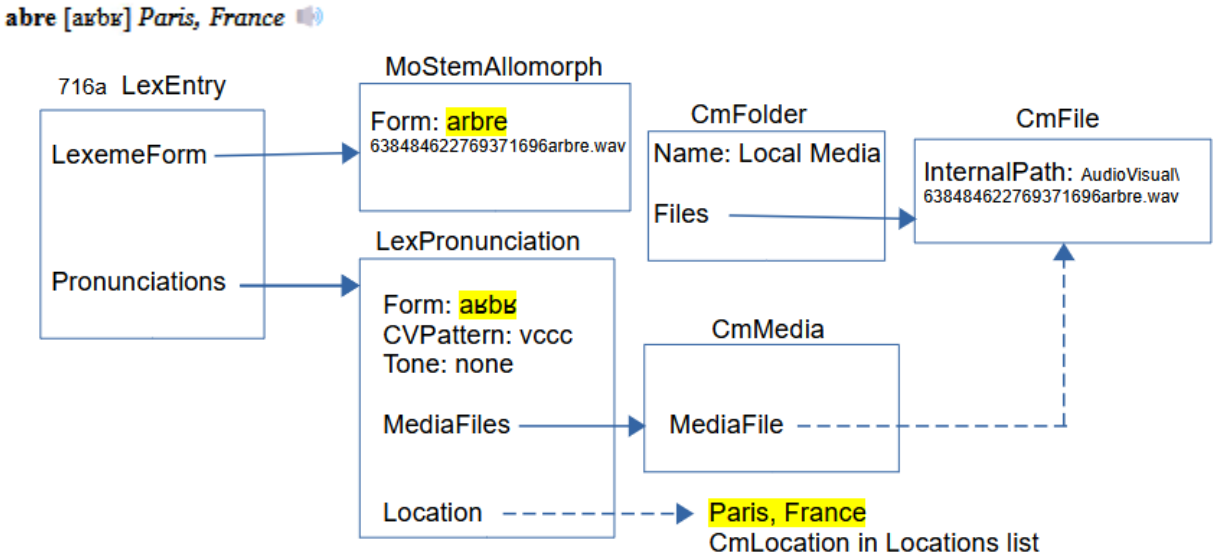
HomographNumbers are maintained automatically by FLEx. Homograph numbers will be applied to headwords that have an identical form and SecondaryOrder. You can force renumbering homographs in FLEx using Tools > Utilities > Reassign Homographs. Sometimes you have to use Refresh to get homograph numbers to display correctly after a change. Homograph numbers are automatically adjusted during dictionary views where some entries are excluded from the dictionary.

4.1.2 Pronunciations

There are basically two ways to store pronunciations in FLEx.

- If there is only one pronunciation per entry, you can use the lexeme form or citation form property using the 'audio' writing system, where the code for the writing system is fr-Zxxx-x-audio for a French recording, and the content is a file name that must be in the project LinkedFiles\AudioVisual directory. FLEx has a button that will record a sound in these fields, or you can attach an existing recording.
- If you have multiple recordings per entry, and/or want to include other information along with the pronunciation, then you can add one or more LexPronunciation objects.

Here's an example of an entry with a pronunciation including a location



This example shows both possibilities for a pronunciation. Normally you would not have both options.

- The first option is using an audio writing system in the LexemeForm field. This option does not include the 4 classes below and to the right of MoStemAllomorph. The structure is very simple in this case. The audio file name is in the fr-Zxxx-x-audio writing system of the MoStemAllomorph Form field. This was recorded using the record button. The record button in FLEx will only write .wav files, but can play .mp3 as well.

- You can add one or more LexPronunciation objects to the Pronunciations field of LexEntry. The LexPronunciation Form MultiUnicode field is storing the form in the fr-fonipa writing system. You can also include a CVPattern in a String field, and a Tone in a String field. The Location field lets you reference one CmLocation item in the Locations list. The MediaFiles property is an owning sequence which holds CmMedia objects. CmMedia has a Label MultiString field which isn't supported in the FLEx interface. It also has a MediaFile reference to one CmFile object. The CmFile object has an InternalPath Unicode property that holds the name of the audio file. In this case, it needs to include the path from the LinkedFiles folder inside the project folder. The files can be in other locations in this case. The CmFile object is owned in a CmFolder named "Local Media" that is owned by the Media owning collection property of LangProject.

4.1.3 Etymology

LexEntry has an owning sequence Etymology property that can hold multiple LexEtymology objects. These are the fields that are available. All except Source Language are MultiString fields that default to first analysis writing system, except Source Form defaults to vernacular and analysis. Source Language allows multiple references to the Languages list.

Data Entry Field	**Property Name**
Preceding Annotation	PrecComment
Source Language	Language
Source Language Notes	LanguageNotes
Source Form	Form
Gloss	Gloss
Following Comment	Comment
Bibliographic Source	Bibliography
Etymology Note	Note

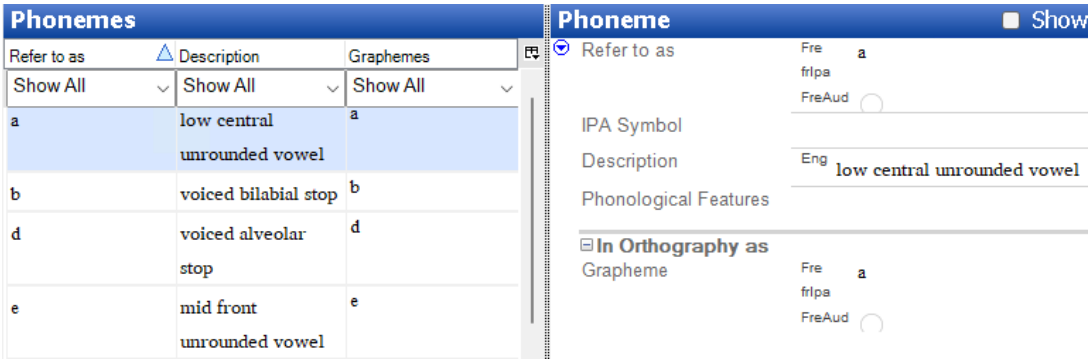
When you start typing to open a LexEtymology object, you are typing in the first analysis writing system in the Source Form field.

4.1.4 Allomorphs

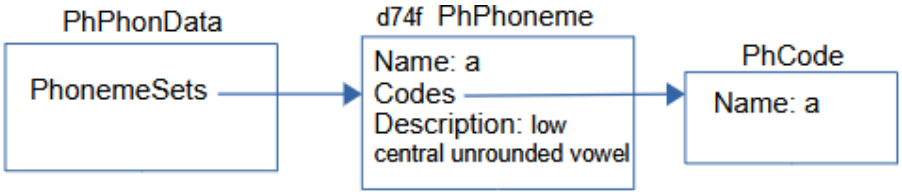
Allomorphs are used to enter and constrain alternate forms of a lexeme form, especially for parsing situations. Allomorphs are stored as MoForm (MoStemAllormorph or MoAffixAllomorph) objects in the AlternateForms owning sequence property of LexEntry.

For example, consider an allomorph 'z' for plural entry with lexeme form -s. In English, when the plural follows a vowel or voiced consonant, the plural is also voiced so could be represented as a -z allomorph in this context.

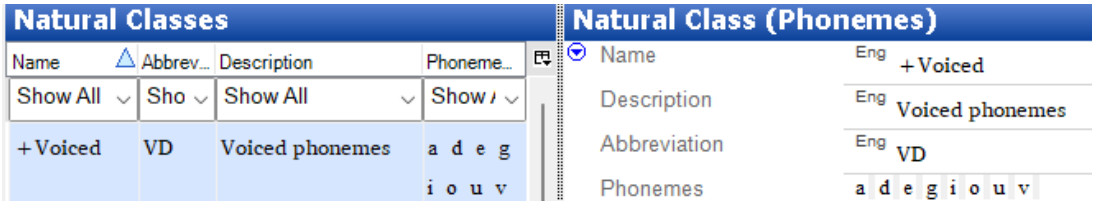
This example gets into the Grammar area of FLEx a well as the lexicon. We start with a list of phonemes in the language. In the grammar you can list all of the phonemes in the language in the Phonemes tool.



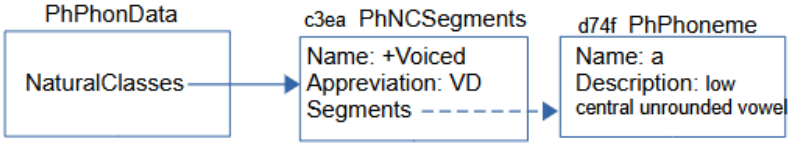
These phonemes are represented by PhPhoneme objects owned in the PhonemeSets property of PhPhonData, which is owned in the PhonologicalData property of LangProject. The PhPhoneme gives the linguistic description of the phoneme. The letter(s) in the orthography that represent this phoneme are stored in PhCode objects with a Name identifying the phoneme. There are other phonemes in addition to what is shown here.



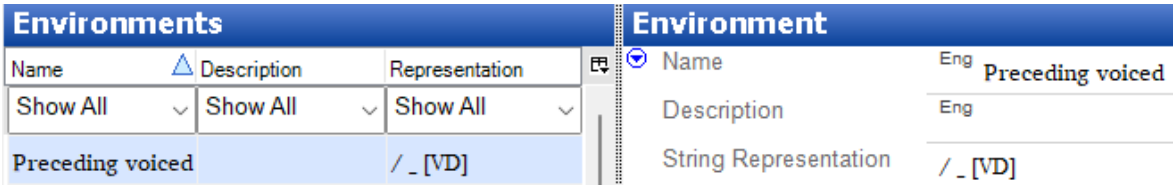
In order to provide a phone environment that applies to this allomorph, we need to define a natural class for voiced phonemes in the Natural Classes tool in the Grammar area.



This is stored in a PhNCSegments object with a name (+Voiced) and abbreviation (VD). The Segments reference property references all of the PhPhoneme objects that are voiced.

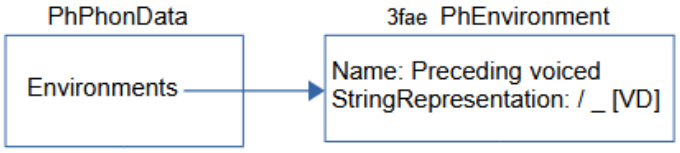


Next is defining an environment for the allomorph in the Environments tool in the Grammar area. This rule states that the environment for the allomorph is that it must follow a voiced phoneme.

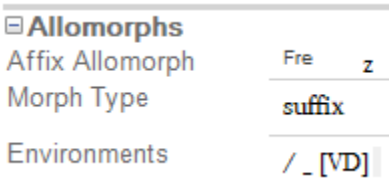


This is stored in a PhEnvironment class with a name (Preceding voiced), and a StringRepresentation that gives a linguistic representation of the environment (/ _ [VD]) This

string is parsed by FLEx to make sure it is valid. In this case, the VD matches the abbreviation in the PhNCSegments object.

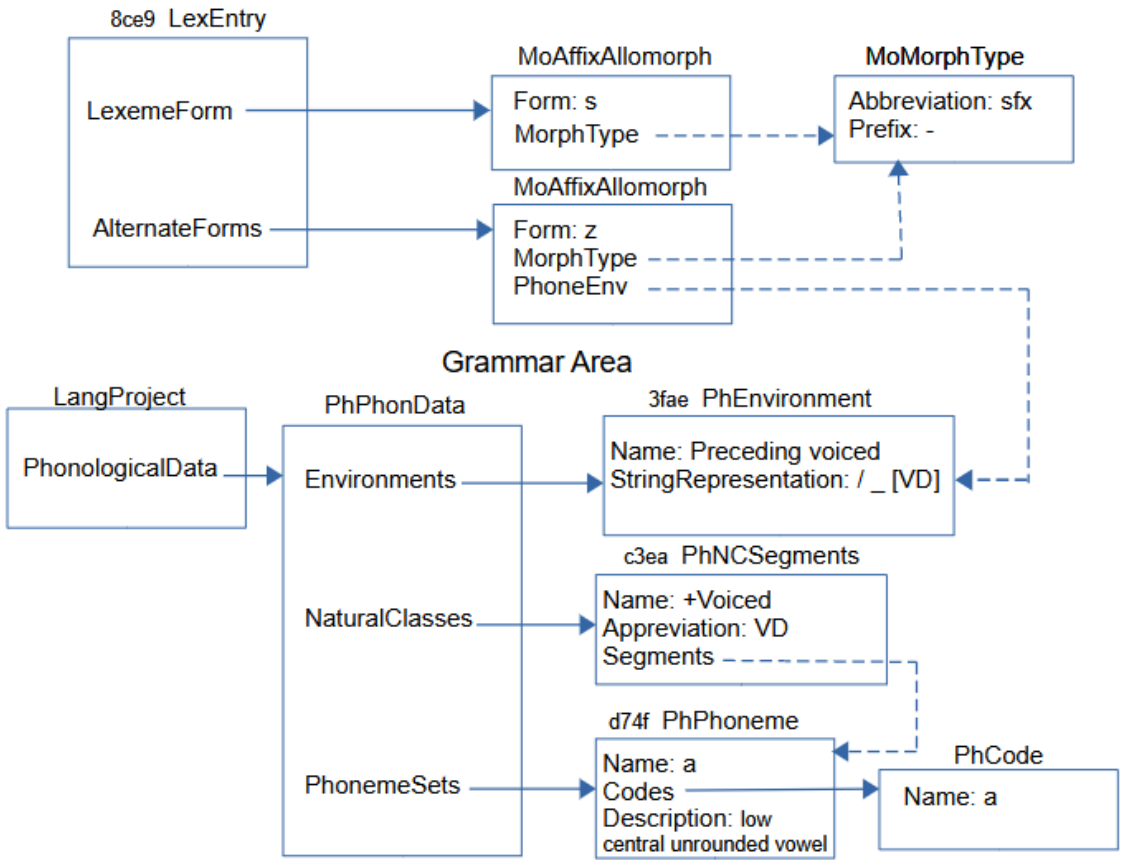


Now back in the lexicon, in the Allomorphs section of the -s entry, we then create a suffix allomorph with z in the form and the Environments set to the environment.



In the following diagram, the allomorph is represented by a MoForm (MoAffixAllomorph in this case) owned in the AlternateForms property of LexEntry, with the form (z), MorphType referencing the suffix MoMorphType, and PhoneEnv referencing the “Preceding voiced” PhEnvironment. There are other grammatical properties on MoAffixAllomorph that are not shown in this example to further specify where it can be used. This includes Inflection Classes, and Required Features in the FLEx UI.

This is combined diagram in FLEx.



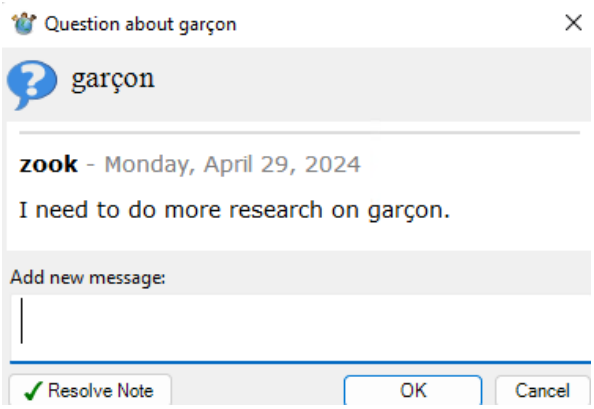
4.1.5 Grammatical information.

In its simplest form, Grammatical Information just references a PartOfSpeech object in the Categories list. For parsing situations, it can include other information that restricts where it can be used. This information is entered in the Grammatical Info Details section at the bottom of an entry. This includes inflection classes and features, exception features, etc. Most of these features are discussed briefly in section 5.

4.1.6 Messages

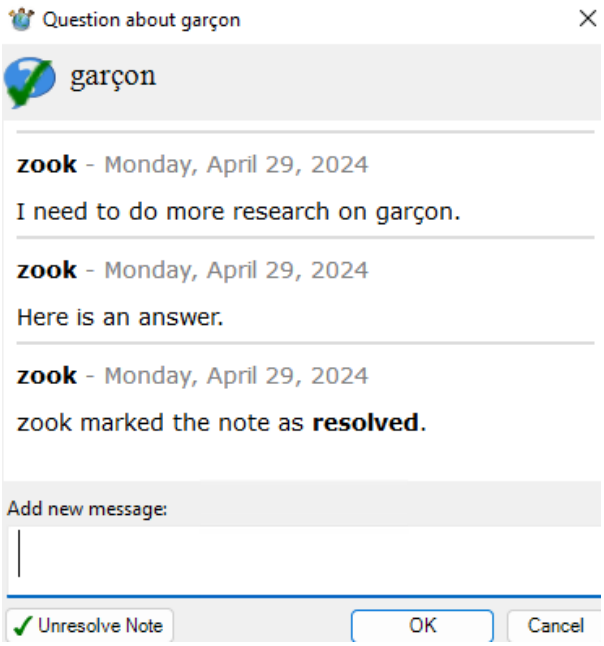
Although not part of the FLEx conceptual model, FLEx does have a messaging capability on LexEntry. It's primarily designed for users doing Send/Receive to have a place where they can ask questions of their colleagues, get feedback, and then resolve the message when done. It uses the FLEx Bridge Send/Receive capability, but it can be used without actually doing Send/Receive, and it gets saved in a FLEx backup. This is how it looks when there is a message on an entry.

When you click the blue button, it opens the "Question about" dialog showing the status of the message(s).



When resolved, the icon changes to

Clicking that button gives history.



The data for questions is stored in Lexicon.fwstub.ChorusNotes in the project folder. The information is stored in annotation elements. Responses and closing information is included as additional message elements.

```
<annotation
  class="question"

ref="silfw://localhost/link?app=flex&amp;database=current&amp;server=&amp;tool=default&amp;guid
=873de097-9b80-487e-a866-0b99831b4b7e&amp;tag=&amp;id=873de097-9b80-487e-a866-
0b99831b4b7e&amp;label=garçon"
  guid="73abfec0-5214-4ace-964d-a6eb990af1b1">
  <message
    author="zook"
    status=""
    date="2024-04-29T09:33:34-05:00"
    guid="8303724a-01b4-4f66-ab4a-0275d9020000">I need to do more research on
garçon.</message>
  <message
    author="zook"
    status=""
    date="2024-04-29T09:47:13-05:00"
    guid="261d5ba3-c665-4870-b604-758ae640bf55">Here is an answer.</message>
  <message
    author="zook"
    status="closed"
    date="2024-04-29T09:47:22-05:00"
    guid="9df8eacc-d0be-46c3-813c-555ab3e06ed6"></message>
</annotation>
```

The message guid attributes are IDs for the actual message. The ref link at the top allow a user to click a link to jump to the entry. The guid in this reference is the guid of the LexEntry.

There is no capability built into FLEx for removing information from this ChorusNotes file. Data can be removed manually in the file, or the file can be deleted at any time to remove all Messages.

4.2 Complex forms and variants

Variants and complex forms (typically subentries) are stored as separate entries in FLEx that are linked to one or more main entries. There are numerous options for displaying complex forms and variants.

4.2.1 Complex forms

This is the display of a main entry and a complex form as a subentry and a minor entry in Root-based view. Showing minor entries is optional. The minor entry can show more information, but typically it's just a small entry that tells the user where they can find the full entry.

lion *n* a large tawny-colored cat that lives in prides.
lionne der. *n* A female lion.

