Flex tips

Ken Zook

April 19, 2016

Contents

1 Moving senses

There are two ways to move a sense to a new entry.

1. To move a sense to a new entry that is a homograph of the *current* entry, click the ⏷ widget next to the sense label and choose "Move Sense to a New Entry".
2. To move a sense *from* one entry to another, use Window…New Window to make a second Flex window, and in that window move to the destination entry.
   - In the first window, click a field label in the sense you want to move, such as Gloss or Definition, and drag the sense to a similar label on an existing sense in the new window (the mouse cursor will turn into a rectangle with an arrow).
   - Drop the sense in that location.
     Result: It will move it from the first entry to the destination entry, below the sense on which you dropped it.
   - To drag a sense into an entry that has *no* senses, create a dummy sense first, and then delete it after the move.

Tip: To move a sense *within* an entry, drag it as described earlier. You *cannot* reorder the senses by dragging a lower sense to the top sense. To do that, drag the top sense down. Alternatively you can click the ⏷ widget next to the sense label and choose "Move Sense Up", "Move Sense Down", "Demote", or "Promote".

2 Moving examples

- You can move an example to a *different* sense if the destination sense already has an example. To move the example, click the Example label and drag it to the Example label on the target sense and drop it when the mouse cursor turns into a rectangle with an arrow.
  Tip: If the target sense does not have an example, insert a dummy example and delete it afterward.
- Move examples *up* and *down* by clicking the ⏷ widget next to the Example label and choose "Move Example Up" or "Move Example Down".
- You can drag examples *within* a sense as described above.
  Tip: You cannot drag an example to the top of the list.

3 Startup crashes or frequent crashes

If Flex crashes every time you start a particular project, hold Shift down from the time you

A. double-click the Flex icon, or
B. click OK in the file open dialog until the program starts.

This is especially relevant if it happens shortly after upgrading to a new version. Occasionally the settings files get out of sync and cause various crashes. Shift causes the settings to get reset to factory defaults. See Settings files and registry.doc.

4 Custom detail view

You cannot reorder the fields in the detail views in Lexicon Edit and similar places. However, you can control when each field shows up. This allows you to focus on certain fields without the extra fields getting in the way. For each field, you can click the widget, choose FieldVisibility, and set one of the following options.

- Always visible (shows whether or not the field is empty)
- Normally hidden unless non-empty (only shows the field if there is data)
- Normally hidden (is hidden even if there is data)

The Show Hidden Fields button at the top of the screen shows all fields regardless of these settings.

5 Custom dictionary view

You can customize the dictionary view using Tools…Configure Dictionary. In this dialog you can enable, disable, or reorder the fields displayed for each object. Check the Help information in the Configure Dictionary View for additional help.

6 Custom interlinear view

You can customize the interlinear view using Tools…Configure Interlinear. The right pane shows the fields that are currently displayed. You can add and remove fields from this pane and reorder fields to change the interlinear layout. You can also add additional rows with the same field in a different writing system.

7 Startup icons

Flex and other FieldWorks programs have a few command line arguments that are useful in setting up desktop shortcuts for special purposes:

• -c	Computer name
• -db	Database name
• -locale	Windows locale code (en = English, zh-CN = Chinese Han, etc.)
• -? -h –help	Command line usage help.

When a FieldWorks program closes, it stores the name of the last database in the registry. The next time you start the program, it will open the same project. Each FieldWorks program uses its own registry setting for this. If you want, you can create special desktop shortcuts to override this default behavior on startup. For example, you could make a new desktop icon with the Target set to C:\Program Files\SIL\FieldWorks\Flex.exe -db German, and Start set to C:\Program Files\SIL\FieldWorks. When you launch Flex from this it will always open the German database regardless of which database you closed the last time.

By putting in an unused database name, you can force Flex to open to the “Welcome to FieldWorks” dialog that allows you to do things such as open a project, create a new project, and restore a project. Setting the Target to C:\Program Files\SIL\FieldWorks\Flex.exe -c ls-hovland\silfw -db "Sena 3" will open the Sena 3 database on the ls-hovland network machine.

8 Changing cursor movement

The default movement of the cursor in non-Graphite fonts jumps over a base character with its diacritics in one step. Typically this is what you want with standard European diacritics. However, when working in some Asian scripts, this becomes a problem when interlinearizing because you need to split the clump of characters into morphemes at times. For these languages, it is desirable to have the cursor move one code point at a time, even though cursor movement may not show in some cases. FieldWorks uses F7 to move left and F8 to move right to provide this character-by-character movement. If you want the regular arrow keys to work this way as well, set a registry setting to make the arrow keys go one character at a time. In HKEY_CURRENT_USER\Software\SIL\FieldWorks, set the “ArrowByCharacter” registry value to true for this mode. To simplify setting this value, double-click c:\Program Files\SIL\FieldWorks\Arrow by character.reg.

Some Graphite fonts, such as Doulos SIL and Charis SIL have a font feature that determines whether the cursor can show up between diacritics and base characters. You can change this feature in FieldWorks applications

1. Choose Format…Setup Writing Systems
2. With the desired writing system highlighted, click the Modify button.
3. Click the attributes tab.
4. Click the Font Features button to the right of the Graphite font and click Diacritic selection (last item) to toggle it.
5. Close the dialogs with OK.

