ICU and writing systems

Ken Zook

June 20, 2014

Contents

1 ICU introduction

International Components for Unicode (ICU) is an open source standard for working with Unicode that is being developed by IBM and other key players in the Unicode Consortium. Go to http://www.icu-project.org/ for documentation and downloads. It is cross-platform, so services are available for Windows, Mac, and Linux, and there are implementations for C, C++, and Java. ICU provides the following services:

- Access to Unicode character properties
- Many functions to deal with Unicode strings, including surrogates
- Converters to transform data to and from standard code pages
- Locale resources based on a language, region, and variant—includes collation customization, date, time, number, and currency display patterns, translations for languages and countries, and lists of exemplar characters
- Transliteration using rule-based or algorithmic processes such as converting Greek to Latin

- Case conversion
- Normalization
- Date and time functions
- Formatting and parsing strings representing things such as numbers, dates, and time
- Searching and sorting capabilities, including regular expressions
- Text analysis for things such as determining line breaks, selecting words, counting graphemes, and cursor movement
- A text layout engine handles contextual forms, bidirectional text, ligatures, character reordering, and character positioning, and uses OpenType tables
- Allows user-customized data tables including PUA

FieldWorks uses ICU because it handles many things needed for complex Unicode data, and saves a lot of programming time. It was especially attractive because it allows users to create their own writing systems (locales) and allows full use of the PUA and surrogates through customized tables.

Their assumption is that developers will do this work and then install these on end-user computers. In FieldWorks, end-users need to be able to define their own locales and PUA characters. This is possible through the tools they provide, but SIL is probably the only organization using ICU in this way. The big problem is that ICU memory-maps various Unicode and locale resources from the disk during operation to make it more efficient. This works fine during normal operations, but whenever users modify a writing system, FieldWorks needs to update some of the ICU tables. As long as the tables are memory-mapped, the operating system will not allow them to be modified. To allow users to modify a writing system, FieldWorks needs to first release all ICU resources. With careful programming, it is possible to do this in the active program, but not if other programs have the ICU files locked.

ICU developers continue to add improvements and support for new code points as the Unicode Consortium approves new Unicode versions. FieldWorks 4.0 uses ICU 3.4, which implements Unicode 4.1. FieldWorks 4.2 through 5.4 uses ICU 3.6, which implements Unicode 5.0. FieldWorks 6.0 uses ICU 4.0, which implements Unicode 5.1.

Typically, ICU developers deliver a large icudt36.dll that contains all the compiled ICU data. Instead of this approach, FieldWorks includes the compiled data files in separate folders, making icudt36.dll a stub that looks for the compiled data in the separate folders. FieldWorks uses this approach because it works best when users need to modify ICU files for their writing systems.

2 ICU files

The ICU files are installed as part of the SIL Encoding Converter package that is built into the FieldWorks installer. It is also available separately from the NRSI Web site. The ICU data files are installed in c:\Program Files\Common Files\SIL\Icu36. This directory has 3 subdirectories:

- The data directory contains source files for building the ICU data files.
- The icudt36l directory has compiled runtime versions of the ICU data.
- The tools directory holds a set of ICU programs needed in various stages of compiling the ICU data files from the data source files.

The Unicode database and locales are the central part of the ICU library. String functions need to know the locale in order to search, sort, and format dates and times. In FieldWorks, a writing system corresponds to an ICU locale, so as the program creates and modifies writing systems it also creates and modifies ICU locales. FieldWorks will *not* modify ICU-provided locale files, but allows users to create new locales and modify them. As FieldWorks defines and modifies PUA characters, it modifies the Unicode database files. Any changes to these files get compiled into runtime versions in the icudt36l directory. During operation, files in this directory are memory-mapped, causing them to be locked by the operating system until the ICU resources are released.

The three source directories under the data directory that are modified by FieldWorks are the unidata, locales, and coll directories.

2.1 unidata

The primary file in the unidata directory is UnicodeData.txt. This contains all the Unicode properties for code points other than some of the large East Asian sections. These definitions are common across all locales. When FieldWorks defines properties for PUA characters, it adds these definitions to this file. The files installed by SIL Encoding Converters have the SIL Corporate PUA characters already defined in this file. They are flagged with an "[SIL-Corp] Added Sep 2005" comment. If you add PUA characters, they are flagged with [SIL-Corp], followed by the language definition file and date and time they were added.

The Hebrew section of UnicodeData.txt was designed for modern day Hebrew. Using standard normalization on Biblical Hebrew results in incorrect reordering of diacritics. To solve this problem, FieldWorks sets the canonical combining classes for Hebrew characters to custom values described in SBLHebrewUserManual1.5x.pdf available at http://www.sbl-site.org/educational/BiblicalFonts_SBLHebrew.aspx. This makes FieldWorks compatible with BART and other sources working in conjunction with the Society of Biblical Literature and SIL International. Some code points between 599-5C7 are affected. These values have been tested by normalizing the entire Hebrew OT text from BART and verifying that it didn't change the input. There were 2 words that had reordered code points (dt29:28 and pr15:31), but upon further investigation, these were due to errors in the original Hebrew text. As a result of these changes, FieldWorks should work with ancient or modern Hebrew text without any improper reordering.

Various other files in the unidata directory are based on UnicodeData.txt, but reordered in ways to make data retrieval more efficient. These files may also be modified as PUA characters are added.

When these files are compiled, they modify runtime files such as uprops.icu, ubidi.icu, ucase.icu, unames.icu, and unorm.icu in the icudt36l directory.