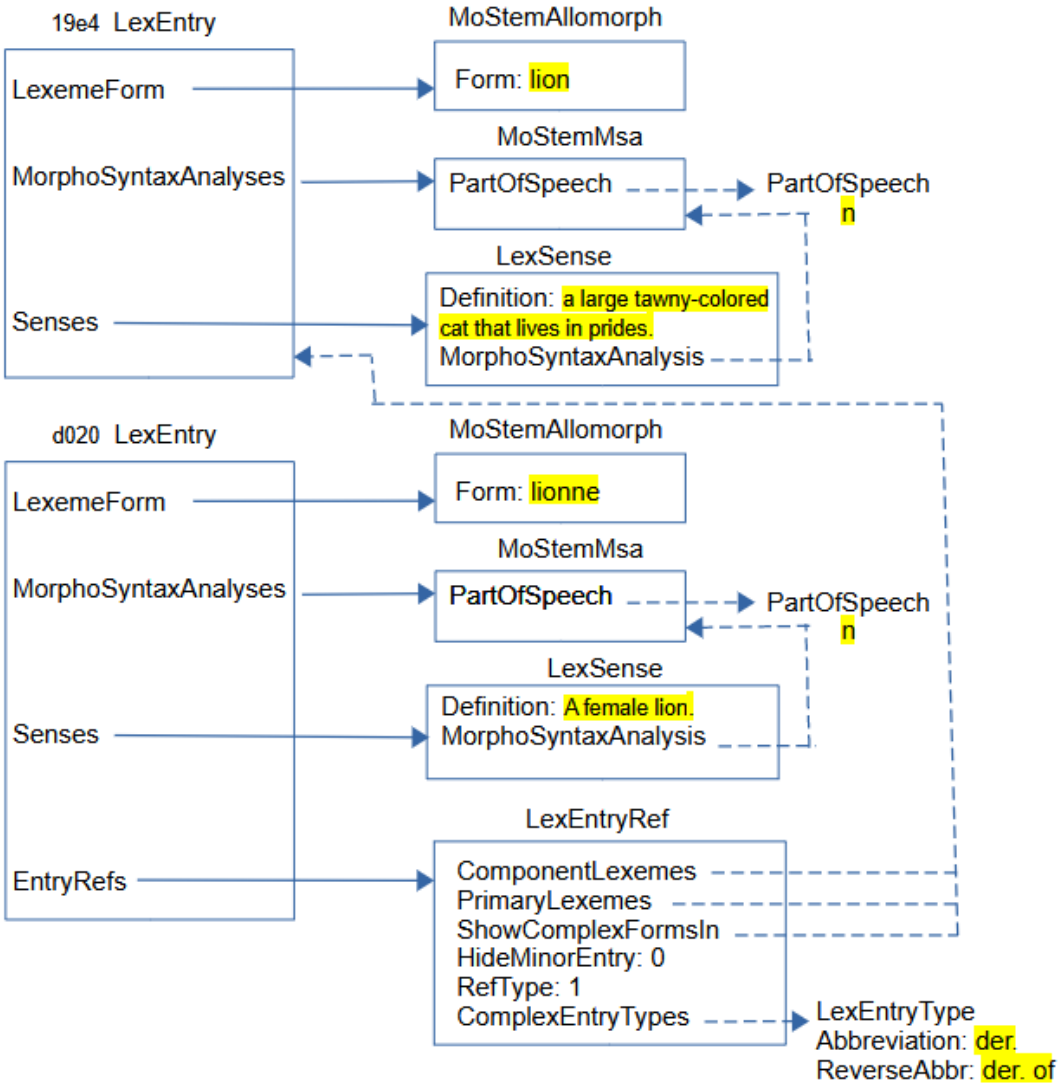
lionne (der. of **lion**)

By switching the display to Lexeme-based view, we get this display, where each entry is displayed as an entry in the lexicon with references to the complex form and the main entry.

lion *n* a large tawny-colored cat that lives in prides. der. **lionne**

lionne (der. of **lion**) *n* A female lion.

Here is how the data is stored in FLEx for a complex form.



The main entry and complex form entry contain typical information in entries and senses. The part that makes the bottom entry a complex form, or subentry, is adding a LexEntryRef in the EntryRefs owning property. In the LexEntryRef:

- RefType is set to 1 to indicate this is a complex form.
- Hide Minor Entry is set to 0 so that the minor entry will show. In the UI, this is controlled by the Show Minor Entry field at the bottom of the complex form entry.
- ComplexEntryTypes references one or more LexEntryType items in the Complex Form Types list. In the UI, this is set in the Complex Forms section in the Complex Form Type field. The Abbreviation in this item is used when displaying the complex form, and the ReverseAbbr is used when displaying a minor entry.
- ComponentLexemes references one or more main entries for this complex form. In the UI, this occurs in the Components field in the Complex Forms section.
- PrimaryLexemes references one or more of the entries in ComponentLexemes. For each PrimaryLexeme, a subentry will be generated for this complex form entry in Root-based views. This property is changed in the UI at the bottom of the complex form entry by the Show Subentry Under field.

- ShowComplexFormsIn references one or more of the entries in ComponentLexemes. For each ShowComplexFormsIn, a complex form will be generated for this complex form entry in Lexeme-based views. This property is changed in the UI at the bottom of the complex form entry by the Referenced Complex Forms field.

Complex forms typically refer to a LexEntry, but they can also refer to a LexSense if the complex form only applies to one sense. Note that there is no direct connection in the model from the main entry to complex form entries, although they can be accessed through virtual properties.

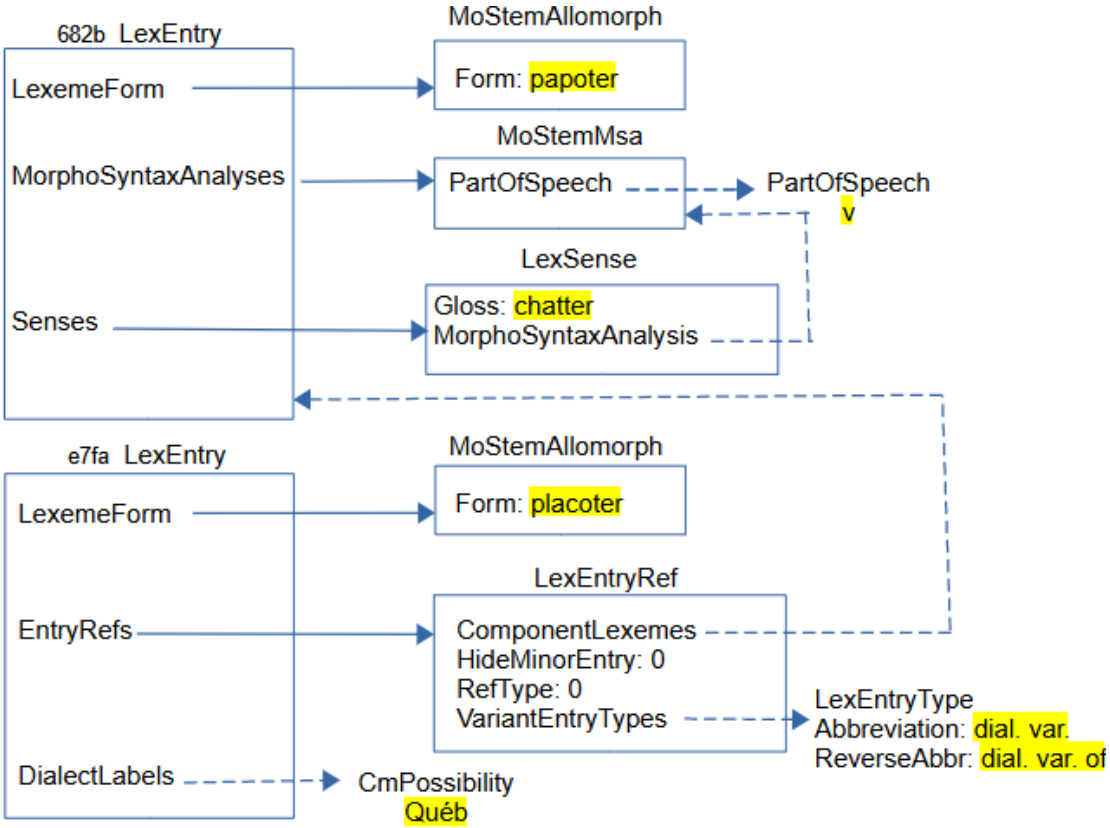
4.2.2 Variants

This is the display of a main entry with a variant, and a minor entry.

papoter (dial. var. **placoter** [Québ]) *v* chatter

placoter [Québ] dial. var. of **papoter**

Here is how the data is stored in FLEx for a variant.



The structure for a variant is similar to complex forms, but needs less information. A simple variant entry does not have a sense. In interlinear text it shows the sense information from the main entry. This means MorphoSyntaxAnalyses are not needed on the entry.

DialectLabels are available on LexEntry and LexSense and allow users to specify one or more dialects that use this entry or sense. The possible dialects are specified in the Dialect Labels list. Since this is a dialect variant, it's appropriate to include the dialect on the variant entry.

The part that makes the bottom entry a variant entry, is adding a LexEntryRef in the EntryRefs owning property. In the LexEntryRef

- RefType is set to 0 to indicate this is a variant.
- Hide Minor Entry is set to 0 so that the minor entry will show. In the UI, this is controlled by the Show Minor Entry field at the bottom of the variant entry.
- VariantEntryTypes references one or more LexEntryType items in the Variant Types list. In the UI, this is set in the Variants section in the Variant Type field. The Abbreviation in this item is used when displaying the variant form, and the ReverseAbbr is used when displaying a minor entry.
- ComponentLexemes references one or more main entries for this variant. In the UI, this occurs in the "Variant of "field in the Variant section.
- PrimaryLexemes and ShowComplexFormsIn fields are not used for variants.

Variants typically refer to a LexEntry, but they can also refer to a LexSense if the variant only applies to one sense. Note that there is no direct connection in the model from the main entry to variant entries, although they can be accessed through virtual properties. Although a variant is typically present without a sense, these entries can have senses and most other information of full entries, if needed.

4.3 Senses

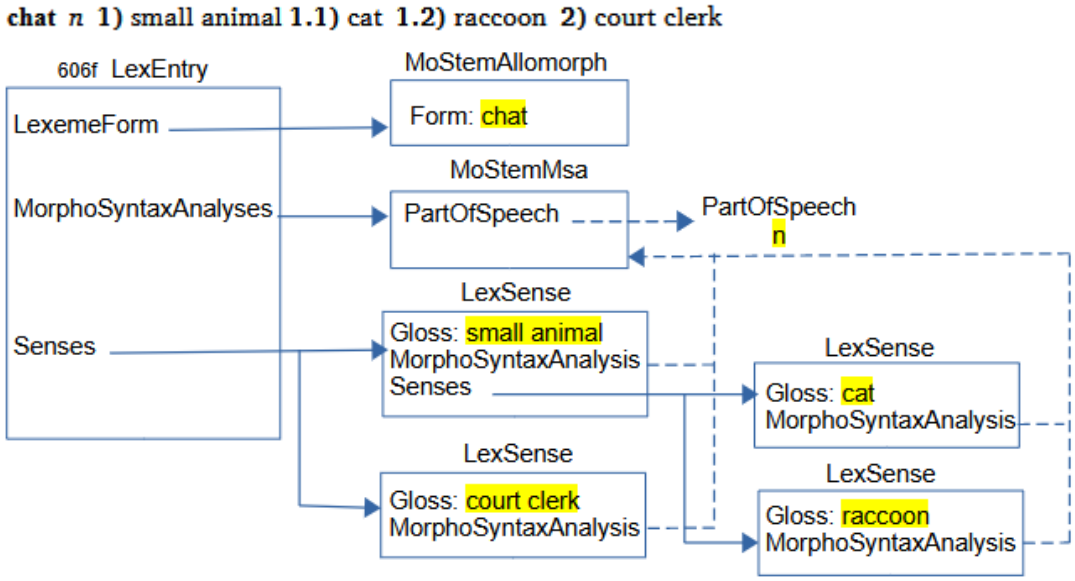
Sense information is stored in LexSense objects which are owned in the Senses property of LexEntry. Senses can also be nested. So LexSense can also be owned in the Senses property of LexSense. Sense numbers are provided virtually by FLEx, so there is no need to store numbers with senses. Dictionary configuration allows you to alter the numbering system in various ways. Configuration also allows you to start each sense in a new paragraph, if desired, and allows indents, and other paragraph properties when they are separate paragraphs.

The LexSense class includes the following types of strings:

Name	**Signature**
Definition	MultiString
Gloss	MultiUnicode
ScientificName	String
AnthroNote	MultiString
Bibliography	MultiString
DiscourseNote	MultiString
EncyclopedicInfo	MultiString
GeneralNote	MultiString
GrammarNote	MultiString
PhonologyNote	MultiString
Restrictions	MultiString
SemanticsNote	MultiString
SocioLinguisticsNote	MultiString
Source	String
ImportResidue	String
Exemplar	MultiString
UsageNote	MultiString

Styles for each field and Before, Between, and After text is controlled via configuration. Each sense has a MorphoSyntaxAnalysis that references one of the MorphoSyntaxAnalyses owned by its LexEntry. Senses can also be configured to show the grammatical category before a group of senses that have the same category.

Here's an example that shows main and subsenses of senses using the configuration option, "If all senses share the grammatical information, show it first".



All 4 senses share the same part of speech. The two senses in the middle are main senses and the two on the right are subsenses of the first sense.

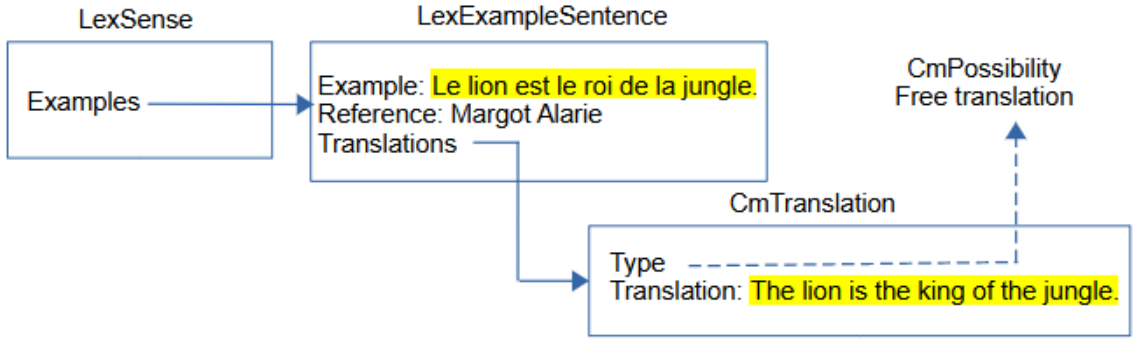
4.3.1 Example sentences

Example sentences and translations can have multiple writing systems with embedding as needed. You can also have multiple translations with different types. Default types are Free translation, Literal translation, and Back translation, but you are free to add other types if needed.

Here's a simple example.

Le lion est le roi de la jungle. **The lion is the king of the jungle.**

This is how it is represented in FLEx:



Multiple examples are owned in an Examples owning sequence property. The MultiString Example and optional Reference String are stored in a LexExampleSentence object. One or more

translations are owned in a Translations sequence property. Each translation is stored in a CmTranslation object with the MultiString Translation and a Type that references a CmPossibility in the Translation Types list.

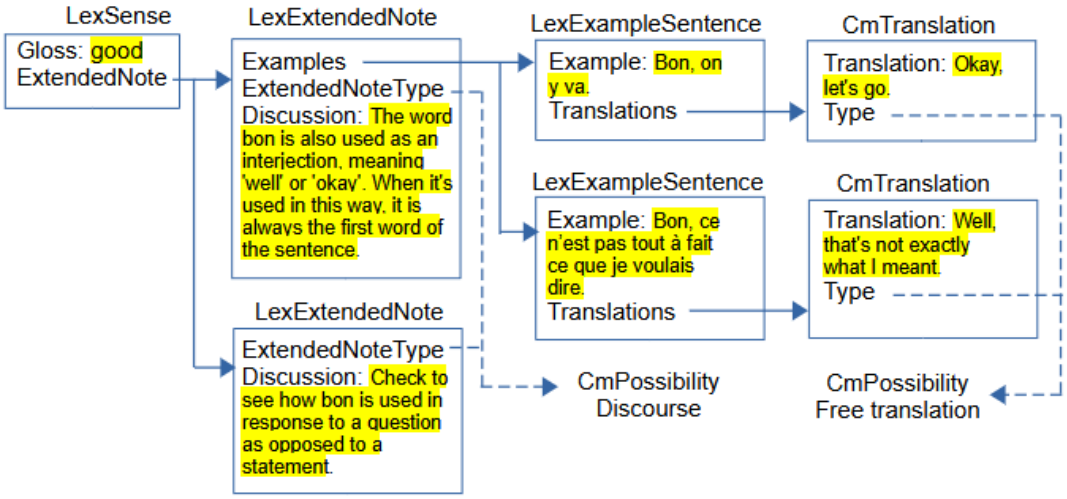
4.3.2 Extended notes

There are times when note fields need to include one or more example sentences to illustrate the discussion. In order to do this, FLEx offers Extended Notes. This works by including one or more extended note objects. Each extended note has a type that refers to an item in the Extended Note Types list, a MultiString Discussion field, and one or more examples. Consecutive extended notes of the same type will be displayed together as one note with configured Before/After text and style. When more than one type is used, the Extended Note node in Dictionary Configuration needs to be duplicated to provide different formatting, if you want to have different configuration for the additional types.

Here's an example with two example sentences illustrating the first part of the note.

bon good [The word *bon* is also used as an interjection, meaning 'well' or 'okay'. When it's used in this way, it is always the first word of the sentence. *Bon, on y va.* Okay, let's go. *Bon, ce n'est pas tout à fait ce que je voulais dire.* Well, that's not exactly what I meant. Check to see how *bon* is used in response to a question as opposed to a statement.]

This is how it is represented in FLEx:



LexExtendedNote objects are owned in the ExtendedNote owning sequence property of LexSense. Both of these notes reference the Discourse CmPossibility in the Extended Note Types list through the ExtendedNoteType property, so they represent one discourse extended note. The text before the examples is in the Discussion field of the first LexExtendedNote, and the text after the examples is in the Discussion field in the second LexExtendedNote. The first LexExtendedNote owns two LexExampleSentence objects through the Examples owning sequence property. These examples use identical classes for normal examples on senses described in section 4.3.1.

4.3.3 Lexical relations

In FLEx, lexical relations (between senses) and cross references (between entries) are implemented using LexRelation objects that represent an instance of a certain kind of relation set, and references entries or senses for this particular relation. FLEx uses virtual properties on entry and sense to find any LexRelation objects that are referencing the entry or sense, and displays appropriate information showing the relevant lexical relations. The specifications for the LexRelation are in LexRefType items in the Lexical Relations list.

There are 6 basic relation set types that are available

- Collection – links to a set of unordered objects (e.g., synonyms). Every object in the list will show all of the other objects in the list.
- Pair – links two objects (e.g., antonyms). Each object will show the other object with the same relation label.
- Pair (different names) - links two objects (e.g., classified nouns). Each object will show the other object, but with different labels for each end.
- Sequence – links an ordered list of objects (e.g., days of week). Each object will show the entire ordered set.
- Tree – links one object to a set of unordered objects (e.g., part/whole). The first object shows all other objects with one label. The other objects show the root object with a different label.
- Unidirectional – links one object with a sequence of objects. The root object shows all other objects. The other objects do not show anything.

