SFM import into existing data

June 17, 2025
Ken Zook

Contents

1 Introduction

SFM (Standard Format Marker) import into FLEx was designed to import a full lexical database from a single file into a FLEx project without any existing entries. For background information that is somewhat out of date, see http://downloads.languagetechnology.org/fieldworks/Documentation/Technical_Notes_on_LinguaLinks_Database_Import.pdf.

The SFM file being imported has all of the entries in the project, and various fields for linking entries and senses within the file for subentries, variants, main entries, lexical relations, cross references, links to grammatical categories, semantic domains, and other lists, reversal entries, pictures, and sound files. During the process, entries are loaded into FLEx with textual Link elements to objects outside the current entry. The Link elements include a name for list items, and target import ID codes to other objects used in the import process. In phase 5 of the import process, FLEx attempts to turn these textual links into actual links to other entries, senses, lists, reversals, etc. This process goes through all entries in the project, making appropriate links, merging variants and subentries with minor entries, etc. This is essential for an initial import to establish all of the links necessary for FLEx to function.

Various things happen in phase 5 when textual links cannot be turned into actual links. If the field is a reference to grammatical categories or lists, a new item will normally be added to the list and the link is made to this new item. When links are to entries or senses that cannot be located, an option in the import dialog choses whether the program should leave a message in ImportResidue saying it could not make the link, or it should create a new entry and/or sense so that the link can be made.

It's possible to import additional entries from an SFM file at a later time, but there are a number of issues that need to be considered. This document explains issues to consider before doing any additional SFM imports after the initial one. Note that you cannot remove any existing data when you do a following import. You are just adding new information to the existing project.

Caution!! SFM imports fail in various ways, especially in Phase 5, where the data is not structurally sound. When this happens a virtually meaningless error message is given during phase 5, and the import is aborted at that point. The new entries are in place at this point, but all links following the failed link are skipped, which may include grammatical category links, variant and subentry links, etc. At this point your data is in a state with missing links that would hard to correct. If this happens, you should delete the project and start over again. To protect yourself against this, you should always make a backup prior to an SFM import to an existing project to make sure you can get back to your original state if anything goes wrong.

2 Owned objects

All owned objects from the SFM file including entries, senses, examples, etc. will create new objects in the import process. No attempt is made to merge anything with existing entries. If you import a lexeme form that already matches an entry in FLEx, a new entry with a new homograph will be created. If you want to merge data with existing data, you should use LIFT, which provides this capability if done properly. For details, see https://downloads.languagetechnology.org/fieldworks/Documentation/Technical%20Notes%20on%20LIFT%20used%20in%20FLEx.pdf.

In summary, you can add new entries to an existing project without affecting existing entries other than possible homographs as long as you are not referencing existing entries.

3 Links to grammatical categories and lists

References to list items (e.g., \ps noun) works the same way it would with an initial import. When a match is found in the list, the object is linked to that item. If a match in the list is not found, a new item is added to the list (except for closed lists as in morpheme types), and the object is linked to the new item.

In summary, you can add references to lists with additional imports without any problems.

4 Lexical relation links

In FLEx, lexical relations (senses) and cross references (entries) are implemented as LexReference objects for a given reference type (e.g., synonyms, antonyms, etc.) that link to multiple entries or senses. In the initial import, if you have 3 senses that are synonyms, there will be one Synonym LexReference object linked to the 3 senses.

Importing this into an empty project

```
\lx cat
\ps n
\sy puss
\sy feline
```

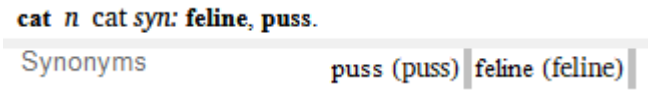
gives the desired dictionary view:

cat *n* cat *syn:* **feline**, **puss**.

feline feline *syn:* **cat**, **puss**.

puss puss *syn:* **cat**, **feline**.

The Lexicon Edit view shows this for ‘cat’.



There is one LexReference that references ‘cat’, ‘puss’, and ‘feline’.

