Bulk edit issues

Ken Zook

October 20, 2008

Contents

1 Introduction

Bulk edit tools provide powerful ways to apply changes to specific fields in the database. However, with power comes the ability to damage your data. If you plan to do much with bulk edit, *back up* your project first, in case things do not turn out the way you want!

Here are some useful possibilities, depending on your language and the state of your lexicon:

- Create lexeme forms from citation forms by stripping off affixes.
- Set the Grammatical Info on senses.
- Produce glosses from definitions.
- Produce reversal entries from glosses or definitions.
- Fill the CV Pattern field.

- Fill the Tone field.
- Fill the phonemic form from the phonetic form.
- Produce a Romanized form.
- Fix spelling or orthographic changes on a given field.
- Move import residue fields to custom fields or other locations.
- Clean up inconsistencies in data.
- Merge imported parts of speech with variant forms.

The bulk edit tools try to make the complex database structure look like a simple table that holds strings which can be modified. This process works very well when the source and target fields are basic properties on the current record object.

For example, when the records are senses, it is easy to work with all the string fields (e.g., gloss, definition, and various note fields) or atomic reference properties (e.g., sense type and status).

However, when more structure is involved, it is often unclear what should happen. For example, what should Flex do if the target is a reference sequence (e.g., domain types, usage types, and anthro codes)? Instead of a one-to-one mapping of sense fields, FieldWorks now has a one-to-many mapping that does not fit well in a simple table view. It is unclear what making a change to these fields would mean.

An even more complex problem is example sentences. This involves a one-to-many owning relationship. In addition, examples have multiple fields within each example. How would you display this in a simple table? What would it mean to change one of these? Bulk edit can show fields from the entry that owns the sense, but these are not properties on the sense. The entry may also own multiple senses, so it is risky to make changes on an entry that is shared by multiple senses without seeing these other senses.

As a result of these difficulties in mapping a complex structure to a table, only certain fields can be processed via bulk edit. The fields that can actually be modified have a ⇒ symbol in the Column Choices dialog. This also explains why there are separate views for Bulk Edit Entries compared to senses.

In Bulk Edit Entries, the records are entries, and the fields that can be modified are generally ones that have a one-to-one relationship with entries. An exception is made for pronunciations which have a one-to-many owning relationship. In this case, any change you make only affects the first pronunciation object. You cannot bulk edit additional pronunciations. In future versions of FieldWorks it would be fairly easy to add additional bulk edit views for allomorphs, example sentences, and similar objects.

Some of the fields that can be bulk edited actually do much more than simply altering a string, as it appears in the bulk edit window. For example, when you bulk edit a sense reversal, it actually finds or creates entries in the reversal index and sets a reference from the sense to that entry. For this reason, some bulk edit operations work much faster than others, where more work is being done underneath the apparent simple change.

1.1 Limitations

There are a number of difficulties with bulk editing in Flex related to objects and properties on the objects. In the two-dimensional layout of bulk edit we show a list of record objects such as entries in Bulk Edit Entries and senses in Bulk Edit Senses. We can easily add any string, integer, date, or Boolean properties of the current object as columns and make them editable. Atomic references to possibility items are also quite easy to add. However, beyond this, things start to get complex.

For atomic owned objects (e.g., Lexeme Form, Etymology fields), we can readily show and change these because there is a one-to-one relationship between the record object and these fields. There is a higher level of complexity with making changes to these, though, because the owned object may not exist, so we are not simply modifying a string, but also creating an object to hold that string.

We run into problems, however, when we want to show and edit fields for sequences (e.g., semantic domains or example sentences in a sense). Except in certain specialized situations, there isn't any clear way to show multiple items for each record object in a two-dimensional display. For semantic domains and other sequence list items, we provide a semicolon-delimited list and a special chooser that allows you to bulk-edit these. Also, when you sort on semantic domains, it actually gives one record per semantic domain, which means senses may show up in the list multiple times. This starts to get confusing.

But how do you handle example sentences which is implemented as a sequence of objects with each one having multiple fields? At this point we don't have a way to do this without creating a new tool called bulk edit example sentences. We could probably show a column that would contain a semicolon delimited list of sentences or translations, but there isn't any easy way to make these editable. We've cheated a bit on this with Pronunciations which are a similar thing and decided we would only show the first one and allow you to change that. But this means additional pronunciations are left out unless we add a Bulk Edit Pronunciations tool.

Other fields, such as reversals on senses are made to look like semantic domains, but underneath it requires considerable work because reversals are actually objects owned in the index that reference senses.

