Technical Notes on FieldWorks Send/Receive

October 25, 2018

Contents

1 Send/Receive Introduction

FieldWorks Language Explorer (FLEx) provides two ways to collaborate with colleagues working on the same project. The first approach (starting in FieldWorks 7.3) includes the entire FieldWorks project: the full lexicon, grammar, interlinear texts, data notebook, scripture, lists, pictures, sound files, writing systems, and some configuration information. This is the approach that should be used whenever you are collaborating with other FLEx users. The second approach is via the Lexicon Interchange Format (LIFT). This approach should only be used if you are

collaborating with WeSay users or some future program that can only use LIFT. LIFT only includes the lexicon, parts of grammar, writing systems, pictures, and sound files. The LIFT format does not cover as much detail in the lexicon as the FLEx project format, so the results are less satisfactory when using features of FLEx that WeSay does not use. There are also some extra issues with custom fields and lists.

With either approach, collaboration can be through any combination of the following

- Internet server (e.g., http://languagedepot.org) which requires some initial setup on the server that needs to be requested ahead of time.
- USB drive (no setup needed) that is passed around to users.
- Local Network using ChorusHub (requires setting up an instance of ChorusHub on one machine on a network).

If you are just collaborating between FLEx users, you would only use a FLEx repository (repo).

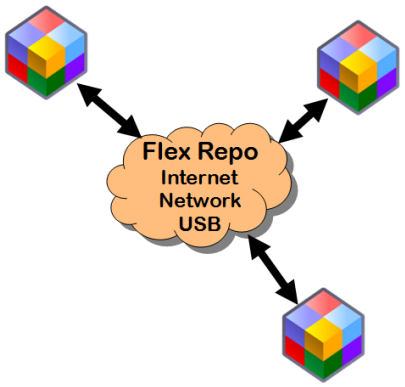


Fig. 1 Flex Collaboration

If you are just collaborating between WeSay users and one optional FLEx user, you would only use a LIFT repo.

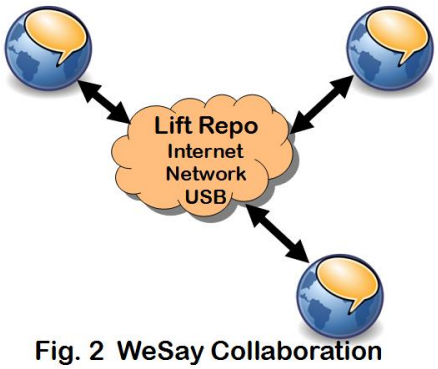


Fig. 2 WeSay Collaboration

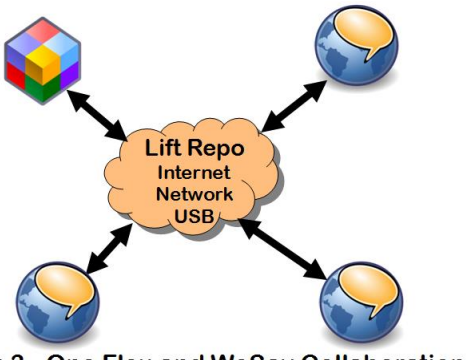
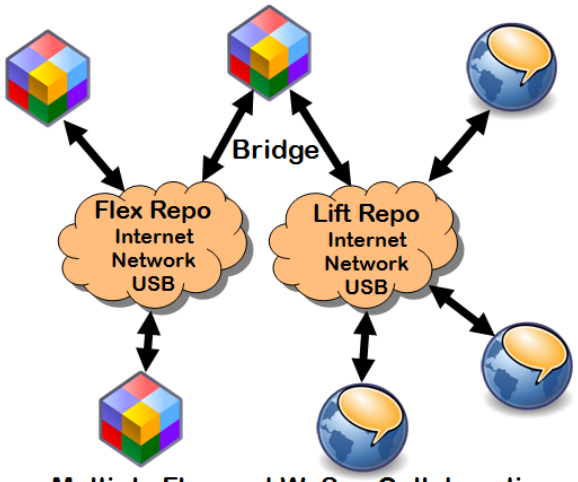


Fig 3. One Flex and WeSay Collaboration

If you are collaborating with multiple FLEx users and one or more WeSay users, then you will need a separate FLEx repo for collaboration between FLEx users, and a LIFT repo for collaborating between WeSay users and a FLEx user. It's critical (explained below) that only one FLEx user act as a 'bridge' between the two repos. This user would periodically sync with the LIFT repo and the FLEx repo. Other FLEx users would just sync with the FLEx repo, and WeSay users can only sync with the LIFT repo.



Multiple Flex and WeSay Collaboration

The FLEx Send/Receive menu has changed in different versions since FW7.3. This document was written for FLEx 8.0.5. Here is a brief summary of how the Send/Receive process works between two or more FLEx users.

One user starts the collaboration process from a master copy of the project. This user does an initial Send/Receive…Send this Project for the first time to store a copy of their project in a Mercurial repository (repo). Each colleague then gets a copy of the project from the repo using Send/Receive...Get Project from Colleague. After that, all colleagues periodically do Send/Receive…Send/Receive Project (with other FLEx users) (S/R) to keep their projects synchronized with their colleagues.

If colleagues are working in different parts of the project, the S/R will merge their changes without any conflicts or loss of data. If colleagues change the same piece of data (e.g., the definition of the same sense), during the S/R process the program will merge the changes the best it can (e.g., picking one of the two definitions), then add a conflict report warning the users about a change that was made that would be wise to review. If an undesirable change was made in a merge, the user will need to fix the appropriate data manually.

The current S/R process does not support different privileges for different users. Any user doing S/R can modify any part of the data. For an Internet server, a user can be given read-only permission to a repository, which allows them to get a copy of the project, but nothing they do will get back into the repo.

With each S/R operation, changes and new information are appended to the repo with a timestamp. Merges use a 3-way process that can generally tell whether the change was an addition by one user, a deletion by one user, or a modification by both users. Each user has a local (hidden) copy of the repo as it was the last time they did a S/R, so this helps the program to know what changed since their last S/R. FLEx does not provide a way to see what changed

during a S/R—it only shows merge conflicts that occurred. Some special programs outside of FLEx let you see the history of changes in the repo, and it is possible to go back in time to an earlier version of the project, but this takes more advanced skills.

When using S/R with a project, you can still back up your project using FieldWorks backup, but you need to be very careful about using a FieldWorks restore, as this will likely cause all colleagues to lose their work since the backup was made.

2 Getting started

2.1 Starting up Project (FLEx) Send/Receive

Before your first S/R you should make sure that you have a single project that can be considered your master project. If two users created their own FieldWorks project and added entries independently, the merge collaboration will not work. Also, if two users started with the same FieldWorks project, but have made independent changes, the merge collaboration will not work. This is discussed in more detail later. If you have multiple projects, you should find some way to merge these before doing your first S/R. Once you do the initial S/R of your project, all other collaborators need to delete or move (not just rename) any copies of the old project from their FieldWorks Projects directory and then start over by getting the master project from the new repo.

If you plan to use Chorus Hub or Internet options, you first need to set these up. See the following sections for details.

The first step is for one user to do an initial S/R from their master project. To do this, follow these steps.

1. Go to Send/Receive...Send this Project for the first time.
2. Optionally type a label in the Send/Receive Project dialog
3. a) Click USB Flash Drive or Chorus Hub if you are not using the Internet button. This will start the Send/Receive process to the specified device.
   b) For Internet
   1. Click the Settings box in the lower right to bring up the Send/Receive Settings dialog.
   2. Make sure Server is set to LanguageDepot.org
   3. Fill in your LanguageDepot project id, user login and user password
   4. Click OK to close the Send/Receive Settings dialog.
   5. Now the Internet button should be enabled, so click that to start the Send/Receive process.

Every other user that wants to collaborate with this project then needs to do a one-time setup to get the project on their machine. To do this, follow these steps.

1. Make sure you have the same version of FieldWorks installed on your machine that was used for the master project.
2. Go to Send/Receive…Check for FLEx Bridge Updates to see if there is a later version of FLExBridge available. If there is, Click Install update and follow the default installation.
3. If you already have an older copy of this FLEx project on your machine use File…Project Management…Delete Project if you can do this from another open project. Otherwise, delete the project folder in C:\ProgramData\SIL\FieldWorks\Projects. On XP this is in

c:\Documents and Settings\All Users\Application Data\SIL\FieldWorks\Projects. The default settings for Windows Explorer will not show this directory. You can still get to it by typing at least part of the path into the edit box at the top.

4. Start FLEx. If the startup screen comes up, choose "Get project from a colleague". Otherwise, go to Send/Receive…Get project from colleague. Either option will bring up a Receive project dialog.
5. Choose the location of the repo holding your project
   a) Click USB Flash Drive or Chorus Hub, or
   b) Click Internet. You'll need to specify some additional information in the Get Project From Internet dialog before it will start.
      1. Make sure Server is set to LanguageDepot.org
      2. Fill in your LanguageDepot project id, user login and user password
      3. Type the project folder name you want to use on your computer (not the full path). The project folder will be created in your FieldWorks Projects directory.
      4. Click Download to start the download.

This process will create a new project folder in your FieldWorks Projects directory which will contain a copy of what was in the repo at that point.

Another way users can get started after the first user did the initial S/R is to copy the entire FieldWorks project directory from the first machine to the other machines. If you do this, the first time each user does a S/R, they should click the Settings link in the Send/Receive Project dialog and change the "Name to show in change history:" to the current user.

Once each user has a current copy of the project on their computer, they can periodically do Send/Receive...Project (with other FLEx users) and click the desired location for synchronization. You can do another S/R to a different location if you want to keep several repos in sync.

2.2 Starting up Lexicon (LIFT) Send/Receive in FLEx.

The master project for starting S/R can be either a FLEx or a WeSay project. Note there are a few compatibility issues between FLEx and WeSay. See Section 4.2 for more details.

If a FLEx user is setting up the initial LIFT repo, use Send/Receive…Send this Lexicon for the first time (to WeSay) and then select the desired destination.

If a LIFT repo has already been set up by WeSay, and there is no FLEx project for this language, then the FLEx user needs to connect to the existing repo using Send/Receive…Get Project from Colleague. FLEx will realize that the project is a LIFT repo, so it will create a new FLEx project based on the LIFT repo.

It's possible that you have been collaborating between FLEx and WeSay in the past, but the connection may get broken. For example, if your FLEx project directory gets deleted and you restore from a current FLEx backup. (Note restoring a backup when using Send/Receive is usually dangerous. See Section 4.3 for more details.) To get reconnected to an existing LIFT repo, you need to use Send/Receive...Get Lexicon (WeSay) and Merge with this Project. This will do the initial sync to the LIFT repo without losing other information in your FLEx project, such as interlinear texts. Before doing this command, inside your FLEx language project folder, look for an OtherRepositories folder. If there is anything inside this folder, delete it first.

Once the initial connection has been established, then use Send/Receive…Lexicon (WeSay) whenever you want to sync with the LIFT repo.

2.3 Starting up Send/Receive in WeSay.

The master project for starting S/R can be either a FLEx or a WeSay project. Note there are a few compatibility issues between FLEx and WeSay. See Section 4.2 for more details.