2.2 locales

Each locale has a separate file in the locales directory. The filename represents the locale and has a .txt extension. ICU factory locales contain a lot of information, including

- local names for languages, countries (regions), and variants
- information for formatting dates, time, and currencies
- the Windows keyboard used with that locale

- exemplar characters, and
- other information.

When a writing system is created in FieldWorks, it creates a file in the locales directory with the ICULocale as the filename with a .txt extension. By default, this file does not have much in it unless users select a similar writing system in the writing system wizard or Writing System Properties page. In that case, it copies information from the similar writing system (ICU locale) into the new locale.

When a locale is modified, FieldWorks also adds information to root.txt, keeping track of added languages, regions, or variants. This information is at the top of the file in a Custom resource bundle. It keeps track of the language definition file, date, and time of modification. It also modifies res_index.txt which keeps a list of installed locales. It adds flag lines with [SIL-Corp] along with the language definition file, date, and time. (FieldWorks 5.0 through 5.4 has a bug that omits this.)

When these files are compiled, they modify runtime files xyz.res (xyz represents the ICULocale code), root.res, and res_index.res in the icudt36l directory.

2.3 coll

Unicode (and ICU) provide a default collation for all Unicode code points, but these can be modified or tailored for given locales. Each locale typically has a .txt file in the “coll” directory that defines collation tailoring for that locale.

When a locale specifying a collation is modified, FieldWorks updates a file for that locale in the “coll" directory. If users specify a similar writing system in the writing system wizard or Writing System Properties page, tailoring for the similar writing system (ICU locale) is copied into the collation tab in the writing system dialog. Users can modify the collation information as desired. This information is then copied into the collation file for that writing system in the “coll” directory.

FieldWorks also updates res_index.txt, a list of all collations defined in ICU. Modifications made to this later file are flagged with [SIL-Corp], along with the language definition file, date, and time. (FieldWorks 5.0 through 5.4 has a bug that omits this.)

When these files are compiled, they modify runtime files xyz.res (xyz represents the ICULocale code) and res_index.res in the icudt36l\coll directory.

3 Languages and dialects

FieldWorks maintains a distinction between vernacular and analysis writing systems. Any writing system for the language in which you are working is considered a vernacular writing system. There may be multiple writing systems for this language such as a standard orthography, IPA (phonetic), Pinyin form, phonemic, and Romanized.

Any language used for glosses, definitions, notes, and translations of examples use analysis writing systems. These languages may also have additional writing systems such as Pinyin and Romanized forms.

FieldWorks has some support for dialects, but it is probably not adequate. Your suggestions in this area would be helpful. There is usually a continuum between dialects and languages which determine to what extent they are separated in FieldWorks. Here are current options for treating dialects.

- The dialect is included in the same writing system as vernacular. This approach can be used when the dialect is almost identical to the vernacular language. Interlinear text would be interspersed with this dialect, so the wordform inventory and lexical headwords would contain these words. Where dialect headwords are different from vernacular, they would be entered as dialectal variants, with the condition or restrictions field indicating the dialect. Where senses for a dialect are different, they could be flagged in the restrictions or source fields.
- The dialect is a different vernacular writing system in the same FieldWorks project. If there are quite a few dialectal variants where the meanings are identical to the vernacular, you could keep track of these in alternate vernacular writing systems. In this case they would have the same language identifier, but the region would be set to indicate the dialect. This would allow you to record alternate spellings for a dialect. There are potential problems with interlinearizing text in this situation if baseline texts have more than one dialect. Currently, Flex would put all of these dialectal words in the wordform inventory as vernacular words, which would not be desirable. We expect to improve the processing of interlinear text to allow the baseline text to be in a different writing system. It would probably work better at that point, but may still be lacking.
- The dialect is an analysis writing system in the same language project. This does not present any problem with headwords, interlinear text, or the wordform inventory. You may want to use a reversal index for the dialect. This approach basically treats the dialect as a different language.
- The dialect is the vernacular writing system in a separate FieldWorks project. If the dialect is quite different and you want to have a separate lexicon and interlinear text for the dialect, it would be best to treat the dialect as a separate language and have a separate FieldWorks project for that language. Although we currently do not support references to other projects, we expect this will be a need as we start using automatic adaptation programs in FieldWorks.

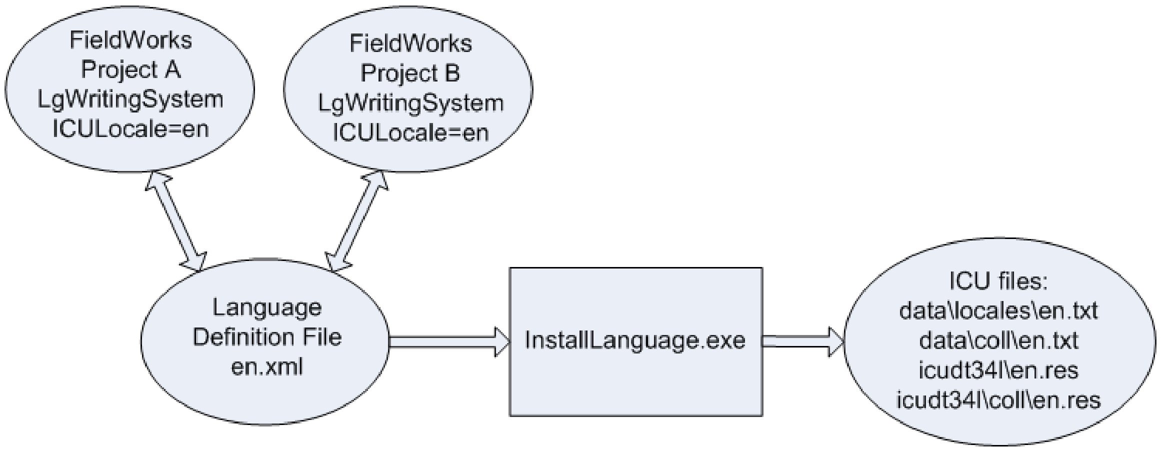
4 Writing systems in FieldWorks

Information about the conceptual model for writing systems is in the FieldWorks conceptual model section. In most cases when you make a change to a writing system, you want that change to propagate to other language projects which use that writing system. If you created a writing system for one language project and want to use that same writing system in another project, you can do so without redefining everything for that writing system.