Later on, if we import this to add ‘pussy’ into the existing project

```
\lx pussy
\ps n
\sy cat
```

the result is not what we wanted. The dictionary view is now:

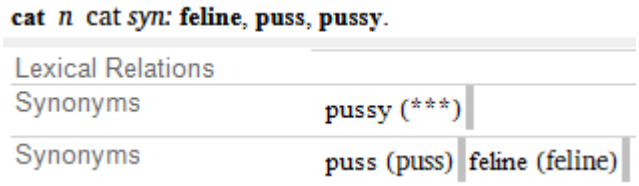
cat *n* cat *syn:* **feline**, **puss**, **pussy**.

feline feline *syn:* **cat**, **puss**.

puss puss *syn:* **cat**, **feline**.

pussy *n* pussy *syn:* **cat**.

The ‘cat’ entry is correct because it shows ‘pussy’ along with the other synonyms. However, ‘pussy’ is not showing up in ‘feline’ and ‘puss’ entries, and the ‘pussy’ entry is not showing ‘puss’ and ‘feline’. This is what you’ll see in the Lexicon Edit view for ‘cat’.



The import process is not designed for this additional import. Instead of adding ‘pussy’ to the original set of synonyms, it just creates a new set for this import giving us two sets. The original LexReference referencing ‘cat’, ‘puss’, and ‘feline’ is unchanged, but a new LexReference has been added referencing ‘pussy’ and ‘cat’.

Instead of using the last import that just added the ‘cat’ synonym for ‘pussy’, if we try to include all of the synonyms for ‘pussy’ like this,

```
\lx pussy
\ps n
\sy cat
\sy puss
\sy feline
```

The resulting dictionary view is even worse.

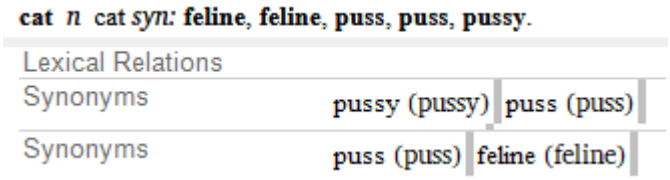
cat *n* cat *syn:* **feline**, **feline**, **puss**, **puss**, **pussy**.

feline feline *syn:* **cat**, **cat**, **puss**, **puss**, **pussy**.

puss puss *syn:* **cat**, **cat**, **feline**, **feline**, **pussy**.

pussy *n* pussy *syn:* **cat**, **feline**, **puss**.

This is what shows in Lexicon Edit for ‘cat’.



Again, the original LexReference referencing ‘cat’, ‘puss’, and ‘feline’ is unchanged, but a new LexReference has been added referencing ‘pussy’, ‘cat’, ‘puss’, and ‘feline’.

In summary, it’s not wise to attempt to add new lexical references in later imports. Later imports will not damage existing lexical relations, but will add new relations that are probably not what you wanted.

5 Reversal links

In FLEx, reversal indexes are separate objects that hold ReversalIndexEntry objects in a target language that references senses in the lexicon. Importing this SFM file in a blank project

```
\lx olmo
\ps n
\ge elm
\re elm
\re tree
\lx roble
\ps n
\ge oak
\re oak
\re tree
```

results in an ‘elm’ reversal entry referencing the ‘olmo’ sense, an ‘oak’ reversal entry referencing the ‘roble’ sense, and a ‘tree’ reversal entry referencing the ‘olmo’ and ‘roble’ senses. The English reversal index shows this:

elm *olmo*

oak *roble*

tree *olmo*; *roble*

Later, if you import this SFM file

```
\lx pino
\ps n
\ge pine
\re pine
\re tree
```

it will add a ‘pine’ reversal entry referencing ‘pino’, and it will add ‘pino’ to the existing ‘tree’ reversal entry.

In summary, reversal indexes work fine in later SFM imports.

6 Variant links

In FLEx, a variant entry is a normal entry that includes a LexEntryRef object in the EntryRefs property that indicates the type of variant, and one or more main entries for the variant, and a flag to indicate if the variant should show as a minor entry in the dictionary view.