The Graphite fonts distributed with FieldWorks 5.4 have this feature turned on by default, which means FieldWorks will move the cursor by code point rather than by character. Hopefully this default will be changed in future releases of these fonts. This feature overrides the ArrowByCharacter registry setting. If the Graphite Diacritic selection feature is off, then the ArrowByCharacter mode will work the same in Graphite and Uniscribe fonts.

9 Hiding the splash screen

In some sensitive areas, users do not want the SIL splash screen to come up when the program is launched. FieldWorks programs check a registry value and disable the splash screen when you set DisableSplashScreen to *true* in HKEY_CURRENT_USER\Software\SIL\Fieldworks. To simplify setting this value, double-click c:\Program Files\SIL\FieldWorks\DisableSplashScreen.reg.

10 Pronunciation writing systems

You can specify the writing system(s) for pronunciations in the detail view by right-clicking the Pronunciation label and choosing Writing System. This shows a list of writing systems for the vernacular language, and allows you to set or clear each writing system you want to see.

11 Reversal indexes

You can have reversal indexes in any language. Use the button to switch between existing reversal indexes.

To add a new index, choose Insert…Reversal Index. This allows you to add a reversal index for any language in the database, whether it shows in the writing system properties or not. Normally, you will only want to add a single index for each language, so pick the writing system for the standard orthography.

The Form field of the reversal entry will show all writing systems for the language of the index checked in Format…Setup Writing Systems. The first writing system is *always* the primary writing system of the reversal index.

Each reversal index has a private Parts of Speech list. Edit these by going to the Lists area and selecting the Reversal Index Categories tool. Select the desired list with the button.

In the Lexicon Edit detail view, the Reversal Entries field shows the primary writing system for each index if that writing system is checked in Format…Setup Writing Systems. From this detail view, you can *only* link to index entries based on the primary writing system of the index. To fill out other writing systems for the index, go to the Reversal Indexes tool.

12 Categorized Entry

In the Categorized Entry tool, type words and definitions for each category. When you leave the current line, Flex will look for an entry with a lexeme form that matches the word you type. If it finds this entry, it checks those senses to see if a definition matches the one you typed. If so, it adds a semantic domain link to that sense. If not, Flex adds a new sense to the entry and sets a semantic domain link to the new sense. If it does not find a matching entry it,

- creates a new entry
- sets the lexeme form
- adds a sense to this entry
- sets the definition, and
- sets a semantic domain link.

13 Interlinearizing with multiple scripts

Language Explorer 2.4, in FieldWorks 5.4 provides the capability for interlinearizing a vernacular language in multiple scripts. In other words, you may have some texts in IPA and some in the standard orthography. It works well as long as you follow careful procedures. If you fail to follow these procedures, you can end up with irregularities that cannot be fixed easily until future versions when more capabilities are implemented.

Refer to Conceptual model overview.doc—2.2.14 Interlinear text for an introduction to the model used in interlinear text. In particular, note that the Word line comes from the Form field in the WfiWordform, once broken into morphemes the Morphemes line comes from The Form field of a MoForm in the LexemeForm or AlternateForms of the LexEntry, and the Lex Entries line comes from the LexemeForm of the LexEntry. Each of these forms are MultiUnicode properties, so they can hold the equivalent string in multiple writing systems.

To interlinearize in multiple scripts, you want to use the same wordform and lexical entry for the same word and morpheme. You just fill in multiple scripts for the forms on these wordforms and lexical entries. For example, if you have ‘cat’ as your orthographic form and ‘kat’ as your IPA form, you should have one wordform and one lexical entry as follows:

	English	English (IPA)

LexEntry_LexemeForm	cat	kat
WfiWordform_Form	cat	kat

Maintaining this state of having both writing systems filled in requires special precautions any time you edit a baseline text in a different writing system or interlinearize using a different writing system. Any time you make this switch, whether the first time you use the new writing system, or switching back to a previously used writing system after interlinearizing in the other writing system, you should first check to make sure all of your wordforms and lexeme forms (and alternate forms if used) have both writing systems filled in. If not, fill them in **prior** to interlinearizing in the different writing system.

Once you have all wordforms and lexical forms entered in both writing systems, you can help to maintain this condition by turning on both wordforms in your interlinear text and as you are interlinearizing make sure you fill in any empty wordforms in the other writing system. You can't fill the missing lexeme form in this way, but it's fairly easy to correct that later using lexicon edit or bulk edit.

To check and fill in missing writing systems for lexeme forms, you can either use a Lexicon Edit browse view or Bulk Edit Entries. In either case, configure the columns to show the lexeme forms and citation forms in both writing systems. You can then set filters to Blanks for each column. If there are any blanks either fill them in by editing, or using a bulk process if you have one defined. There currently isn't any way to easily find missing forms for alternate forms.

To check and fill in missing writing systems for wordforms, use the Bulk Edit Wordforms view in the Texts & Words area. Configure the columns to show the wordform in both writing systems. If there are any blanks either fill them in by editing, or using a bulk process if you have one defined. It is only possible to edit forms that do not have interlinear text connected to that form. If you need to change the spelling after the form is being used in text, you need to use Tools...Spelling...Change Spelling in the Texts & Words area.

