**SFM import of variants and subentries**

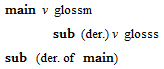
Ken Zook  
November 15, 2016

Understanding how the current (FW8.2.8 and some FW8.3.1 Beta notes) SFM import works with variants and subentries is very important to get desired results. There are numerous limitations with the current import that require special strategies (e.g. preprocessing) in order to have a successful import. Let’s look at various examples using root base view to see how the import process works at this point.

# Subentries

Example 1

\lx main  
\ps v  
\ge glossm  
\se sub  
\ps v  
\ge glosss

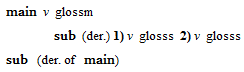


During the import process when we encounter a subentry (\se) we automatically create a second entry and link it as a subentry of the entry in which the \se occurs. So the default view will show it as a subentry of the main entry and will also show a minor entry pointing to the main entry. If you don’t want to include the minor entry for a given entry, you can uncheck the Show Minor Entry field in the minor entry. Note that FW8.3.1 now defaults to “Unspecified Complex Form” as the type if you do not specify something else. So you’ll normally want to change this during import setup. In these examples, I set it to Derivative, which is what earlier versions used by default.

Example 2

But often the SFM dictionary will already have a minor entry like this:

\lx main  
\ps v  
\ge glossm  
\se sub  
\ps v  
\ge glosss  
  
\lx sub  
\mn main  
\ps v  
\ge gloss



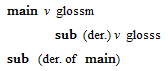
What happened during the import? Just like the first example, we process the \se and related fields as a new entry that is a subentry of the main entry. But as a result we have two ‘sub’ entries; the one we created for the \se, in addition to the minor entry that was already there. But we don’t want to leave the user with a bunch of extra entries in this process, so after the data is imported, we go through the results and when we find an existing minor entry, for the one we created, it combines the two into a single entry. It does this right now by appending the sense we created to the existing sense of the minor entry. Unfortunately, our current merge process doesn’t detect that the two senses are really identical. In this case is should have only had one sense. Appending a generated sense that is different from the existing would make sense.

For the merge to work properly between the subentry and the minor entry, the subentry \se field must be identical to the \lx field of the minor entry. Also, the minor entry must have a \mn field that matches the \lx field off the main entry. As long as these two reciprocal links are present, then Flex will recognize the two entries as equivalent and will merge the results.

Example 3

There are three ways you can work around the existing merge limitation that results in two identical senses. The first is to store the sense content in the minor entry and remove the sense fields following the \se.

\lx main  
\ps v  
\ge gloss  
\se sub  
  
\lx sub  
\mn main  
\ps v  
\ge glosss



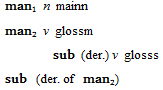
The result here is what we want. A second way to get the same result is to put all of the sense content following the \se and remove sense content from the minor entry.

\lx main  
\ps v  
\ge glossm  
\se sub  
\ps v  
\ge glosss  
  
\lx sub  
\mn main

A third way, would be to visit each minor entry after the import and use the Flex merge sense menu option.

Example 4

\lx main  
\hm 1  
\lc man  
\ps n  
\ge mainn  
  
\lx main  
\hm 2  
\lc man  
\ps v  
\ge glossm  
\se sub  
\ps v  
\ge glosss  
  
\lx sub  
\mn man2



Notice that the \mn field in the minor entry needs to specify the main entry correctly for the merge to work. First, if there is a citation form on the main entry, the \mn field should specify the citation form since that is what gets displayed in the final result. Also, when there are homographs, the homograph number should be specified as an appended number in the \mn field.

The above examples are all clean data which result in good imports. Without clean data the import will be a mess. You may see things like this:

\lx bre2  
\lx bre / bree / brap, etc.  
\lx ' , or q, glottal consonant  
\lx aban , abin (PC)  
\se housesit, see note in appendix  
\se 0

Where homograph numbers are appended to the \lx instead of using a \hm, more than one form may be included in the same field, or other commends and glosses. When you don’t have proper \lx \se \mn fields, Flex will do what it can, but the results will not be pretty.