If a WeSay user is setting up the initial LIFT repo, In the WeSay Home tab, click the Send/Receive button, and then choose the destination for the repo.

If a LIFT repo has already been set up by FLEx or another WeSay user, and there is no WeSay project for this language on your machine, then you need to use the WeSay Configuration tool to get connected. In the WeSay Configuration Tool opening dialog, choose "Get from USB drive" or "Get from Internet", choose the desired repo, then click Copy To Computer. This will create a WeSay project on your machine that will be in sync with the repo.

Once the initial connection has been established, then use Send/Receive in the WeSay Home tab whenever you want to sync with the LIFT repo.

2.4 Chorus Hub startup

Older versions of Chorus Hub used a process on the server computer. Recent versions use a service on the server computer. These two versions are described in the next two sections. It should be possible to use a mixture of versions of FLEx Bridge and Chorus Hub without causing problems. But it is recommended that collaborators use current versions of software.

2.4.1 FLEx Bridge 2.2 – Chorus Hub process

In this version of FLEx Bridge, the Chorus Hub 2.4.266 program is installed in the FLEx Bridge directory, and can be started on a single machine on the network. It is not a service, so it is stopped any time the user that started it logs off, or the machine goes into sleep mode. Only one instance of Chorus Hub can be running at one time on a network. If you try to start Chorus Hub when there is already an active Chorus Hub on the network, it will let you know that it can't be started because there is already a Chorus Hub running on the network. It uses a directory (c:\ChorusHub) on the machine that is running Chorus Hub to store repos for all users on the network. Anyone on the network has access to Chorus Hub whether it is running on your local machine or on another computer on the network. Chorus Hub supports repos from several different programs including FieldWorks, WeSay, and Bloom. Each program limits access to repos it can use.

From FW8.0.6 through FW8.1.4, chorushub.exe was installed in the FLEx Bridge directory and could be started from there. C:\Program Files (x86)\SIL\FieldWorks 8\Installers\ChorusHubInstaller.msi was also copied to your machine during installation. This installer can be run on any machine, even without FieldWorks, to run the Chorus Hub 2.4.7 program on that machine. After installing the program, you can launch it from: c:\Program Files (x86)\SIL\Chorus Hub\ChorusHub.exe (omit " (x86)" for 32-bit Windows)

When Chorus Hub is running, a Chorus Hub dialog is present that gives information about the program. Other users that start a Send/Receive operation on the network will have access to the Chorus Hub button in the Send/Receive dialog as long as some machine on the network is

running Chorus Hub. You'll have to wait a second or two for this button to be activated. Before you can close the Chorus Hub dialog, you must click the "Stop Chorus Hub" link at the bottom. The project directories will remain in the ChorusHub directory and can be used again by restarting Chorus Hub.

Initial startup of Chorus Hub may ask to permit hg.exe through the firewall. This should be granted. It may also need ChorusHub.exe clearance through the firewall. This seems to happen automatically with Windows Firewall, but with other firewalls you may need to allow these permissions before it works.

Chorus Hub 2.4 will give continuous warning messages, "this machine has more than one IP address", and may or may not work if it detects more than one IP address on your machine. This will happen if you have more than one network card, if you have wireless as well as wired connections, or if you are running virtual machines. In general, you should pick a machine to run Chorus Hub that has a single IP port. See Section 4.1 for further information on virtual machines.

2.4.2 FLEx Bridge 2.3 – Chorus Hub service

Starting in FW8.2.0, FLEx Bridge 2.3.5 or later is installed as part of the installation. This version uses a newer version of Mercurial (3.3.2) to solve some difficult bugs occasionally encountered with the older version. ChorusHub.exe is no longer installed in the FLEx Bridge folder. A new ChorusHubInstaller.msi file is copied to c:\Program Files (x86)\SIL\FieldWorks 8\Installers. This is Chorus Hub 2.5.20 or later. This version installs a service called Chorus Hub Sharing Service on the computer instead of running a process. As a service, it will be available on the machine whenever it is running and is not dependent on a user being logged in.

Firewall: At least on some machines it is important to add c:\Program Files (x86)\SIL\Chorus Hub\mercurial\hg.exe to the inbound rules in Windows Firewall for Chorus Hub to work. If you get a S/R message, "abort: error: A connection attempt failed because the connected party did not properly respond after a period of time, or established connection failed because connected host has failed to respond" it probably indicates your firewall is blocking hg.exe.

When Chorus Hub is installed as a service, it can't easily warn you about another Chorus Hub already running on the network since services do not use dialogs. If there is more than one machine running Chorus Hub service on a network, as could happen in the past, FLEx will pick one, seemingly the last one that was started. In the S/R dialogs Chorus Hub will identify the machine on which it is running. If for some reason you want to have Chorus Hub server installed on more than one machine, you should stop all services except the one you want to use. During installation, the service will be set to Automatic which means it starts whenever the machine is booted. To start or stop the service, you can type Services in the Start dialog, then click Services at the top of the menu. In this dialog, find Chorus Hub Sharing Service in the right pane. You can right-click this and choose to Start or Stop the service. With more than one installation, you should go to the Properties on this service and set the Startup type to Manual on all but your primary machine which should be set to Automatic so it will start up automatically when you reboot.

When you uninstall Chorus Hub through Programs and Features, it will automatically stop and uninstall the Chorus Hub service.

The Chorus Hub service will send messages to the Event Viewer when there are problems. To see these, type Event Viewer in the Start dialog, then click Event Viewer at the top of the menu

to open the event dialog. In the left of this dialog, open the Windows Logs node and click the Application node. Any messages from Chorus Hub will show up in the right pane.

If you install Chorus Hub service on a machine when another machine on the network is already running Chorus Hub, The installation takes place, but when it tries to start the service, it fails and gives a red error icon in the Application log. When you click this you'll see a message, "Only one ChorusHub can be run on a network but there is already one running on …" The message will list the machine that is currently running Chorus Hub. The service will remain installed and set to Automatic. Thus when the computer is restarted it will try to start again and will succeed if the other Chorus Hub is no longer running. Otherwise it remains off.

Chorus Hub 2.5 service has the same limitations with multiple IP addresses on the Chorus Hub machine. It currently doesn't communicate this via the event log, but will cause a crash when someone tries to access Chorus Hub with an error report similar to this:

Msg: There was an error on the Chorus Hub Server, which was transmitted to the client.
…..
**Inner Exception:
Msg: Could not connect to net.tcp://192.168.196.1:5912/. The connection attempt lasted for a time span of 00:00:20.9952009. TCP error code 10060: A connection attempt failed because the connected party did not properly respond after a period of time, or established connection failed because connected host has failed to respond 192.168.196.1:5912

See Section 4.1 for further information on how to solve this.

2.5 LanguageDepot Internet startup

LanguageDepot is an Internet server (http://languagedepot.org) that hosts repos for various programs including FieldWorks. As of FieldWorks 8.0.6, in order to use the Internet option, a repo needs to be set up on the server prior to your use.

The first step is to have user accounts set up for each user that will be working on your project. You can do this by going to the site and clicking the Register link at the top right. You'll need to fill in a Login (preferably firstName_lastName) and enter a Password. Do not use a password you use for other secure operations since the password is not encrypted as it is used. You will need to use this information when you set up Internet Send/Receive from your FieldWorks project. Fill out the other required fields and then click Submit.

Next, you need to e-mail admin@languagedepot.org to have the repo created. Let the administrator know whether you need a FLEx repo and/or a LIFT repo for your work, and give the vernacular language and ISO-639 code (see www.ethnologue.com) used for your project. Give the user Login name of the manager for this project. The administrator will then send back a project id that you'll need to use in Send/Receive setup.

A project can have one or more managers. A manager can sign in to http://languagedepot.org using their user login and password. They can then type in their project name in the search box and click on their project link to access information on their project repo. From the Settings tab and the nested Members tab, any manager can add other users as Manager, Contributor, or Observer. A manager or contributor can send and receive changes in the repo. An observer can get the project, but cannot make changes. If you click the outer Repository tab for your project, you can see information on each S/R to your repo. Clicking the number at the left will list files

that were modified for that S/R. Clicking a file link will attempt to show changes that were made to that file, although this may be very slow due to the size of many files and speed of the Internet. There are better ways to get detail on the repo if you need it. The Tortoise Hg Workbench will be described later.

3 How it works, or why it doesn't

For the most part, changes from all colleagues will be merged successfully. However, remember the program has definite limitations when merging results between users. If you don't consider these limitations, you may get results that are baffling and disappointing.

When users change different things in the project, the changes will merge without problem. However, when users modify the same thing, this results in a conflict that needs to be resolved. To keep the S/R process functioning quickly, we don't want to stop and ask the user about each conflict before finishing the S/R. So instead, the process picks one of the two conflicting changes and then adds a merge conflict report describing the conflict and what the program did to resolve it. Most common conflict descriptions will be clear to the user. However, there are some things that can change that are internal to the FieldWorks model, and there isn't a simple way to explain this to the user, so it may just show some underlying XML in the conflict report in these cases. Sometime after the S/R, the user can investigate the conflict reports and decide whether the program's guess was good or not, and fix any that were incorrect. At this point it requires a manual edit to fix any conflicts that were resolved incorrectly. The conflict report has a link that will try to take you to the spot that changed in FLEx. Technically, if there is a conflict, the person doing the merge will win (except for deletions described later), but because it's not easy to determine which user will actually initiate the merge for a given object, it's best to assume that the winner is random. A conflict report is displayed after a S/R that results in conflicts. You can also see this report using Send/Receive…View Project Messages (for a FLEx repo) or Send/Receive...View Lexicon Messages (for a LIFT repo).

Conflict information is stored in special XML files (*.ChorusNotes) that are merged along with the data. The user can mark conflicts as resolved so they don't normally show in the conflict dialog, but there isn't any way in the program to actually delete the conflicts. So if you get a huge number of conflicts, these conflict files can get very large.

3.1 Lexicon examples

Merging entries is not based on the headword, but on an internal unique identifier (id) that is assigned to an entry when it is created. The reason for this id is that headwords do not uniquely define entries since you can have homographs, and homographs are not unique because they can be renumbered or adjusted when switching to a different writing system, etc. Entries can also have the same headword but have a different morpheme type (prefix, suffix, root). Headwords can also be misspelled, but when corrected, we don't want the entry to be considered a different entry, especially when lexical relations and interlinear text depend on it. The computer needs something that is created once when an entry is created and it will never change for the life of the entry. The technical name for this is a Globally Unique Identifier (guid). Anything inside FLEx that references this entry uses that id.

If I create an entry for 'house', and my colleague also creates an entry for 'house', each one will have a different id, so when we merge our projects using S/R, the result will have two entries for

'house'. Duplicate entries can be merged one at a time using the Tools...Merge with entry option, but it doesn't happen automatically. If two users take a list of words and both add them to the lexicon, after they are merged, as far as humans are concerned, there will be duplicates of every word. The computer knows they have different ids, so it thinks it did a great job of merging the entries. On the other hand, if one user adds words from A-M and another adds words from N-Z, and they merge their projects, everything will be great since the human and computer agree that these are different entries.