The catch in editing a wordform that is already filled in is that it could damage your interlinear text because you are only modifying the wordform. The baseline text will not be changed by these edits. Thus when you edit any of your baseline texts that contain the edited word, the program will create a new wordform for that word using the original spelling, unless you carefully do a search and replace in all baseline texts to fix the spelling there as well. The Change Spelling tool changes the spelling of wordforms as well as all occurrences in baseline texts, so it solves this problem.

Once you have both writing systems filled out in your wordforms and lexeme forms (and alternate forms if needed), then you can safely interlinearize a text in either writing system. The important thing is that you set the writing system for your baseline text before you begin to enter the baseline text. Flex will use the baseline text writing system when looking up or creating new wordforms and lexical entries from the interlinear text. If a given wordform or lexeme form cannot be found, it will create a new one using the baseline writing system. This is what you want to do. It will only be filling in the one writing system, but the other one isn't essential until you try editing a baseline text in the other writing system or interlinearizing in the other writing system. At that point, you need to again make sure all writing systems are filled in for each wordform and lexical entry.

13.1 Things that can go wrong

There are various things that can go wrong if you do not follow the above procedures.

13.1.1 Forms have wrong writing system

The problem here is entering baseline text in the wrong writing system. For example, if you enter your baseline text in IPA but forget to set the writing system via the writing system combo in the toolbar, you will end up with wordforms that have IPA text in a standard orthography field. This may not be apparent if the font covers both writing systems, but your data is compromised and you'll have various problems until it is fixed. Here's how it would look.

	English	English (IPA)
Baseline text	mai kath	
LexEntry_LexemeForm	kat	
WfiWordform_Form	kat	

If you get in this state, you should set the writing system correctly for all baseline texts that are wrong. To fix up the lexeme forms, you could use bulk edit to bulk copy the IPA forms from the English column to the English (IPA) column. To fix the wordforms you would either have to manually edit them via the concordance view, reinterlinearize the texts and then delete the old wordforms, or use the hidden Bulk Edit Wordforms view. The goal is to have

	English	English (IPA)
Baseline text		mai kath
LexEntry_LexemeForm		kat
WfiWordform_Form		kat

13.1.2 Duplicate wordforms and lexical entries

This situation happens if you try interlinearizing a text in a different writing system without first making sure that all wordforms and lexeme forms have both writing systems filled in.

	English	English (IPA)
LexEntry_LexemeForm	cat	
LexEntry_LexemeForm		kat
WfiWordform_Form	cat	
WfiWordform_Form		kat

In this case you have two wordforms for cat, one with the English form and a missing IPA form, and the other with an IPA form and a missing English form. You'll also have two lexical entries. The reason for this is that you already have a wordform and lexical entry for ‘cat’, but when Flex tries to find one for ‘kat’, there isn’t any way it can tell that these entries already exist, so it creates new ones instead of using the existing entries.

Flex has a merge entries option for merging lexical entries, although doing these one by one will take some time if you have many duplicates. There isn’t an option to merge wordforms at this point, so it will require a considerable amount of work fixing these manually.

13.1.3 Wordforms can get out of sync with baseline texts

Suppose you used ‘kath’ for IPA but later decided it should be ‘kat’. If you ‘fix’ this by editing the ‘kath’ wordform in an interlinear bundle and edit the lexeme form, you will likely end up with this type of situation. Initially you would have:

	English	English (IPA)
Baseline text		mai kath
LexEntry_LexemeForm	cat	kath
WfiWordform_Form	cat	kath

After editing the two forms you would have:

	English	English (IPA)
Baseline text		mai kath
LexEntry_LexemeForm	cat	kat
WfiWordform_Form	cat	kat

Now you have the correct form in the wordforms and lexeme forms, but the baseline text still has the old spelling. If you fail to fix this, the next time you edit anything in the paragraph containing ‘kath’, you’ll have:

	English	English (IPA)
Baseline text		mai kath
LexEntry_LexemeForm	cat	kat
WfiWordform_Form	cat	kat
WfiWordform_Form		kath

After editing the baseline paragraph, Flex reprocesses the paragraph, and finds that ‘kath’ doesn’t exist as an IPA wordform, so it creates a new wordform.

The problem can be fixed by using search and replace in the baseline text and correcting any misspelled words. As long as you do this before other edits in the paragraph, you won’t end up with the bad wordform(s). Otherwise, you’ll have to delete the bad wordform(s) after you’ve fixed the baseline text.

14 Interlinear Cautions

14.1 Losing interlinearization in Translation Editor

FieldWorks 5.4 allows users to interlinearize books from Translation Editor. If you import the interlinearized book from an SFM file, you could lose this interlinearization. The TE import process gives you a choice whether to keep or discard your interlinearization when loading reloading a book.

14.2 Damaging wordforms and interlinearlization

When Flex breaks words at improper places, you can currently correct this using WordFormingCharOverrides.xml. See the Wordform characters section of XML configuration files.doc for details on this.

15 LIFT Export & Import

Flex provides the ability to export lexicons in the LIFT XML standard. This output can be loaded directly into recent releases of Lexique Pro without doing a lot of Lexique Pro setup that you would need to do if you used SFM export. Also, when importing from LIFT, Flex provides merging capability that is not available when importing from SFM. This provides one way to merge changed data from one project to another.