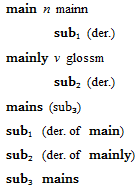
If a person has more than one subentry in a single \se field  
\se abarn, aban  
it would look for a \lx field with the same words. In most cases, that will not be found. Instead, the input needs to have each form in a separate \se field  
\se abarn  
\se aban

There are other issues where people have an \se in an entry with the same form as the \lx field that will cause problems.

Example 5

If there are multiple \se fields with the same form will not attempt to merge the results, so it creates a new minor entries with homographs . Also, if a \mn field refers to something that doesn’t exist, we create an entry as a main entry for the subentry. When this happens it defaults to a variant (see below).

\lx main  
\ps n  
\ge mainn  
\se sub  
  
\lx mainly  
\ps v  
\ge glossm  
\se sub  
  
\lx sub  
\mn mains



# Variants

Variants work much like subentries. Here’s a simple example.

Example 6

\lx main  
\va var  
\ps n  
\ge main



When Flex encounters a \va field, it creates a variant entry that points back to the main entry. Note that FW8.3.1 now defaults to “Unspecified Variant” as the type if you do not specify something else. So you’ll normally want to change this during import setup. In these examples, I set it to “Irregularly Inflected Form”, which is what earlier versions used by default.

Example 7

\lx main  
\va var  
\ps n  
\ge mainn  
  
\lx var  
\mn main  
\ps v  
\ge glossv

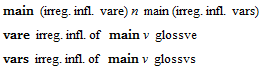


As with subentries, if there is a minor entry already present for the variant, in the process Flex creates a second variant entry, but then in the cleanup after the import, it recognizes that the variant entry already exists, so it merges the results into a single variant minor entry. The merge process depends on the \va field matching the minor \lx field, and the \mn field matching the \lx of the main entry. As with subentries.

Unlike subentries, variants can be imported to senses as well as the main entry. Any \va fields that occur before the first sense will be variants of the entry, while \va fields after a sense starts will be variants of the current sense. The \mn field for variants of senses should include the sense number.

Example 8

\lx main  
\va vare  
\ps n  
\ge main  
\va vars  
  
\lx vare  
\mn main  
\ps v  
\ge glossve  
  
\lx vars  
\mn main 1  
\ps v  
\ge glossvs



Example 9

Multiple variants, or subentries on the same line need to be broken into multiple fields. Each field should contain only one form that must match the citation or lexeme form of the target. For example, the following file would result in a variant with lexeme form “vara, varb” and a complex form with lexeme form “suba, sub).

\lx main  
\va vara, varb  
\ps n  
\ge main  
\se suba, subb

Example 10

Assuming these are two variants and two subentries, you would need to have this.

\lx main  
\va vara  
\va varb  
\ps n  
\ge main  
\se suba  
\se subb

Example 11

Suppose we have a \mn without a target, or with only a partial target.

\lx entry  
\mn main  
\ps n  
\ge entry

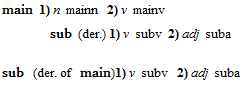


In this case, the \mn refers to an entry that doesn’t exist. So on import, Flex will create a ‘main’ entry to match the reference. The main entry will have this comment in Import Residue, “This was automatically created to satisfy a Components link, and it should be checked.” So you can filter on these after the import to find them. But \mn by itself is not enough information to know whether ‘entry’ is a variant or a subentry. If it had a corresponding \va in the main entry, then it would be clear that this is a variant. If it had a corresponding \se in the main entry, then it would be clear that it is a subentry. Since it is ambiguous at this point, the import process will assume it is a variant.

Flex will not import subentries of senses without special processing. Variants have limited fields, so it is possible for the import process to determine the end of a variant cluster, whether it’s in an entry or a sense. But subentries may have one or more full senses with anything that can go into a sense.

Example 12

\lx main  
\ps n  
\ge mainn  
\ps v  
\ge mainv  
\se sub  
\ps v  
\ge subv  
\ps adj  
\ge suba



In this case it is ambiguous whether the final ‘adj’ sense is part of \se sub, or part of \lx main. It’s not even clear whether the ‘v’ sense after the \se is part of the subentry or the main entry. Since it is ambiguous, Flex assumes that once you hit a \se field, everything following it is part of a subentry of the main entry until another \se field, or a \lx for a new entry. Because of this ambiguity, Flex will only import subentries on the main entry. Yet, internally it does support complex forms on senses as well as entries. So the problem is how to get subentries to import.