The Lexical Relations list owns LexRefType objects, which are subitems of CmPossibility. LexRefType has six significant properties.

- Name: The name (or forward name) for the relation of the first linked object.
- Abbreviation: The abbreviation (or abbreviation) for the relation of the first linked object.
- ReverseName: The name for the relation when showing non-first linked objects.
- ReverseAbbreviation: The abbreviation for the relation when showing non-first linked objects.
- Members: Owns LexReference instances for lexical relation sets of this type.
- MappingType: An enum specifying the type of set types and whether they are between senses, between entries, or between entries or senses. This table displays the possibilities. The enum represents the integer when used in program code.

Int	Type of relation	enum
0	sense collection	kmtSenseCollection
1	sense pair	kmtSensePair
2	sense pair with two relation names	kmtSenseAsymmetricPair
3	sense tree	kmtSenseTree
4	sense sequence	kmtSenseSequence
5	entry collection	kmtEntryCollection
6	entry pair	kmtEntryPair
7	entry pair with two relation names	kmtEntryAsymmetricPair
8	entry tree	kmtEntryTree
9	entry sequence	kmtEntrySequence
10	entry or sense collection	kmtEntryOrSenseCollection
11	entry or sense pair	kmtEntryOrSensePair

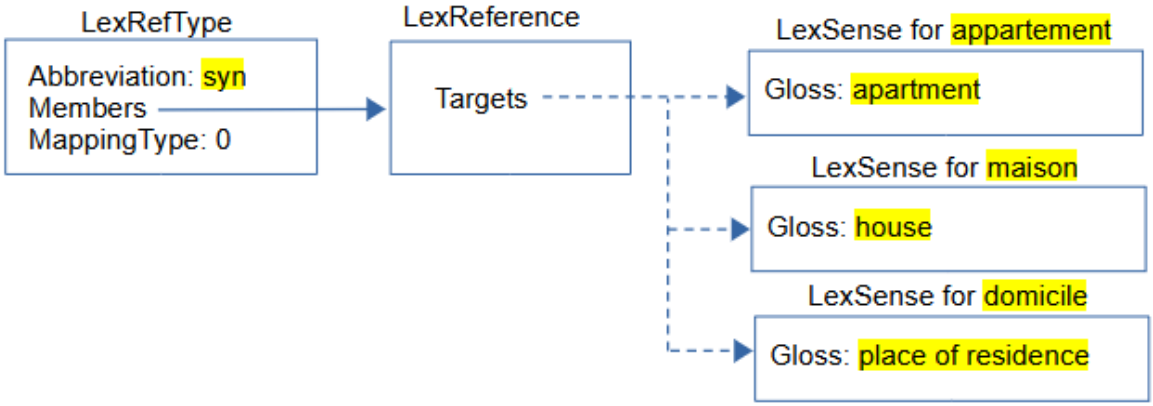
12	entry or sense pair with two relation names	kmtEntryOrSenseAsymmetricPair
13	entry or sense tree	kmtEntryOrSenseTree
14	entry or sense sequence	kmtEntryOrSenseSequence
15	sense unidirectional	kmtSenseUnidirectional
16	entry unidirectional	kmtEntryUnidirectional
17	entry or sense unidirectional	kmtEntryOrSenseUnidirectional

4.3.3.1 Collection

Here is an example of a synonym set:

appartement *n* apartment *syn:* **domicile** place of residence, **maison** house.
maison *n* house *syn:* **appartement** apartment, **domicile** place of residence.
domicile *n* place of residence *syn:* **appartement** apartment, **maison** house.

This is how it is represented in FLEx.



LexRefType is a subclass of CmPossibility owned in the Lexical Relations CmPossibilityList owned in the References owning property of LexDb, owned in the LexDb atomic owning property of LangProject. The Abbreviation for this object is ‘syn’ and the name is ‘Synonyms’. The MappingType 0 means the targets are a collection of senses.

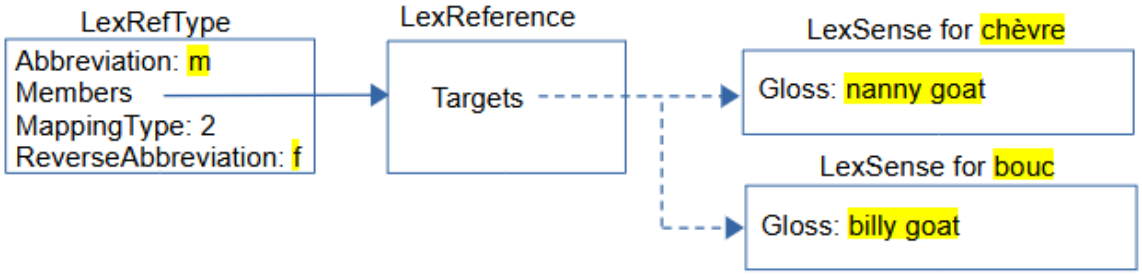
In this diagram, we are not showing the entries and related classes for each sense to keep the diagram simpler. LexReference Targets is a sequence reference, so it has a fixed order dependent on when items were entered into the set. In this example, the order happens to be appartement, maison, domicile, but the order is ignored in displaying synonyms. Instead, FLEx displays the synonyms sorted on the headwords. For this synonym relationship, the display for appartement will list the other two senses in the set: maison and domicile. As soon as the user adds the three items to the set from some entry, all of the references show automatically on all of the senses.

4.3.3.2 Pair with different names

Here is an example of a Male Female set for classifying animal names. This uses a pair with different names. Showing glosses following the related words is an option in Dictionary Configuration. In this case, and the following examples, that is unchecked.

chèvre *n* nanny goat *m:* **bouc**.
bouc *n* billy goat *f:* **chèvre**.

This is how it is represented in FLEx.



In this case, MappingType is 2 to indicate we only have two targets in each LexReference, and they use different names from both sides. For this type the LexRefType uses a ReverseAbbreviation to indicate the reverse side. We started from ‘chèvre’ and chose “Insert Male Relation (to this Female)” in the UI to connect to ‘bouc’, so the female is first in the targets and the male is second. From the female sense we use the Abbreviation ‘m’ to mark the male. From the male sense we use the ReverseAbbreviation ‘f’ to mark the female.

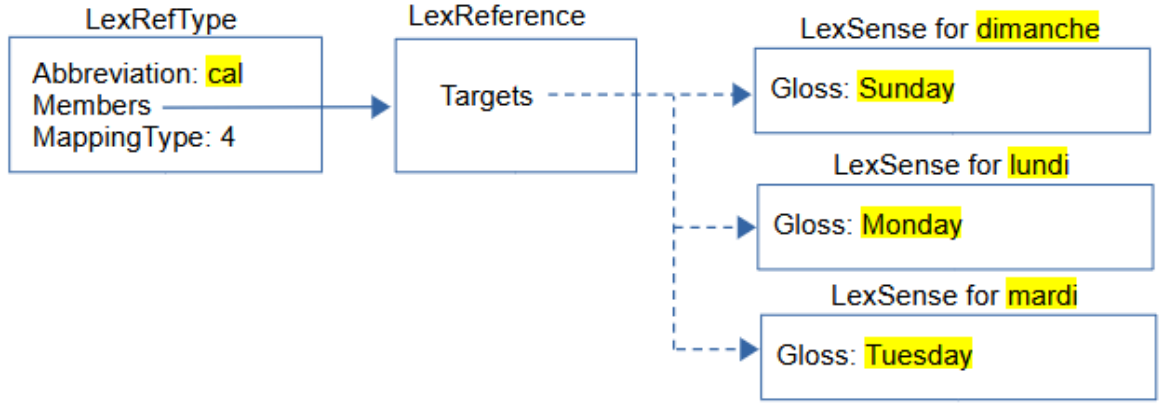
The Pair MappingType 1 would be similar to this but there would be no ReverseAbbreviation, and the Abbreviation would be used from both sides of the relation. Antonyms would typically use this type.

4.3.3.3 Sequence

Here is an example of a Calendar sequence set for classifying days of the week, months, etc. The order in this set is critical, and FLEx allows the user to move items left or right in the sequence if needed. The full sequence is shown on all senses.

dimanche *n* Sunday *cal:* **dimanche, lundi, mardi**.
lundi *n* Monday *cal:* **dimanche, lundi, mardi**.
mardi *n* Tuesday *cal:* **dimanche, lundi, mardi**.

This is how it is represented in FLEx.



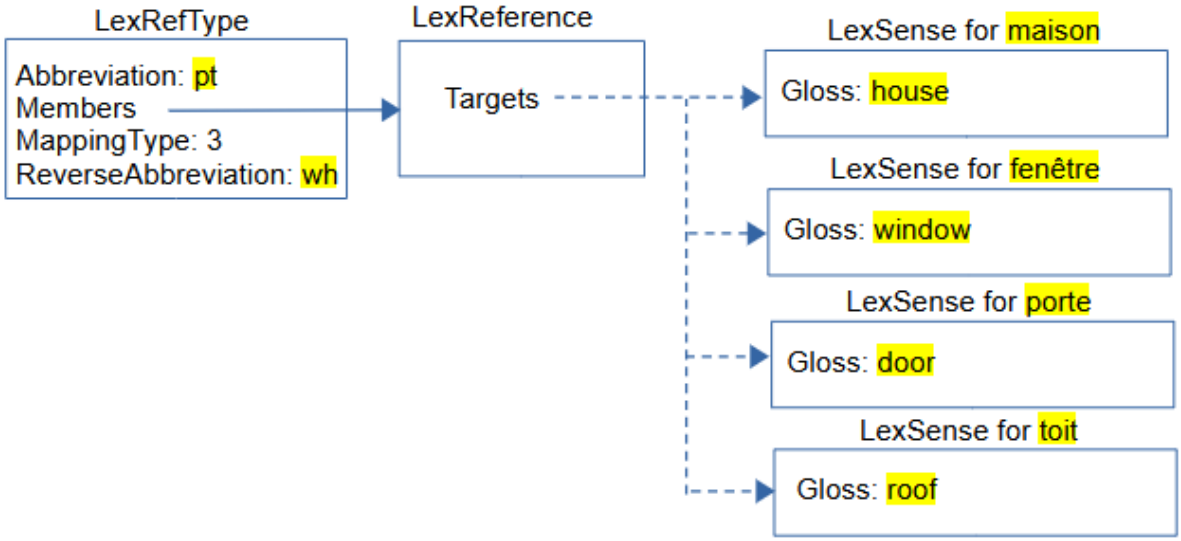
In this case, MappingType is 4 to indicate we have an ordered sequence in each LexReference, and all targets are shown from each sense.

4.3.3.4 Tree

Here is an example of a part-whole Tree relation that can provide a semantic hierarchy of senses.

maison *n* house *pt:* **fenêtre, porte, toit**.
fenêtre *n* window *wh:* **maison**
porte *n* door *wh:* **maison**.
toit *n* roof *wh:* **maison**

This is how it is represented in FLEx.



In this case, MappingType is set to 3 to indicate this is a Tree relation. The first item in Targets must be the root of the tree. The order of remaining items is not relevant as they are all leaves of the tree and will be sorted when they are displayed. The Abbreviation is used when displaying in the root item, and ReverseAbbreviation is used when displaying in the remaining items. Additional tree relations could be used for fenêtre, porte, or toit to further break these parts into subparts, thus providing a tree structure with branches.

4.3.3.5 Unidirectional

There are times when it is desirable to reference senses from one sense without anything showing from the target senses back to the first sense. All of the relations discussed so far always show relations from both ends of the links. The Unidirectional options provide for this need. Unidirectional Targets will be displayed in the order they occur without any sorting. The first Target is the sense that will display the relation. The remaining targets are all listed in order following the relation abbreviation or name. They are not sorted based on the headwords.

When one of the Entry types are used, everything works the same except the links are to entries instead of senses, and the relations would show up under Cross References instead of Lexical Relations in Lexicon Edit and Dictionary views. An ‘Entry Or Sense’ type would allow either Entries or Senses in the set.

Caution: Items in the Lexical Relations list are not simply objects that are referenced by other objects. Each item in this list also owns a collection of all of the lexical relation sets that are specified for this type of relation. So deleting an item in this list removes every instance of that type of relation. When a user tries to delete these, FLEx gives a warning message indicating how many relations will be deleted if they proceed.

4.3.4 Reversals

In FLEx, for each analysis language, there is a ReversalIndex object that is owned in the ReversalIndexes owning collection of LexDb. Each ReversalIndex has

- Entries owning collection of ReversalIndexEntry objects.
- PartsOfSpeech atomic owning property holding a CmPossibilityList. This list owns PartOfSpeech objects so that language-specific parts of speech categories can be used in the reversal display.
- Unicode WritingSystem property that contains the writing system code for this reversal index.

ReversalIndexEntry objects have

- MultiUnicode ReversalForm property that has the headword for the reversal entry in the language of the reversal list.
- PartOfSpeech atomic reference property to one of the items in the ReversalIndex parts of speech list.
- Senses reference sequence property that references LexSense objects.
- Subentries owning sequence property that owns ReversalIndexEntry objects allowing for hierarchical reversals.

LexSense uses a virtual property to find ReversalIndexEntry objects and displays this in the Lexicon Edit window. When typing something into a reversal field, FLEx finds or creates a ReversalIndexEntry in the appropriate language ReversalIndex and adds a link to the current sense to this item. A colon indicates a subentry.

Here is a simple English reversal index into the French dictionary:

D d

days of week
Monday n *lundi*
Sunday n *dimanche*

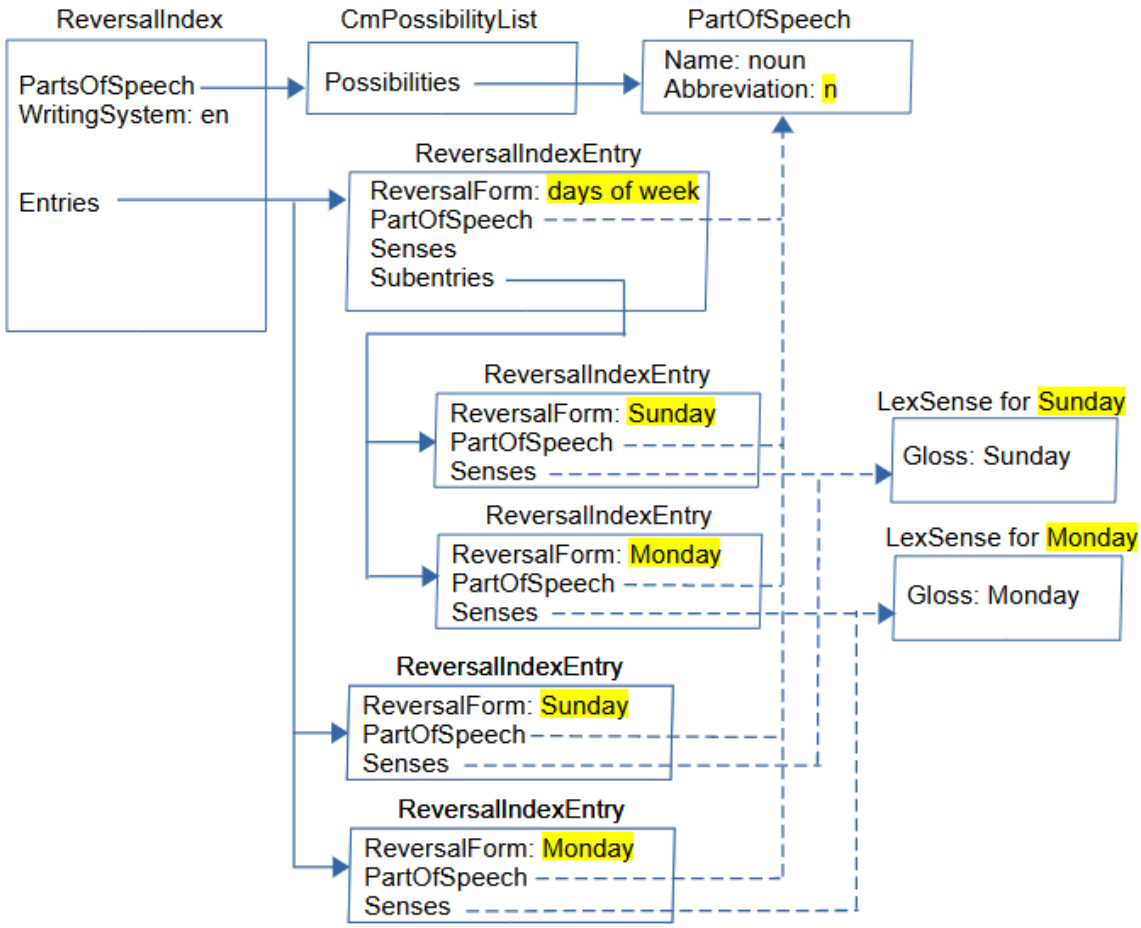
M m

Monday n *lundi*

S s

Sunday n *dimanche*

This is how it is represented in FLEx.



This is the English ReveralIndex that has an English Parts of Speech list with a noun category that is referenced by all 5 ReversalIndexEntry objects via the PartOfSpeech property. Three ReversalIndexEntry objects are owned through the Entries owning collection in the ReversalIndex. Two ReveralIndexEntry objects are owned in the Subentries owning sequence property of the first ReversalIndexEntry. The Senses reference sequence properties in the ReveralIndexEntry objects reference one or more senses to show with this reversal entry.

Use of Part of Speech categories in reversal entries is optional. Another option would be to include the sense part of speech using Reversal Configuration. Another option is to include a category marker at the end of the ReversalIndexEntry ReversalForm where necessary.

In the Lexicon Edit area, this is how it appears in the ‘lundi’ Reversal Entries field. There are two references to Monday. The second one is a subentry since it follows a colon in the ‘days of week’ reversal.

Reversal indexes are sorted and filtered using the Bulk Edit Reversal Entries tool in Lexicon Edit. The collation comes from the writing system of the reversal index. Sorting in this area only affects the main reversal entries. To sort the reversal subentries you have to use Tools > Utilities > “Sort Reversal Subentries”. If this subentry sort puts some subentries in an undesired order, you can correct that order in Reversal Indexes using the “Move Subentry Up/Down” right-click menu.

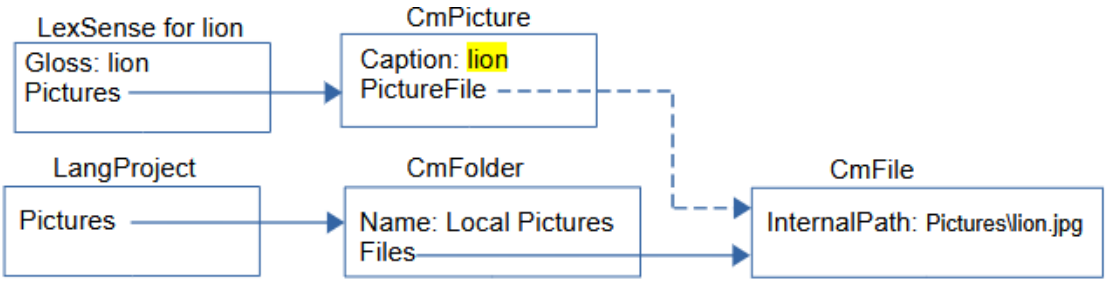
4.3.5 Pictures

FLEx allows multiple pictures on each sense. When inserting a picture, you should normally choose the option to copy the file to the project LinkedFiles\Pictures folder. It will then be backed up with FLEx backups and also included in Send/Receive. Here's an example of a picture with a caption and copyright information.

lion *n* a large tawny-colored cat that lives in prides. *Le lion est le roi de la jungle.* The lion is the king of the jungle.

lion Designed by Freepik, Copyright © 2024, Freepik Company

This is how it is represented in FLEx.