Likewise, the computer assigns a unique id for every sense that is created within an entry. Again, there isn't any other unique way a computer can identify a sense because some senses may have the same or missing gloss, definition, part of speech, etc. If I added a new sense for 'house' with a gloss 'political body', and my colleague did the same on his machine on the same entry, when we merge the projects, 'house' is going to have two identical senses from a human perspective, but the computer considers them different because they have different ids. These duplicate senses can be merged one at a time using the Merge Sense into menu option on a sense, but it is not automatic.

During the Send/Receive merge process, once the computer realizes two people changed the same entry or sense based on the id, then it will do the best it can at merging the changes within that 'object'. At least it knows it is working with the same entry or sense. If I add example sentences to senses, and my colleague adds semantic domains to senses, when we merge, the result will be exactly what we want. Likewise if I add Spanish glosses and my colleague adds French glosses, the merge will again be flawless.

If I change a gloss on a sense and my colleague doesn't do anything with that gloss, then when we merge, my change will be made in both projects. However, if I change a gloss for a sense, and my colleague changes the same gloss to something else, now when we merge we have a conflict, so the program will pick one change and reject the other, then display a merge conflict report after the merge.

3.2 General concepts

In FLEx, every 'object' has a unique id. Objects are defined in the underlying model of the data. Some examples of objects are lexical entries, senses, example sentences, etymology, pronunciations, allomorphs, grammatical info, lexical relations, reversal entries, interlinear texts, wordforms, wordform analyses, word glosses, parts of speech, notebook records, list items, etc. In WeSay only entries and senses have unique ids. The merge process basically adds any new objects, and merges the contents of existing objects with identical ids.

Now suppose I delete an entry or sense. As long as the other user does not modify this entry or sense, when we merge, the deletion I made will remain deleted. However, if my colleague changed something on that same entry or sense, the merge process doesn't want to potentially lose some important work, so the object I deleted will remain after the merge with the change my colleague made, and a merge conflict report will be added. So after the merge I may think the S/R messed up because the thing I deleted came back again.

When two or more people are working on a project, the longer the interval between S/R, the bigger the chance of having many merge conflicts. Naturally, if multiple users make a lot of changes to the same fields of the same entries or senses, a lot of merge conflicts will also result. So if you are working in the same areas, more frequent Send/Receives will be helpful. If you

plan to do extensive bulk editing or orthography changes, you can avoid a lot of conflicts if everyone does a S/R and then stops work until the extensive changes are made, then they all do another S/R before continuing their work to get the bulk changes.

3.3 Interlinear examples

Interlinearization in FLEx is complex internally because the baseline text paragraph has pointers to wordforms in the word analyses section and analyses in these wordforms have pointers to lexical entries, categories, and senses. All of these objects in the project have unique IDs (guids) that are assigned when the object is first created. During Send/Receive (S/R), if any of these objects are added or changed, FLEx attempts to merge the changes the best it can. When two or more colleagues do interlinearization or text charting, especially on the same text, and use Send/Receive (S/R) to merge their work, the results may be disappointing.

Here are some guidelines to avoid or reduce the problems.

- Editing baseline text, adding/removing final punctuation, and changing paragraph breaks should only be done by one user during an S/R cycle. Colleagues should sync as quickly as possible following these changes. Otherwise merging can produce very poor results with duplicated paragraphs, etc.
- Avoid more than one person adding new entries for the same word in a S/R cycle. Otherwise you'll get homograph entries where there should be one. You'll need to use Merge Entry and Merge Sense menu options to remove the duplicates.
- Avoid more than one user adding a new analysis to the same wordform in the same S/R cycle. Otherwise you'll get duplicate analyses where there should be one. Tools..Utilities...Merge Duplicate Analyses may eliminate these duplicates.
- Avoid more than one person working on the same text chart in the same S/R cycle, otherwise one user's work will likely be lost.
- It's safest for only one user to modify a specific interlinear text in a S/R cycle.
- In general use S/R frequently so all colleagues receive new data created by others, thus avoiding undesirable duplicates and merge conflicts resulting in lost data.

What is an S/R cycle? After a full cycle, all colleagues have the same content on their machines. Consider colleagues A, B, and C. If A wants to make extensive changes with a minimum of merge conflicts involving losses, before making the changes, B should do an S/R which merges his changes with what's in the repo and then stops working. C then does S/R to merge his changes with what's in the repo and stops working. A then does S/R to get all changes from B and C. A then makes the extensive changes, then does S/R to send them out to the repo. B and C both do S/R to get the accumulated changes and then continue their work.

A full cycle is rather disruptive for work, especially if colleagues are scattered around the world. A less rigorous S/R cycle doesn't require stopping work, but it basically involves each user doing two S/R operations. If A and B are both working, when A does the S/R his changes are now on the repo. The next time B does S/R, he'll pick up A's changes and put the merged changes out to the repo. Then the next time A does a S/R, he'll pick up the changes that B made and again merge his ongoing work to the repo. When these S/R cycles are done fairly frequently, newly

created items from one user become available to the other user without the other user creating new objects that then cause duplications that need to be cleaned up.

When the baseline is modified, FLEx attempts to update existing links to keep from losing data. With small changes in the baseline, it can usually keep up. But with large changes, it may not be possible to keep existing links, so the interlinearization linkages may be lost. Even if linkages are lost, the wordforms, analyses, and lexical entries are still present, so fixing any losses is much faster than starting from scratch.

Suppose two users simultaneously (e.g., in one S/R cycle) interlinearize the same word in the same text location for the first time. Both users create a new lexical entry with different guids. Both users add a new analysis with different guids to the wordform. When FLEx merges the results, both lexical entries remain, so they now have homographs. The original wordform will now have two analyses, one going to the one homograph, and the other going to the other homograph. The paragraph in the text has segments for each sentence (typically), and this segment has a pointer for every word in the sentence either to a wordform, if unanalyzed, or an analysis, or a word gloss when it is fully analyzed. But since each user created a new word gloss during the analysis, and the segment pointer can only point to one, the merge picks one and then adds a merge conflict stating that two users edited the same part of the data (related objects in Analyses), and the program had to pick one.

So the result is a lot messier than we would like, but it would be difficult for the program to do any better considering the complexity of the objects being merged. To clean up the lexicon, you would need to go to one homograph and choose Merge Entry and merge with the other homograph. Then since that results in two senses in the merged entry, you need to go to one of those and choose Merge Sense Into and choose the other sense. Merging the entry and senses will remove the linkage from one of the wordform analyses, so the other one can be deleted. The simplest way is to choose Tools...Utilities...and run Merge Duplicate Analyses. This will merge any duplicates in all wordforms and the result in this case is the unused analysis is deleted. The merge conflict can be ignored since there are no longer two analyses anyway. So now the data is as clean as if only one user had analyzed this new word.

If the same word was processed in different texts simultaneously, it would still produce homograph lexical entries and duplicate word analyses, but there would be no merge conflict in this case because the changes were in different paragraph segments. But the manual cleanup work remains the same. However, if one user did the analysis, and did S/R, and the other user did S/R to get the results, then did the analysis, all they would have to do is pick the analysis that is already there from the first user and there would be no problems to deal with.

If two users simultaneously add a new paragraph to the same text and type the same sentence and interlinearize it, there are additional problems when merged. If all of the words were new words, as soon as they are typed, new wordforms will be created with different ids but the same form. So in addition to lexical homographs, the merge will find two wordforms with the same form. Since FLEx attempts to avoid having more than one wordform with the same form, the merge will merge these two wordforms together, but keep the original analyses, so we'll basically have the same result as described above. But each new paragraph created by the users will have a different guid, and rather than attempting any merge of paragraph data, the merged result will now have two paragraphs with the same data. There will also be a merge conflict report because two users added a new paragraph at the same place, but FLEx didn't know which one should be

first. So to clean up this situation, you would need to delete the duplicate paragraph from the text in addition to cleaning up lexical homographs and duplicate word analyses.

An interlinear paragraph contains the baseline text. FLEx automatically breaks the baseline text into segments based on final punctuation (period, question mark, exclamation point, section sign). Each segment contains an offset into the baseline, and has pointers for every word in the segment that initially points to wordforms. During analysis these pointers are changed to analyses or word glosses within the wordform. The segment also holds the free or literal translation and any notes. As you edit the text in a paragraph, FLEx has to make adjustments to these offsets and pointers.

When final punctuation marks are edited in an interlinear paragraph, FLEx attempts to maintain the interlinear work you've done, which means pointer segments have to be readjusted, and segments may have to be split or merged. If they are split or merged, free translations will need to be edited to clean up the results.

When interlinear paragraphs are merged or split, FLEx adjusts segment information in an attempt to not lose any interlinearization. Merging paragraphs will end up with the original paragraph object being deleted after merging the data. Splitting a paragraph creates a new paragraph with a new guid.

When baseline text is edited in a paragraph, after a S/R that involves a merge, FLEx sets a flag in the paragraph indicating that the next time the text is opened, or the next time you go to Word Analyses or Concordance, then any paragraphs that have this flag set will be reparsed. Typically this process will not alter anything because the parse is already correct. However, if there were changes to the baseline, the merged data may have a mismatch between the baseline and the segment pointers. So the reparse in this case will need to make some adjustments to the pointers. This means the paragraph changes without you doing any editing. In S/R, if one person edits an object and another one deletes it, the deletion will be ignored in favor of the edit. In one case where a user merged multiple paragraphs together, and then later did S/R, the result was that all of the deleted paragraphs reappeared in the baseline without the original interlinearization because new segments were created. Merge conflicts were generated for each paragraph that had been modified warning that deletion were ignored. This was because the one user inadvertently modified the paragraphs while the other one deleted them as part of the merge. The solution in this case was to delete the paragraphs that reappeared in the merge.

Merges and deletions of analyses or word glosses in Word Analyses affect all interlinear texts that use the modified wordform. If another user makes use of the modified analysis or word gloss in the same S/R cycle, their analyses involving the modified wordform will likely be lost.

Because of all of these possibilities for conflicts when baselines are changed or paragraph breaks are changed, it's wise for only one user to work in a given text during any S/R cycle. It's also best to establish your paragraphs and ending punctuation (period, question mark, exclamation point, section sign) early on before doing interlinearization.

3.4 WeSay/LIFT collaboration

When collaborating between multiple FLEx users and one or more WeSay users, it's critical that only one FLEx user sync with the LIFT repo. This is because the logic for merging gets fuzzy when dealing with lexical relations, variants, complex forms, example sentences, etc. since these

are not simple strings, but objects similar to entries or senses. In FLEx these each have unique ids, but the LIFT file used by WeSay does not record the unique id for these. Every time a S/R happens between a LIFT repo and FLEx, for any new objects other than entries and senses, FLEx will give them a unique id. If multiple FLEx users sync with the same LIFT repo, each FLEx will have different ids for the same object, which would result in duplicate objects when the FLEx users merge via the FLEx repo.