In Bulk Edit Senses, we developed a way to show a property directly on the owning entry, but we have not made these editable via bulk edit because of the confusion of what it means if a person checks some senses of an entry but not others of the same entry, and then specifies a change to an entry field. Also, trying to bulk edit sense fields from Bulk Edit Entries is disabled because entries can have multiple senses, and there isn't any clear way to designate what it really means to bulk edit them.

Other kinds of changes are desirable, but involve more structure than is possible in this environment. Future versions of FieldWorks should include ways for doing some of these more complex operations. Otherwise, these complex operations either need to be done manually or automatically via

- SQL scripts in the database
- CC (or some other process on the XML dump file), or
- Python (or some other programming language that can work at the FieldWorks Data Object level).

These techniques are described in other papers.

1.2 Basic bulk edit steps

1. Choose the bulk edit view that gives you the desired fields (e.g., Entries, Senses, or Reversal Entries).
2. Select the columns (fields) and writing systems you want to use by clicking the column selector (just above the right scroll bar), click More Column Choices, then add, delete, or reorder fields in the right column of the Column Choices dialog to get the desired fields.
3. Select one or more filters to limit the records (rows) and use the combo box at the top of a column to activate a filter on that column.
4. Narrow the choices further by globally or individually setting or clearing check boxes on the left side of each record.
   Tip: The ☑ checkmark in the column header brings up a menu to set or clear all checkboxes.
5. Select the type of bulk edit operation you want to use (e.g., List Choice, Bulk Copy, Click Copy, Process, Bulk Replace, or Delete).
6. Select the target field and fill out the remaining elements for the bulk edit operation.
7. Click Preview to check that the changes you want to make work properly.
8. If not, click Clear, fix the problems, and try again.
9. When satisfied with the changes, click Apply to make the changes permanent.

1.3 Bulk edit tab summary

- List Choice allows you to set or change references to list items.
- Bulk Copy allows you to copy the contents of one field to another field, with options to overwrite, leave, or append to what is already there.
- Click Copy allows you to click a word in one field and copy it to the target field. In the target field, the copied word can overwrite or append to what is already there. It can also copy a string with the clicked word coming first.
- Process allows you to apply some process when copying from the source to the target field (may be the same field), with options to overwrite, leave, or append to what is already there. Processors use Unicode to Unicode SIL Encoding Converters, so you can use CC, ICU transducers, TECkit, Regular Expressions, Python, or Perl (Python and Perl require a special setup).
- Bulk Replace allows you to replace a given string with a replacement string in the target field. You can use Regular Expressions in this operation, which provides quite a bit of power.
- Delete allows you to clear target field contents or delete the selected records.

Caution: You may lose embedded writing systems or styles in strings that are changed in bulk edit. Find/Replace only affects characters that are *actually* changed. Process will lose *all* style and writing system information if you make *any* change to the string.

1.4 Known issues

FieldWorks might have an intermittent bug that sometimes keeps bulk edit operations from happening. If you encounter this, stop and restart the program and it then seems to work fine.

2 Normalization issues

Normalization is covered in Unicode Introduction.doc. FieldWorks stores *all* data internally in NFD format for consistency. NFD not only decomposes diacritics, but also other composite characters such as Korean Hangul syllables. FieldWorks makes every attempt to support canonical equivalence so users cannot tell the difference in the way it is stored. However, in a few cases it is important for users to realize that data is stored in NFD format:

- The Process tab of bulk edit feeds NFD data to the processor, so the processor *must* be prepared to work with UTF-8 NFD data.
- The Find Lexical Entry dialog does *not* provide regular expression searching. If you type "ne", it will include *all* forms that start with "ne" whether the ‘e’ has diacritics or not. This is generally not a problem as you are looking for a single word here. By typing the next letter, you eliminate this ambiguity.
- When setting filters or doing Search and Replace, in a few subtle situations the underlying NFD format makes a difference.

Example: You may have two forms of ‘maiden’—one with a grave accent over the “e” and the other without. If you use Bulk Replace to replace “de” with “da”, you get various results depending on what options you checked in the dialog.

- With “no” options checked:
  maiden ⇨ maidan
  maidèn ⇨ maidan
  It ignores diacritics, thus they are replaced along with the base character.
- With “Match diacritics” checked:
  maiden ⇨ maidan
  maidèn
  Words with diacritics are not changed.
- With “Use regular expressions” checked:
  maiden ⇨ maidan
  maidèn ⇨ maidàn
  Since the diacritics follow the vowel in NFD, the search changes the “de” without altering the following diacritic.

Any one of these options might be desirable, depending on what you want.