Every writing system used on your computer has a corresponding Language Definition File. This file stores information from LgWritingSystem, LgCollation, and other data, including the ICU locale it is based on (similar writing system) and PUA characters needed for this writing system. These files are stored in the %ALLUSERSPROFILE%\Application Data\SIL\FieldWorks\Languages directory. The name of each file is the ICULocale code for the writing system with an .xml extension.

Note: %ALLUSERSPROFILE%\Application Data is c:\Documents and Settings\All Users\Application Data on Windows XP and c:\ProgramData on Vista.

In order for ICU to use these writing systems, some information needs to be transferred from the language definition file to the ICU source data files, and then they need to be compiled. The FieldWorks InstallLanguage program updates ICU source files based on the language definition file, and then compiles the changes into the ICU runtime files.


FieldWorks Project A LgWritingSystem ICULocale=en
FieldWorks Project B LgWritingSystem ICULocale=en
Language Definition File en.xml
InstallLanguage.exe
ICU files: data\locales\en.txt data\coll\en.txt icudt34l\en.res icudt34l\coll\en.res

This diagram illustrates the process flow for a single writing system in English.

When users create a new writing system and close the Writing System Wizard dialog, FieldWorks

1. creates and updates the language definition file
2. updates the LgWritingSystem in the current language project, setting LastModified to the modified time of the language definition file, and
3. executes InstallLanguage to create the new ICU locale files and compiles these into runtime files.
   Tip: InstallLanguage will only update ICU files if the locale is not a factory locale.

If users modify an existing writing system, it follows the same process except it modifies existing files instead of creating new ones.

When FieldWorks needs to use a writing system that is already defined in the database, it checks first to see if a language definition file exists for this writing system and does the following:

- If the language definition file exists and the date is identical to LastModified in the database, it does not update anything.
- If the language definition file exists with a different date than LastModified in the database, it updates the database information from the language definition file and sets LastModified to the date of the language definition file but does not change the date of the language definition file.
- If the language definition file does not exist, it creates it from the database information and uses InstallLanguage to install it into ICU.
  Note: This does not include PUA characters or similar writing system since these are not stored in the database.

Using this process, the language definition file contains the master information for the writing system. Any database on the local computer aligns itself to what is in this file the next time it is opened. InstallLanguage is only called when you create or update a language definition file. Thus, FieldWorks assumes that if a language definition file is present, it is already installed in ICU.

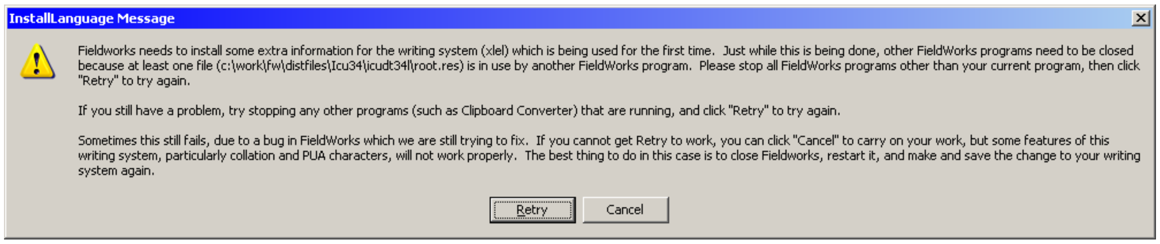
This process of keeping ICU files and databases in sync via the language XML file works well for single users on one computer. However it does cause confusion in some situations.

- In a lab situation where multiple users work on different projects on the same computer, suppose user A sets a writing system in some special way for his purpose. When user B opens his database that uses the same writing system, user A’s customizations will be loaded into B’s database, which is usually not desirable.
- When a user makes a change to his writing system and backs up his database and restores it on another computer that already has a definition for that writing system, the dates will usually differ, so the restored database will take on the definition from the language definition file on the target machine, thus losing the change made on the source machine.
- When a user accesses a database across the network, FieldWorks checks the dates stored in the remote database against the language definition file on the local machine. If they are different, the database is updated from the local machine.
- When a user installs an updated version of FieldWorks, the language definition files are not removed. Thus if the new version of FieldWorks has made some change to a particular writing system, that change may be lost on the user’s machine since the dates will be different and the user’s information will override what was in the installed database.

To overcome these problems, you should make sure each machine that runs FieldWorks has the same copy of the language definition file, and that it is properly installed on each machine. To do this for the xyz writing system

1. Copy xyz.xml to the %ALLUSERSPROFILE%\Application Data\SIL\FieldWorks\Languages directory.
2. In a command window (Start…run, type cmd then press Enter) execute the following
   InstallLanguage xyz
   This step updates the ICU files to match the language definition file.