Any program that uses LIFT files is supposed to read and modify what it understands, and leave the rest of the information intact for other programs that need it. FLEx uses a hidden LiftResidue field on several objects in the lexicon to store information that isn't in the FLEx model, so that it can be exported back to the LIFT file without being lost. Any time you import a LIFT file, every entry and sense will have a LiftResidue field added to hold this information. If you are only using FLEx to FLEx collaboration, the LiftResidue fields are excess clutter that can be deleted.

3.5 FLEx/ParaTExt collaboration

If a ParaTExt (PT) project is associated with a FLEx project, and you are collaborating with other users via PT Send/Receive (S/R), and the other users also have FLEx installed and are collaborating using FLEx S/R Project, and FLEx users are doing anything in the Texts & Words area, there are some cautions that need to be taken to avoid corrupting FLEx data.

A PT administrator is the only one that can associate a shared PT project with a FLEx project. Once associated, PT can access lexical data from FLEx, and FLEx can access current PT data in the FLEx Texts & Words area. The association is stored in the PT project settings file with the name of the associated FLEx project. These settings will go to colleagues during PT S/R. If the colleagues have a FLEx project with the same project name, they will automatically be connected with the PT project on their machine. If these users also use FLEx S/R Project to keep the FLEx projects in sync, there is potential for problems if they also use the Texts & Words area in FLEx.

Important! When you are using S/R with FLEx and PT, one user should load all of the data from PT, then go to Word Analyses. After doing this, he should S/R Project in FLEx. Then all other FLEx collaborators should do S/R Project to get all of this new data before they open the Choose Texts dialog in Texts & Words...Interlinear Texts. After this initial S/R cycle, it will work best if collaborators sync FLEx and PT projects at the same time. When new books are added to PT or significant changes are made, again, you should go through this same process with a single FLEx user initially getting the changes and then passing them around via S/R.

As with any interlinear text in FLEx where users are collaborating with S/R, you should avoid more than one user making changes in a given book during a single S/R cycle. For Scripture books, each section is treated as an interlinear text, so this caution would apply to each section in a scripture book.

Here is some technical detail to understand what is happening and what can go wrong.

When scripture is first loaded from PT by opening the Choose Texts dialog in FLEx and selecting scripture, for the books that are checked, it creates ScrBooks, ScrSections, ScrFootnotes, StText (for titles and headings), ScrTxtParas with ParseIsCurrent set to False, and Segments without Analyses. As part of the import, a checksum is made of the PT file and stored in ImportedCheckSum of ScrBook. Once loaded, when a user opens the Choose Texts dialog, if

the stored checksum matches the current PT checksum, then FLEx will not reimport that book. If the book is modified in PT, and a portion of that book is included in the current Text & Words area, when the user goes to the Choose Texts dialog, FLEx will recognize that the PT book has changed, so it reimports the book, trying to maintain any interlinearization that has been done on that book. These checksums are included in FLEx S/R.

When you click a scripture text in the Texts pane (actually a ScrSection, ScrFootnote, or StText for titles and headings) that text is parsed creating WfiWordforms and PunctuationForms as needed, setting default Analyses on Segments, and setting ParseIsCurrent on ScrTxtParas to True.

If you go to the Word Analyses (or Concordance) tool, it will parse all texts and scripture that is currently imported, creating WfiWordforms and PunctuationForms as needed, setting default Analyses on all Segments used in those texts, and setting ParseIsCurrent on all ScrTxtParas to True.

All of the objects above, except Analyses which are initially pointers to WfiWordforms or PunctuationForms, have unique IDs. So if two FLEx users import the same books in the same S/R cycle, there will be a lot of duplications that would be hard to clean up after a S/R merge. FLEx does not provide a way to delete duplicate scripture books or sections that get imported in this way. Although wordforms have unique IDs, they will actually merge during S/R if the form is the same, so that helps some.

The duplicated portions can result in the load process failing. This will continue until the book is cleaned up enough for the import process to work. With Translation Editor (TE) you could run a verse chapter check on a bad book which will usually show serious problems in the data. If you clean these up enough in TE, you should get to a point where the import will work again.

If something is messed up too badly, it may be necessary to delete a book entirely. This is best done using FDOBrowser. This is a low-level utility program that is in c:\Program Files (x86)\SIL\FieldWorks 8 directory. You should not have FLEx or TE open when using this program. To delete a book of scripture, you can type FDOBrowser at the Windows start command and it should find the program.

Once FDOBrowser is open, go to File…Open Language Project. In the open dialog, you'll have to navigate to your project *.fwdata file. Project directories are normally under c:\ProgramData\SIL\FieldWorks\Projects\. After the project is open, you'll see two other tabs. Click the LangProj tab then scroll down to TranslatedScriptureOA and click the + to the left to expand the node. Under the TranslatedScriptureOA node, open the ScriptureBooksOS node. You'll see a node for each book that has been imported from PT. To delete a book, right-click the book and choose Delete. When done, do File…Save Current Language Project and then close FDOBrowser. FDOBrowser does autosaves the same as FLEx.

Various things can go wrong when users sync FLEx and PT at different times and the user goes to the Choose Texts dialog. For example, if user A imports scripture and then does FLEx S/R, his import and checksum will go to user B. But if PT is not in sync with users A and B and user B does FLEx S/R and opens the Choose Texts dialog, they will reimport the scripture from B's version of PT data, and conflicts will likely happen the next time they do FLEx S/R with user A. Currently, the FLEx merge conflicts for scripture typically give information that is difficult even for technical people to understand. Also, the merge conflict hotlinks into FLEx do not work for

scripture at this point. So it's best if colleagues work in a way that reduces or eliminates scripture merge conflicts.

By following the Important paragraph above, most conflicts will be avoided. If one user imports books from PT and then does FLEx S/R and PT S/R, and another colleague does FLEx S/R and PT S/R, their FLEx and PT projects will normally be in sync so that nothing will be reloaded by going to the Choose Texts dialog. However, if multiple users are importing scripture in one S/R cycle, the FLEx S/R merging capability may have trouble sorting everything out.

Interlinearization on Scripture is stored with each paragraph in a section. The entire section is loaded into FLEs as an interlinear text. When FLEx imports from PT, it will update the baseline text and then try to maintain the interlinearization of the revised text. If you make a lot of changes to paragraphs, and add and remove sections heads in PT, there is a good possibility that FLEx will not be able to keep up with the changes which means interlinearization may be lost. Also, if you delete a scripture book as described above to get past a messed up book, you'll lose all interlinearization in that book.

When we talk about interlinearization being lost, it doesn't mean everything is lost. There are 3 places that are affected by interlinearization. First, you may create a wordform in the wordform inventory and add analyses to the wordform. Second, these analyses are connected to entries and senses in the lexicon, which may involve creating entries or senses in the lexicon. Third, in the interlinear text itself, for each wordform in a paragraph you are linking to one of the analyses in the wordform inventory. When we talk about losing interlinearization, what we are actually losing is the pointers to specific analysis in wordforms which means it will default to the wordform. So the actual work in creating wordforms, analyses, entries, and senses are not lost and are still available when doing future interlinearization. So although you do lose work by deleting a scripture book (or interlinear text), recovering should be much faster because everything you previously created in the lexicon and wordform inventory is still there and you simply have to link the wordforms again. FLEx will propose analyses for the entire text, so the proposals can either be approved immediately, or adjusted to get the correct analysis.

The process of associating a PT project with a FLEx project was designed for one FLEx project being associated with one PT project. If you happen to have multiple PT projects for a single FLEx project, you should stick to associating a single PT project to the FLEx project. If you switch the associations around between different PT projects, it will likely confuse the process resulting in corrupted Scripture and loss of interlinearization.

3.6 Linked files

Pictures, sound files, and other linked files are included in the S/R process as long as they are stored under the default LinkedFiles directory inside the FieldWorks project directory. FLEx allows you to use an external directory for linked files which is advantageous if you have multiple projects that refer to a master set of pictures and sound files. But if you have chosen this approach, they will not be included in S/R. Also, because repo size and time for S/R, especially to the Internet, can become too great if you use high resolution pictures, sound files, movies, etc., the S/R process currently limits files to 1 Mb, and only accepts certain file extensions. For images, it accepts these extensions: bmp, jpg, jpeg, gif, png, tif, tiff, ico, wmf, pcx, and cgm. For audio, it accepts these extensions: wav, snd, au, aif, aifc, aiff, wma, and mp3. Anything that doesn't meet these requirements is skipped during S/R. A warning message will be given in the

S/R log if files are greater than 1 Mb. Files with doc and txt extensions are included, but mp4 files are not.

3.7 FieldWorks version

Each version of FLEx works with a single underlying data model. Any version of FLEx can share projects from other versions of FLEx as long as the data model is the same. When a user upgrades FLEx to a version that uses a different data model, in any project he opens, data will be migrated to the new model. At this point users with older versions of FLEx will no longer be able to open the project until they upgrade to a FLEx version that supports that data model.

When using S/R for a FLEx project, all collaborators should use the same version of FLEx, or at least versions that use the same data model. You can see the model version in a FieldWorks project directory involved in S/R by opening the file, FLExProject.ModelVersion in a text editor. It contains a version, such as 7000070). For any project, the data model can also be seen in the second line of the fwdata XML file. Recent data models and associated versions of FLEx are
7000068 FW8.0.4 – FW8.2.9
7000069 FW8.3.1
7000070 FW8.3.2 –

FLEx stores data in the mercurial repo in a unique branch for each data model. A repo can have different branches, and support users in each branch, but data will not be merged or made available between branches until the last colleague upgrades to the most recent version.

Thus if users A, B, C, and D are collaborating in version FW8.3.1 everything works fine. If user A upgrades to FW8.3.2 and does a S/R the repo now has two branches. If user B also upgrades to FW8.3.2 and does S/R, A and B can now collaborate with each other, and C and D can collaborate with each other, but they can't see any work done by the other two users. If C upgrades, his work as well as the work merged with user D will merge with A and B so that all three of them can collaborate. Once D upgrades and does S/R, the two branches will be merged so that everyone will again be working on a single branch.

For this reason it is best for collaborators to upgrade about the same time so that they can continue to see each other's work.

3.8 FieldWorks project name

Repos also have a unique id that is assigned when it is first used. The id stays with the repo when you do a S/R to different locations, or copy the repo. This is used to ensure that you are really working with the correct repo. During a S/R the program uses this repo id to find the project in the FieldWorks Projects directory. If one user changes the name of their FieldWorks project, they can still S/R with the same repo because the repo id hasn't changed. For this reason, if you want to receive a new version of the repo, you can't just rename the old folder or give it a different name in the Receive dialog. In order to do this, you need to actually move the old project folder outside the Projects folder, or move it inside a subfolder inside the Projects folder so that it won't cause a conflict. This also means that different colleagues can specify their own name to a project when it is received the first time, although this could be rather confusing.