If the subentries on senses are not too frequent, you could manually move the subentry from a sense to the end of the entry prior to import, then find these entries after the import and reset the main entry to the original sense. You could even add some special field containing the sense number to these subentries that would allow you to filter on them after the import to simplify converting them to the desired sense.

In order to provide a reliable way to import subentries of senses, we would need a new SFM code to mark the end of a subentry (e.g., \see or \-se), thus making it clear where it ends, then the import process would need to be improved to interpret this end marker.

# Hack to import subentries of senses

With appropriate preprocessing of the SFM file, and post-processing the fwdata file after import, there is a way we can import subentries of senses. Suppose in example 12, the subentry really only had one sense, and it was a subentry of sense 2. Here’s how it can be handled. The first step is to pull the subentry out of the main entry, making it a minor entry referring to the second sense.

Example 13

\lx main  
\ps n  
\ge mainn  
\ps v  
\ge mainv  
\ps adj  
\ge maina  
  
\lx sub  
\mn main 2  
\ps v  
\ge subv



Notice what has actually happened is that Flex imported the minor entry as a variant of the second sense instead of a subentry of the second sense. If the SFM file already had a minor ‘sub’ entry, the preprocessing would need to combine this with the part that was pulled out of the main entry so that there would still be a single ‘sub’ minor entry referring back to the second sense.

FieldWorks before 8.3.1 Beta1

In the fwdata file the only significant distinction between a variant and a complex form is a RefType property on LexEntryRef. For a variant, this is 0 and for a complex form, this is 1. Here is the result of importing example 13 .

<rt class="LexEntryRef" guid="77c53630-35bb-4b96-bd2f-42adbcd3881a" ownerguid="7c145a75-1e74-4c5d-922a-802735e6f0c3">  
<ComponentLexemes>  
<objsur guid="656f3b06-2f77-4dba-a3d5-8061b116151a" t="r" />  
</ComponentLexemes>  
<HideMinorEntry val="0" />  
<RefType val="0" />  
</rt>

In addition to the RefType, a complex form normally has a ComplexEntryTypes property that points to the desired Complex Form Type. So here is the result to convert this minimal variant into a full complex entry instead of a variant.

<rt class="LexEntryRef" guid="77c53630-35bb-4b96-bd2f-42adbcd3881a" ownerguid="7c145a75-1e74-4c5d-922a-802735e6f0c3">  
<ComplexEntryTypes>  
<objsur guid="98c273c4-f723-4fb0-80df-eede2204dfca" t="r" />  
</ComplexEntryTypes>  
<ComponentLexemes>  
<objsur guid="656f3b06-2f77-4dba-a3d5-8061b116151a" t="r" />  
</ComponentLexemes>  
<HideMinorEntry val="0" />  
<RefType val="1" />  
</rt>

"98c273c4-f723-4fb0-80df-eede2204dfca" is the guid for the Derived form complex entry type. (You’ll want to verify you have the correct guid for what you want.) When modifying LexEntryRef objects, you’ll need to make sure you don’t modify fully formed variant or complex form objects. Normally after import those will have either a ComplexEntryTypes property or a VariantTypes property already filled in. The only ones without either of these would be entries that were created to fulfill a missing \mn reference. So for this to work, you’ll need to be careful of \mn fields that do not have valid entry/sense targets. If you keep track of the number of subentries you move into separate minor entries, you should be able to tell whether there are more empty LexEntryRefs. Note, Flex 8.3.0 is needed to get the sense subentries to display correctly.