LangProject has an owning collection Pictures property that holds CmFolder objects. Pictures are normally stored in a CmFolder in this property with a name "Local Pictures". CmFolder has an owning collection Files property that owns CmFile objects. CmFile has a Unicode InternalPath property to record the location of a picture file. This path is normally a relative path starting at the project LinkedFiles directory, so it would include the Pictures folder as part of the path. Files can be located outside the project folder, and in that case it would be a full path to the file on the hard drive. CmFile has other properties that are not used in FLEx.

LeSense has an owning sequence Pictures property that holds CmPicture objects. CmPicture has a MultiString Caption property, and an atomic reference PictureFile property that references a CmFile holding the file name. CmPicture has various other properties that haven't been implemented in FLEx.

FLEx has a dialog for copyright information for pictures. This information is not stored in the model, but is stored in the picture file.

4.3.6 Sound files

See section 4.1.2 on Pronunciations for information on sound files in FLEx.

4.4 Virtual ordering

The default ordering of subentries within an entry is an alphabetical sort. There are times where this is undesirable. It's possible to override this order in given entries using the right-click "Move Left/Right" options in the "Subentries" and "Referenced Complex Forms" fields at the bottom of

the entry. When this is done, FLEx inserts an unowned VirtualOrdering object. Choosing Alphabetical Sort in these menus will remove the VirtualOrdering object and go back to default sorting.

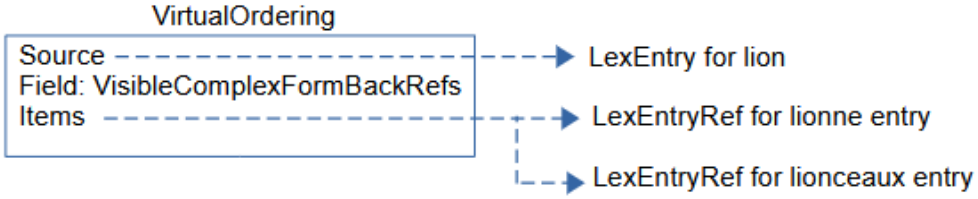
VirtualOrdering has

- Source - atomic reference to CmObject to which it applies.
- Field - Unicode name of the field it overrides on this object. It can be real or virtual.
- Items - sequence reference to CmObject. This is the order that is desired for this property on this object.

The sample project has the order switched in the ‘lion’ entry in Lexeme-based view by changing the “Referenced Complex Forms” field at the bottom of the entry.

lion *n* a large tawny-colored cat that lives in prides. *Le lion est le roi de la jungle.* The lion is the king of the jungle. der. **lionne**, **lionceaux**

This is how it is represented in FLEx.



The source is the ‘lion’ entry. The Field is a virtual VisibleComplexFormBackRefs property on LexEntry that returns the LexEntryRef items for the ‘lionne’ and ‘lionceaux’ entries.

When VirtualOrdering is used on the Subentries property, the Field is virtual Subentries, and the Items point to the ‘lionne’ and ‘lionceaux’ LexEntry objects.

The code for VirtualOrdering is pretty generic for VectorReferenceView. It may or may not work to add this class on other attributes.

4.5 Publications

FLEx allows multiple publications where each publication can pick which parts of the data to include in that publication. This filter, along with custom Dictionary Configuration views for selecting what fields to include and how to format them provides a flexible way of using the same data in FLEx to be printed in various publications.

FLEx has a Publications list for adding different publications. FLEx comes with a standard “Main Dictionary” item in this list. By default, everything you enter in FLEx will be included in this publication.

To exclude an object from a given publication, there are DoNotPublishIn properties on these classes:

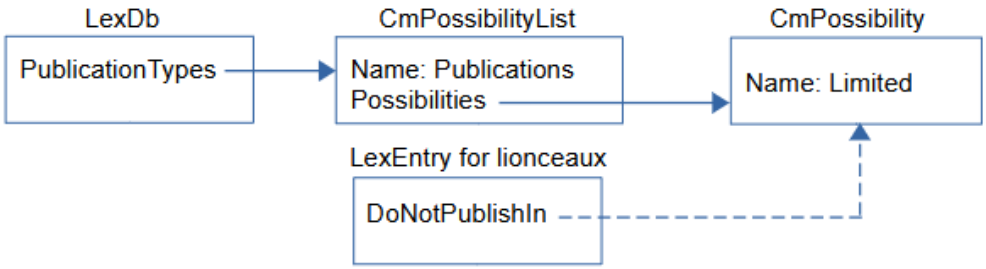
Class	Lexicon Edit Field name
LexEntry	Publish Entry In
LexSense	Publish Sense In
LexExampleSentence	Publish Example In
LexPronunciation	Publish Pronunciation In

CmPicture	Publish Picture In

DoNotPublishIn is a reference collection to CmPossibility in the Publications list. In Lexicon Edit there are fields on each of these classes with the opposite polarity. In the UI, FLEx shows which publications will include the current object, and you have the option to remove publications from each of these fields. When you remove a publication from an object, FLEx adds a reference to the removed publications to the DoNotPublishIn field.

In the example project, the 'lionceaux' entry has 'Limited' removed from the Publish Entry In field. In the Dictionary view, if you choose the Limited publication from the left chooser in the title bar, you will see that 'lionceaux' no longer shows in the dictionary view. But if you choose the Main Dictionary publication, then 'lionceaux' is included.

This is how it is represented in FLEx.



The Publications CmPossibilityList has a "Limited" CmPossibility item. The LexEntry for 'lionceaux' has a reference from DoNotPublishIn to the 'Limited' item, thus it will not be printed in this publication.

5 Grammar model

Most objects in the grammar area come from factory lists that are provided with FLEx. The user normally chooses from these lists to provide the features they want. The core part of the grammar is the Parts Of Speech list seen in the Category Edit tool in the Grammar area. In addition to providing names and abbreviations for categories, there are other details that can be added to a category including Affix Templates, Affix Slots, Inflection Class Info, Features, and Stem Names. Using these features along with lexical entries, a user can use morphological parsers to validate morphology as specified by the grammar. These specifications affect the XAmple, HermitCrab, and interlinear parsers.

In addition to Category Edit, in the Grammar area, there are various tools to list phonemes and phonological features and rules, natural classes, environments, inflection features, etc. There is also a Grammar Sketch tool that will provide a sketch of your grammar at any point in time. The Feature Types list in the Lists area is also related to Grammar and is not a true CmPossibility list.

To illustrate some of the grammar capabilities, the sample project illustrates some basics for parsing French verbs that follow the ER paradigm. Only the first and third person suffixes are currently defined. This can parse the verb 'manger' into 'mange' for first person singular (I eat), and 'mangeons' for third person singular (we eat). Once this is set up, it could parse other verbs that use the same affixes to determine person and number, as long as the other main entries are linked to the 'ER Verbs' Inflection Class in their Grammatical Info Details section.

3.1	**Word**	Je	mange	
	Morphemes	je	mang	-e
	Lex. Entries	***	mang	-e
	Lex. Gloss	***	eat	PRS.1SG
	Lex. Gram. Info.	***	v (ER)	v:vPersNum
	Word Gloss	I	eat	
	Word Cat.	pro	v	

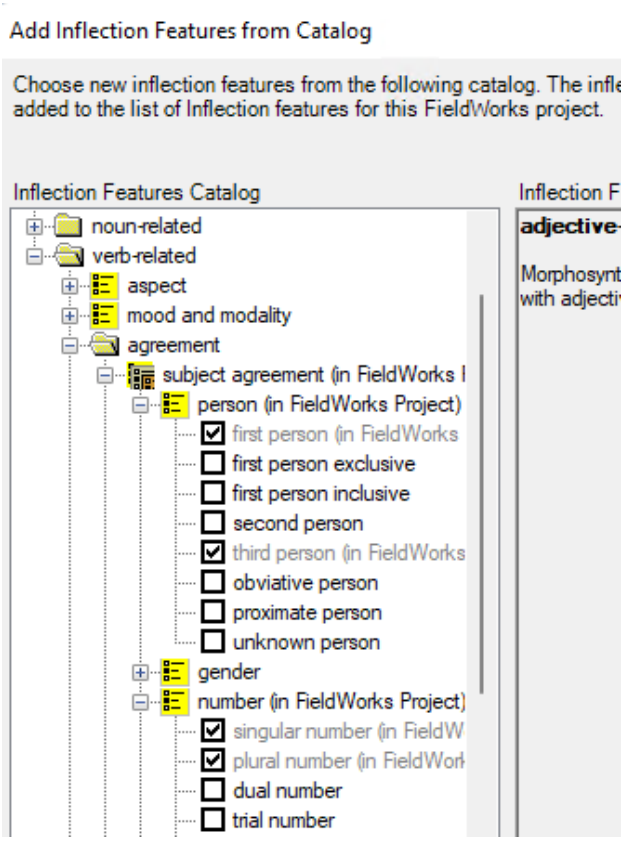
Free I eat.

3.2	**Word**	Nous	mangeons	
	Morphemes	nous	mang	-eons
	Lex. Entries	***	mang	-eons
	Lex. Gloss	***	eat	PRS.3SG
	Lex. Gram. Info.	***	v (ER)	v:vPersNum
	Word Gloss	we	eat	
	Word Cat.	pro	v	

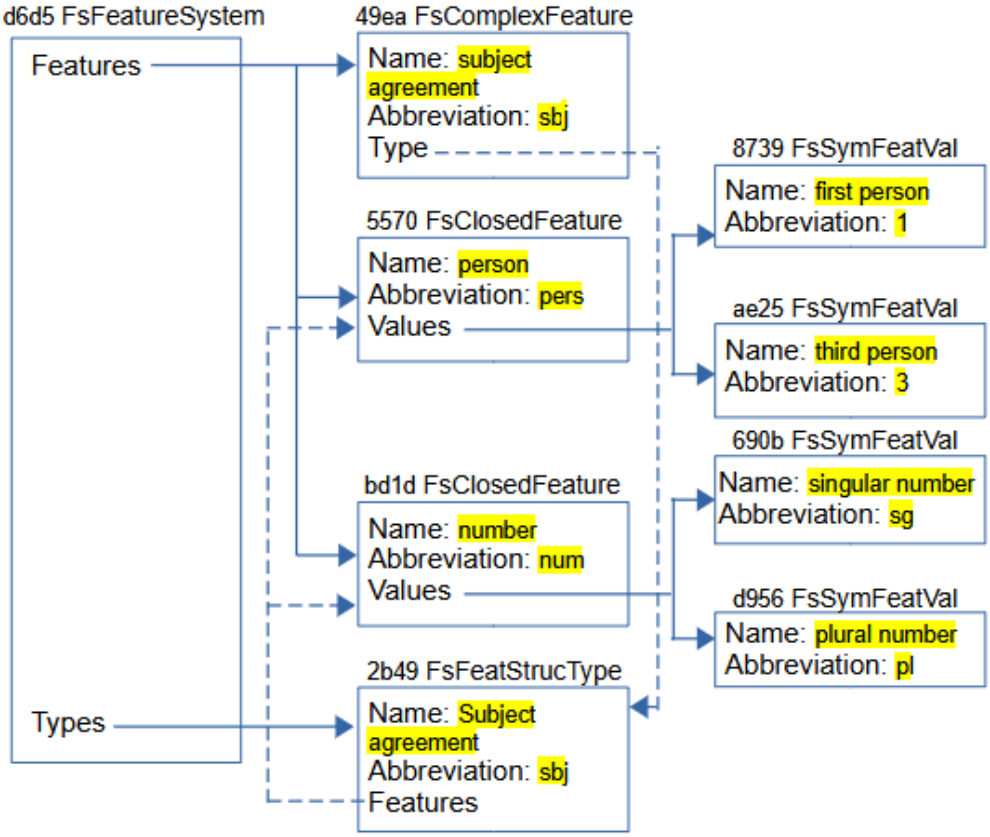
Free We eat.

5.1 Inflection features

In order to do this, in the Inflection Features tool, add a new Complex Feature with check marks in person and number sections under subject agreement.



This shows what happens in FLEx.



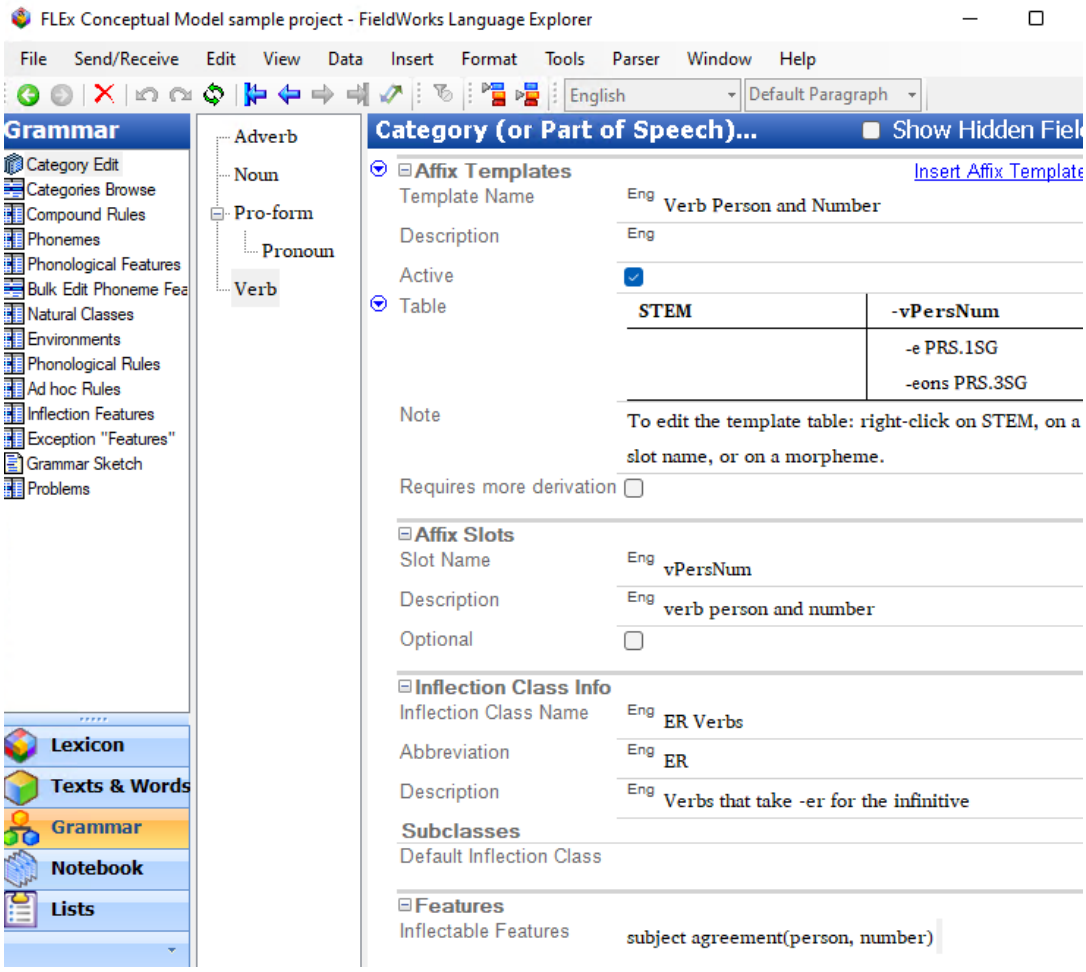
The feature system is stored in a FsFeatureSystem object owned in the MsFeatureSystem atomic owning property of LangProject. FsFeatureSystem has a Features owning collection of FsFeatDefn objects, and a Types owning collection of FsFeatStrucType objects. FsFeatDefn is an abstract class with the possibilities: FsClosedFeature, FsComplexFeature, and FsOpenFeature.

In this example we have

- A FsComplexFeature for subject agreement and it references the FsFeatStrucType for Subject agreement in the Type property. The Features on this type reference two FsClosedFeature objects.
- A FsClosedFeature for person owns two FsSymFeatVal objects in the Values property, one for first person, and the other for third person.
- A FsClosedFeature for number owns two FsSymFeatVal objects in the Values property, one for singular number, and the other for plural number.

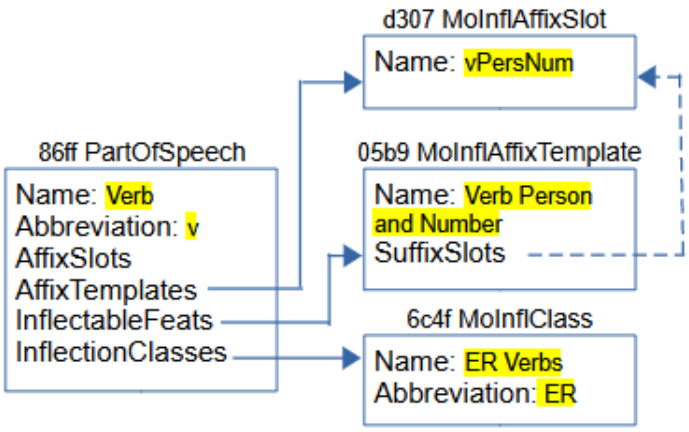
5.2 Categories

In Category Edit, add an Affix Template, an Affix Slot, and an Inflection Class Info to the Verb category.



At this point, the affix template will not have any entries in the vPersNum column

This is how it is implanted in FLEx.

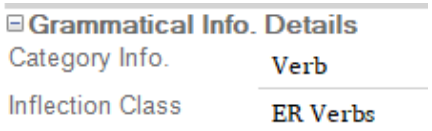


In the Verb PartOfSpeech, we have a ‘vPersNum’ MoInflAffixSlot owned in the AffixTemplates. This gives the ‘vPersNum’ column in the ‘Verb Person and Number’ affix template. The InflectableFeats property owns a MoInflAffixTemplate for ‘Verb Person and

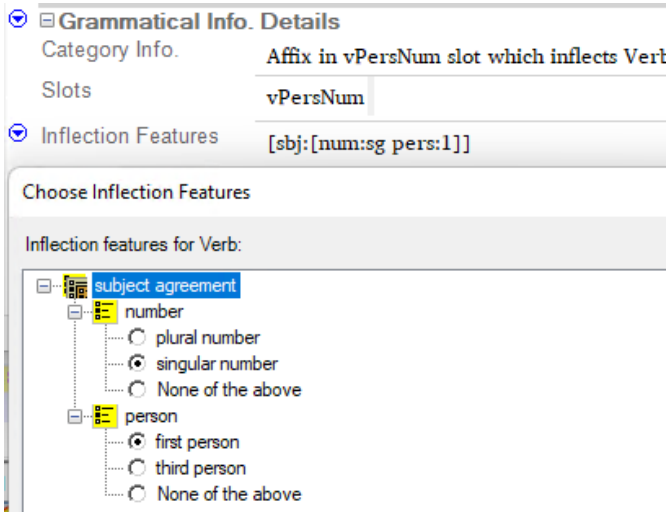
Number' and the SuffixSlots in that object is set to the 'vPersNum' MoInflAffixSlot object. InflectionClasses owns an 'ER Verbs' MoInflClass object.