A major problem with this process happens when InstallLanguage tries to update locked ICU files. In this case, InstallLanguage displays a message like this and waits for the operator:


InstallLanguage Message

Fieldworks needs to install some extra information for the writing system (xlel) which is being used for the first time. Just while this is being done, other FieldWorks programs need to be closed because at least one file (c:\work\fw\distfiles\Icu34\icudt34l\root.res) is in use by another FieldWorks program. Please stop all FieldWorks programs other than your current program, then click "Retry" to try again.

If you still have a problem, try stopping any other programs (such as Clipboard Converter) that are running, and click "Retry" to try again.

Sometimes this still fails, due to a bug in FieldWorks which we are still trying to fix. If you cannot get Retry to work, you can click "Cancel" to carry on your work, but some features of this writing system, particularly collation and PUA characters, will not work properly. The best thing to do in this case is to close Fieldworks, restart it, and make and save the change to your writing system again.

If some other application is open, close it to release the ICU lock, then click Retry to successfully update the ICU files. Having multiple Flex windows open should not cause this problem since they are all part of the same application. Having a TE window open while modifying a writing system in Flex will normally give this problem. Some other programs such as the SIL Encoding Converter Clipboard Converter may also lock the ICU files. Non-SIL programs never use our ICU version, so they will not cause this problem.

A single FieldWorks application such as Flex is supposed to release all locks on ICU *before* calling InstallLanguage. Unfortunately, some elusive bugs remain where these locks are not released, so InstallLanguage gives this error even though no other application is open. When this happens, you can only click Cancel, which means the ICU files are *not* updated.

To update the ICU files:

1. Exit the current application, and start it again.
2. Make a change to the writing system.
3. Close the writing system dialog to force the ICU update.

Note: This *usually* solves the problem. If you ever get into a state where you can't get around this message even when restarting the program, stop all FieldWorks programs, then use InstallLanguage manually to install the writing system (see next section). If that happens to fail, try rebooting. Windows occasionally gets files in locked states where only a reboot will clear the lock.

5 InstallLanguage

c:\Program Files\SIL\FieldWorks\InstallLanguage.exe is a separate program you can run from a command line when desired. If you simply type InstallLanguage from the command line, it lists the options. It uses registry settings to find the language definition files, so it can be run from any directory. When you specify a language definition file, the .xml file extension is optional.

The following is the typical way to install a language definition file (xyz.xml) into ICU:

```
InstallLanguage -i -c xyz
```

This installs the locale (-i) and any PUA characters (-c).

The following command lists custom locales currently installed in ICU:

```
InstallLanguage -s
```

The -customLanguages flag appears to be broken.

You can remove a locale and delete the language definition file with the following command:

```
InstallLanguage -r xyz
```

Note: If the database still has the "xyz" writing system defined in LgWritingSystem, the next time you open that database, it recreates the language definition file and reinstalls it into ICU. To completely get rid of a writing system, you need to delete it from all databases that use it. The only *safe* way to do this is in the FieldWorks XML dump file. See the section on "Removing a writing system" in FieldWorks XML model.doc. If you remove the writing system from each database and then use InstallLanguage to remove the language definition and ICU information, you should be rid of the old writing system. Of course, if you restore a backup that still has the writing system, it comes back again.

If your ICU files ever get corrupted, rename the Icu36 and languages directories, then run the installer in repair mode to install a fresh copy.

Whenever InstallLanguage modifies an ICU factory file, it makes a copy of the original file appending _ORIGINAL to the file name. The command

```
InstallLanguage -o
```

attempts to restore ICU to its original condition, then reinstalls all of the language definition files in your Languages directory.

InstallLanguage has a logging capability that, when turned on, writes information to a log file whenever it executes a command. This is particularly helpful if you get InstallLanguage failures you do not understand. The log file usually shows what is going wrong, whether you execute the program from the command line or it is called from within Fieldworks. Two registry values need to be enabled to get logging to work. The FieldWorks installer installs these values, but keeps the logging disabled. The registry key is

```
HKEY_LOCAL_MACHINE\SOFTWARE\SIL\FieldWorks
```

The InstallLanguageLog string value lists the path and name for the log file.
The InstallLanguageUseLog string value controls logging to this file. When set to Y (yes), normal logging information is written to the log. When set to V (verbose), more information is written to the log. When set to N (no), logging is disabled.

One error you may encounter when upgrading older FieldWorks databases is an error related to locale names. If ICU recognizes a 3-letter locale code that has an equivalent 2-letter code (e.g., English = eng = en), it automatically switches to using the 2-letter code. InstallLanguage attempts to catch this and warns users to use the 2-letter code instead. A similar problem can happen if you try to use a 3-letter country (region) code. For example, Kenya uses KEN as the 3-letter code and KE as the 2-letter code. If you enter KEN, ICU "helpfully" changes it to KE.

6 Writing systems during FieldWorks installation

Uninstalling FieldWorks does *not* alter your FieldWorks database files or the language definition files in the Languages directory.

When you install FieldWorks, it empties the ICU directory to clear any problems that may be there, then copies a clean installation of ICU files. It then sets a DWORD InitIcu registry flag to 1 in

```
HKEY_LOCAL_MACHINE\SOFTWARE\SIL
```

Whenever you start a FieldWorks application, it checks this flag. If set to 1, it calls InstallLanguage with the -o flag to reinstall all of the language definition files into the new copy of ICU.

If you have a lot of extra language definition files you no longer want, delete them after uninstalling FieldWorks. When you install a clean version they are not installed again (assuming you do not open any FieldWorks projects that still have those writing systems in the database).

7 Custom Character setup

Note: This section is written for FieldWorks 6.0.1 or later. Earlier versions supported the Private Use Area (PUA), but did not support other areas. For information on Unicode code points and the PUA area, see Unicode Introduction.doc and Unicode Introduction.ppt.