FieldWorks 8.3.1 Beta1 and later

The import above will result in

<rt class="LexEntryRef" guid="73e7d909-00f2-47ce-88ff-ad1e9c510e22" ownerguid="b6c69b4a-d940-4157-81e2-123567db3d1f">  
<ComponentLexemes>  
<objsur guid="f82b0d5d-67fc-42d3-8e58-e04d2f76b211" t="r" />  
</ComponentLexemes>  
<HideMinorEntry val="0" />  
<RefType val="0" />  
<VariantEntryTypes>  
<objsur guid="3942addb-99fd-43e9-ab7d-99025ceb0d4e" t="r" />  
</VariantEntryTypes>  
</rt>

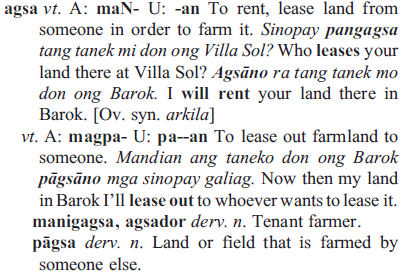
Notice here that even though we didn’t specify a variant type, the VariantEntryTypes is automatically filled in during the import. The 3942… guid in this case refers to the “Unspecified Variant”. Even for normal variants, if you do not set the variant type for the \va field, it will default to “Unspecified Variant”. So there would be no way to distinguish the real variants from the fake variants generated from \mn fields. To avoid this dilemma, make sure you specify a real variant for the \va field during imports so that it will have a different guid in the VariantEntryTypes field. Then your search for the fake ones would be to find a LexEntryRef with a VariantEntyTypes field referring to the “Unspecified Variant”. These would be the ones you want to convert. But again, you need to make sure every real \mn will match a \va or \se field in the main entries. If not, these will generate the same type of fake variants.

# Agutaynen examples

Example 14

Consider the Agutaynen “agsa” entry.

\lx agsa  
\ps vt  
\oc A:  
\oi maN-  
\oc U:  
\oi -an  
\gl To rent, lease land from someone in order to farm it.  
\ie lease farmland  
\ie tenant farming  
\ex Sinopay +pangagsa tang tanek mi don ong Villa Sol?  
\tr Who +leases your land there at Villa Sol?  
\ex +Agsāno ra tang tanek mo don ong Barok.  
\tr I +will +rent your land there in Barok.  
\ro arkila  
\ps vt  
\oc A:  
\oi magpa-  
\oc U:  
\oi pa--an  
\gl To lease out farmland to someone.  
\ex Mandian ang taneko don ong Barok +pāgsāno mga sinopay galiag.  
\tr Now then my land in Barok I'll +lease +out to whoever wants to lease it.  
\ld manigagsa, agsador  
\ps n  
\co Tenant farmer.  
\ld pāgsa  
\ps n  
\co Land or field that is farmed by someone else.



It has 2 \subentry fields, but the first one is really two separate subentries delimited by a comma, so they need to be split into 2 \ld fields, and in this case since they share the same \ps and \co fields, those fields would need to be duplicated into the second \ld. So we start out with this:

\ld manigagsa  
\ps n  
\co Tenant farmer.  
\ld agsador  
\ps n  
\co Tenant farmer.  
\ld pāgsa  
\ps n  
\co Land or field that is farmed by someone else.

But we already have minor entries for ‘agsador’ and ‘manigagsa’, but we have none for ‘pāgsa’.

\lx agsador  
\rf agsa  
\ps n  
\gl \*Tenant \*farmer.  
\rs manigagsa



\lx manigagsa  
\rf agsa 2  
\ad manig-  
\ps n  
\gl \*Tenant \*farmer.  
\rs agsador  
\re agsa



Agutaynen often has \ld fields at the end of senses, but since this is at the end of the entry, it’s not clear whether these subentries are on the full entry or on sense 2. Let’s assume these are really subsenses of sense 2. So what we need to merge the first 2 subentries into their corresponding minor entries, and convert the 3rd subentry to a minor entry. The result would be no \ld fields in the original ‘agsa’ entry, and these three minor entries.

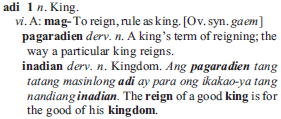
\lx agsador  
\rf agsa 2  
\ps n  
\gl \*Tenant \*farmer.  
\rs manigagsa  
  
\lx manigagsa  
\rf agsa 2  
\ad manig-  
\ps n  
\gl \*Tenant \*farmer.  
\rs agsador  
\re agsa  
  
\lx pāgsa  
\rf agsa 2  
\ps n  
\gl Land or field that is farmed by someone else.