5.3 Lexicon

In the entry for 'manger', we specify the lexeme form as 'mang' and the citation form as 'manger'. In the Grammatical Info Details section the entry is classified as a verb, and the Inflection Class is set to ER Verbs.

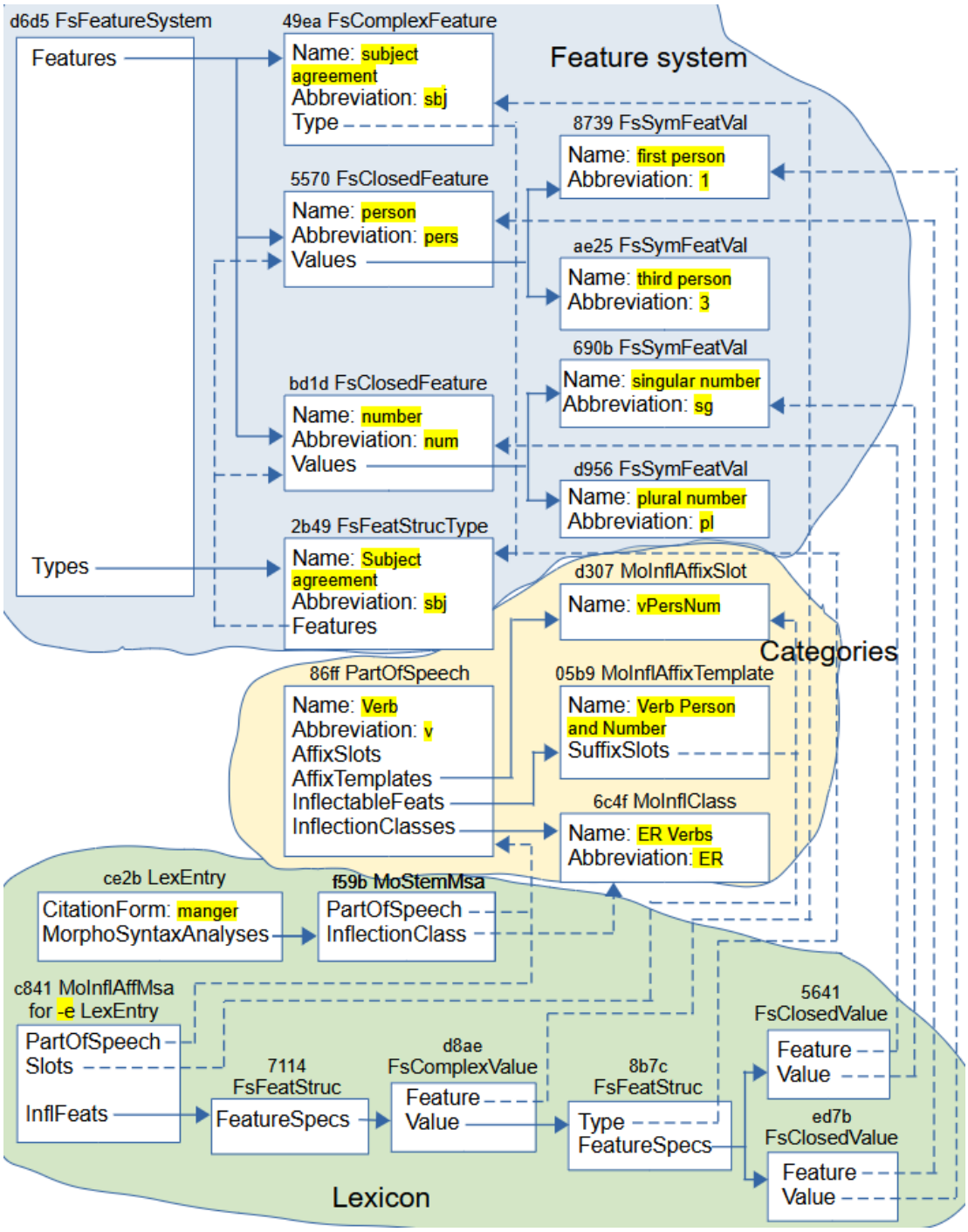


In the '-e' suffix entry, we add the entry to the 'vPersNum' slot, and set the Inflection features to 'singular number' and 'first person'.



The '-eons' entry, we make similar changes except person is set to 'third person', and number is set to 'plural number'.

This is represented in FLEx data as



In the lexicon, the ‘manger’ entry owns a MoStemMsa object that references the ‘Verb’ PartOfSpeech object in the PartOfSpeech reference property. It also references the ‘ER Verbs’ MoInflClass, indicating that this entry is one that follows the standard ER paradigm.

The MoInflAffixMsa object owned by the ‘-e’ LexEntry references the ‘Verb’ PartOfSpeech object via the PartOfSpeech reference property. The Slots reference property references the ‘vPersNum’ MoInflAffixSlot object so that it will show up in the ‘Verb Person and Number’ affix template. The InflFeats property owns a FsFeatStruc object.

The 7114 FsFeatStruc objects owns a FsComplexValue object in the FeatureSpecs property.

The FsComplexValue object has a Feature reference property that references the ‘subject agreement’ FsComplexFeature object. This references the person FsClosedFeature and the number FsClosedFeature objects that are relevant to the FsComplexValue object in the ‘-e’ entry. The FsComplexValue object also has a Value property that owns a FsFeatStruc object.

The 8b7c FsFeatStruc has a Type property that also references the ‘Subject agreement’ FsFeatStrucType object. The FsFeatStruc also has a FeatureSpecs property that owns two FsClosedValue objects.

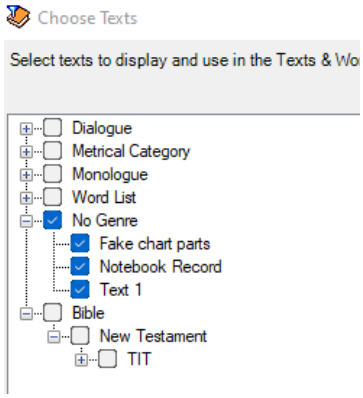
The 5641 FsClosedValue object references the ‘number’ FsClosedFeature object in the Feature property and the ‘singular number’ FsSymFeatVal obejct in the Value property.

The edb7 FsClosedValue object references the ‘person’ FsClosedFeature object in the Feature property and the ‘first person’ FsSymFeatVal obejct in the Value property.

There wasn’t room to show the ‘-eons’ entry in this diagram, but it would be very similar to the ‘-e’ entry diagram, except the final FsClosedValue objects would have a value set to the ‘plural number’ FsSymFeatVal object, and the ‘third person’ FsSymFeatVal objects.

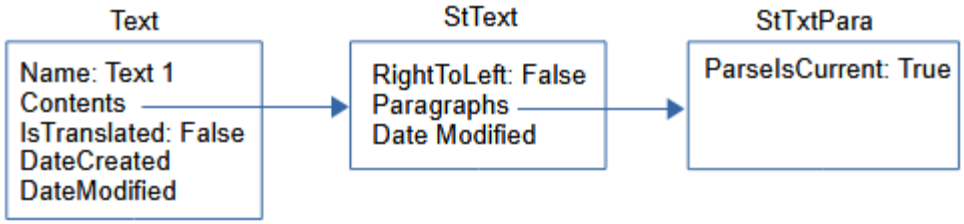
6 Interlinear model

FLEx maintains a wordform inventory based on unique words in baselines of interlinear texts in the Texts & Words area. The wordform inventory can be viewed in the Word Analyses tool in the Texts & Words area. Initially capitalized and uncapitalized words are both included because it is unknown whether they should be different or not. The wordforms shown in Word Analyses are limited to ones in the texts filter. To change the filter, go to View > Choose Texts, or use the Choose Texts icon in the toolbar. This will show all of the texts in the project. Ones with checkmarks are in the current filter that affects what you see in Word Analyses and Interlinear Texts.



6.1 Structured Text

Suppose we create a new Text 1 interlinear text without typing anything in the baseline. This is what we have in FLEx.



An interlinear text is implemented by a Text object that has

- Name - MultiUnicode inherited from CmMajorObject shown as the Title in the UI.
- Abbreviation – MultiUnicode inherited from CmMajorObject
- Contents – atomic owning holding a StText
- IsTranslated - Boolean to indicate if it's translated text, such as scripture, or original text.
- DateCreated and DateModified inherited from CmMajorObject
- Source - MultiString
- Genres – reference collection to CmPossibility in the Genres list. This allows texts to be hierarchically arranged in the Choose Texts dialog making it easier to filter on certain types of text.

Text objects do not have owners. Most of these Text properties can only be seen or modified in the Info tab in the Interlinear Texts tool.

StText (Structured Text) is a class in FLEx that can hold any number of paragraphs (StTxtPara) that can have styles associated with paragraphs and text in paragraphs. An StText can be used kind of like a Google document. Whenever you press the Enter key, it starts a new StTxtPara at that point. StText has

- Paragraphs – an owning sequence of StPara which is abstract, and is implemented as StTxtPara or ScrTextPara (for Scripture).
- DateModified – Time when modified
- RightToLeft – Boolean. True if the text is right-to-left, or False if left-to-right.
- Tags – owning collection of TextTag (used in text tagging).

StTxtPara stores the text for the baseline of the paragraph, plus additional information for interlinearization. StTxtPara has

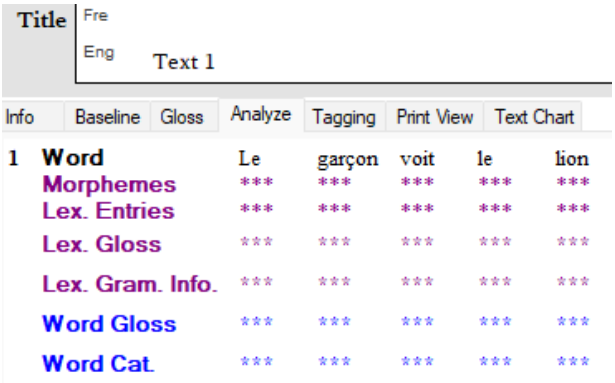
- Contents – String holding the baseline text.
- Segments – an owning sequence of Segment for interlinearization.
- ParseIsCurrent – Boolean. True if the Segments have not changed since they were last parsed, or False if they need to be reparsed.
- StyleRules – Used where paragraph and text styles are allowed.

There are some other properties that are not relevant at this point.

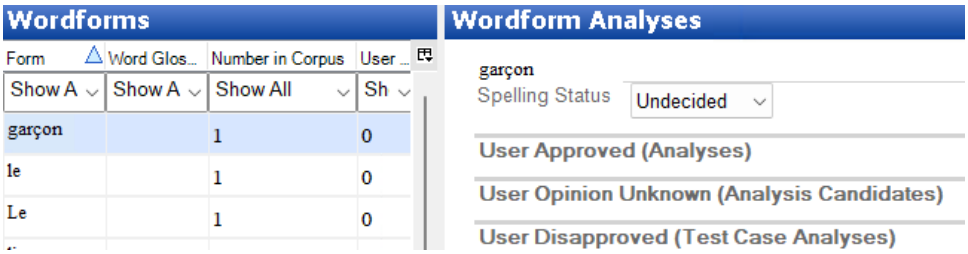
In the empty Text diagrammed above, the only real information is the title of the text in the Name property of Text. The StTxtPara does not have any text in the baseline at this point.

6.2 Segment

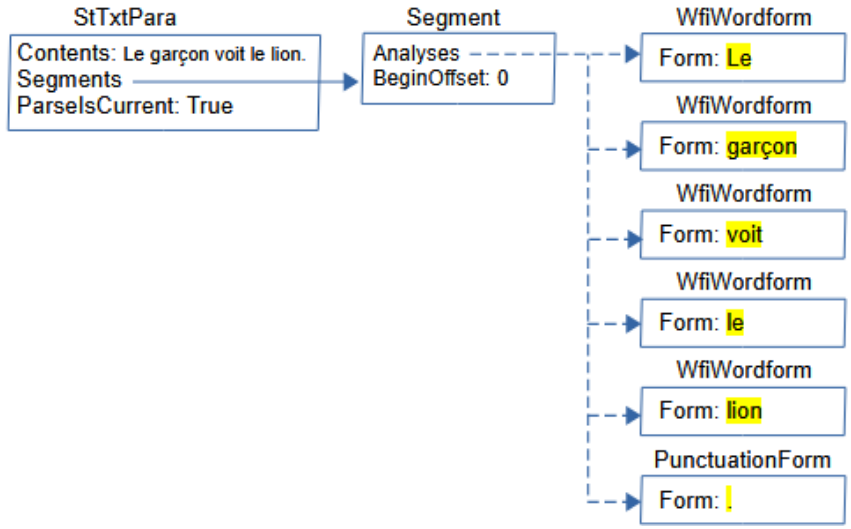
If we now type a text in the baseline and go to the Analyze tab, it will look like this, assuming none of the words are in the lexicon and you haven't done any previous analyses.



Word Analyses will show this:



At this point, the implementation in FLEx starting at StTxtPara is this.



Segment is a class that holds all the information that is relevant to the analysis of a sequence of words ending in period, question mark, exclamation point, or section sign (U+00A7). Segment has

- Analyses – reference sequence to IAnalysis, which is an interface that includes WfiWordform, WfiAnalysis, WfiGloss, and PunctuationForm.
- BeginOffset – An integer giving the offset from the beginning of the Contents string in StTxtPara to the beginning of this segment. This allows FLEx to find the section of baseline text in the StTxtPara Contents string that is covered by this Segment.
- FreeTranslation – MultiString giving the free translation of the text in this segment.
- LiteralTranslation – MultiString giving the literal translation of the text in this segment.
- Notes – owning sequence of Note, allowing notes to be added to this segment.

- Other properties used to store information from ELAN, but not currently covered by FLEx UI include BeginTimeOffset, EndTimeOffset, MediaURI, Reference, and Speaker.
- Custom fields are also possible on Segment.

The baseline is stored in the Contents String of StTxtPara. FLEx breaks the baseline from the paragraph into segments ending in period, question mark, exclamation point, or section sign (U+00A7). Since there is only one period in this example, we only need one Segment in the Segments owning sequence property. In this case, the segment starts at the beginning of the string, so BeginOffset is set to 0.

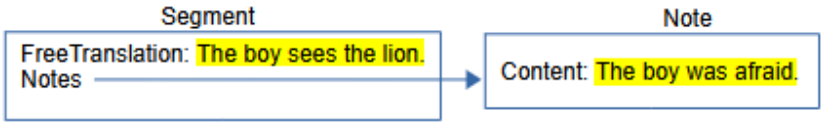
As editing takes place, FLEx attempts to keep the Analyses accurately matching the baseline. During Send/Receive, it's not always possible to keep everything current. When this happens, FLEx sets ParseIsCurrent on StTxtPara to False. The next time that Text is opened, FLEx will reparse the baseline text and resets the Analyses to reflect the current state of the data. A user can also force this operation at any time by using Tools > Utilities > Force Rechecking Word Breaks. This sets all of the StTextPara ParseIsCurrent flags to false. When you go to the Word Analyses tool, or the Concordance tool, FLEx will automatically reparse all Texts and reset the flags to True.

This is a free translation and a Note at the bottom of a Segment.

Free The boy sees the lion.

Note The boy was afraid.

This is how it is implemented in FLEx (omitting other Segment fields)



6.3 Wordforms

FLEx breaks the baseline text for this segment into wordforms based on space and punctuation, then finds an existing WfiWordform that matches the wordform, or creates a new WfiWordform with the text from the wordform which is stored in the MultiUnicode Form field of WfiWordform. WfiWordform objects do not have owners. The Analyses reference sequence property on Segment stores a reference to each WfiWordform or PunctuationForm. When Analyses reference WfiWordform, it means there is no analysis for this word at this point. In the display, this is shown by 3 asterisks in each field. Punctuation is stored in PunctuationForm objects and referenced from Analyses. The Form field for PuncutationForm is a String property.

When creating a new interlinear text, if you have more than one vernacular writing system, FLEx will ask which writing system you will be using for this text. This is not stored in any property. The writing system of the first character of the baseline is the only way to determine this. Any runs that are not in this writing system in the baseline are stored in PunctuationForm objects rather than WfiWordform objects. Any text in PunctuationForm objects cannot be analyzed.

PunctuationForm objects are never deleted. Because they will be created if a person uses any writing system other than the vernacular writing system for the baseline, they should avoid this as much as possible. If the project gets cluttered with many unneeded PuncuationForm objects,

they can all be deleted. In Notepad++, you can delete all PunctuationForm objects from fwdata by using this regular expression search and replace.

Search <rt class="PunctuationForm".*\r\n<Form>\r\n<Str>\r\n<Run.*\r\n</Str>\r\n</Form>\r\n</rt>\r\n
Replace (leave empty)

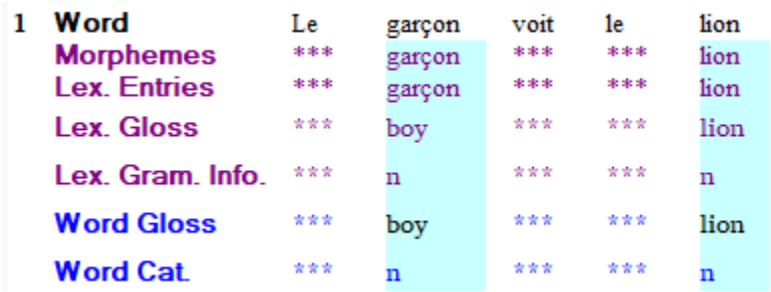
After doing this, you need to run Tools > Utilities > Find and fix… from another project to process your repaired project to remove any references to the deleted punctuation forms. Then open the project and use Tools > Utilities > Force Rechecking Word Breaks. Then go to Concordance or Word Analyses which will reparse everything and add new PunctuationForm objects where needed.

Wordforms should be unique. If duplicates somehow occur, you can run Tools > Utilities > Merge Duplicate Wordforms to merge them together. This will copy the analyses from the duplicate ones into a single wordform. After running this, you should run Tools > Utilities > Merge Duplicate Analyses. This will merge any duplicate analyses in the wordforms.

FLEx can interlinearize texts in different vernacular writing systems. There are various issues involved in doing this. See section 13, Interlinearizing with multiple scripts in https://downloads.languagetechnology.org/fieldworks/Documentation/Flex_tips.pdf before attempting to do this.