4 Technical details

4.1 Chorus Hub and virtual machines

Chorus Hub 2.4 will give continuous warning messages, “this machine has more than one IP address”, and may or may not work if it detects more than one IP address on your machine. Chorus Hub 2.5 is unable to display these warning messages, but has the same limitations. Instead, when a user attempts to access Chorus Hub 2.5 under these conditions, they will get a crash with a message similar to the following:

Msg: There was an error on the Chorus Hub Server, which was transmitted to the client.
…..
**Inner Exception:
Msg: Could not connect to net.tcp://192.168.196.1:5912/. The connection attempt lasted for a time span of 00:00:20.9952009. TCP error code 10060: A connection attempt failed because the connected party did not properly respond after a period of time, or established connection failed because connected host has failed to respond 192.168.196.1:5912

If the problem is due to a virtual machine, and you really need to use Chorus Hub on the virtual machine or on the machine that is running a virtual machine, there are ways to make this possible. Here are some notes for using Oracle VM VirtualBox on Windows 7. There are probably similar things that can be done with other virtual machines.

Virtual Box defaults to creating an IP address for the virtual machine. The default attachment is NAT which is a separate IP address that is not available outside the virtual machine. If you need to run Chorus Hub in the virtual machine, you can change this setting to Bridged Adapter. In this mode it uses the same IP as your main machine. So when you run Chorus Hub inside the virtual machine, users outside the virtual machine will be able to use it.

If you want to run Chorus Hub on a machine that is using a virtual machine, you’ll need to disable the IP address for the virtual machine. You can do this using Control Panel, Networking and Sharing. Click “Change adapter settings”, then right-click the Virtual Box Host-Only Network and choose disable. Now Chorus Hub will run on the main machine without complaining about extra IP addresses. You can still access the Chorus Hub running on your main machine from within the virtual machine.

4.2 Using FieldWorks backups and Send/Receive

Making FieldWorks backups is not as critical when using Send/Receive Project because the S/R process is storing a type of backup on the repo. If your machine should fail or something happens that makes your version of FLEx unusable, you can delete your FLEx project directory and start over again by using Get Project from colleague. This will restore your project to the current state of the repo.

If you are only using S/R Lexicon, then the repo only stores the lexical data, but none of the other data that may be in your FLEx project. So in this case, you should definitely do regular FieldWorks backups to prevent data loss in case of some emergency.

While making FieldWorks backups is perfectly safe when using S/R, you normally should not do a restore from a backup when using S/R. For one thing, if you restore your project from a backup

and then do S/R, it will basically set all other users back to your restored state, thus cancelling all work they have done since the state of the project when the backup was made.

Also, a FieldWorks backup does not include the LIFT or FLEx repo that is normally inside your project directory. So if your project directory gets deleted, when you restore from a backup, you are now disconnected with both the LIFT and FLEx repos.

If you are using S/R Project, instead of restoring from a FieldWorks backup, you should delete your directory and use Get Project from colleague to restore your project. Then it would be in sync with the repo. If you have made a FLEx backup since your last S/R, the next section describes how you can restore from this.

If you are using S/R Lexicon, and you need to restore your project from a FieldWorks backup, then you should use Get Lexicon (WeSay) and Merge with this Project. But keep in mind that this will probably cause a loss of the WeSay work since the time the FieldWorks backup was made.

4.3 Restoring a FieldWorks backup

Restoring from a backup is usually not a good thing when using S/R, so to prevent accidentally losing a lot of data, the program currently refuses to let you restore to the project, and forces you to restore to a different name. Say your original project is QQQ and you restore to QQQ-01. There isn't a direct way to get QQQ-01 hooked back up to S/R. The only way you can get back to S/R with the restored data is to copy the QQQ-01.fwdata file from the QQQ-01 project to the QQQ project and rename it QQQ.fwdata, replacing the bad QQQ.fwdata and then continue S/R from the QQQ project. This will be safe to do IF the backup from which you restored was made since your last S/R. Otherwise this would likely cause loss of data for everyone on the project.

The normal way to recover if you mess up the project in a way that you don't want to pass on is to delete your QQQ project and then do Get Project From Colleague to get back to a normal S/R project. Note, if you have QQQ-01 in your Projects directory, you will not be able to do Get Project From Colleague because it searches all projects to see if any have the same unique project id stored in the fwdata file. This project id will be the same if you restore a project from a backup or rename the project. So you need to make sure there is no project in the projects directory that was made at some point from the project you want to download.

If you are going to do something potentially dangerous to a project that is connected with S/R, one approach is to first make a copy of the fwdata file in the project directory using Windows Explorer. If you need to restore, just delete the fwdata file and rename the copy back to the original name. But again, this should only be done if you haven't done S/R since the time you made the backup.

4.4 Viewing Send/Receive history

FieldWorks provides a very limited view of history when doing S/R. If there is a merge conflict, a conflict report is generated and displayed after S/R completes. These conflict messages will continue to be displayed on succeeding S/R operations until the conflict is Resolved. Even after they are resolved, you can see them by going to Send/Receive…View Project Messages, and clicking the filter icon under Project Notes and turning on Show Resolved Items. These conflict reports attempt to show the data for both users and say which change the merge accepted. The

FieldWorks model is very complex in some areas such as interlinear text, and the merge conflicts are difficult if not impossible to understand in some cases. But this is the best we have at this point.

Send/Receive uses Mercurial repositories which maintain consecutive revisions for each S/R. When a merge is performed, there are actually two revisions. The first is what the user submitted prior to the merge, and the second one is the result after the merge. With appropriate tools outside of FieldWorks, you can see full details of what happened during each S/R operation, and who was responsible for each change. However, this usually gets very technical, and you'll need to understand the underlying data model to be able to interpret the results. http://software.sil.org/fieldworks/support/technical-documents/ contains several files that are helpful for understanding the model. "Conceptual model overview" is out of date at this point, but gives useful background information. "FieldWorks model diagrams" provides model diagrams that make things a little easier to understand, but it was last updated for FW7.2. "Current classes and fields" gives current detail on all classes and fields (properties).

Most of the data in a FieldWorks project is stored in a single .fwdata file directly under the project directory with the same file name as the project directory. The data is actually XML data, but in a form that is made for ease in loading and saving the project. It has a separate rt element for every object in the project that identifies the class of the object, an identifying guid, and the data that it contains. You can learn more about this file by reading https://software.sil.org/fieldworks/wp-content/uploads/sites/38/2016/10/FieldWorks-7-XML-model.pdf. Here's a short example showing how a simple entry is stored in fwdata for this entry: **maison** *n* house *Ils ont construit une maison en bois.* They built a wooden house.

```
<rt class="LexEntry" guid="41be6cd4-4d51-4137-9f2d-d0a66f84467d">
<DateCreated val="2018-10-19 16:41:42.916" />
<DateModified val="2018-10-19 16:42:22.635" />
<DoNotUseForParsing val="False" />
<HomographNumber val="0" />
<LexemeForm>
<objsur guid="93d68a3c-c31a-473f-917f-e539b93a8710" t="o" />
</LexemeForm>
<MorphoSyntaxAnalyses>
<objsur guid="c7208b5d-dbd1-4f13-a405-a660d50fd6ab" t="o" />
</MorphoSyntaxAnalyses>
<Senses>
<objsur guid="1f3d34f7-85d6-4cbf-af75-43405d0fb242" t="o" />
</Senses>
</rt>

<rt class="MoStemAllomorph" guid="93d68a3c-c31a-473f-917f-e539b93a8710"
ownerguid="41be6cd4-4d51-4137-9f2d-d0a66f84467d">
<Form>
<AUni ws="fr">maison</AUni>
</Form>
<IsAbstract val="False" />
<MorphType>
<objsur guid="d7f713e8-e8cf-11d3-9764-00c04f186933" t="r" />
</MorphType>
</rt>

<rt class="MoStemMsa" guid="c7208b5d-dbd1-4f13-a405-a660d50fd6ab" ownerguid="41be6cd4-
4d51-4137-9f2d-d0a66f84467d">
<PartOfSpeech>
<objsur guid="a8e41fd3-e343-4c7c-aa05-01ea3dd5cfb5" t="r" />
</PartOfSpeech>
</rt>

<rt class="LexSense" guid="1f3d34f7-85d6-4cbf-af75-43405d0fb242" ownerguid="41be6cd4-
4d51-4137-9f2d-d0a66f84467d">
```

```
<Examples>
<objsur guid="b0432314-b417-4f51-a6c3-8064b34d3978" t="o" />
</Examples>
<Gloss>
<AUni ws="en">house</AUni>
</Gloss>
<MorphoSyntaxAnalysis>
<objsur guid="c7208b5d-dbd1-4f13-a405-a660d50fd6ab" t="r" />
</MorphoSyntaxAnalysis>
</rt>

<rt class="LexExampleSentence" guid="b0432314-b417-4f51-a6c3-8064b34d3978"
ownerguid="1f3d34f7-85d6-4cbf-af75-43405d0fb242">
<Example>
<AStr ws="fr">
<Run ws="fr">Ils ont construit une maison en bois.</Run>
</AStr>
</Example>
<Translations>
<objsur guid="df53c7a6-47f3-4e8d-aee6-3aca2e50bd9a" t="o" />
</Translations>
</rt>

<rt class="CmTranslation" guid="df53c7a6-47f3-4e8d-aee6-3aca2e50bd9a"
ownerguid="b0432314-b417-4f51-a6c3-8064b34d3978">
<Translation>
<AStr ws="en">
<Run ws="en">They built a wooden house.</Run>
</AStr>
</Translation>
<Type>
<objsur guid="d7f7164a-e8cf-11d3-9764-00c04f186933" t="r" />
</Type>
</rt>
```

Note that there are individual elements (class instances) for the main entry, the lexeme form, the grammatical category, the sense, the example sentence, and the example translation. The individual classes are connected using guids. The defining objects for the morph type, the part of speech, and the translation type are not shown here.

The .fwdata file is much too large and complex to process efficiently in S/R. To keep it smaller and more manageable in merging, the fwdata file is split into 70 or more individual files. In these files, the data is stored in a hierarchical form of XML that keeps objects together rather than breaking it down into individual class elements. Some parts, such as the lexicon and wordform inventory, are broken into 10 separate files to keep them smaller. This is how the lexical entry shown above would be represented in one of the 10 lexdb files.

```
<LexEntry
  guid="41be6cd4-4d51-4137-9f2d-d0a66f84467d">
  <DateCreated
    val="2018-10-19 16:41:42.916" />
  <DateModified
    val="2018-10-19 16:42:22.635" />
  <DoNotUseForParsing
    val="False" />
  <HomographNumber
    val="0" />
  <LexemeForm>
    <MoStemAllomorph
      guid="93d68a3c-c31a-473f-917f-e539b93a8710">
      <Form>
        <AUni
          ws="fr">maison</AUni>
      </Form>
      <IsAbstract
        val="False" />
      <MorphType>
```

```
        <objsur
          guid="d7f713e8-e8cf-11d3-9764-00c04f186933"
          t="r" />
      </MorphType>
    </MoStemAllomorph>
  </LexemeForm>
  <MorphoSyntaxAnalyses>
    <MoStemMsa
      guid="c7208b5d-dbd1-4f13-a405-a660d50fd6ab">
      <PartOfSpeech>
        <objsur
          guid="a8e41fd3-e343-4c7c-aa05-01ea3dd5cfb5"
          t="r" />
      </PartOfSpeech>
    </MoStemMsa>
  </MorphoSyntaxAnalyses>
  <Senses>
    <ownseq
      class="LexSense"
      guid="1f3d34f7-85d6-4cbf-af75-43405d0fb242">
      <Examples>
        <ownseq
          class="LexExampleSentence"
          guid="b0432314-b417-4f51-a6c3-8064b34d3978">
          <Example>
            <AStr
              ws="fr">
              <Run
                ws="fr">Ils ont construit une maison en bois.</Run>
            </AStr>
          </Example>
          <Translations>
            <CmTranslation
              guid="df53c7a6-47f3-4e8d-aee6-3aca2e50bd9a">
              <Translation>
                <AStr
                  ws="en">
                  <Run
                    ws="en">They built a wooden house.</Run>
                </AStr>
              </Translation>
              <Type>
                <objsur
                  guid="d7f7164a-e8cf-11d3-9764-00c04f186933"
                  t="r" />
              </Type>
            </CmTranslation>
          </Translations>
        </ownseq>
      </Examples>
      <Gloss>
        <AUni
          ws="en">house</AUni>
      </Gloss>
      <MorphoSyntaxAnalysis>
        <objsur
          guid="c7208b5d-dbd1-4f13-a405-a660d50fd6ab"
          t="r" />
      </MorphoSyntaxAnalysis>
    </ownseq>
  </Senses>
</LexEntry>
```

When we do S/R, we use a process similar to this.