15.1 Merging LIFT data

A LIFT file from Flex and WeSay store unique identifiers (GUIDs) on each entry and sense. At this point Flex will only attempt to merge entries or senses that have identical GUIDs. When a LIFT file is imported, Flex tries to merge data with matching GUIDs. If a definition or other field is missing in the database, but is present in the LIFT file, it will be added to the database. If any extra items for sequences (e.g., semantic domains, senses, examples) exist in the LIFT file, they will be added to whatever is already present in the database. If a field is present in the database but not in the LIFT file, the field will not be changed in the database. Thus you cannot use a LIFT import to remove anything from the database. If a field has different non-empty content in the database than in the LIFT file, then the import process uses one of three settings you choose at the beginning of the import.

1) The field is not changed in the database, thus skipping the LIFT field.
2) The field in the database is replaced with the field from the LIFT file.
3) A duplicate entry or sense will be created so that you can see the contents later and merge them manually.

If entries or senses in the LIFT file do not have identical GUIDs, then the entire entry or sense is added to the database. Thus, if two users develop lexicons independently, a LIFT import will typically have many duplicate entries or senses since the GUIDs will all be different. Likewise, if a LIFT file is created by some other program such as Paratext, Lexique Pro, or Solid, the LIFT file will likely not have GUIDs, but even if it does, they would be entirely different, so the import would simply import all the data as separate entries with no attempt to merge.

Another option can speed up import by trusting the modification dates on entries. If the modification dates for an entry are the same in the database and the LIFT file, the entry is automatically skipped without trying to analyze and merge all of the fields in the entry.

15.2 Importing LIFT data from SOLID or WeSay

WeSay (www.wesay.org) data is stored in LIFT files by default. These LIFT files can be imported into Flex.

SOLID (http://palaso.org/solid) provides an alternative way to clean up SFM files prior to import into FieldWorks. It can also produce a LIFT file that can be imported without specifying further mappings inside of FieldWorks. SOLID produces a LIFT file that can then be imported into Flex.

SOLID and WeSay can produce writing system codes that are currently illegal in Flex. Flex 2.4 (FW 5.4) will convert a few codes to valid FieldWorks codes (e.g., bth-fonipa or bth-IPA will be converted to bth__IPA), but it cannot handle the script portion of the LDML BCP47 XML standard. If you have these codes in your LIFT file, they will be added to Flex, but they give some odd results in the writing system dialog and might cause crashes in some circumstances.

For example, zh-Hans-CN is a valid BCP47 language code for Chinese (Simplified Han in China). This gets imported into FieldWorks as zh_Hans_CN, but it is incorrectly shown as zh_CN_CN in the writing system wizard with no name for the final CN variant. It can be corrected to zh_CN by setting the Variant to some other value and then clearing the name. The result will be zh_CN. When you click Ok, it will give you an option to merge the data into the existing zh_CN. The writing system merge in FieldWorks 5.4 will fail under certain circumstances, but as long as it doesn't fail, this will remove the original zh_Hans_CN writing system and switch all data to use the built in zh_CN.

By default, SOLID uses eng for English since this is a valid ISO 639-3 code. However, the ICU component of FieldWorks requires a 2-letter ISO 639-1 code when available. This is based on the BCP47 standard that specifies the shortest ISO 639 code for language identifiers. So if you try to import a SOLID LIFT file that uses eng for English, the import will fail with a -18 error message trying to register the language eng.xml in ICU. When English is chosen inside FieldWorks, the program automatically converts this to the two-letter equivalent, en. At this point the LIFT import does not do this, so it's important to have the correct language codes in the LIFT file prior to import.

15.3 Spell checking

FieldWorks will use Open Office 2.x (not later versions) spelling dictionaries directly. If you don't have Open Office 2.x installed, you can download spelling dictionaries from http://wiki.services.openoffice.org/wiki/Dictionaries and copy them to C:\Documents and Settings\<username>\Application Data\enchant\myspell or C:\Users\<username>\AppData\Roaming\enchant\myspell. Then in the FieldWorks Writing System Properties dialog for the writing system you are setting up, set the Spelling dictionary combo to the desired dictionary.

Using a custom Flex writing system????

16 Searching and Filters

You are getting into an area that gets pretty technical. Let me explain a bit about searching as a starter.

Here are some sample words:

1 aba

2 ãbã a tilde

3 ã́bã́ a tilde acute

4 á̃bá̃ a acute tilde

5 ãbcdé̃fg a tilde and e acute tilde

6 abénda e acute

7 abõnó̃ o tilde and o acute tilde

Here are some examples of how FieldWorks search and filtering work.

- Setting a filter for ab without checking diacritics brings up #1-7 since diacritics are ignored.
- Setting a filter for ab with diacritics checked brings up #1,6,7 because diacritics are now significant to the search and these three are the only ones that contain ab without any diacritics.
- Setting a filter for ã (a tilde) with diacritics checked only brings up #2 and #5. When diacritics are checked, every combination of base form with diacritics is considered as a unique entity, and #2 and #5 are the only words that contain just an a with a tilde.
- Setting a filter for ã́ (a tilde acute) only brings up #3 since this is the only word with this combination.
- Setting a filter for á̃ (a acute tilde) only brings up #4 since this is the only word with this combination.
- Setting a filter for ~ with or without diacritics doesn't match anything. This is apparently a limitation in the way the search code works since it is probably looking for a base form plus diacritics.