Notice we have to add the 2 sense number to the first two \rf fields so they refer to the second sense. Since the rest of the data is already in the minor entries, we don’t need to do anything else with these entries. The 3rd subentry is turned into an \lx field, an appropriate \rf is added to link to the second sense, and the \co is turned into a \gl field. When this is imported, the 3 minor entries will become variants of ‘agsa’ sense 2. Then we need to convert these in the fwdata file to make them complex entries (subentries).

Example 15

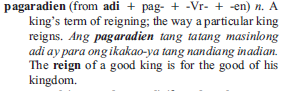
Consider sense 1 of the Agutaynen ‘adi’ entry.

\lx adi  
\ms 1  
\ps n  
\gl \*King.  
\g2 hari  
\ps vi  
\oc A:  
\oi mag-  
\gl To reign, rule as king.  
\ro gaem  
\ld pagaradien  
\ps n  
\co A king's term of reigning; the way a particular king reigns.  
\ld inadian  
\ps n  
\co Kingdom.  
\ex Ang +pagaradien tang tatang masinlong +adi ay para ong ikakao-ya tang nandiang +inadian.  
\tr The +reign of a good +king is for the good of his +kingdom.

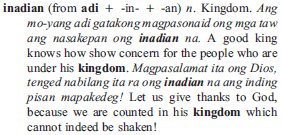


There are 2 \ld subentries. Both of these subentries have corresponding minor entries:

\lx pagaradien  
\rf adi  
\ad pag- + -Vr- + -en  
\ps n  
\gl A king's term of reigning; the way a particular king reigns.  
\ie reign of a king  
\ex Ang +pagaradien tang tatang masinlong adi ay para ong ikakao-ya  
tang nandiang inadian.  
\tr The +reign of a good king is for the good of his kingdom.



\lx inadian  
\rf adi  
\ad -in- + -an  
\ps n  
\gl \*Kingdom.  
\g2 kaharian  
\ex Ang mo-yang adi gatakong magpasonaid ong mga  
taw ang nasakepan ong +inadian na.  
\tr A good king knows how show concern for the people who  
are under his +kingdom.  
\ex Magpasalamat ita ong Dios, tenged nabilang ita ra  
ong +inadian na ang inding pisan mapakedeg!  
\tr Let us give thanks to God, because we are counted in his  
+kingdom which cannot indeed be shaken!



In Flex a subentry is just one view of an entry, and the minor entry is another view of the same entry. We have flexibility on what fields to show in each location, but we can’t select certain examples to show in one location and a different set in another location. And the selections apply to all subentries and minor entries. You can’t show some fields on some minor entries and other fields on other minor entries. So this example presents a problem since the examples in the subentry are different from the examples in the minor entry.

Assuming the linguist wants to maintain what they currently have in both places, we don’t have the power to show different examples in the two locations. So the only choice we have at this point is to format the body of the subentries into a single field that will be used only in subentries. Typically this is the summary definition field on entry. So when merging the \ld field into the minor entry, we would have something like this for the second minor entry where \sd is imported into the entry summary definition, and the subentry display would just show the summary definition.

\lx inadian  
\rf adi  
\ad -in- + -an  
\ps n  
\gl \*Kingdom.  
\g2 kaharian  
\ex Ang mo-yang adi gatakong magpasonaid ong mga  
taw ang nasakepan ong +inadian na.  
\tr A good king knows how show concern for the people who  
are under his +kingdom.  
\ex Magpasalamat ita ong Dios, tenged nabilang ita ra  
ong +inadian na ang inding pisan mapakedeg!  
\tr Let us give thanks to God, because we are counted in his  
+kingdom which cannot indeed be shaken!  
\sd |iderv. n.|r Kingdom. |ivAng|r |ivbpagaradien|r |ivtang tatang masinlong|r |ivbadi|r |ivay para ong ikakao-ya tang nandiang|r |ivbinadian|r. The |breign|r of a good |bking|r is for the good of his |bkingdom|r.

We may need to do the same thing for the other subentries so they are all consistent.

MORE WORK NEEDED:

Explain problems that cause the import to give useless error messagesw while trying to clean up after the import.

Discuss how to handle minor entries that have more than one main entry.