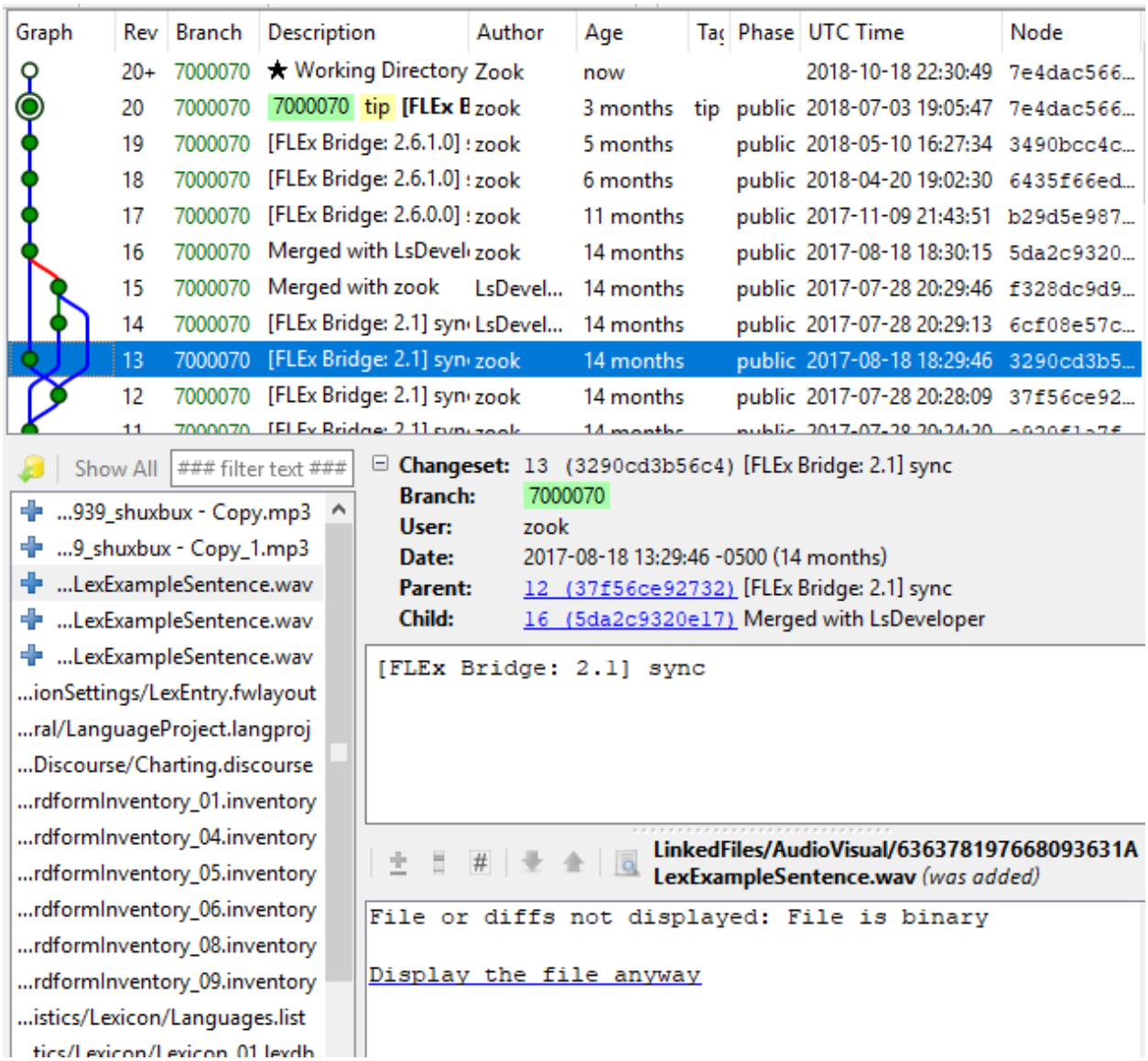
1. Split the single file into many smaller files with different extensions. These files are stored in Anthropology, General, Linguistics, and Other directories under the project directory.

2. Commit any files that have changed in these directories as well as ConfigurationSettings, WritingSystemStore, and LinkedFiles directories to the local repository (.hg directory under your project folder)
3. Get outstanding changes from LanguageDepot (LD), ChorusHub (CH), or USB, depending on the user's choice.
4. If the same file is changed by both users, we call a special merge process that understands the XML data in that type of file and merge the incoming results with the current changes.
5. Merge all of the individual files back into a single .fwdata file.
6. Run validation code to fix any resulting problems. (repeat split and merge if changes are made)
7. Commit changes to the local repo.
8. Send the merged revision with file changes back to the remote repo.
9. Reopen FLEx with the new data.

This process is controlled by FLEx Bridge (FB) and Chorus that are designed for our specific version of mercurial, and our special XML data. The mercurial program we use is installed with FLEx Bridge in c:\Program Files (x86)\SIL\FLEx Bridge\mercurial (or c:\Program Files (x86)\SIL\FLExBridge3\mercurial for FW9 and FB3). You should never use standard mercurial merge programs for modifying a FieldWorks repository as it will likely cause data corruption or loss.

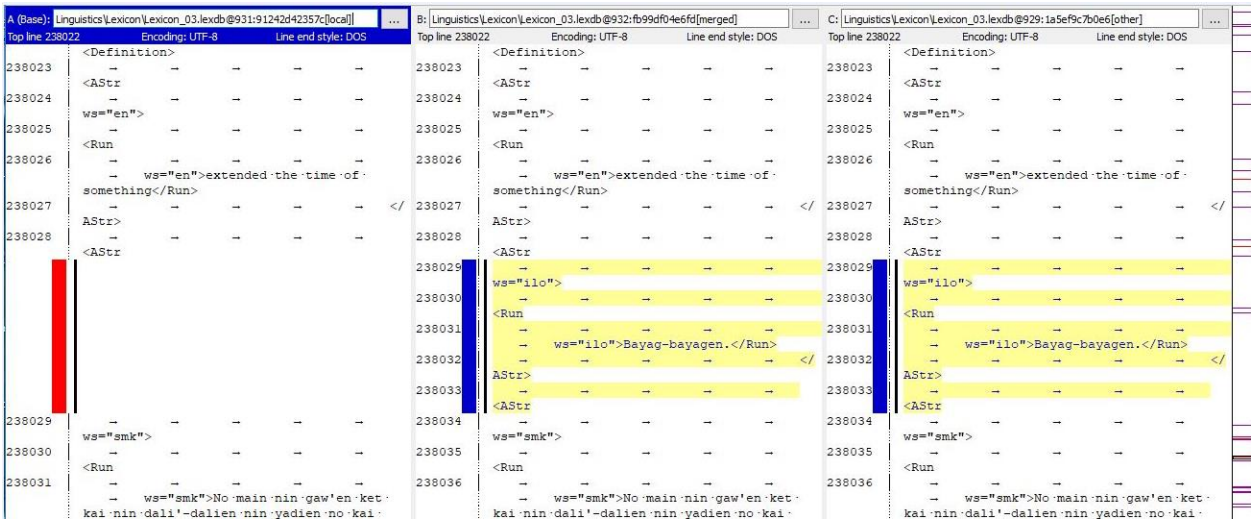
While it is not safe to modify a FieldWorks repository with other mercurial tools, it is fine to use other tools to look at the data in the repository. One of the best programs that is freely available for doing this is TortoiseHg which you can download from https://tortoisehg.bitbucket.io/. When comparing file versions, an excellent program is KDiff3, freely available from http://kdiff3.sourceforge.net/. In TortoiseHg you can set kdiff3 as the comparison program, then they work nicely together. You can also set the default editor to use to open a file. Notepad++ is an excellent program that is freely available from https://notepad-plus-plus.org/, although it can take a long time to open some of the larger FieldWorks files.

TortoiseHg is a Windows shell that allows you to explore a FW repository. To open a repository, from Windows File Explorer, go to C:\ProgramData\SIL\FieldWorks\Projects and then right-click your project folder and choose Hg Workbench. Here's an example window from Hg Workbench.



A revision contains all of the changes made in this particular S/R. Keep in mind that the Rev column gives consecutive numbers for a given instance of the repo. When revisions are merged with other repos the result may have different rev numbers. Each rev has a unique number that is shown in the Node column. This number will always be the same for each set of changes regardless of which repo you are observing. This is important when comparing several repos of the same project. The files on the lower left are individual files that were modified in this revision. There are various options in the right click menu. If you want to see the file in an editor, you can choose View at Revision. This will open the file in the specified editor. If you choose Save at Revision, it saves the file with a revision number appended. If you want to see what changed, you can right-click the file and choose Diff to Parent.

Here's an example using KDiff3 that shows a merge between rev 921 (right) and 931 (left) resulting in rev 932 (center). The user that submitted 921 added an Ilocano translation of the definition field, and this was added to the merged results.



Mercurial uses a 3-way merge. It keeps track of the common ancestor between two revs being merged so that it can determine who actually made a change since the common ancestor, and it tries to maintain that change in the merged results. If both users modify the same string, FLEx chooses one of the changes and adds a merge conflict report to notify the user what it did. If one user makes a change to an object while another user deletes that object, FLEx will always ignore the deletion, and it will add a merge conflict. Usually you can see the common ancestor from the Graph column in Hg Workbench, but a guaranteed way to find it is to use the following mercurial command from a Cmd window on the project folder.
"c:\Program Files (x86)\SIL\FLEx Bridge\mercurial\hg" debugancestor 23 35
where the two numbers are the revision numbers being merged.

As you can see, this is not a user-friendly way to see the changes that were made and who made them, but it's the best that is currently available.