When you can’t get the matching to work the way you want using these simple techniques, the next step is to try regular expressions. Regular expressions provide a powerful syntax for specifying complex sort and replace specifications. In the Filter and Replace dialogs if you click Help, it will take you to some help on regular expressions and give various examples of useful features.

- Setting a filter for \p{M} with “Match for regular expression” checked matches #2-7. \p{M} is the way to look for any diacritics anywhere in a word.
- Setting a filter for \N{combining tilde} with “Match for regular expression” checked gives #2-5,7. This expression looks for any word containing one or more ‘combining tilde’ code points. You have to use the technical Unicode name for the code point when using the \N{} syntax.

- Setting a filter for \x{303} with “Match for regular expression” checked gives #2-5,7. This is identical to the previous example. Instead of using the technical name for the code point, the \x{} syntax uses the Unicode numeric value for the code point. 0303 is the code point stored in the data for a combining tilde.
- Setting a filter for \x{303}\x{301} with “Match for regular expression” checked gives #3. In this case we are looking for any word that contains a tilde (303) followed by an acute accent (301) regardless of the vowel.
- Setting a filter for \x{301}\x{303} with “Match for regular expression” checked gives #4,5,7. In this case we are looking for any word that contains an acute accent (301) followed by a tilde (303) regardless of the vowel.

See Bulk edit issues.doc section 4 for additional information on regular expressions.

17 Lexical Relations

17.1 Introduction

Lexical relations are used to reference other senses or entries. Unlike Toolbox and similar programs where relations are expressed textually, in Flex lexical relations are pointers to senses and entries. When displaying relations, Flex uses the target headword, homograph number, sense number, and optionally a sense gloss to display the targets. This approach has several advantages. Relations change automatically if headword, homograph number, sense number, or gloss changes in the target. References are removed if an entry or sense is deleted, so you never have references to non-existent senses or entries.

17.2 Defining a Lexical Relation

Lexical relations are defined in the Lexical Relations section of the Lists area. New projects come with several predefined relations, but you are free to add, delete, or change relations as needed.

Relations are actually stored as sets pointing to senses or entries. The Reference set type selector determines what is allowed in a set, and how the set will be displayed from the target senses or entries. There are 5 basic relation types:

- Collection: the set holds two or more targets and every target will list all of the other targets in the display. This is typically used for synonyms.
- Pair: the set holds two targets and each target will list the other target in the display. This is typically used for antonyms.
- Pair--two relation names: the set holds two targets and each target will list the other target in the display, but will use a different name/abbreviation at each end.
- Tree: the set holds two or more targets. The first target has a special relation to all of the other targets. For example, the first target may be a whole, and the remaining targets are parts. Then when displaying from the whole, all targets after the first are listed, but when displaying from a part, only the whole is displayed.
- Sequence/Scale: the set holds two or more targets. The targets will be listed in the same order when displayed from any member of the set. This could be used to show age, days of week, etc.

In addition to the type of reference, you also need to specify whether the relation set points to senses, entries, or entries and senses. This choice determines whether the relations will show up under Lexical Relations on senses, or Cross References on entries.

For each relation, you'll need to define a name, and abbreviation. The name is used when creating a new relation set and is used as the label in the detail view, and the abbreviation is usually used when displaying the relation in the dictionary view. You can also fill in a description of the relation, or discussion giving more detail about how it is used. For Pair—two relation name and Tree relations, there is also a reverse name and a reverse abbreviation. The name and abbreviation is used from one end of the relation, and the reverse name and reverse abbreviation is used from the other end. All of these fields can be entered in any number of analysis writing systems.

Flex does not block you from changing the Reference set type when there are existing relation links, but doing so may cause undesirable results. Changing the Reference set type back to the original setting should restore everything as it was. The names and abbreviations can be changed at any time without affecting existing data.

17.3 Configuring Lexical Relations

In Tools…Configure…Dictionary you can alter various aspects of how relations will appear in the final dictionary. For relations on entries, you configure the Cross References node on entry. For relations on senses, you configure the Lexical Relations node on senses.

When you click the top level of the node you can check which relation types you want to show at this level of the display. It's possible to show some relations at one location in the display and others in other locations. To do this you would duplicate the node and then check different relation types for each copy. Also at this level, you can choose the character style for the relation contents, and enter text to show before, between, or after the targets in the relation.

By default, Flex always shows relation information for every member of the relation set. However, it is possible in 2 cases to change this in the dictionary output. For most relations, there is only one item in the relation types list discussed in the previous paragraph. However, for Pair—2 relation names, and Tree, since these relations also have reverse names and abbreviations, both the name and the reverse name are listed in this list. So for these two cases you can uncheck the reverse name, and Flex will not show it in the dictionary view. This feature will be discussed in further detail below in the Unidirectional example.

You can choose to show the name and/or abbreviation at the beginning of each of the targets in the relation. For each of these you can assign a style for displaying the string, and you can also enter text to show before, between, or after each type of relation.