In the Characters tab of the Writing System Properties dialog the user can define custom characters needed for a particular writing system. In most cases you will not need to define any custom characters because Unicode or the SIL Non Roman Script Initiative

(NRSI) have defined the vast majority of characters needed for languages around the world.

There are a few situations, though, where it is necessary to define or modify Unicode characters for a given writing system. The following sections describe these.

7.1 You have a font that supports a character you need, but FieldWorks does not know about it.

The font may be a recent font from NRSI or some other source. The new character is in the PUA area, or it is in an area recently adopted by Unicode, but standard ICU tables used by FieldWorks do not yet contain a definition for this code point. The Custom Characters section of the Characters tab of the Writing System Properties dialog can be used to resolve this issue. For each undefined code point, use the dialog to add a definition for that code point using the instructions in section 7.5.

7.2 Unicode defines a new character you need, but FieldWorks does not know about it.

If you have a font that supports these new characters, you can see section 7.1 for instructions. If you don’t have a font, you will need to obtain a font before it can be used in FieldWorks.

7.3 You have a required character that neither Unicode nor NRSI have defined.

You may have a character that is unique to your language that hasn’t been approved by Unicode, and NRSI has not provided this character in their area of the PUA. If possible, you should correspond with NRSI about this need before proceeding. For one thing, you will need to have a font made to support this character. NRSI may be able to help with this and can advise on what you should do. It may be necessary for you to add this character to the SIL entity-assigned area of the PUA. Once you have a font that supports your characters, see section 7.1 for instructions.

7.4 Unicode or NRSI have defined a character, but there is a problem with their definition.

The code point definitions in Unicode is a standard that has been adopted around the world. Any time you override these definitions, you will likely get in trouble later on if your data is used anywhere outside of FieldWorks. As such, this need should be extremely rare, and you should seek directions from NRSI before proceeding. (One example of this is that FieldWorks installations have overridden a few Hebrew normalizations because this is a standard supported by several major organizations supporting Biblical Hebrew, and the Unicode standard is based on modern Hebrew.) After appropriate consultation, if you do need to change an existing definition, the Custom Characters section of the Characters tab of the Writing System Properties dialog can be used to override an existing Unicode definition or NRSI PUA definition.

7.5 Defining a Custom Character

To define a custom character for a writing system or override an existing definition:

1. Go to the Writing Systems tab of the FieldWorks Project Properties dialog (In FieldWorks Language Explorer, you can use Format…Setup Writing Systems.)
2. Select the writing system from the vernacular or analysis section and click Modify.
3. In the Writing System Properties dialog, click the Characters tab, then click the Add button.
4. In the Add Custom Character dialog, enter the hex Unicode value in the Code value box.
   Tip: Valid ranges for SIL entity PUA definitions are E000-EFFF, F0000-FFFFD, and 100000-10FFFD. Values under 1000 need to have leading zeros to fill out 4 digits.
5. Type a name for the character, then fill in the desired properties.
6. Close the Add Custom Character dialog with OK.
7. If you need to add these custom characters to your valid characters, you may need to close the Writing System Properties dialog and reopen it before clicking the Valid Characters button.

Normally your font designer gives details for each code point property. For more information, go to www.unicode.org/ucd.

Tip: One way to find appropriate code point properties is to look at similar code points in the Add Custom Character dialog. You can type any existing code point in the Code value, and the dialog will fill in the values for that code point. You can type in other values to investigate, then Cancel the dialog without making any changes.

In the Writing System Properties dialog, the Custom Character window lists all the custom characters defined from any of your language definition files. It checks the ones defined in the current writing system. You can check or uncheck boxes to add or remove the definitions from your language definition file. A custom character definition affects all writing systems on your computer. The only reason to keep track of specific custom definitions in a writing system file is to copy that file to a new computer and install the custom characters on that computer.

Note: Although the Add Custom Character dialog lets you override any Unicode code point, some alternations will be ignored. For example, the 'Bidirectional character type' property allows you to set a character to be Left to Right. However, there are ranges of Unicode code points that have been reserved for Right to Left characters (e.g., 0800-08FF). If you attempt to change this property in a code point in one of these ranges, ICU will ignore your request and will return the value to Right to Left. However, as long as you are using the override feature to allow FieldWorks to process newly assigned Unicode code points, this will never be a problem since the Unicode Consortium will abide by the ranges they have reserved for special uses.

7.6 Removing Custom Characters

If you have predefined a code point that is subsequently defined by Unicode or NRSI in a more recent version of FieldWorks, you should eliminate your definition and resort to the installed definition. You can use this same approach to delete a custom definition you no longer want. At this point, the easiest way to do this is

1. Uninstall FieldWorks

2. Review the language definition files in your FieldWorks\Languages directory. For each one that contains a CharDef element that defines the code point you want to remove, delete that element from the file.
3. Install FieldWorks. The custom definitions should now be gone.

If you need to do this frequently, or help multiple people, there is another way. Instead of uninstalling and reinstalling FieldWorks, make a copy of the C:\Documents and Settings\All Users\Application Data\SIL\FieldWorks\Icu40\data and icudt40l directories from a new installation and restore these originals to your machine, then clean up the language definition files (step 2 above), and then execute

```
InstallLanguage -o
```

Here is another way you can get or custom character definitions, if you want.