4.5 Recovering lost data via S/R

The previous section summarizes the strategy FLEx uses in S/R, describing how you can see historical changes made to a project via the S/R mercurial repository. Suppose you would like to undo or roll back some of those changes. FLEx does not provide a built-in capability for doing this at this point, but with care, and using third party software, it is possible to roll back to an earlier version. At this point you should normally contact flex_errors@sil.org for help. However, if you have highly skilled personnel that have familiarity with FLEx data and mercurial, the following information will give some tips for them.

For a single user, rolling back is fairly easy. However, when you have multiple colleagues with outstanding work you don't want to lose, it becomes much more complex, especially if it involves interlinear text which touches multiple parts of the project. So the first step is to understand how many colleagues are involved, whether they are just trying to keep up with current changes, or whether they are making modifications to the data that have not yet been merged via S/R.

With mercurial, all colleagues have a complete local copy of the repo (.hg in their project folder) at the state it was in after their last S/R. There are also remote copies of the repo on LanguageDepot (LD), Chorus Hub (CH), and/or one or more USB drives. With USB drives, it's

easy to delete a repo directory to start over. With CH it's also fairly easy to delete a repo to start over as it only involves deleting the repo directory on the one machine on the network that is hosting CH. However, when deleting or changing a folder name under c:\ChorusHub, you should go to Start…Services and stop the Chorus Hub Sharing Service, then after making the change, start the service again. With LD, in order to reset the repo, you need to contact flex_erros@sil.org and ask an LD administrator to do this for you.

If S/R is done frequently, there will be minimal changes that would be lost if everyone had to go back to some earlier point. At most, they would lose the work they all did during one day. Suppose there are three colleagues, all syncing on a daily basis. Suppose one colleague does an S/R and discovers that data has been trashed in some way that would be hard to undo. If the other two colleagues have not synced since then, the damage at this point is limited to the first user's project and the remote repo. The safest and quickest way to recover in this case is to reset the remote repo and have a remaining colleague sync to the reset repo. Then the one that had the problem can delete their FieldWorks project directory and do Get from Colleague. This way only the one user would lose their work for the day. Trying to recover by rolling back a repo and getting everyone back in sync will probably involve more than a day with potential for getting things more messed up in the process. So most often it's quickest to take the hit on one user's work than to try to keep all work without causing further damage. Of course, no one wants to lose work, but once a bad merge has happened, the local and remote repos are both damaged as well as the user's FieldWorks data. Trying to pull out of a bad situation like this to save the one user's work will probably be more disruptive to the overall project.

Another fairly simple approach to resolving problems is to take one FLEx project as the new master copy and start everyone over at this point. To do this, the shared remote repo must be reset as mentioned above. Once the remote repo is reset, clear out the local repo in the master FLEx project. To do this, delete the .hg directory, and the Anthropology, General, Linguistics, and Other folders. (If you want to maintain merge conflict reports, only delete the .hg directory.) Now you can do Send Project for the first time, then have all colleagues delete their project directories and do S/R Get from Colleague to get started again.

Merge conflicts are stored in *.ChorusNotes files. There is one file, Lexicon.fwstub.ChorusNotes, that is directly under the project folder. This holds notes the users have entered via the Messages field in entries. All other ChorusNotes files in other directories under the project including ConfigurationSettings and WritingSystemStore directories hold merge conflict reports. You can delete these files at any time if you want to get rid of conflict history.

4.5.1 Repairing lost or damaged local repo

When a user does an S/R sync, their local repo will match the remote repo at that point in time. As the user works in FieldWorks, changes are being stored in the fwdata file, but nothing is happening to the repo including its associated split files until they do the next S/R. If the local repo (.hg directory) gets deleted or corrupted in some way, there are basically two choices. First, if changes since the last sync were insignificant, the safest thing is to delete the project directory and do Get From Colleague to get back to a healthy working copy. However, if they have done a lot of work that would cause a significant loss, then they need to get a new .hg directory at the state of their last sync. With the new local repo, they can then do a normal S/R which will merge their work as usual.

One way to get a new repo at a certain point is to start with a clone of the remote repo. One way to do this is temporarily move any projects that are based off this project to some directory outside of their projects directory, then do Get from Colleague and save it to some temporary name. Right-click this project folder and choose Hg Workbench to open the repo. Find the last sync/merge revision from the affected user. This point in the repo is what the user had before they lost their repo. So the goal is to remove everything from the repo after this revision. Say their last revision is 35. Now move your damaged project back under projects, replace the current .hg directory (if present) and copy the .hg directory from the temporary project, then open a Cmd window and execute these two commands.
"c:\Program Files (x86)\SIL\FLEx Bridge\mercurial\hg" strip 36
"c:\Program Files (x86)\SIL\FLEx Bridge\mercurial\hg" update –C 7000068
For FLEx Bridge 3, installed by FW9, the path to the hg program is c:\Program Files (x86)\SIL\FLExBridge3\mercurial\hg. The final number in the second command is the data model which comes from the second line of your fwdata file <languageproject version="7000068">.
The first command removes all revs from 36 and above, and the second command restores the split files in your project folder to their state at that revision level. In Hg Workbench on the repaired repo, if there are still other revisions with dates later than rev 35, repeat strip commands to get rid of these before executing the update command. At this point, your local repo has been restored to the state it was in after your last sync, so it should be safe to do a normal S/R which will include everything you've changed since that point in time.
Note, before mercurial will accept the strip command, you'll need to edit c:\users\<computerLoginName>\mercurial.ini and add these 2 lines:
[extensions]
mq =

A similar approach can be used to roll back a project to a certain revision. If you strip everything from the repo as described in this section and use the update command, you can then copy this stripped repo (.hg directory) to an appropriately named empty directory on CH or USB and then delete your current project folder and do Get From Colleague from the stripped repo. This will build a new Flex project from that point in history. Unlike the rollback described below using RepositoryUtility, this approach will not clutter the repo with extra revisions undoing the damage.

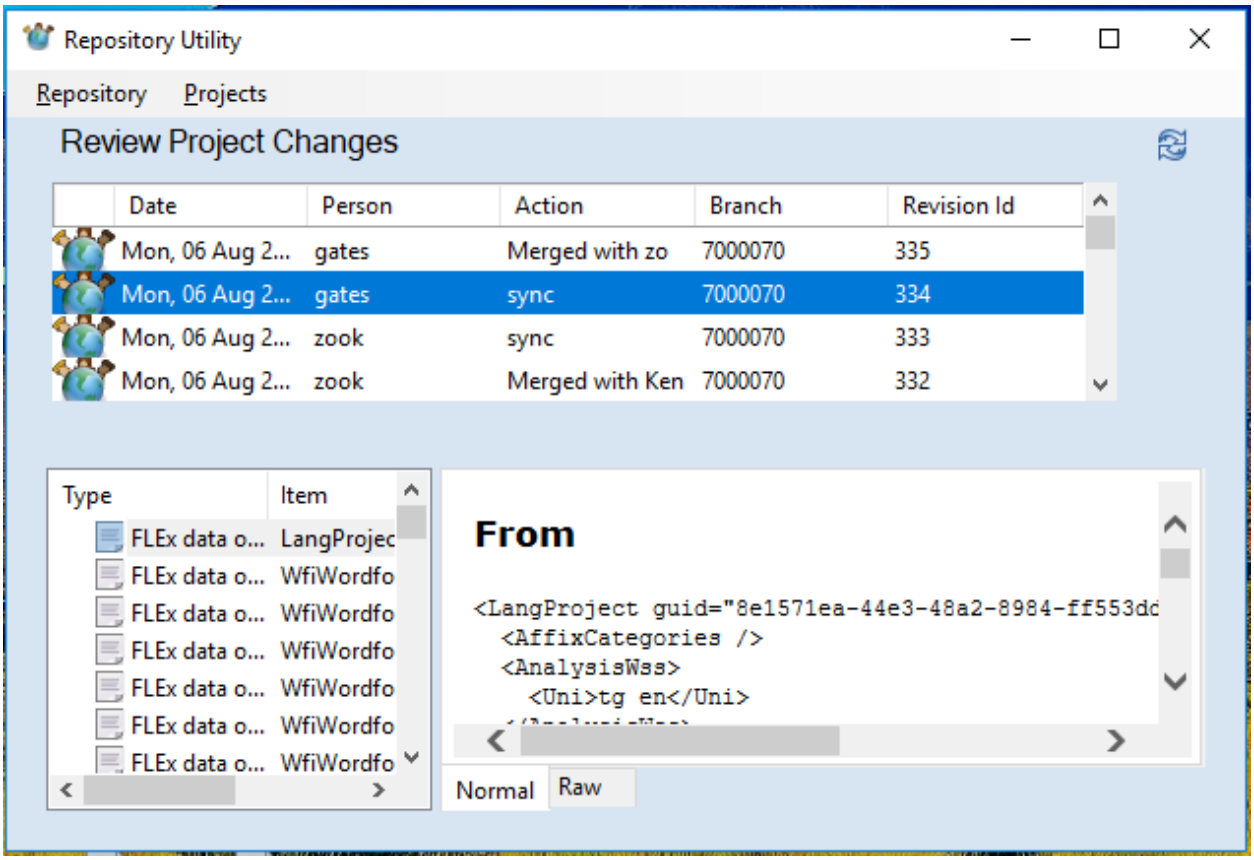
If you want to remove merge conflicts in this process before getting the project into Flex, you could delete all of the ChorusNotes files in subdirectories under the project folder, then execute these two commands to remove them from the repo
"c:\Program Files (x86)\SIL\FLEx Bridge\mercurial\hg" addremove
"c:\Program Files (x86)\SIL\FLEx Bridge\mercurial\hg" commit -m "comment" -u username
Then when you Get from Colleague, it will not bring back the old ChorusNotes files.

4.5.2 Using Repository Utility

Any time you are going to attempt changes in a repository, you should provide an easy way to restore things to the starting point in case the results are not what you want. This way you can make sure things will work the way you want before messing up repos that affect multiple users. The best way to do this is to make zipped copies of the project folder from each colleague as well as the main repo from languagedepot (LD), ChorusHub (CH), or USB. You can then restore

these on your machine and perform tests using a USB or CH. If the results are undesirable, you can use the zip files to restore all repos back to their original state. LD is more difficult to restore as it requires help from a LanguageDepo administrator to reset a repo. With ChorusHub (on your machine) or USB you can easily make copies and restore them as needed. When everything is fixed as desired, then you can S/R with LD to get your changes online.

As mentioned in the previous section, you should only use the version of Mercurial installed with FLEx Bridge when making changes to a repo. The normal way to do this is to use a Cmd window and execute hg commands. But there is one program built with FLEx Bridge that provides some very useful functions, but it is not included in the FLEx Bridge installer because it is not a user-friendly program, but has a rather crude interface It provides some useful features for knowledgeable FieldWorks support personnel. To get the program, you should contact flex_errors@sil.org, letting them know which version of FLEx Bridge you are using. The RepositoryUtility.exe program should be copied to your FLEx Bridge program directory and run from there. It is also only available for Microsoft Windows. Here's how it looks with a file open.



