Technical Notes on FieldWorks Send/Receive

November 4, 2013

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.

If you are just collaborating between Flex users, you would only use a Flex repository (repo). If you are just collaborating between WeSay users and one optional Flex user, you would only use a Lift repo. 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 Flex users. 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.

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).

The Flex Send/Receive menu has changed in different versions since FW7.0. 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

The master project for starting S/R can be either a Flex or a WeSay project.

If a Flex user is setting up the initial repo, use Send/Receive…Send this Lexicon for the first time (to WeSay).

If a 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 you delete the Flex project directory and then restore from a current Flex backup. 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 Chorus Hub startup

Chorus Hub is a program that can be run 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. 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 a local or remote computer. Chorus Hub supports repos from several different programs including FieldWorks, WeSay, and Bloom. Each program limits access to repos it can use.

To start Chorus Hub, using Windows Explorer, launch this program:

c:\Program Files (x86)\SIL\FLEx Bridge\ChorusHub.exe (omit “ (x86)” for 32-bit Windows)

When it 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 will likely ask to permit c:\Program Files (x86)\SIL\FLEx Bridge\mercurial\hg.exe through the firewall. This should be granted. It may also need c:\Program Files (x86)\SIL\FLEx Bridge\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 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 LanguageDepot Internet startup

LanguageDepot is an Internet server (http://languagedepot.org) that hosts repos for various programs including FieldWorks. As of FieldWorks 8.0.5, 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 sent. 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. 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 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 a conflict. You can also see this report using Send/Receive…View Project Messages.

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 merge entry capability, but it doesn’t happen automatically. If two users take a list of words and 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. So after merging our entries, if I add a new sense for ‘house’ with a gloss ‘political body’, and my colleague does the same on his machine, 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 capability, 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 to 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 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 based on their id.

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

Interlinear texts and charts are complex objects in Flex with many sub-objects with ids and references pointing into the text via offsets. Even cutting and copying the baseline will generate a whole new set of objects with different ids. This is normally not a problem, but when using Send/Receive it presents major difficulties with merging if another user was working on the same text and will likely result in duplications of text and/or problems with interlinearization. So in order to avoid these potential merge problems, coordinate with your colleagues so that only one person works on a given text in any round of S/R operations, especially if one member makes any change to the baseline. Even adding or removing a segment break character (period, question mark, exclamation point, section sign) in the baseline will alter the underlying segment objects so that interlinearization of the affected segments from a colleague in the same S/R cycle will likely be lost.

Even when working in different texts and not modifying the baseline, two users doing interlinearization can create duplicate objects when working in the same S/R cycle. For example, if user A analyzes the word ‘cat’ for the first time, and user B also analyzes the word ‘cat’ in the same S/R cycle, when they merge, there will be two ‘cat’ entries in the lexicon as homographs, even though they have identical content. So one user will need to use the ‘Merge with entry’ option to combine these entries, then the ‘Merge sense into’ option to merge the two senses in the merged entry. Also, under Word Analyses, there will be two duplicate analyses for the wordform ‘cat’. So one user will need to delete the duplicate analysis that isn’t being used, or possibly first use the ‘Assign Analysis’ option to move any analyses from one analysis to the other analysis.

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.

In summary, when working on interlinear texts, only one user should modify a given baseline in one S/R cycle. Any interlinearization that adds new entries, wordform analyses, or word glosses will produce duplicates if another user does similar interlinearization on the same wordform in the same S/R cycle. Any merges or deletions in the Word Analyses area will cause loss of analyses that were made to the modified wordforms in the same S/R cycle. Analyses that use existing entries, senses, and wordform analyses and glosses should survive S/R operations. Frequent use of S/R will minimize the chances of encountering these problems.

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 assign a 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 our internal 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 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. ??? doc and txt files are included, but mp4 aren't. What is and isn't included???

3.6 FieldWorks version and project names

When using S/R for a Flex project, all collaborators should use the same FieldWorks version. FieldWorks projects have a data model version number that is stored in the repo. If one user upgrades to a newer version of FieldWorks that has a newer data model version, and then does a S/R, the version in the repo will now be the newer version. At this point, users that haven't upgraded will be blocked from using the repo until they upgrade to the same version. Thus when you are using S/R Flex collaboration, participants need to upgrade to a new version together.

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 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. 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.

UNDER CONSTRUCTION!