The minimal way to import a variant is with this SFM file:

```
\lx main
\va variant
```

After import, there will be two entries in the lexicon; a ‘main’ entry, and a ‘variant’ entry that is linked to the main entry via a LexEntryRef. In the dictionary view, the main entry indicates that it has a variant (unspecified type by default), and the ‘variant’ entry is shown as a minor entry that indicates it is a variant of ‘main’, so the user would look in the ‘main’ entry to find its semantic information.

main (unspec. var. **variant**)
variant unspec. var. of **main**

Often, an SFM dictionary will include the minor variant entries for many variants, since that was the only way to get the minor entries to show in the printed lexicon. The SFM file in this case would be:

```
\lx main
\va variant
\lx variant
\mn main
```

where \va indicates the headword for the variant entry, and \mn indicates the main entry for the variant entry. The result on import would be:

main (unspec. var. **variant**)
variant unspec. var. of **main**

It ends up the same in FLEx. During an import, when a \va is listed, FLEx will automatically create a ‘variant’ minor entry. It then creates a second ‘variant’ entry when it encounters the \lx variant. During phase 5 of the import process, after importing these two variant entries, it notices two variant entries with the same form, and if the \mn field matches the \lx field with a \va field that matches the variant entry, it merges these two variant entries into a single variant entry. When looking for variants, the import process uses the import Link ID rather than the actual headword. As a result, it will never link to an existing entry in the lexicon prior to import. These IDs can be seen in “%temp%\Language Explorer” following an SFM import.

SFM does not support multiple \mn fields in a minor variant entry, even though internally, a variant can have multiple main entries.

After an initial import, if you try to add a new variant to an existing main entry using this:

```
\lx main
\va varianta
```

The resulting dictionary is:

main$_1$ (unspec. var. **variant**)

main$_2$ (unspec. var. **varianta**)

variant unspec. var. of **main$_1$**

varianta unspec. var. of **main$_2$**

The new entries are totally new entries that are linked to each other, but make no connections to existing entries and variants in the lexicon. However, since there are now two ‘main’ entries, they have homograph numbers to distinguish them. With this import, however, it would be simple to move ‘varianta’ to the original ‘main$_1$’. If you go to ‘varianta’ in Lexicon Edit, you could change the ‘Variant of’ field to main$_1$, and then delete main$_2$.

If you tried to connect a second main entry to ‘variant’ using:

```
\lx variant
\mn maina
```

The result would be:

main$_1$ (unspec. var. **variant**)

main$_2$ (unspec. var. **varianta**)

variant unspec. var. of **main$_1$**

varianta unspec. var. of **main$_2$**

The results are the same as the previous example. The imported entries are new entries and the import does not affect the original entries.

In summary, main entries and variants can be imported in a later import, but the only variant links that occur are ones between entries included in the new SFM file. None of the links affect existing entries.

7 Subentry links

In FLEx, a subentry is a normal entry that includes a LexEntryRef object in the EntryRefs property that indicates the type of subentry (complex form), and one or more main entries for the subentry, and a flag to indicate if the subentry should show as a minor entry in the Root-based dictionary view. For a Stem-based dictionary, a minor entry for a subentry is shown if the “Show As Headword In” field includes the Publication (‘Main Dictionary’ by default) being displayed.

The minimal way to import a subentry is with this SFM entry:

```
\lx main
\se subentry
```

After import, there will be two entries in the lexicon; a ‘main’ entry, and a ‘subentry’ entry that is linked to the main entry via a LexEntryRef. In the Root-based dictionary view, the main entry displays the subentry body indented below it. A minor ‘subentry’ entry is displayed that includes the type (unspecified type by default), and it indicates that it is a subentry of ‘main’, so the user would look in the ‘main’ entry to see the body of the subentry for its semantic information.

main
subentry unspec. comp. form

subentry (unspec. comp. form of **main**)

Often, an SFM dictionary will include the minor subentry entries for many subentries, since that was the only way to get the minor entries to show in the printed lexicon. The SFM file in this case would be:

\lx main
\se variant
\lx subentry
\mn main

where \se contains the headword for the subentry entry, and \mn contains the main entry for the subentry entry. The result on import would be:

main
subentry unspec. comp. form

subentry (unspec. comp. form of **main**)

It ends up the same in FLEx. During an import, when a \se field is encountered, FLEx will automatically create a 'subentry' subentry entry. It then creates a second 'subentry' entry when it encounters the \lx subentry. During phase 5 of the import process, after importing these two subentry entries, it notices two subentry entries with the same form, and if the \mn field matches the \lx field with a \se field that matches the subentry entry, it merges these two subentry entries into a single subentry entry. When looking for subentries, the import process uses the import Link ID rather than the actual headword. As a result, it will never link to an existing entry in the lexicon prior to import. These IDs can be seen in "%temp%\Language Explorer" following an SFM import.

SFM does not support multiple \mn fields in a minor subentry entry, even though internally, a subentry can have multiple main entries.

After an initial import, if we try to import 'subentrya' as an additional subentry to 'main' using this file:

\lx main
\se subentrya

The resulting dictionary is:

main$_1$
subentry unspec. comp. form

main$_2$
subentrya unspec. comp. form

subentry (unspec. comp. form of **main**$_1$)

subentrya (unspec. comp. form of **main**$_2$)

The new entries are totally new entries that are linked to each other, but make no connections to existing entries and subentries in the lexicon. However, since there are now two 'main' entries, they have homograph numbers to distinguish them. With this import, however, it would be simple to move 'subentrya' to the original main$_1$. If you go

to ‘subentrya’ in Lexicon Edit, you could change the Components field to ‘main$_{1}$’, and then delete ‘main$_{2}$’.

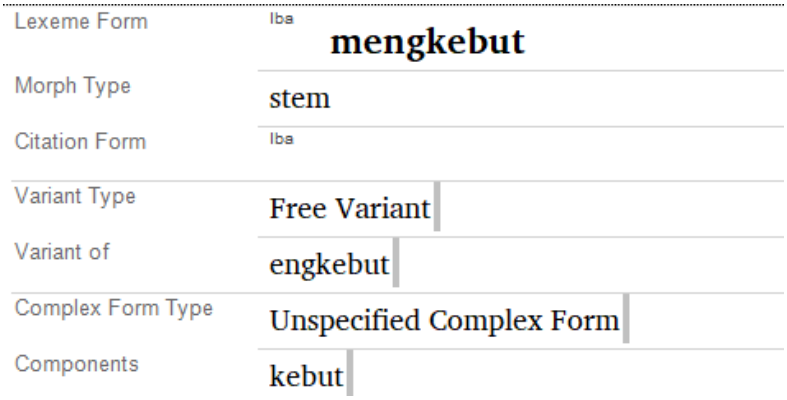
In summary, main entries and subentries can be imported in a later import, but the only subentry links that occur are ones between entries included in the new SFM file. None of the links affect existing entries.

8 Other issues

There is only one somewhat surprising result of importing into an existing project. It may result in some new senses being added to the project. This is described in the next section.

8.1 Missing senses may be created.

Even if you only import one headword, Phase 5 processing will add a missing sense linked to a missing MoStemMsa to every entry that has a variant and component, and is missing a sense as in this situation.



Here’s an example of the added Sense and Msa from the fwdata file:

```
<rt class="MoStemMsa" guid="020adda0-5ca3-4411-869b-d07e7b7ad944"
ownerguid="2d72e126-f5cc-4725-ade6-dd5162217192" />
<rt class="LexSense" guid="02ba0dd6-6eeb-41b1-8488-fa1716445486"
ownerguid="2d72e126-f5cc-4725-ade6-dd5162217192">
<MorphoSyntaxAnalysis>
<objsur guid="020adda0-5ca3-4411-869b-d07e7b7ad944" t="r" />
</MorphoSyntaxAnalysis>
</rt>
```

Thus, after any import, you may find that multiple senses have been added to your project. This is probably good in that subentries normally have senses, although variants may not have senses, and in this case the new sense affects both the subentry and the variant.