1. In all *.xml files in C:\Documents and Settings\All Users\Application Data\SIL\FieldWorks\Languages delete the CharDef element(s) you want to remove.
2. In C:\Documents and Settings\All Users\Application Data\SIL\FieldWorks\Icu40\data\unidata, rename any file with _ORIGINAL to the name without _ORIGINAL, replacing the existing file with that name.
3. In a cmd window, type the following command:
   ```
   InstallLanguage -o
   ```

Note that custom definitions are only stored in the language definition file. They are not stored in the database. If you transfer your database to a new computer, you should also copy your language definition file(s) to that computer as well for the custom definitions to work. If this is not done before installing FieldWorks, you should install the writing system on the new machine using

```
InstallLanguage xx
```

where xx is the name of your language definition file without the .xml extension.

8 Collation setup

FieldWorks uses ICU collation (based on the Unicode Collation Algorithm) for sorting data (see http://www.unicode.org/unicode/reports/tr10/). FieldWorks provides one sort specification for each writing system that users can access in the Writing Systems Properties…Sorting tab. When this is empty, the writing system uses the default Unicode collation. The window in the Sorting tab contains rules that override the default Unicode collation. Using the ICU/Unicode approach, rather than giving a list of characters to determine collation, users normally give a set of rules to change specific code points that do not sort correctly by default.

In the Writing System Wizard or the Writing System Properties dialog (Sorting tab), users can load sort specifications from some other writing system. If they select another writing system, FieldWorks copies the collation rules from that locale into the sorting tab, allowing use as is, or allowing further customization. When you close the dialog, this information is written to the writing system language definition file and then InstallLanguage incorporates this into the ICU files (assuming it is not an ICU factory locale). If users select a different writing system that already has something in the sorting window, FieldWorks asks if they want to overwrite the existing collation rules. The program is not smart enough to merge new rules with existing rules, so users either need to overwrite what is already there or leave it. Copy what is currently in the window to

ZEdit (UTF-8 mode) or some other Unicode editor *first,* then let the similar writing system overwrite it so users can compare the results.

The sorting window uses Unicode data and is displayed using the default font for the writing system. Prepare the collation rules in Word or ZEdit in UTF-8 mode and then paste the results into the window. If you have trouble displaying characters, you can use the ICU \uNNNN syntax for Unicode values.

8.1 ICU rules introduction

The following resources are available:

- See the ICU tailoring rules at http://userguide.icu-project.org/collation/customization.
- ICU provides a useful Web site for testing collation rules at http://demo.icu-project.org/icu-bin/locexp?_=root&d_=en&x=col. Click the root language link, then near the bottom under Collation rules, click Demo.
  Note: This demo uses the currently released version of ICU which may not apply to what is currently available in FieldWorks. For example FieldWorks 6.0 and 6.0.1 use ICU 4.0, but as of this date, ICU has released ICU 4.2 which adds a new \ quoting character. So although the demo works with \, FieldWorks will not currently accept this.
- Martin Hosken wrote an excellent tutorial on using ICU collation for users who need to delve into this area. The pdf file is zipped in sort_trainer.zip. He also provides a sorting trainer program to work on the exercises or to test needed collations. This Python package, complete with its own default ICU version is in sort_trn.zip. Unzip this to a directory (24 MB) and execute sort_trn.exe to get started. Enter the words that need to be sorted in the top left pane, one per line, and the rules in the lower pane. Click the Sort button to see the sorted results in the upper right pane. It also provides a Sort Keys button to show the sort keys that were generated for each word.

Each ICU rule starts with an ampersand followed by an anchor point. The rest of the rule specifies how characters are collated compared to the anchor point. There is no need to start a new line for each rule, but it makes it more readable.

Here is a simple example of a rule using a primary level (single left wedge):

```
&c<k
```

This rule states that “k” comes immediately after “c” (e.g., cat, kite, dog).
Note: This does *not* handle uppercase.

Diagraphs can be handled as well:

```
&n<ng
```

This rule states that the “ng” diagraph occurs after “n” (e.g., nang, nung, ngang).
Note: This does not handle uppercase.

The Unicode default collation sequence ignores diacritics unless the rest of the word is identical. In that case, words are sorted based on the diacritic (e.g., bad, bád, bàd, bâd, båd, bäd, bãd).

Two left wedges are used for secondary level collation which only comes into effect if the primary levels are identical. The secondary level is typically used for diacritics. You can change the way diacritics are sorted with the following rule:

```
&a<<à<<á<<â<<å<<ä<<ã
```

This example changes the default collation of diacritics to include grave before acute (e.g., bad, bàd, bád, bâd, båd, bäd, bãd).

Note: In addition to inserting the actual character in a rule, you can also give the code point. The following commands are identical:

```
&a<<à
&\u0061<<\u00e0
```

Three left wedges are used for tertiary level collation which is typically used for case. Tertiary level sorting only affects strings that are identical through the secondary level.

```
&n<ng<<<Ng<<<NG
&c<k<<<K
```

The first rule moves “ng” (regardless of case) to follow “n” (e.g., nang, Nang, NANG, nung, ngang, Ngang, NGANG). The second moves “k” (regardless of case) to follow “c” (e.g., cat, kite, Kite, dog).

To sort “á” at a primary level after all other “a's”, use this rule:

```
&a<á<<<Á
```

This sort order gives ade, ãde, apple, Azure, áde, Áde.

If you need to sort a character before another one instead of after, (e.g., āb, Āb, aa, Aa) you can do it two ways.

```
&[before 1]a<ā<<<Ā
```

In this case the right-hand side goes before the A anchor instead of after. The digit 1 indicates this is a primary level.

```
&9<ā<<<Ā
```

The other way is to use an anchor point before the desired letter. Since 9 normally sorts before A, we can use the normal way to specify that ā immediately follows 9, so therefore it will be before A.