Finally, you can choose how to display the targets in the relation, specifying style, and text before, between, or after the target, For entries, you can also choose to display the summary definition of the entry, and for senses you can also choose to display the gloss of the sense.

17.4 Entering Lexical Relations

17.4.1 Collection example

Suppose you have entry A and you want to add B as a synonym.

Click just left of the Lexical Relations field and choose Insert Synonyms Relation. This brings up an Add Reference (Synonyms) dialog that lets you find an existing sense or create a new entry. If you type B in the Find box and click Create, it opens a New Entry box with B filled in for the Lexeme Form. You can fill in a Gloss, then click Create. If you now go to the dictionary view, you'll see this:

A agloss *syn:* **B.**

B bgloss *syn:* **A.**

Flex has added B as a synonym of A. But in this process it also shows A as a synonym of B. This is one Synonym set that points to A and B.

Now in the B entry, you can add C to this set by clicking the middle of the Synonyms field and a button appears on the right side. Click that button and it brings up the Add Reference (Synonyms) dialog. If you use this to create entry C, you'll now have this in the dictionary view.

A agloss *syn:* **B, C.**

B bgloss *syn:* **A, C.**

C cgloss *syn:* **A, B.**

Notice we now show 6 synonyms. A has B and C as synonyms, B has A and C as synonyms, and C has A and B as synonyms. At this point there is one synonym set that points to A, B, and C, and Flex shows the relationships on all of the members of this set.

Now suppose we want to add D as a synonym of C. If you use the same procedure as last time, and add D from the little button on the right, we get this.

A agloss *syn:* **B, C, D.**

B bgloss *syn:* **A, C, D.**

C cgloss *syn:* **A, B, D.**

D dgloss *syn:* **A, B, C.**

What we have now is one synonym set that points to A, B, C, and D, so each of these entries shows all of the other entries as synonyms. But what if D isn't a very good synonym for A, so you go to the A entry, click the D synonym in the Synonyms field and press Delete. That deletes it from the A entry. But now notice what happens in our dictionary display.

A agloss *syn:* **B, C.**

B bgloss *syn:* **A, C.**

C cgloss *syn:* **A, B.**

D dgloss

What we did was delete the D sense from the synonym set, so it took D out of all of the entries of this set. So how can we get D to be a synonym of C without affecting the other synonyms of C? To do that, instead of adding D to the existing synonym set, we need to create a new synonym set with just C and D. So in the C entry, instead of using the little button to add to the existing set, click to the left of the Lexical Relations label and choose Insert Synonyms Relation, and add D to this new set. Now this is what we have in the dictionary.

A agloss *syn:* **B, C.**

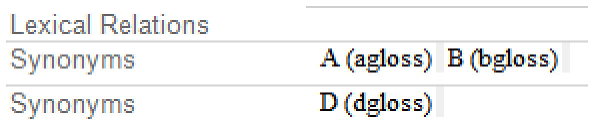
B bgloss *syn:* **A, C.**

C cgloss *syn:* **A, B;** *syn:* **D.**

D dgloss *syn:* **C.**

Note that C now has two synonym sets, as indicated by the two "syn:" labels. One is with A and B, while the other is with D. D has a synonym relation with C, but A and B do not share that relation. So this accomplishes what we want, although the dictionary view is a little unwieldy because it shows two syn labels where you probably want only one. At present Flex is not able to display these with a single label, but when using Pathway for output, you can use a transform to merge these together in the final dictionary.

In the data entry view you can tell when you have multiple sets of a given relation type because each set shows up as a separate field. So at this point the data entry fields for C would look like this.



Lexical Relations	
Synonyms	A (agloss) B (bgloss)
Synonyms	D (dgloss)

So this works reasonably well if you only want to add D to the C synonyms. But what if you wanted D as synonyms of A and C, but not B. You could do this by adding a new synonyms set to A that includes D. The synonym (or any collection) set works well when all of the members can be added to a single set. It becomes more unwieldy when you have overlapping sets with different members.

Returning to the last example, suppose entry B is deleted, and the C entry is edited to be Cc. The resulting dictionary would be.

A agloss *syn:* **Cc.**

Cc cgloss *syn:* **A;** *syn:* **D;** *syn:* **D.**

D dgloss *syn:* **Cc;** *syn:* **Cc.**

OOPS this is buggy, but crashed when trying to delete one of the duplicates and when restarting it was OK.

A agloss *syn:* **Cc.**

Cc cgloss *syn:* **A;** *syn:* **D.**

D dgloss *syn:* **Cc.**

Notice that every occurrence of B in the synonmys is automatically removed. Also, every place that used to have C now has Cc. This demonstrates how Flex maintains integrity of relations as members change.

If you want to delete an entire synonym set, right-click the Synonyms field label in one of the senses, and choose Delete Relation. This removes the entire set, so that relation will disappear from all of the members that were in the set.

17.4.2 Pair example

Returning to a single A entry, if you click to the left of the Lexical Relations label and choose Insert Antonym Relation, and enter B as an antonym of A, then you'll get this in the dictionary.

A agloss *ant:* **B.**

B bgloss *ant:* **A.**

Notice B automatically shows that it is an antonym of A.