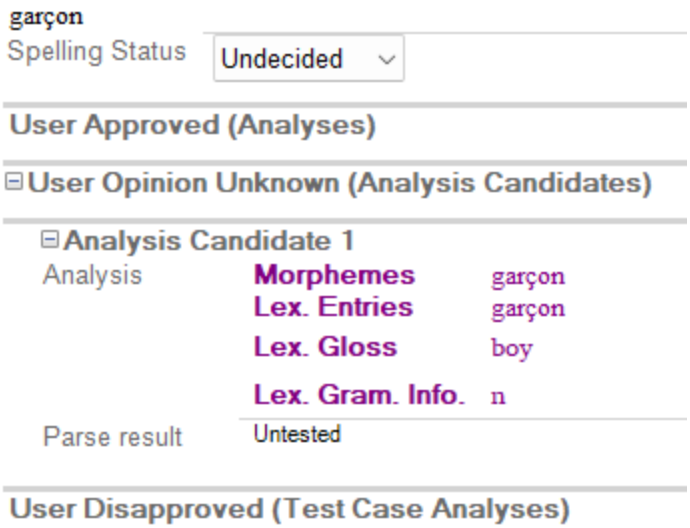
6.4 Analyses

Suppose the lexicon already has a definition for ‘lion’ and ‘garçon’? The interlinear display will now show this in the Analyze tab



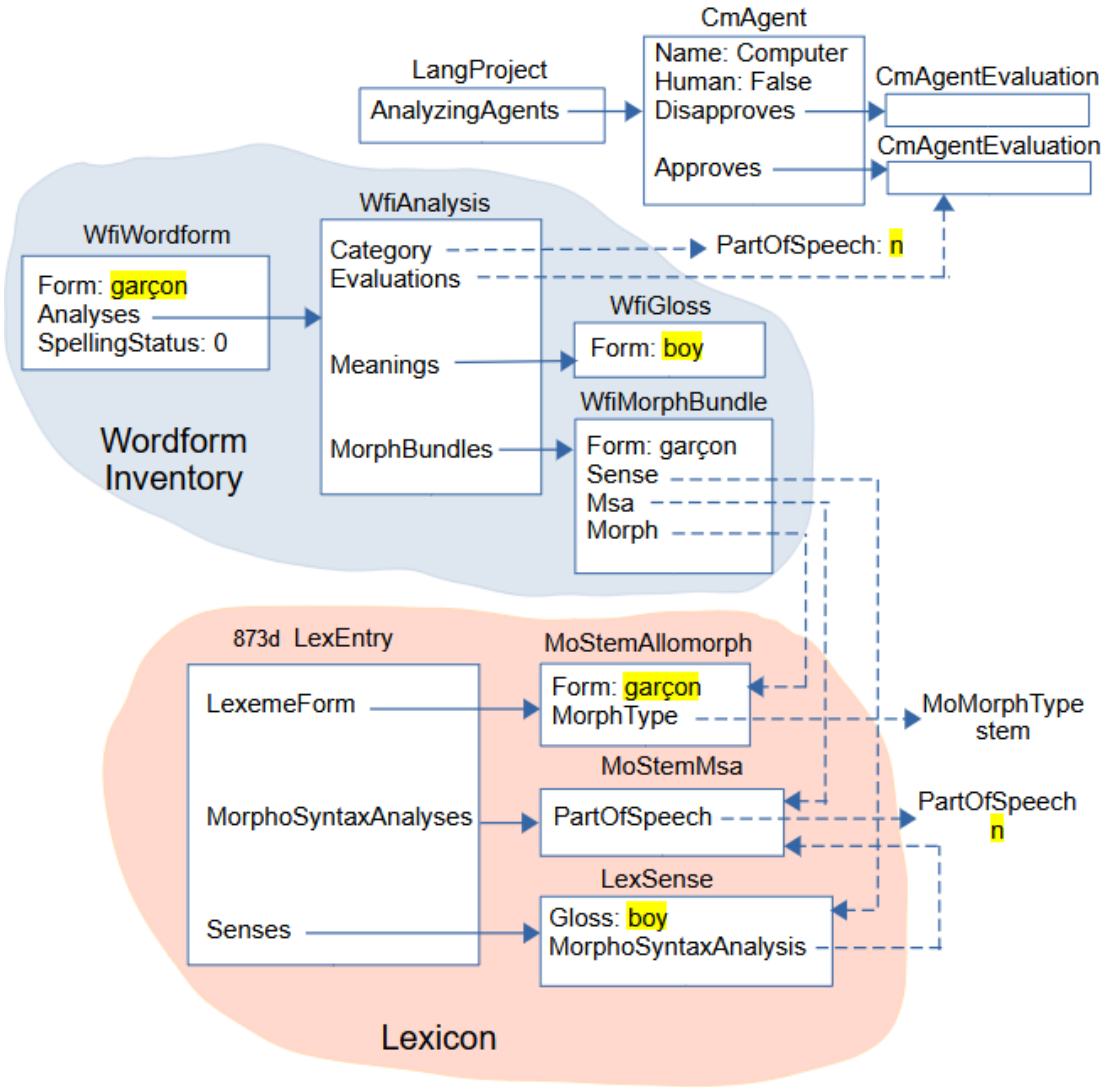
1	Word	Le	garçon	voit	le	lion
	Morphemes	***	garçon	***	***	lion
	Lex. Entries	***	garçon	***	***	lion
	Lex. Gloss	***	boy	***	***	lion
	Lex. Gram. Info.	***	n	***	***	n
	Word Gloss	***	boy	***	***	lion
	Word Cat.	***	n	***	***	n

and Word Analyses will now show this for ‘garçon’



The user hasn't done anything to the text, but FLEx found two words in the lexicon that matched a wordform, so it automatically created an analysis candidate that is shown in blue in the interlinear Analyze tab.

This is what FLEx now has for the 'garçon' wordform.



LangProject owns the following CmAgent objects in the AnalyzingAgents owning collection.

Name of CmAgent	Human	Description
M3Parser	False	approved or disapproved by the XAmple parser
HCParser	False	approved or disapproved by the Hermit Crab parser
Computer	False	approved or disapproved by the interlinear parser
Default user	True	approved or disapproved by the user

Each CmAgent has an Approves, and a Disapproves property owning a CmAgentEvaluation. Analyses can be approved or disapproved by any of these CmAgents by referencing the desired CmAgentEvaluation through the Evaluations reference collection property on WfiAnalysis. The current suggestions are generated and approved by the Computer parser. At this point it's up to the user whether they want to approved the analyses or not.

The 'garçon' WfiWordform now owns a WfiAnalysis object. A WfiAnalysis objects has

- Category – atomic reference to a PartOfSpeech in the Category list in the Grammar area. This shows up in the Word Category line of the Analyze tab of an interlinear text.

- Meanings – owning collection holding WfiGloss objects. One of these WfiGloss objects will show up in the Word Gloss line of the Analyze tab of an interlinear text.
- MorphBundles – owning sequence of WfiMorphBundle. There is one WfiMorphBundle for each morpheme in the wordform analysis.
- Evaluations – reference sequence to CmAgentEvaluation approving or disapproving the analysis for one or more CmAgents.

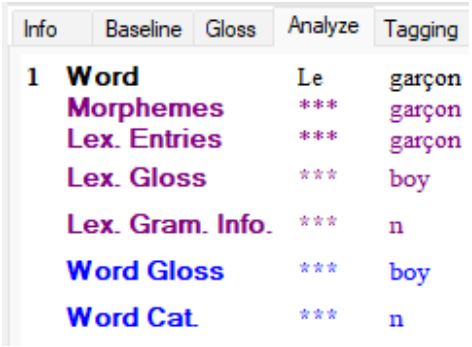
In this case the WfiAnalysis references a Noun PartOfSpeech through the Category property. It owns one WfiGloss with form ‘boy’ in the Meanings property. It references a CmAgentEvaluation for being approved by the computer. It has one WfiMorphBundle in MorphBundles. WfiMorphBundle has

- Form – MultiString form for the morpheme. This form will be displayed in the Morphemes line in the Analyze tab of an interlinear text if the Morph property is empty. Once the Morph property is set, the Morphemes line displays the Form from the referenced MoForm,
- Morph – atomic reference to a MoForm, either in the LexemeForm or AlternateForms property of a LexEntry. This Form is displayed in the Lex Entries line in the Analyze tab of an interlinear text.
- Msa – atomic reference to a MoMorphSynAnalysis in the Morph entry. This PartOfSpeech in this analysis is displayed in the Lex Gram Info line in the Analyze tab of an interlinear text.
- Sense – atomic reference to a LexSense in the Morph entry that shares the Msa referenced by this WfiMorphBundle. The Gloss from this sense is displayed in the Lex Gloss line in the Analyze tab of an interlinear text.

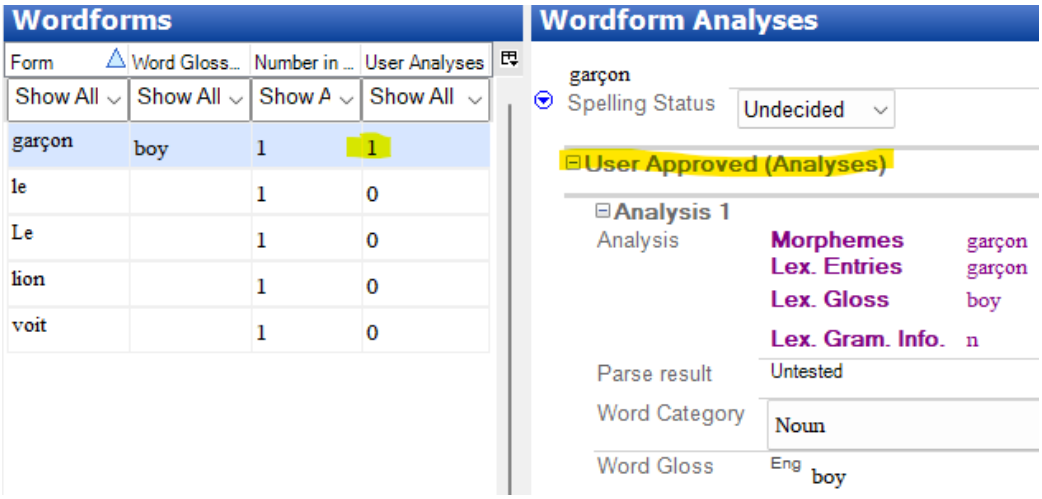
In the sample WfiMorphBundle, the Form contains ‘garçon’, but it is unused because the Morph property is referencing the MoStemAllomorph of the ‘garçon’ LexEntry. The Msa references the MoStemMsa from the ‘garçon’ LexEntry and displays ‘n’ part of speech referenced by the MoStemMsa. The Sense references the only sense in the ‘garçon’ LexEntry and displays ‘boy’ from the Gloss. The Word Gloss displays ‘boy’ from the WfiGloss Form. The Word Cat displays ‘n’ from the WfiAnalysis Category.

6.5 Approving analyses

If the user clicks the green Approve button in ‘garçon’ in the word focus box, the Analyze tab shows



and the Word Analyses tool shows



Note that now ‘garçon’ shows 1 User Analysis, and the analysis now appears under the User Approved (Analyses) section.

Two changes occurred in the FLEx data.

- In the Segment Analyses list of references for wordforms, the second one that used to reference the ‘garçon’ WfiWordform now references the ‘boy’ WfiGloss object owned in the WfiAnalysis owned by the ‘garçon’ WfiWordform.

  <rt class="Segment" guid="78220579-5e21-47df-82b1-1fd72a02748d" ownerguid="a51ed87e-d6fe-44d6-a317-1b726d218fe3">
  <Analyses>
  <objsur guid="b0460a42-7e19-460e-9ec8-4779c4a5f3ff" t="r" /> Le
  <objsur guid="7dec7244-62f4-441f-bfc9-c9564093c2f0" t="r" /> <- garçon WfiGloss 'boy'

- In addition to the WfiAnalysis Evaluations referencing the Computer Approves CmAgentEvaluation, it now also references the “default user” Approves CmAgentEvaluation.

  <rt class="WfiAnalysis" guid="be5aec0f-ff8d-4718-a97d-018a1fcd23d6" ownerguid="40b5890e-abc9-4741-9ef7-8dea5fb88c37">
  <Evaluations>
  <objsur guid="41581e3a-7ed3-41ac-94c7-6e91dbcc108b" t="r" /> <- Computer approves
  <objsur guid="8caa11bb-cac4-4836-a081-1666245106b9" t="r" /> <- Default user approves
  </Evaluations>

In summary, the Analyses property of Segment can reference 4 types of objects.

- A WfiWordform with a Form that matches the wordform in the baseline. This indicates the wordform is unanalyzed.
- A WfiAnalysis owned by a WfiWordform with a Form that matches the wordform in the baseline. This indicates the wordform is partially analyzed. The analysis is known, but the word gloss has not been chosen.
- A WfiGloss owned in a WfiAnalysis owned by a WfiWordform with a Form that matches the wordform in the baseline. This indicates the wordform is fully analyzed to an exact WfiGloss.
- A PunctuationForm contains punctuation in the baseline and any strings that are not in the vernacular writing system of the text. They cannot be analyzed.

Here's an example of a word with two morphemes:

Word	lions	
Morphemes	lion	-s
Lex. Entries	lion	-s
Lex. Gloss	lion	PL
Lex. Gram. Info.	n	n:Any
Word Gloss	lions	
Word Cat.	n	

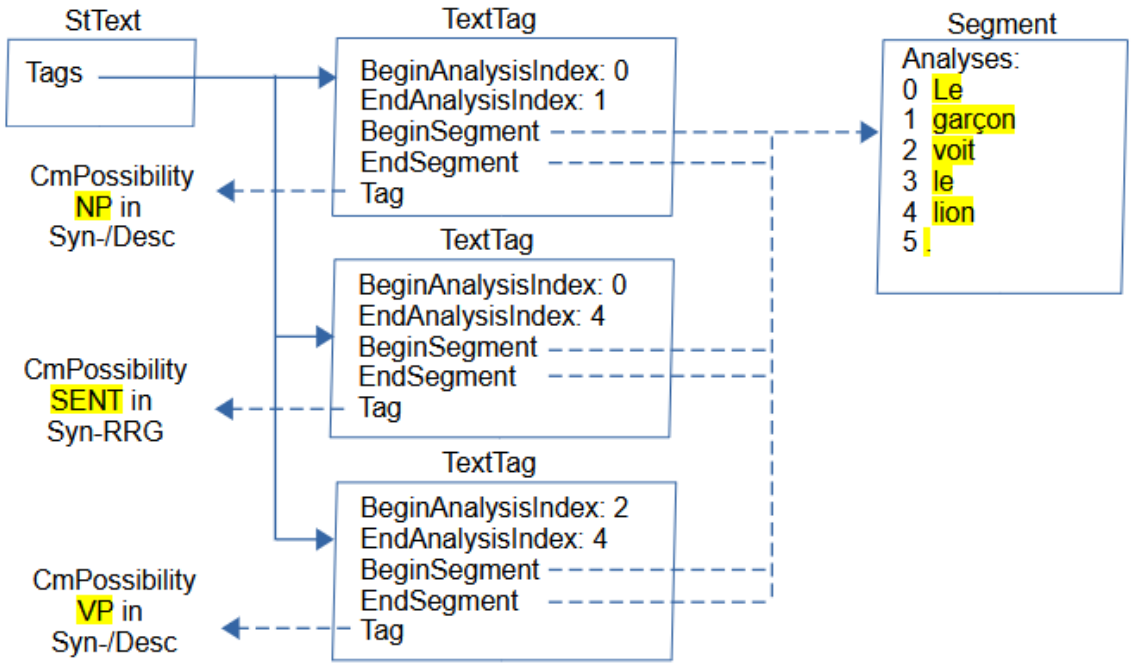
The FLEx diagram would be very similar to the previous diagram for 'garçon', but the WfiAnalysis MorphBundles property now has two WfiMorphBundle objects; one for 'lion' and one for '-s'. The first MorphBundle would reference the 3 parts of the 'lion' entry, and the second MorphBundle would reference the 3 parts of the '-s' entry.

6.6 Text Tagging

Interlinear text offers a Tagging tab that allows a user to tag sections of the baseline for Grammatical Relations – Functional, Semantics – RRG, Syntax – Descriptive, or Syntax – RRG. You can also create your own tagging structure. The tags are defined in the Text Markup Tags list in the Lists area. The top-level tags define rows in the tagging system, and subitems provide the tag markings within that line.

In the Interlinear Text Tagging tab you can highlight a string of wordforms and right-click to choose a tag to assign to these wordforms. You can choose from more than one level. The result might look like this

This is represented as follows in FLEx



The Text Markup Tags list is owned in the TextMarkupTags property of LangProject. When you add a tag, FLEx adds a TextTag object in the Tags owning collection of StText.

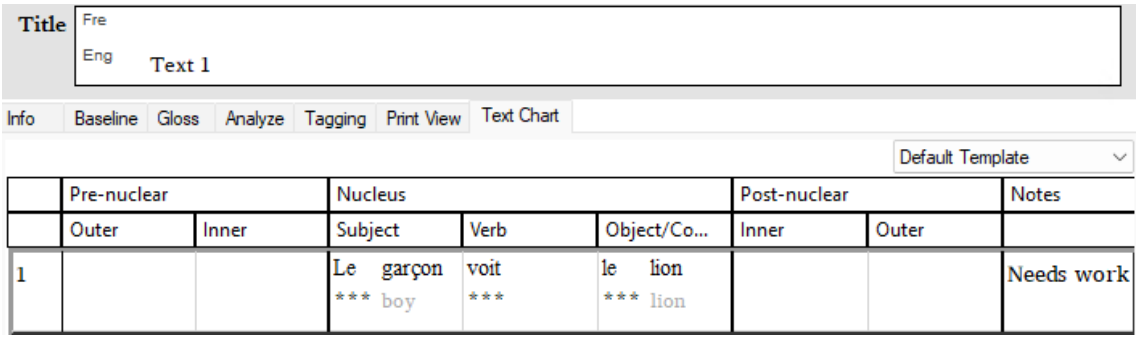
TextTag has

- BeginAnalysisIndex – Integer indicating the beginning index of IAnalysis references in the Segment Analyses property.
- EndAnalysisIndex – Integer indicating the ending index of IAnalysis references in the Segment Analyses property
- BeginSegment – atomic reference to the Segment where the tag begins
- EndSegment – atomic reference to the Segment where the tag ends
- Tag – atomic reference to CmPossibility in the Text Markup Tags list.

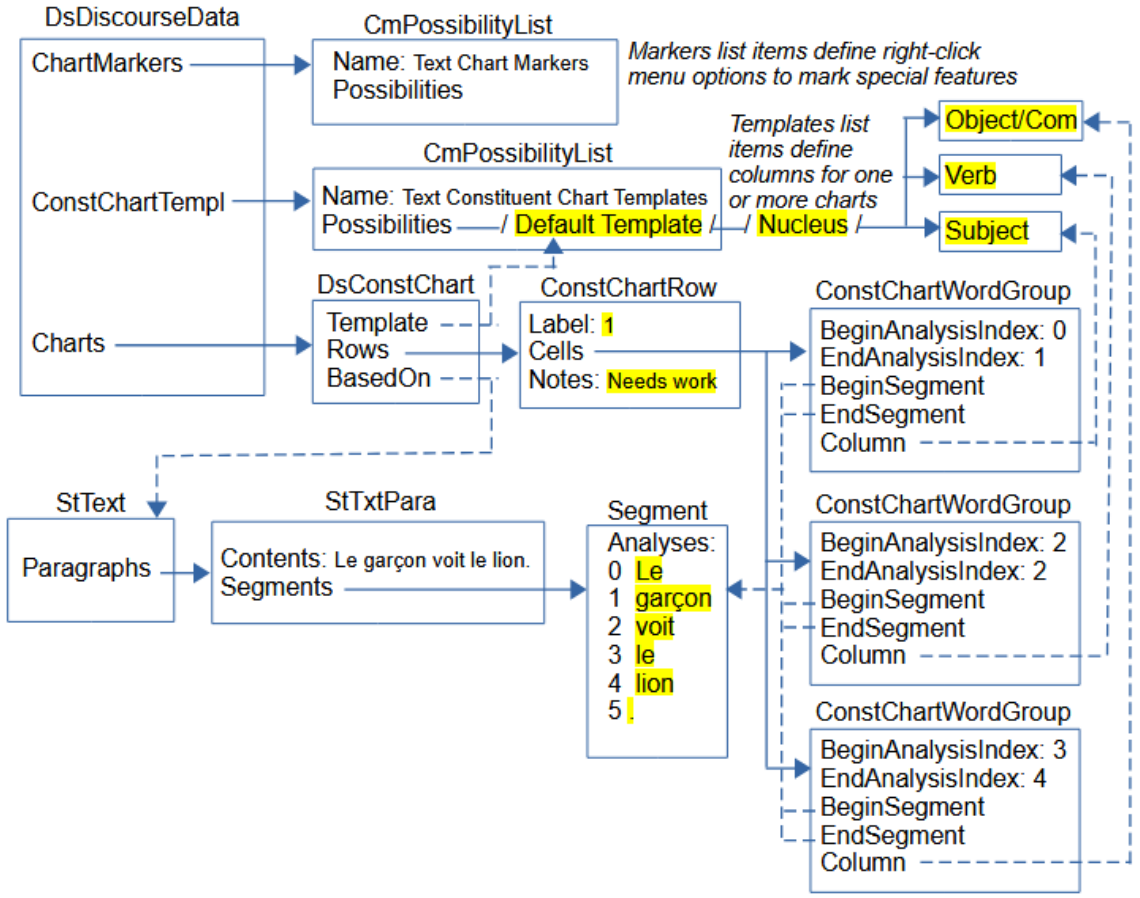
The 3 TextTag objects in this example all reference the same Segment. The Segment here is showing the wordforms following the index into the Analyses list. The top and bottom TextTag objects reference two subitems of the same top-level tag, so these two will be displayed on one line. The top one puts NP around the first two words. The bottom one puts VP around the last 3 words. The middle TextTag is shown on a separate line and puts SENT around all 5 words.