To sort phonetic script in “p p^h b ɸ β m ʍ w” order, use either of the following identical rules:

&p < p^h < b < ɸ < β < m < ʍ < w
&p<\u0070\u02b0<b<\u0278<\u03b2<m<\u028d<w

Note: This approach can be used to turn a Shoebox sort sequence (that does not have case distinctions) into a rule. Shoebox has a list of characters, one per line in the desired order. Put an “&” in front of the first character and change each new line into “<”.This rule can be pasted into the FieldWorks sort tab. Use only UTF-8 characters, *not* ANSI.

To sort uppercase and lowercase in “c C b B a A” order, use these two rules :

```
&c<b<<<B
&b<a<<<A
```

This can also be combined into a single rule:

```
&c<b<<<B<a<<<A
```

Note: In an ICU rule, any non-alphanumeric ASCII character is reserved for syntax characters. If you need to control collation of any of these characters, you must quote them with a \ (only ICU 4.2 or greater) or enclose them in apostrophes. A single apostrophe can also be represented as two apostrophes. Here are some examples of alphanumeric and punctuation characters with or without the \u syntax. (See the Note in the second bullet under section 8.1 regarding the backslash limitation.)

a	letter a
\u0061	letter a
3	digit 3
ng	digraph ng
'ng'	digraph ng (quotes are optional for alphanumeric characters)
\u006e\u0067	digraph ng
\-	hyphen [not currently in FW]
'-'	hyphen
' '	space
\	space (there is a space following the \) [not currently in FW]
'\u0020'	space
\\u0020	space [not currently in FW]
\'	apostrophe [not currently in FW]
''	apostrophe
\u0027\u0027	apostrophe

To control the collation of an apostrophe you would thus add two apostrophes (not a double quote). To sort t' after t, you would use the rule

```
&t<t''
```

The following rules would be one way to handle IPA sorting

```
&d<d͡ʒ
&e<ɛ<f<ɸ
&i<ɨ
&k<k''
&n<ŋ
&p<p''<r<ɾ
&s<ʃ<ʂ
&t<t''<t͡s<t͡s''<t͡ʃ<t͡ʃ''<ʈ͡ʂ<ʈ͡ʂ''
&z<ʒ<ʐ<ʔ
```

Suppose you want to ignore an apostrophe after m and n, but you want ng to sort after n, and ng' to sort after ng. The following rules allow for this.

The = syntax states that the right side is identical to the left side.

```
&m=m''
&M=M''
&n=n''
&N=N''
&n<ng<<<Ng<<<NG<ng''<<<Ng''<<<NG''
```

Suppose you want to ignore 02BC;MODIFIER LETTER APOSTROPHE in sorting. There are two ways you could handle this. The following rule doesn’t totally ignore the apostrophe, but it treats it in a secondary level so that it is ignored unless words are identical otherwise. In this case it always comes after other diacritics.

&\u030E<<\u02BC

This would result in the following order: ba, bad, bäd, ba’d, b’ad, bade, bat, bät, ba’t, b’at, bate.

The second approach is to totally ignore 02BC.

&[last tertiary ignorable] = \u02BC

This would result in the following order ba, ba’d, bad, b’ad, bäd, bade, ba’t, bat, b’at, bät, bate. Since ba, ba’d, and b’ad all have identical sort keys, their order is random.

If you need to ignore more than one character, use = to separate the list of characters. The following rule would ignore an apostrophe, a question mark, a hyphen, a space, and the ng digraph

&[last tertiary ignorable] = '' = '?' = '-' = ' ' = ng

or [not currently in FW]

&[last tertiary ignorable] = \' = \? = \- = \ = ng

This could also be represented as

&[last tertiary ignorable] = \u0027\u0027 = '\u003f' = '\u002d' = '\u0020' = \u006e\u0067

If you simply want to ignore all punctuation as well as white space, you can use the following rule

[alternate shifted]

The ICU default collation of Thaana characters, used by Divehi, treat diacritics as primary instead of secondary characters, making it impossible in Flex to filter on baseforms without diacritics. This can be solved, however, by using this collation rule (unfortunately it looks mixed up because of RTL characters, but if you copy it and paste it into Flex, it will work):

&[last primary ignorable]<<◌ް>>◌ޮ>>◌ޯ>>◌ޭ>>◌ެ>>◌ޫ>>◌ު>>◌ީ>>◌ި>>◌ާ>>◌ަ

Refer to the references at the beginning of this section for more complex sorting issues.

9 Keyboard setup

See the separate documentation for Keyboard input for details on installing Windows input languages and Keyman files. Also refer to c:\Program Files\SIL\FieldWorks\Language Explorer\Training\Technical Notes on Writing Systems.doc about Keyman setup in section 3.

10 Setting up a complex language project

This summarizes the steps for setting up a complex FieldWorks project. When setting up writing systems, you do not need to have everything worked out ahead of time. You can always go into the Writing System Properties dialog and change things such as the keyboard, fonts, and sort specifications.

It works best if you set up the writing system codes correctly from the beginning. See c:\Program Files\SIL\FieldWorks\Language Explorer\Training\Technical Notes on Writing Systems.doc for more details. In particular, pay attention to section 2.3 “Multiple Writing Systems for the same language” (e.g., orthographic, phonetic, phonemic, and Romanized).