17.4.3 Pair—2 relation names example

Suppose we have a relation defined with a Reference set type of Sense Pair – 2 relation names, with Name: Forward, Abbreviation: fwd, Reverse Name: Reverse, and Reverse Abbreviation: rev. If we start in entry A and click to the left of Lexical Relations and choose Insert Forward Relation and set it to B, we would have this dictionary view.

A agloss *fwd:* **B.**

B bgloss *rev:* **A.**

This is similar to a Pair relation, but notice that instead of both entries using the same relation abbreviation, this type of relation is directional. It lists B as a fwd relation for A, but lists A as a rev relation for B.

17.4.4 Tree example

Starting with a 'house' entry, click to the left of Lexical Relations and choose Insert Part Relation, then add 'roof' as a part. Again in the 'house' entry, click the button to the right of the 'roof' relation and add 'walls'. The resulting dictionary will show these entries.

house house *pt:* **roof, walls.**

roof roof *wh:* **house.**

walls walls *wh:* **house.**

Notice 'house' now lists 'roof' and 'walls' as parts. Also, 'roof' shows 'house as a whole, and 'walls' also shows 'house' as a whole.

It's also possible to add senses using the Whole relation. For example, if you add entry 'foundation', then click to the left of the Lexical Relations label and choose Insert Whole Relation, then you can enter 'house' to add it to the existing 'house' relation. So the dictionary view will show this.

foundation foundation *wh:* **house.**

house house *pt:* **roof, walls, foundation.**

roof roof *wh:* **house.**

walls walls *wh:* **house.**

Notice as each part is added, it goes to the end of the list. If you want a different order, go to the 'house' entry and in the Part relation field, right-click a sense and choose Move Left, or Move right to change the order. By moving the 'foundation' sense to the left two times, we would have this.

foundation foundation *wh:* **house**.

house house *pt:* **foundation**, **roof**, **walls**.

roof roof *wh:* **house**.

walls walls *wh:* **house**.

The example so far shows a single node that has multiple parts. We can extend this into a tree by adding parts to 'roof'. Go to the 'roof' entry and choose Insert Part Relation, then add 'shingles', then add 'trusses' to the same relation. The dictionary would now have these entries.

foundation foundation *wh:* **house**.

house house *pt:* **foundation**, **roof**, **walls**.

roof roof *wh:* **house**; *pt:* **shingles**, **trusses**.

shingles shingles *wh:* **roof**.

trusses truses *wh:* **roof**.

walls walls *wh:* **house**.

Notice the 'roof' entry shows that it is part of a 'house', but it also contains its own parts, 'shingles', and 'trusses'

17.4.5 Sequence/Scale example

Starting with a 'January' entry, click to the left of Lexical Relation and choose Insert Calendar Relation and add 'February'. Then to the same relation, click the chooser button and add 'March'. The dictionary has these entries.

February Feb *cal:* **January**, **February**, **March**.

January Jan *cal:* **January**, **February**, **March**.

March Mar *cal:* **January**, **February**, **March**.

With this kind of relation, each entry contains the entire sequence of set members including the current entry in the same order.

17.4.6 Unidirectional example

There may be some relations where you want to refer to other senses, but you don't want all of the senses to show the reverse relation. The 'See' relation may be this type because you just want to have the user look at related entries, but don't want to list the specific relation involved. Flex does not currently have a unidirectional relation, but it's on the list for improvements. In the meantime there is a way you can accomplish this in Flex that works well for the dictionary view, but has some messy side effects in the data entry view that you'll have to live with until we can improve the user interface.

For this experiment, create a new Lexical Relation in the Lists area called 'See'. Set the Reverence set type to Sense Tree, use 'See' as the name and 'Ignore See' as the reverse name. Go to Tools…Configure…Dictionary, expand the Senses node and click the Lexical Relations node. In the Lexical Relation Types box on the right, uncheck 'Ignore See'. Now in entry A, click to the left of Lexical Relations and choose Insert See Relation and add B. Then go to B and add a See relation to C. The dictionary now shows this.

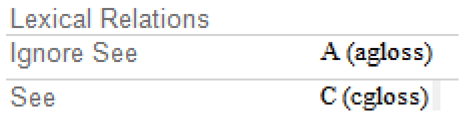
A agloss *see:* **B**.

B bgloss *see:* **C**.

C cgloss

Notice we no longer have any automatic back references showing. We only show references that were explicitly set in each sense. There is never any confusion with making a change in one sense that affects other senses as well. Of course, this means where you actually want to show the reflexive relationships, using this approach you need to add the relation to both senses.

There are two confusing factors in the user interface at this point.



Lexical Relations	
Ignore See	A (agloss)
See	C (cgloss)

Notice in the data entry for the B entry, we show both the ‘See’ relation going to C, and also the automatic ‘Ignore See’ relation. So although we don’t show the ‘Ignore See’ relation in the dictionary view, there isn’t any way to keep it from showing in the data entry view. Furthermore, you should never modify what’s in the ‘Ignore See’ field since this would be removing the desired relation from the other sense (A in this example).

The other undesirable feature is that when you click to the left of the Lexical Relations field, the dropdown menu includes ‘Insert Ignore See Relation’. You should never choose this option.

My anticipation is that in the future we can refine the Tree relation so that if the Reverse name is left empty, it would automatically work like our above exampe, except you wouldn’t need to uncheck the option in Configure Dictionary, the Ignore fields wouldn’t show up in data entry, and they wouldn’t be an option in the insert relation menu.