Keep in mind that making changes in the repo outside of FLEx only changes the split version of the project files. You still need some way to rebuild the fwdata file from these split files. The RepositoryUtility program provides these options via the Repository menu.

- Clone – Get a FieldWorks repo from LanguageDepot, ChorusHub, or USB.
- Open local repository - Open a current project on your machine.
- Once open, you can see the revision history, and you can click different revisions to see what changed. This operation is quite slow and not as useful as Hg Workbench.

- Update to revision – This temporarily reverts to some other revision you have selected. It updates the split files to the selected revision, and then builds the fwdata file at that level. This can be very helpful for going back to earlier revisions to see the data in FLEx at that revision. If you use this option, before closing RepositoryUtility, make sure you use "Update to revision" at the top revision if you want to continue using the project. This option can be useful in getting a version of the fwdata file or any of the split files at an earlier revision level, which can be useful in certain situations.
- Reset Repo to revision (cannot be undone) – This rolls a repo back to the selected revision and rebuilds the fwdata file at that point. The way it does this is it adds a couple new revisions that undo the changes made since the selected revision. If you use this option and you send it back to an external repo, you should have all users delete their project folder and do Get from Colleague to get the rolled back version. Using a normal S/R operation would likely cause undesirable merges that you may regret.
- Send Back to Source – This pushes the current repo back to a remote location. So for instance, you can clone from LD, make a change, then send the changes back to LD.
- Pull file from revision range – This allows you to specify a starting revision and an ending revision, a file name, and a destination folder for the extracted files. This will save a copy of the specified file at each revision in the range, appending revision numbers to the file names. You can then compare versions in KDiff3 or some other tool to see what changes were made in each revision. This is similar to "Save at revision" in Hg Workbench, but saves multiple files in one step.

4.5.3 Using S/R to repair or change data

Suppose a significant amount of work has been done interlinearizing a given text, then through various S/R operations, something happens that loses this work. How can you recover the work?

One option that might work is to use RepositoryUtility to temporarily roll back the project to the revision that had good data (e.g., Update to revision), then export the text to flextext. Then in a current project, import that flextext file, choosing the option to overwrite the existing text. This would restore the baseline and segment annotations, while trying to keep the current analyses. If you import a flextext file without overwriting an existing text, it will import the baseline with segment annotations, but it will not include any morph analyses.

Another option that may restore the original interlinearization is to replace the current split text file with an older version. This technique also provides a way to restore a possibility list (e.g., Semantic Domains) to an earlier state, or any other file that may need fixing. Since each text and each list is stored as a single file in the repo, if you replace the current file using an earlier version of the file, or a file that has been modified in some way, you can use mercurial commands and then pick up these changes via S/R. When doing this, however, keep in mind that all guid references to other split files, which is especially true of texts, will only work properly if the guids in the current project match the guids in the earlier version. This may or may not be the case. These are the steps to doing this kind of repair.

1. In your current project, do an S/R to USB or CH where the remote location does not have the current project.

2. Open a Cmd window on the remote repo directory. Execute this command to extract the actual files from the repo:
   "c:\Program Files (x86)\SIL\FLEx Bridge\mercurial\hg" update -C -r7000068
   where the number following r is the top branch of the repo (FLEx model number).
3. Now modify/replace the file(s) you want to change in the split files (e.g., "Linguistics\TextCorpus\Text_73d2051c-9800-4eec-a333-2b2a6c6ad7aa.textincorpus" for an interlinear text)
4. Add the modified file(s) to the current commit. Note hg addremove adds or removes any files that changed. If you want to limit it to one file you can use hg add <filename> where filename could be Linguistics\TextCorpus\Text_73d2051c-9800-4eec-a333-2b2a6c6ad7aa.textincorpus.
   "c:\Program Files (x86)\SIL\FLEx Bridge\mercurial\hg" addremove
5. Commit the changes to the remote repo.
   "c:\Program Files (x86)\SIL\FLEx Bridge\mercurial\hg" commit -m "comment" -u username
6. Now do S/R from your current project with the modified remote repo. This will pick up the file(s) you committed to the remote repo and you should see those changes in FLEx.
7. You can then delete the remote repo directory since this was just a temporary repo.

4.5.4 Dealing with defective repos

Occasionally something might happen that leaves a repo in a broken state that will not allow S/R. Using a Cmd window in the directory above the .hg directory, you can use this command to verify that the repo is OK.
"c:\Program Files (x86)\SIL\FLEx Bridge\mercurial\hg" verify

If something interrupts a S/R, it may occasionally leave the repo with an abandoned transaction. If so, the verify operation will note an abandoned transaction was found. You can usually correct this using this command from the Cmd window.
"c:\Program Files (x86)\SIL\FLEx Bridge\mercurial\hg" recover

Beyond a recover fixing the problem, it's not likely you'll be able to fix a broken repo. So the solution is to replace the repo from a good copy. Note, if this is a local repo, you must have it set to the state it was in during your last S/R from this project. The process for doing this is described above in "Repairing lost or damaged local repo".

4.5.5 Recovering data in difficult situations

Whenever you need to make serious repairs to data and there are multiple colleagues involved with outstanding changes, it's quite possible that after you have repaired one repo, when other users sync their projects, there will be undesirable merges between the projects that may undo some of your changes or damage the fixed data in some way. The safest way to deal with this is to have all colleagues delete their projects and do Get from Colleague to get started from the repaired repo. This guarantees that the changes you made will remain unaltered, but it also means that any outstanding work the colleagues had done will be lost. If you don't do this, and things go awry when colleagues sync, then you have yet another mess to try to fix which often involves needing to have someone reset the LD repo, etc.

Any colleague who is not making intentional changes should never do a normal S/R to get the latest project data since they may inadvertently make undesirable changes they did not realize

they were making. The safe way to get the latest is to always delete their project directory and do S/R Get from Colleague to avoid any problems. (Some day we should have a read-only S/R for these users, but we do not currently have this capability.) So when you are dealing with multiple colleagues in trying to resolve problems, you can ignore those colleagues who are not actually making changes. Those colleagues can just get the latest copy after things are fixed using their normal delete project and Get from Colleague.

There are two approaches to fixing serious issues, depending on the nature of the problem.

1. Have all users S/R and then stop working. Then you can do S/R to pick up all changes. Make whatever changes are needed, then send the results to the remote repo. Then have all colleagues delete their projects and do Get from Colleague to get the changes and then continue working.
2. If having all colleagues do an S/R is not possible for some reason, the best approach is to get a zipped copy of project directories from all colleagues with outstanding work and they stop working. Get a clone of the LD repo copied to CH or a USB that you can use as a temporary LD repo. Make a zipped copy of the remote repo and your project directory. Analyze the project directories of each colleague, especially checking on the status of their local repos. Using techniques described above, and any other techniques you might find useful, merge all of the projects from your colleagues into the temporary repo on CH or USB. If the results are bad, start over with the original zip files. When you have merged the data as best as possible to CH or USB, then do a S/R to LD. Now have all colleagues delete their directories and do Get from Colleague and then get back to work. This way they should have their data merged as well as possible without damaging the LD repo in the process.

If you have both FLEx and WeSay colleagues, it adds another level of difficulty to repairing data. You now need to consider two repos in the FieldWorks poject that is hosting the WeSay bridge since you have two different repos to deal with: the .hg directory under your project folder for the local FLEx repo, and .hg directory under OtherRepositories\projectname_LIFT\ for the local LIFT/WeSay repo.

Another consideration is when there are a mixture of FieldWorks versions doing S/R to a repo. If the FieldWorks model version is different, the new FW users have their changes stored in a different branch in the repo, and the two branches will not merge until the last user upgrades to the new program.

4.6 To switch a WeSay bridge user to another FLEx user

Remember that only one FLEx user should provide a S/R bridge between FLEx and WeSay users. Assume FLEx users A and B are using S/R Project to keep in sync, while user A is also doing S/R Lexicon to sync with WeSay users. Now they would like to switch the WeSay bridge operation to user B instead of A. If this process is not done correctly, you can duplicate senses, examples, etc. To avoid these problems, use the following process.

1. All WeSay users do S/R and stop work.
2. All FLEx users do S/R Project and stop work.
3. User A does S/R Lexicon to get the latest changes from WeSay users.

4. User A does S/R Project to send these merged changes to the FLEx repo.
5. User B does S/R Project to get the latest from the FLEx repo.
6. User B does Send/Receive...Get Lexicon and merge with this project to get started as the bridge.
7. User B does S/R Project to push any changes to the FLEx repo.
8. At this point all users can do S/R and then continue working.

At this point user B would be the only one that does S/R Lexicon to function as the bridge between WeSay and FLEx users.

When done, user A should delete the *_LIFT directory inside their FieldWorks project folder under the OtherRepositories folder to make sure they don't accidentally do S/R Lexicon.

The reason for needing these steps is that S/R Get Lexicon and merge this project uses the 3rd LIFT import option that keeps objects from both users when there is a conflict rather than one user overwriting the other. The above process makes sure there will be no merge conflicts during the process of switching the bridge.

4.7 FLEx and WeSay compatibility issues

There are some compatibility problems with writing systems between WeSay and FLEx for non-standard ISO codes. Lists are maintained in different ways between the two systems and are not included in S/R. Custom fields are also stored in different ways.

WeSay is heavily biased toward lexeme forms and definitions, and this bias cannot be removed in configuration. In WeSay 1.5.38 you can check Gloss in the dictionary configuration, but it doesn't show the field in the editing mode unless it currently has data. So when adding a new entry or editing an existing entry without gloss, you have to first click "Show Uncommon Fields" in the lower right to open the Gloss field. When adding a new sense in WeSay, you must enter a Meaning (definition) before you can add a gloss. But in the Semantic Domains tab, it does not give any option other than Meaning, which is definition.

If I create an entry in FLEx with only a gloss, and Glosses are enabled in WeSay Configuration, it shows my gloss in both the Meaning (definition) and Gloss fields. If I edit the gloss in WeSay, only the gloss is changed. Likewise if I edit the definition in WeSay, only the definition is changed. If I add an example in WeSay, the definition will now contain the same content as the gloss even though I didn't touch either one. If you don't edit anything in the entry, WeSay doesn't add the definition to the data even though it shows the gloss in the Meaning field.

If Gloss is not checked in WeSay Configuration, if an entry comes from FLEx with only a gloss, it shows the gloss content in the Meaning field in WeSay and if you edit it, the edited field becomes the Definition and the Gloss is unchanged. If you don't edit the meaning, but make some other change in the entry (e.g., add an example) the definition becomes real and is a copy of the gloss.

WeSay stores lexeme forms in the Word field. It is possible to configure WeSay to show Citation Forms, but this is limited. It is possible to click "Show Uncommon Fields" to add a Citation Form without filling in the Word (lexeme form) field. But in the list of entries, it shows a blank

instead of showing the Citation Form. And in the Semantic Domains tab you can only add a Word (lexeme form).

If you plan to use FLEx and WeSay with Send/Receive during a rapid word collection process, you need to use lexeme form and definition in both programs. If you want to do anything else, then you should only use FLEx during the process and configure it for a simple view that only shows what you want.