If you want to search for a grave diacritic without regard to the base character, enter U+300 COMBINING GRAVE ACCENT in the Filter or Find dialog. One way is to use the Insert Character tool in ZEdit while in UTF-8 mode.

- If you check “Match diacritics” it finds words with this diacritic, but misses the word-final diacritics. This is probably a FieldWorks bug.
- If you check “Use regular expressions” it finds all occurrences. You can also use \u0300 to find the grave accent instead of pasting it in.

3 Identifying underlying code points

There are at least three ways that you can identify underlying code points if it is not obvious.

A. Use ZEdit.

1. Launch ZEdit.exe from c:\Program Files\SIL\FieldWorks.
2. Choose Options...ReOpen As...Unicode (UTF-8).
3. Copy some text from FieldWorks and paste it into ZEdit.
4. In ZEdit, choose Tools...Show Character Codes.
   Result: Shows the hex value of code points before and after the cursor and updates as you move the cursor.
   Tip: To add specific code points in ZEdit you can choose Tool...Insert Characters. You then type hex codes with code points separated by spaces. When you click OK, the code points will be inserted into the file. You can then copy those and paste them into Flex, if you desire.

B. Use the process feature of bulk edit.

1. Go to Bulk Edit Senses or Bulk Edit Entries and make sure the field you want is showing.
   Tip: Use the column selector just above the right scroll bar to change columns.
2. Use the column header ☑ checkmark to Uncheck All rows, then check the box on the row you want to test.
3. Go to Process and set the source and target fields to the column you want to test, then click "Overwrite".
4. Choose *one* of these options:
   A. If you already have a Hex process set up, go to step 5.
   B. If not, click Setup and do the following:
   - Click Add.
   - For Processor Type, choose “ICU Unicode to Unicode transducer”.
   - For Process options, choose “Any to Hex Escape/Unicode”.
   - For Name, type “hex” and then click OK.
5. In the Process combo, choose Hex and click Preview.
   Result: Shows all the Unicode code points for the chosen string.
   Caution: Make sure you *do not* click Apply! But if you do, there is another ICU Unicode to Unicode transducer called Hex Escape to Any/Unicode transducer that will convert the escape sequences back into normal code points.

C. Use Alt+x in Microsoft Word.

1. Copy the code point you want to check into Microsoft Word.
2. With the cursor to the right of the character you want to identify, press Alt+x.
   Result: Word inserts the hex value to the right of the character.
   Tip: To add a specific code point in Word, type the 4-digit hex code, then with the cursor to the right of the code, press Alt+x. Word will convert the code into the actual character. It can then be copied to Flex, if you desire.

4 Regular expressions

Regular expressions add power to filters and also to search and replace. They work somewhat like wildcards in Microsoft Word, but provide a much richer set of options. As with other powerful features, it takes experience to use them reliably. It is quite easy to make a mistake in the regular expression and not get what you wanted. If you are replacing data, carefully check the Preview to make sure things are working as expected *before* clicking Apply.

The Language Explorer help file contains useful information on regular expressions. Type in Regular Expression in the index and read the subsections under this topic.

To use regular expressions in either the Filter dialog or the Find and Replace dialog, make sure you check “Use regular expressions”. If this does not show in Find and Replace, click More. When in the regular expression mode, a button to the right of the edit box(es) provides a list of many regular expression elements as well as a link to help information. The Help file lists regular expression operators and metacharacters and quite a few useful examples. With experimentation and experience, you can do many useful things with regular expressions.

When using regular expressions, watch for boundary conditions. For example, “^\s+->” removes leading spaces and “\s+$->” removes trailing spaces. But if you put them together as “^\s*(\S*)\s*$->$1”, it only removes leading and trailing spaces for strings that *do not* have internal spaces.

The “de\P{M}” expression filters on everything containing “de” without a diacritic but fails to catch “de” at the end of the string since nothing at the end matches the “\P{M}”. A more reliable approach is to use the negative look-ahead option “de(?!\p{M})”. This is true if the diacritic does *not* occur at the input position and does *not* advance the input pointer. Thus, it will also match “de” at the end of the string. Entering “de\p{M}” will find every “de” with a diacritic over the “e”.

“\p{x}” and “\P{x}” match Unicode general categories. Many of these categories have two letters.

Category Examples

- Mn is Mark, Non-Spacing.
- Mc is Mark, Spacing Combining.
- Me is Mark, Enclosing.

Just specify “M” to match any category starting with “M”. “\p{P}” finds any type of punctuation, and “\p{N}” finds any type of number. For a list of general categories, see www.unicode.org/Public/UNIDATA/UCD.html.