17.5 Importing Lexical Relations from SFM

Since lexical relations (and cross references) have always been reciprocal in Flex to date, we assumed that SFM files would also be reciprocal, and when the SFM file is missing the reciprocal relation, we automatically supply it. Thus if we have

```
\lx A
\cf B

\lx B
```

The default definition for the compare relation is a collection of entries. On the A entry, we create a compare entry set (LexReference) with pointers to A and B. So in Flex, you’ll see that we automatically supplied the missing \cf A in the B entry.

A cf: B
B cf: A

This was seen as helpful when we initially designed SFM import because we assumed the user just failed to add the missing \cf A in their SFM file.

Now suppose we have

```
\lx A
\cf B
```

\lx B
\cf A

When we import this, in entry A we create a compare entry set pointing to AB. Then in entry B we create another compare entry set pointing to BA. If we displayed this at this point we would have

A cf: B cf: B
B cf: A cf: A

because we have two entry sets pointing to A and B. This obviously isn't what the user wants. So after the initial import, we run a process (phase 5) that tries to set and clean up all of the references (lexical relations, cross references, variants, complex forms). In this process we find that we have two compare sets that point to the same entries, so we delete the extra set giving the desired:

A cf: B
B cf: A

Now suppose we have

\lx A
\cf B

\lx B
\cf C

In entry A we create a compare entry set pointing to AB. In B we create a compare entry set pointing to BC. Now during the cleanup process we discover that there isn't an entry C, so we create that entry to fulfill the compare relationship. In the C entry we add "This was automatically created to satisfy a Cross Reference, and it should be checked." to the Import Residue field to indicate to the user that this entry couldn't be found, so it was created automatically. There is a JIRA issue asking that we change this to not create these entries automatically but instead insert a note in Import Residue saying the compare relation couldn't find the entry as we did in the early days.

In the early days we assumed that in this situation where A is related to B, and B is related to C, then C is probably related to A as well, so we combined these two compare sets to a single set with ABC. Thus we had

A cf: B, C
B cf: A, C
C cf: A, B

However, we discovered when we did this we often ended up with compare sets with a huge number of entries in each set, resulting in relations to far more entries than the user desired. So we changed the algorithm so that it would only combine sets that had the same set of entries. Since the current sets are AB and BC, they are not identical, thus they are not combined, so the result is

A cf: B
B cf: A, C
C cf: B

Now suppose we have this input

```
\lx A
\cf B
\cf C

\lx B
\cf A
\cf C

\lx C
\cf A
\cf D

\lx D
\cf C
```

In entry A we create a compare set pointing to ABC. In entry B we create a compare set pointing to BAC. In entry C we create a compare set pointing to CAD, and in entry D we create a compare set pointing to DC. I think what used to happen here is that one of the ABC sets would be deleted since they were identical, so we would have sets ABC, CAD, and DC, which would display

```
A cf: B, C cf: C, D
B cf: A, C
C cf: A, B cf: A, D cf: D
D cf: C
```

What we actually get now (FW8.2.7) is a single set with ABCD. I don't know why this is happening and it appears to be a bug. It seems to be going back to our earlier attempts at merging.

```
A cf: B, C, D
B cf: A, C, D
C cf: A, B, D
D cf: A, B, C
```

However, it's highly unlikely that anyone would want either of the above two results for this input. What they probably want is what they started out with:

```
A cf: B, C
B cf: A, C
C cf: A, D
D cf: C
```

I don't think it is possible to produce this kind of output with our standard reciprocal relationships. The only way to make this work is to use a unidirectional relation which we do not currently have, other than the current hack described in section 17.4.6 above for using a Tree relationship and disabling the reverse relationship from showing in the dictionary view. With a tree relation (or unidirectional relation) we would keep the original ABC, BCA, CAD, and DC sets during the merge phase.

Our recommendation at this point for these kinds of relationships is to create Tree relations with bogus reverse names and eliminate the bogus names from dictionary configuration. The import process can then be mapped to the appropriate special tree relationship.

18 Bugs

18.1 Loss of interlinear information

FieldWorks 5.4.1, 5.4, and possibly some older versions can lose interlinear information using the Find and Replace dialog on the baseline text. If you use the Find Next and Replace buttons you won't have any trouble. However, if you use Replace All, Flex will damage the first paragraph that has a change. Note, this could be your whole interlinear text if you only have one paragraph. After Replace All, all interlinear choices in this paragraph will be lost and all free & literal translations and notes for this paragraph will be lost. Also, if you have done discourse charting, the discourse chart information for this paragraph will be lost.

In most cases, you should not be using Find and Replace on baseline text once it has been interlinearized. This only changes the baseline text, but does not change the wordforms and lexeme forms that are associated with the interlinear text. The Change Spelling dialog is designed to change a word in all baseline texts as well as the wordform and lexical entry, maintaining the wordform analyses when practical. It also allows you to uncheck any instances you don't want the change to affect. To get to the Change Spelling dialog, go to Word Analyses and find the wordform you want to change. Right-click the word or use the blue arrow to the left of the word and choose Change Spelling from the context menu. The dialog shows all instances and provides checkboxes, Preview, and Apply buttons similar to bulk edit.