Getting started

1. Make sure you install any special Unicode font(s) needed for your vernacular and analysis languages.
   Tip: If you do not have any, search for some on the Internet and/or contact NRSI. The new Reprise program from NRSI may be helpful here, as well.
2. When needed, make sure you set up Windows input languages and Keyman keyboards needed for *all* writing systems (see Keyboard input.doc for details).
3. If you need to import legacy data, make sure you have a way to convert it to Unicode (see SIL Encoding Converters.doc for details).
4. From any FieldWorks program, create a new project with File…New FieldWorks Project.
5. Enter the project name (usually the language).
6. Select or define the primary *vernacular* writing system by setting up fonts, keyboards, converters, PUA characters, and sorting information as needed.
7. Select or define the primary *analysis* writing system by setting up fonts, keyboards, converters, PUA characters, and sorting information as needed.
8. In the new project, define any *further* vernacular or analysis writing systems needed in the project by setting up fonts, keyboards, converters, PUA characters, and sorting information as needed.
9. Back up the FieldWorks project.

Importing files

1. If you need to import *lexical* data, use Language Explorer. Carefully check the results to make sure the data imported correctly. If not, restore the backup and try again.
   - For SFM import, see Help…Resources…Technical Notes on SFM Database Import.
   - For LinguaLinks import, see Help…Resources…Technical Notes on LinguaLinks Import.
2. Back up the FieldWorks project.
3. If you need to import *Scripture* data, use Translation Editor. Carefully check the results to make sure the data imported correctly. If not, restore the last backup and try again.
4. Back up the FieldWorks project.
5. If you need to import *anthropological* data, use Data Notebook. Carefully check the results to make sure the data imported correctly. If not, restore the last backup and try again.
6. Back up the FieldWorks project.

11 Moving a writing system to another computer

FieldWorks works fine when users transfer a database with simple writing systems to a new computer and start up a FieldWorks program with that database. For more complex cases (special keyboards, fonts, encoding converters, or PUA characters), users still need to do several things:

- If you use Far East or right-to-left languages, enable them in Windows XP Regional and Language Options.
- Install any fonts that are not on the target computer.
- Install any Keyman keyboards that are not on the target computer.

- Install any Windows input languages that are not on the current computer and enable them in Windows Regional and Language Options.
- For any encoding converters that depend on programs such as TECkit or CC tables, copy the appropriate files to the target computer and install them into SIL Encoding Converters (preferably with the same name used on the source computer).
- If the language requires PUA characters, *first* make sure these definitions are in the desired language definition file(s). Then copy the language definition file to the FieldWorks\Languages directory on the target computer and use InstallLanguage -i -c to install it into ICU.

After transferring the FieldWorks project to the new computer, you should be able to start up and have everything work correctly. If you used a different name for the encoding converter on the source computer than for the destination computer, go into the Writing System Properties dialog and change the encoding converter to match the new name.

12 Changing the underlying writing system code

It should not be possible to create illegal writing systems in FieldWorks. It was possible in older versions of WorldPad. It's also possible to do this via SOLID and WeSay imports and under the hood via XML files or SQL queries. If this happens, you'll probably encounter numerous crashes of the type:

```
Msg: InstallLanguage failed on file C:\Documents and Settings\All Users\Application
Data\SIL\FieldWorks\Languages\kaz_CHIARB.xml with code -18
COM message: Unspecified error
```

The -18 error message reported from InstallLanguage is "The Language Name is already used as an ISO3 value". The problem is the ISO 639-3 value of Kazakh is kaz, but where there are equivalent ISO 639-1 two-letter codes, ICU automatically translates the value to the two-letter codes. The ISO 639-1 value for Kazakh is kk. So in this case, InstallLanguage issues an error message to warn the user that the code they are using has conflicts with ICU and it should be changed.

When creating a writing system in Flex, Data Notebook, or TE the new writing system dialog automatically converts kaz to kk and everything is fine. These programs use the Ethnologue database delivered with FieldWorks to make this substitution. Since WorldPad is designed to work independently from other FieldWorks applications and databases, it does not provide this automatic conversion. WorldPad 5.0 warns the user when this condition is encountered and requires them to change the code to something legal.

If you do encounter these error messages, the solution is to change the underlying code of the writing system. There may be other cases where changing this code would be desirable to make it more standard. The easiest way to do this is.

1. Close all FieldWorks apps.
2. Start...Run and type dbmt
3. Click OK to the logon dialog
4. Select a project name from the combo box at the top
5. Paste this query in the box and press F5 to execute

```
select IcuLocale from LgWritingSystem where IcuLocale = 'kaz_CHI'
```

6. If something shows up in the bottom pane, then paste this into the box and press F5
```
update LgWritingSystem set IcuLocale = 'kk_CHI' where IcuLocale = 'kaz_CHI'
```
7. Repeat 4-6 for any other projects that may use this same writing system.
8. From a Cmd window, execute the following command
```
InstallLanguage -r kaz_CHI
```

These steps will remove the old code from ICU and will change the code in existing databases. When a project is restarted, it will use the correct kk_CHI code. However, if you restore from an older database that has the older code, it will reappear and you'll need to repeat this process to get rid of it again.

Also, if you have WorldPad .wpx files that used it, you'll need to fix it in those as well. You can do this by opening the files(s) in a text editor, such as ZEdit (Start...Run zedit), and doing a search and replace to replace kaz_CHI with kk_CHI.

If you have more than one writing system that needs to be renamed, you should use the following query in step 5:
```
select IcuLocale from LgWritingSystem where IcuLocale like 'kaz%'
```

Then in step 6 you’ll need to execute separate queries to fix each writing system that is returned from step 5, making appropriate changes to the old and new codes. In step 8 you’ll need to repeat the command for each old writing system code. Also, in WorldPad files, you’ll need to make appropriate changes for each writing system.