6.7 Discourse Charting

Interlinear text offers a Text Chart tab that allows a user to add analyses into a discourse chart. The chart structure and markers are in possibility lists that the user can modify to meet their need. Here's a simple example.



This is represented as follows in FLEx



Text charts are stored in DsDiscourseData, which is owned in the DiscourseData atomic owning property of LangProject. DsDiscourseData owns two CmPossibilityList objects.

- The “Text Constituent Chart Templates” list is owned in the ConstChartTempl atomic owning property and defines the chart structure. This list is hierarchical. The top level is the name of a type of chart. The only one by default is “Default Template”. The second level defines the primary level of columns and headers (Pre-nuclear, Nucleus, and Post-nuclear) separated by black lines. The third level defines secondary columns and headers. (e.g., Nucleus has Subject, Verb, and Object/Complement items) separated by gray lines.
- The “Text Chart Markers” list is owned in the ChartMarkers atomic owning property and defines submenus for the right-click menu within cells to mark special features. This list is

also hierarchical. The top level defines a right-click subitem. The defaults are TenseAspectMood, Pronouns, and Demonstratives. Lower levels provide actual markers.

DsDiscourseData also has a Charts owning collection that owns DsChart objects. DsChart is a subclass of CmMajorObject, and both of these are abstract. The only actual subclass that is available is DsConstChart.

DsConstChart has

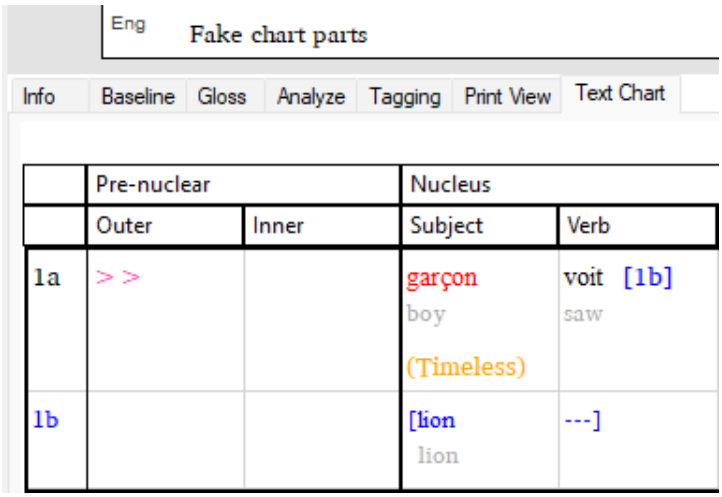
- Template – atomic reference to CmPossibility. This is one of the top possibilities in the "Text Constituent Chart Templates" list. In this example, it is the "Default Template" item. This is the item that defines the columns for this chart.
- BasedOn – atomic reference to StText. This is owned by the Text being charted.
- Rows – owning sequence of ConstChartRow objects. There is one ConstChartRow object for each row in the chart. A row corresponds to a StTextPara and is shown in black lines and numbers on the left designating the paragraph number.

ConstChartRow has

- Label – String containing the paragraph number and possible letters when there are sub-rows within the row.
- Cells – owning sequence of ConstituentChartCellPart which is abstract. Possibilities include
  - ConstChartClauseMarker – controls dependent clauses.
  - ConstChartMovedTextMarker – allows word groups to be moved to a different location.
  - ConstChartTag – inserts a tag before or after a column
  - ConstChartWordGroup – groups 1 or more Wordforms into a cell.
- Notes – String for compiler notes.

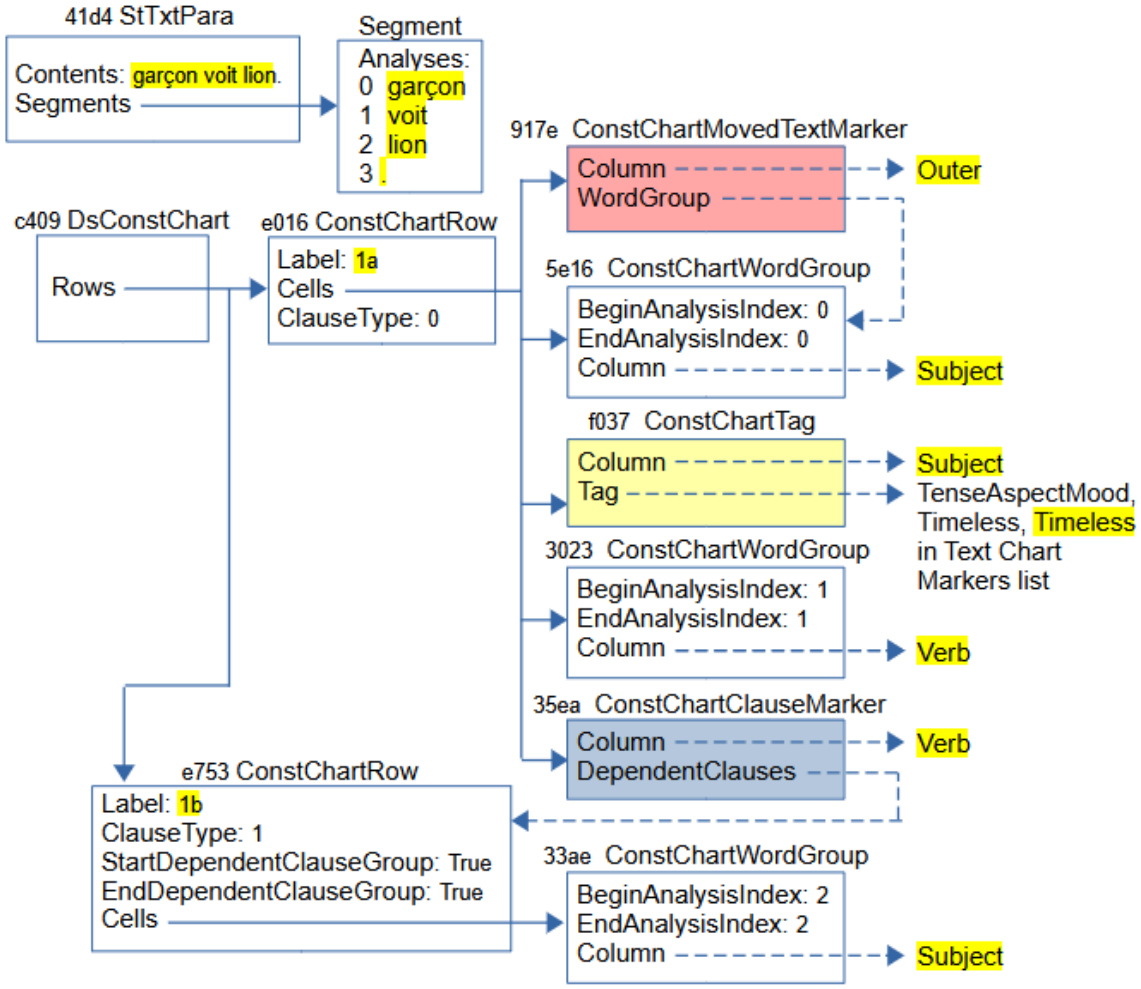
In our simple example, DsConstChart uses the "Default Template" for laying out columns, it's based on the Test 1 StText, and there is only one ConstChartRow. The label for the row is 1 because it's the first paragraph, and Notes has "Needs work" that appears in a Notes column on the right. In the diagram, the first two levels of CmPossibility in the templates list are included in the owning line to simplify the diagram. The Segment is a simplification showing the wordforms for the 5 analyses. There are 3 cells in this example, each one is a ConstChartWordGroup and is owned in the Cells property of ConstChartRow. All 3 are in one Segment, so BeginSegment and EndSegment all reference this Segment. The first cell references the 'Subject' secondary column under 'Nucleus' and includes the first 2 analyses (0 Lu, and 1 garçon). The second cell references the 'Verb' secondary column under 'Nucleus' and includes the 3rd analysis (2 voit). The third cell references the 'Object/Complement' secondary column under 'Nucleus' and includes the last 2 analyses (3 le, and 4 lion). The Word and Word Gloss fields are shown for each analysis.

A "Fake chart parts" example text that makes no linguistic sense, but demonstrates the remaining ConstituentChartCellPart classes looks like this in the sample project.



This chart was created with baseline ‘garçon voit lion.’, and bringing ‘garçon’ into the Subject column, ‘voit’ into the Verb column, and ‘lion’ into the Subject column. Right-click ‘garçon’ and choose Mark as Postposed from > Outer. Right-click ‘garçon’ again and choose Mark TenseAspectMood > Timeless > Timeless. Right-click ‘voit’ and choose Make a Dependent Clause out of > Next Clause.

This is represented as follows in FLEx



This diagram omits various details that were included in the previous diagram to make it easier to understand.

(pink) When you right-click a cell and choose “Mark as Preposed from” or “Mark as Postposed from”, and choose a target column, a ConstChartMovedTextMarker object (e.g., 917e) is placed in the target column. It has a Column property that references the target column CmPossibility, and a WordGroup property that references the source word group from which you gave the command. FLEx adds two wedges in red in the target cell, and it shows the source cell in red.

(yellow) When you right-click a cell and choose one of the last 3 menu items (Mark TenseAspectMood, Mark Pronouns, or Mark Demonstratives) and pick a subitem, a ConstChartTag object (e.g. f037) is inserted in the current cell. It has a Column property that references the current column CmPossibility, and a Tag property that references the selected CmPossibility from the Text Chart Markers list. FLEx adds the CmPossibility name in parentheses in a different color (e.g., Timeless).

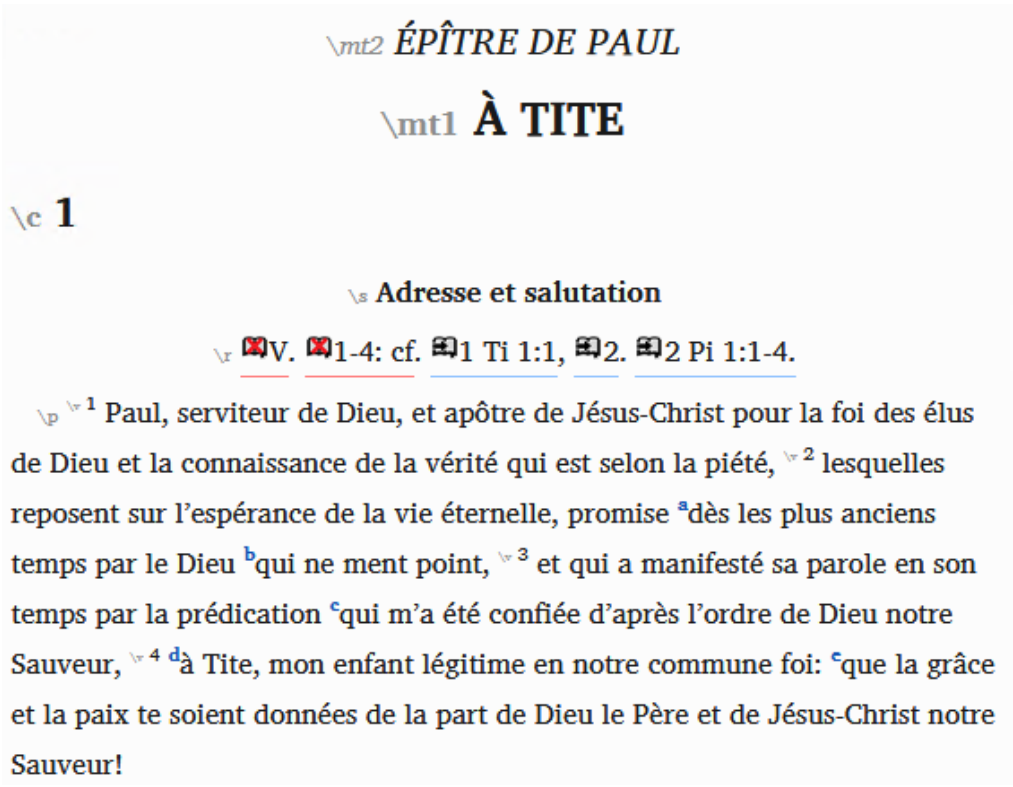
(blue) Clauses are in a sub-row by themselves. When you right-click a cell in one sub-row, you can choose one of the ‘Make a <Type> Clause out of’ menu items, you can choose the next or previous clause, a ConstChartClauseMarker object (e.g., 34ea) is inserted in the source cell. It has a Column property referencing a source column CmPossibililty, and a DependentClauses

property that references the ConstChartRow that contains the clause. FLEx puts square brackets around the clause, and in the source cell it adds the target row label in square brackets.

6.8 Scripture

FieldWorks used to have a Translation Editor program for working on Scripture. Development on TE was stopped when we chose to help develop and work with Paratext for Scripture editing. The last version of FieldWorks with TE was FW8.3.12. Although FieldWorks no longer allows editing scripture text, when a Paratext project is associated with a FLEx project, books from Paratext can be imported into FLEx. The data shows up as interlinear texts. Interlinear analysis can be done on imported scripture, but FLEx does not provide any capability for editing or deleting the imported scripture. If you need to remove scripture from a project, you can use LCM Browser to delete the ScrBook objects (see section 3.3).

This is how a section of Titus from the LS1910 French public domain Bible looks in Paratext.


\mt2 *ÉPÎTRE DE PAUL*

\mt1 **À TITE**

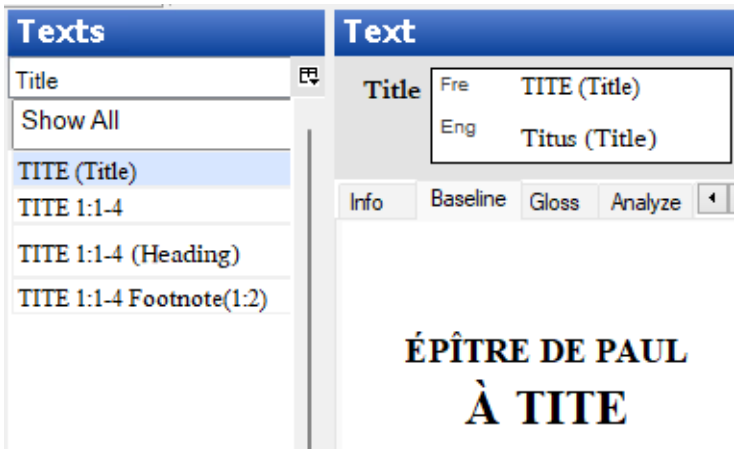
\c **1**

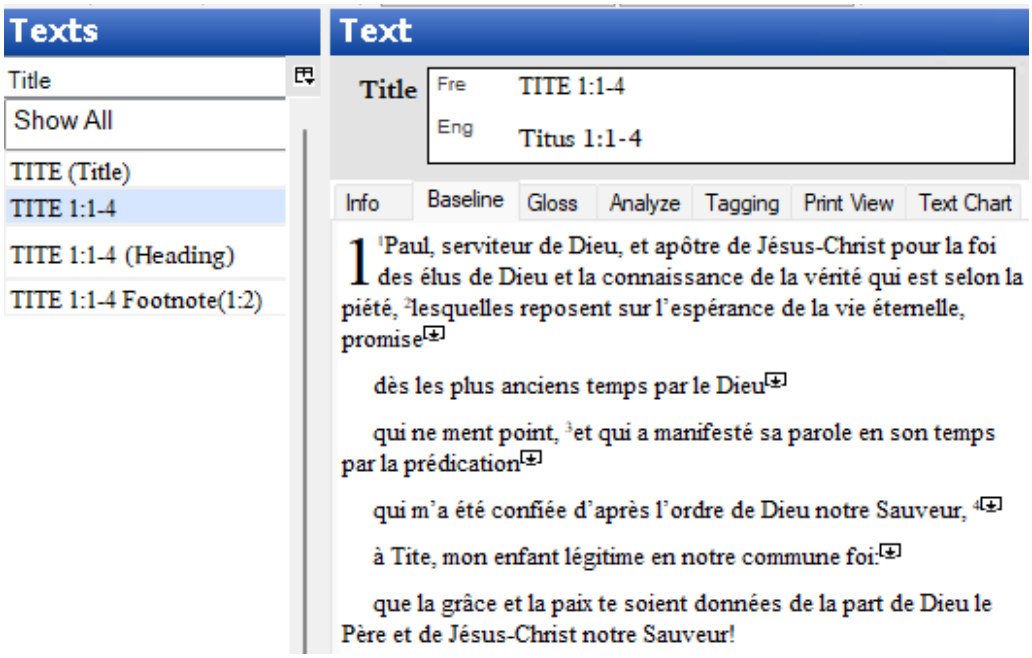
\s **Adresse et salutation**

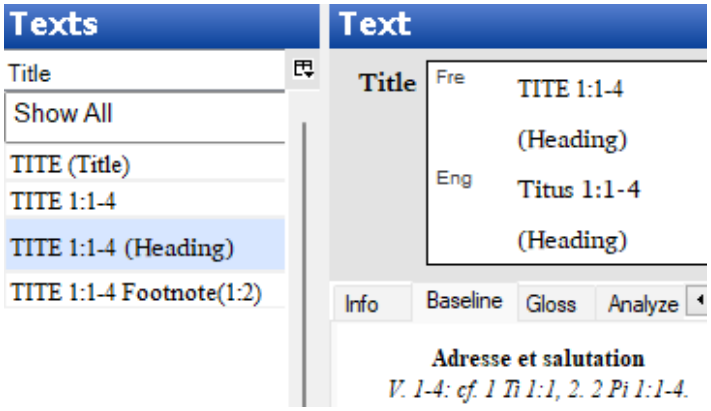
\r V. 1-4: cf. 1 Ti 1:1, 2. 2 Pi 1:1-4.

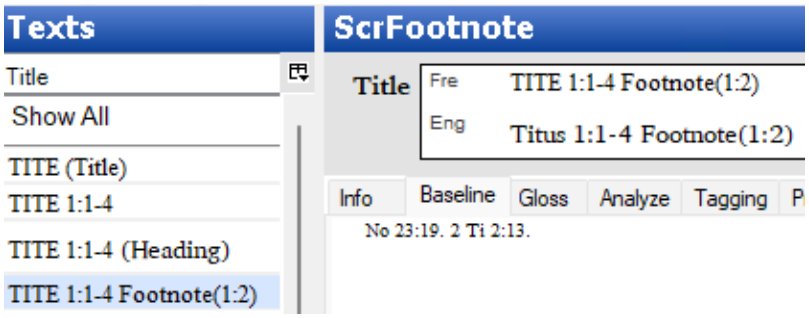
\p \v 1 Paul, serviteur de Dieu, et apôtre de Jésus-Christ pour la foi des élus de Dieu et la connaissance de la vérité qui est selon la piété, \v 2 lesquelles reposent sur l'espérance de la vie éternelle, promise [a]dès les plus anciens temps par le Dieu [b]qui ne ment point, \v 3 et qui a manifesté sa parole en son temps par la prédication [c]qui m'a été confiée d'après l'ordre de Dieu notre Sauveur, \v 4 [d]à Tite, mon enfant légitime en notre commune foi: [e]que la grâce et la paix te soient données de la part de Dieu le Père et de Jésus-Christ notre Sauveur!

This section shows up in FLEx as 4 Texts

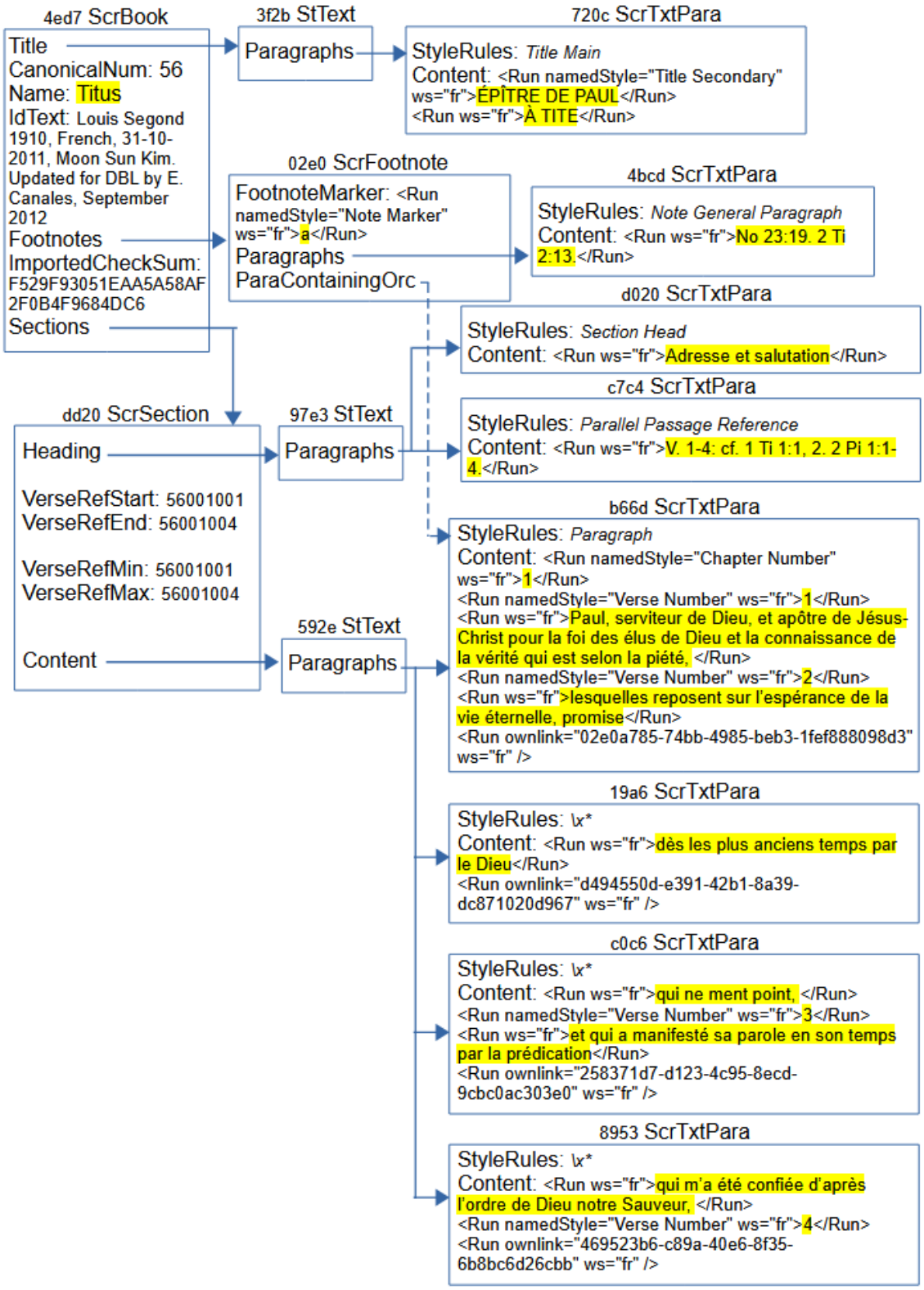

Texts
Title
Show All
TITE (Title)
TITE 1:1-4
TITE 1:1-4 (Heading)
TITE 1:1-4 Footnote(1:2)
Text
Title
Fre
TITE (Title)
Eng
Titus (Title)
Info
Baseline
Gloss
Analyze
ÉPÎTRE DE PAUL
À TITE


Texts
Title
Show All
TITE (Title)
TITE 1:1-4
TITE 1:1-4 (Heading)
TITE 1:1-4 Footnote(1:2)
Text
Title
Fre
TITE 1:1-4
Eng
Titus 1:1-4
Info
Baseline
Gloss
Analyze
Tagging
Print View
Text Chart
1 [1]Paul, serviteur de Dieu, et apôtre de Jésus-Christ pour la foi des élus de Dieu et la connaissance de la vérité qui est selon la piété, [2]lesquelles reposent sur l'espérance de la vie éternelle, promise
dès les plus anciens temps par le Dieu
qui ne ment point, [3]et qui a manifesté sa parole en son temps par la prédication
qui m'a été confiée d'après l'ordre de Dieu notre Sauveur, [4]
à Tite, mon enfant légitime en notre commune foi:
que la grâce et la paix te soient données de la part de Dieu le Père et de Jésus-Christ notre Sauveur!


Texts
Title
Show All
TITE (Title)
TITE 1:1-4
TITE 1:1-4 (Heading)
TITE 1:1-4 Footnote(1:2)
Text
Title
Fre
TITE 1:1-4
(Heading)
Eng
Titus 1:1-4
(Heading)
Info
Baseline
Gloss
Analyze
Adresse et salutation
V. 1-4: cf. 1 Ti 1:1, 2. 2 Pi 1:1-4.



This is how the data is represented in FLEx.



LangProject has a TranslatedScriptures atomic owning attribute that owns a Scripture object. Scripture has various properties that are no longer used, but it has an owning sequence of ScrBooks representing books of Scripture, each book in a ScrBook object. Scripture uses the

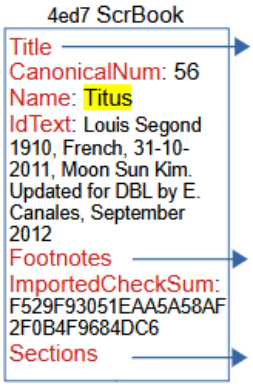
FLEx structured text classes starting with StText, but instead of StText owning StTxtPara objects, Scripture uses ScrTxtPara objects. In the Interlinear Texts tool, FLEx shows StText objects and ScrFootnote objects as interlinear texts. When showing StText, in the Interlinear Texts it

- appends (Title) for the StText owned in the ScrBook Title property
- appends (Heading) for the StText owned in the ScrSection Heading property
- it doesn't append anything for the StText owned in the ScrSection Content property
- it appends Footnote(c:v) for ScrFootnote with the chapter and verse number in parentheses.

The vernacular ScrBook Name (or Abbrev) is used for the book name, and for StTexts owned in ScrSection, it appends the chapter/verse range for the section.

ScrBook has

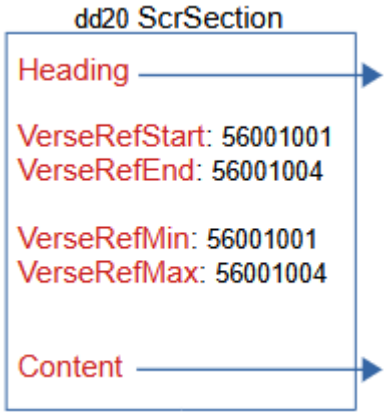
- CanonicalNum – Integer giving the book number in Scripture starting at 1 for Genesis. The number for Titus, which is used in this example is 56.
- Name – MultiUnicode name for the book, used in displaying the Title for FLEx interlinear text.
- IdText – Unicode string showing identifying information for the book.
- ImportedCheckSum – A calculated checksum of the Paratext book file that is stored when FLEx imports a book from Paratext. When the user goes to the Choose Texts tool and the FLEx project is associated with Paratext, and a ScrBook for this book is already in the FLEx project, then FLEx calculates a current checksum for the Paratext book file, and if it is different, it means the file has been edited in Paratext, so it reimports the book from Paratext and stores the new checksum. When reimporting, FLEx attempts to maintain any interlinearization a user did in FLEx.
- Title – atomic owning property holding a StText with the title for the printed book.
- Footnotes – an owning sequence of ScrFootnote holding all of the footnotes for the book.
- Sections – an owning sequence of ScrSection, which usually includes a section heading and all of the verses until a new chapter or section head occurs.



ScrTxtPara objects use a StyleRules property to store paragraph style information in a namedStyle prop element giving the name of the style for the paragraph. The Contents String property includes namedStyle attributes with the name of special styles designating a "Chapter Number", "Verse Number", "Title Secondary", etc.

ScrSection has

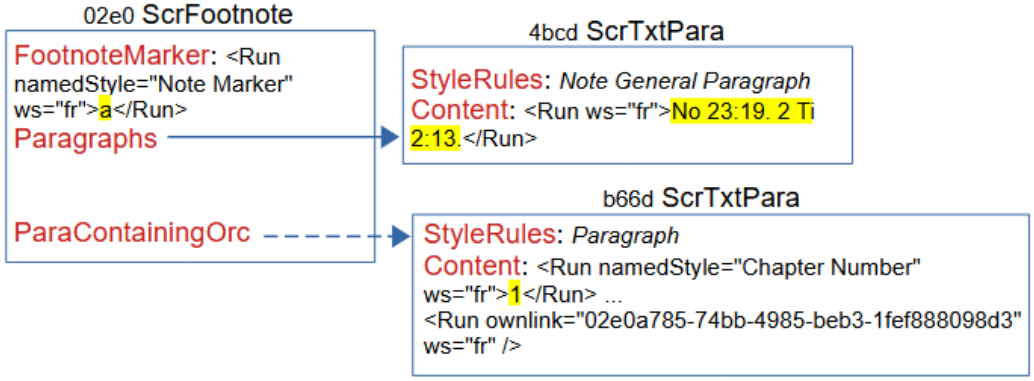
- VerseRefStart – an Integer representing the book, chapter, and verse that starts this section in the form BBCCCVVV.
- VerseRefEnd – an Integer representing the book, chapter, and verse that ends this section in the form BBCCCVVV.
- Heading – atomic owning property that holds an StText for the section heading.
- Content – atomic owning property that holds an StText for the rest of the paragraphs in a section.



Footnote text is stored in ScrFootnote objects. They are owned in the Footnotes property of ScrBook.

ScrFootnote has

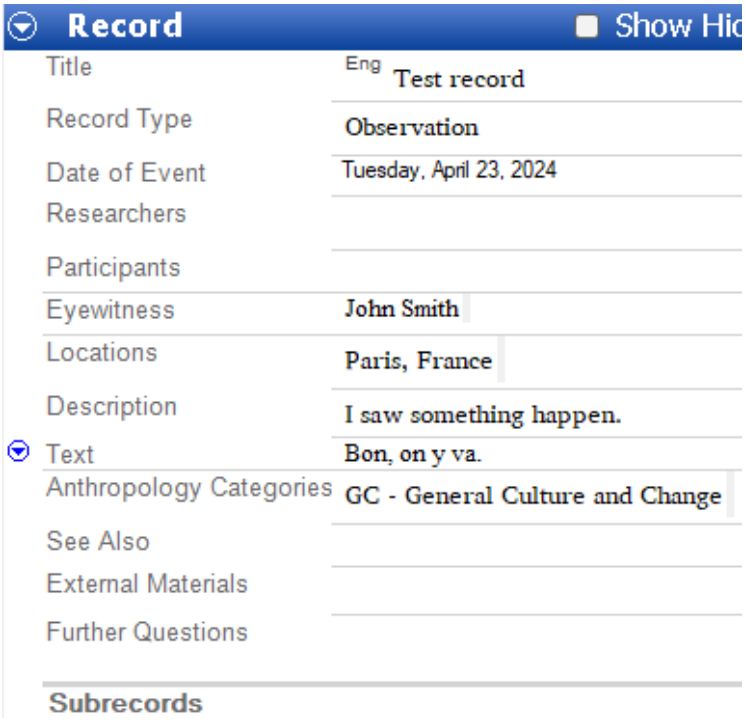
- FootnoteMarker – a String with the footnote marker in a “Note Marker” style.
- ParaContainingOrc – an atomic reference to ScrTxtPara that references the paragraph that contains the footnote. In the paragraph, the footnote is represented by an ownlink attribute in the Run with the guid of the ScrFootnote.
- Paragraphs – an owning sequence of ScrTxtPara that contains the content of the footnote.



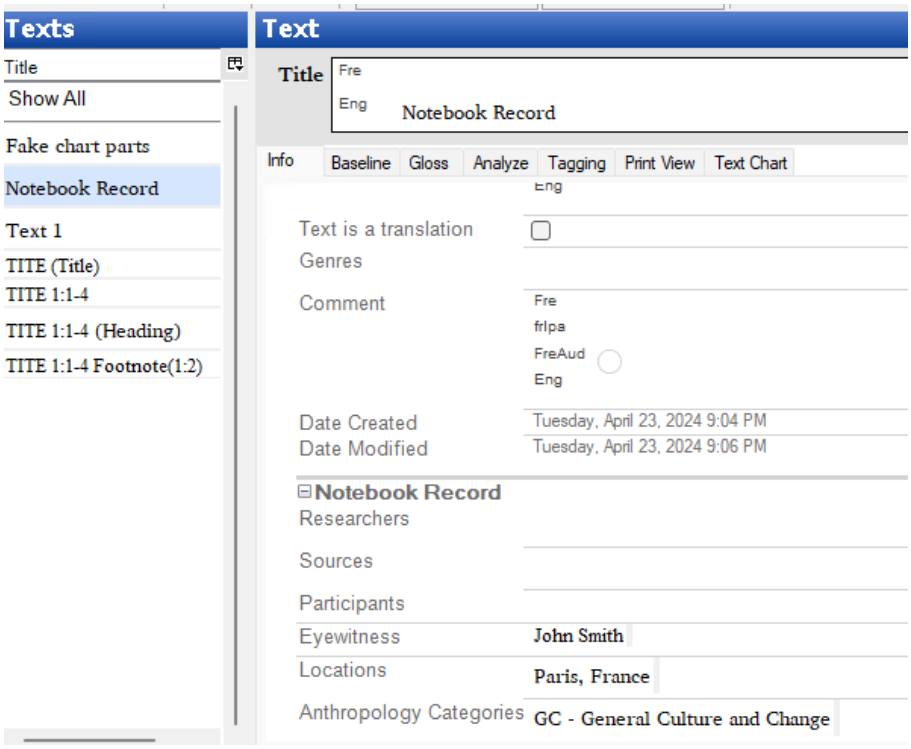
7 Notebook model

FLEx has a Notebook area that allows a user to record information relating to the culture. Most of the fields in the Notebook are Multi-Paragraph fields with styles for paragraphs and text. A notebook record can reference a text that is an actual Text in Interlinear Texts.

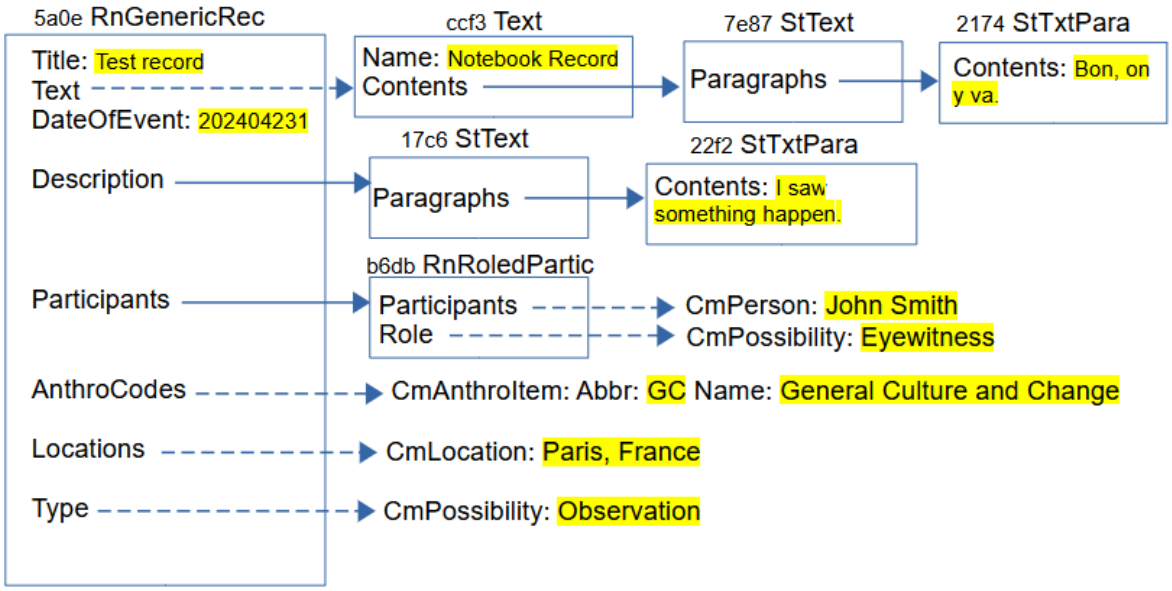
Here's a sample record with a few fields filled in.



And this is how it looks from the linked interlinear text.



This is represented in FLEx as



RnResearchNbk is owned in LangProject in atomic owning ResearchNotebook property. RnReserachNbk has a Records owning collection of RnGenericRec which includes the properties shown in this example, plus other ones including custom fields. The Title is a String. A RnGenericRec can be linked to a Text that shows up in Interlinear Text. It does this through the atomic reference Text property. The view in the Notebook just shows the baseline text. The view in the Interlinear Text Info tab shows some of the information from the Notebook record.

The Description is an owning atomic property holding a StText, which owns any number of StTxtPara objects that can have paragraph and character styles. The Participants field is an owning collection of RnRoledPartic. RnRoledPartic has reference collection Participants property referencing CmPerson objects from the People list. It also has a Role atomic reference to CmPossibility from the Roles list. RnGenericRec has an AnthroCodes reference collection to CmAnthroItem. The display typically shows the abbreviation and name of the anthro items. RnGenericRec has a Locations reference collection to CmLocation objects in the Locations list. It also has an atomic reference Type property referencing one of the CmPossibility items in the Notebook Record Types list.