4.1 Examples

- \x{300} matches Unicode U+0300, which is a combining grave accent. You can enter any Unicode hex value inside the braces to find that character.
- \wing\b matches words anywhere in a string with an “ing” suffix. \w matches a word character and \b matches a word boundary.
- \W finds all strings with spaces or other non-word characters, such as punctuation.

- .+ matches all characters in a string with at least one character. If you replace this with nothing, you can clear out fields. (You can also use the bulk edit Delete tab.)
- d(e|a)f matches all strings with “def” or “daf”.
- b[aeiou]\p{M}*g matches all strings with “bVg”, where “V” is a vowel with or without diacritics. In this case, any one of the characters in the square brackets *must* be present. “\p{M}” matches any diacritic. The “*” following the diacritic means it can occur 0 or more times. As long as the Match Case check box is clear, this matches uppercase as well as lowercase letters.

4.2 Known issue

At least in some Indic writing systems, FieldWorks 5.4 has some problems with the filter dialogs. In these cases, the radio buttons for Anywhere, Whole item, At end, and At start may not work properly, possibly leaving out many matches. This appears to have something to do with normalization and some ICU problems. If you suspect that you aren’t seeing all of the expected results, try using regular expressions instead. It seems to be more reliable. You can use regular expressions to perform the same functions as the radio buttons:

- By default, regular expressions should match Anywhere as long as you don’t include any special regular expression characters in the string.
- ^abc should be the same as searching for abc At start.
- abc$ should be the same as searching for abc At end.
- ^abc$ should be the same as searching for abc Whole item.

5 Bulk Edit Process tab

Why use the Process tab in bulk edit? You can often use Find/Replace with or without regular expressions to make desired changes. When you need more power than this method provides, use CC or other processors.

In the Process tab, any embedded writing systems or styles are lost on strings that are changed. The bulk edit processor gets a plain string with no indication of embedding.

The input strings to the processor are UTF-8 NFD data, so they need to work with decomposed data. The processor output *must also* be UTF-8 data. The normalization is *not* important, since FieldWorks forces NFD prior to saving it in the database.

The bulk edit Process combo *only* shows Unicode-to-Unicode converters. Thus, it does *not* show any converters you may have installed for converting legacy fonts to Unicode or vice versa.

5.1 CC processor

Here are the steps to set up a CC process for bulk edit:

1. Create a CC table in a separate text file.
2. In the bulk edit Process tab, click Setup.
3. In the dialog, set the Processor Type to CC.
4. Select your CC table for the Processor File.
5. Enter a name for the process.

6. Click OK to close the dialog and make sure it is selected in the Process combo.

5.1.1 Considerations when writing CC tables for bulk edit:

- The table is called separately for each string. It is as though each field being processed is a separate file. This requires special techniques if you want to make changes at the beginning, end, or entire string as opposed to anywhere in the string.
- The CC table only applies to text. Your table *cannot* detect formatting or writing system changes. In fact, if changes are made in the field, *all* embedding is lost!
- CC tables are best prepared in a text editor (e.g., ZEdit or Notepad) instead of Word. In particular, quote marks *must* be ASCII quotation marks. Word typically converts these to Unicode equivalents (smart quotes) which breaks the CC processor. Word also typically inserts extra spaces in various places and capitalizes things you do not want capitalized, among other irritating things.
- In bulk edit, all strings fed to the CC program are UTF-8 NFD data. This means that ‘a’ with an acute diacritic will be u61 u301 instead of ue1. For more information on normalization, see Unicode Introduction.doc/.ppt.
- Since bulk edit always works on UTF-8 data, at the top of your table it’s best to include
  ```
    begin > utf8
  ```
  to make sure *fwd, back*, *omit*, *prec*, *fol*, *any*, and *word* commands treat your code points as characters rather than bytes (part of a UTF-8 character). For example, if you have the change
  ```
    'ΣΣΣ' > dup back(2) 'x'
  ```
  with the utf8 command the result will be the expected ΣxΣΣ, while without the utf8 command, you’ll get ΣΣxΣ.You can also use UTF-8 specific commands such as *fwdu, backu, omitu, precu, folu, anyu,* and *wordu* regardless of whether you have the utf8 command.
- Non-ASCII code points can be entered using the hexadecimal code point with the uN syntax,
  ```
    u3ba u3b9 u3bb > u39a u399 u39b
  ```
  or using actual UTF-8 code points
  ```
    'κιλ' > 'ΚΙΛ'
  ```
  Caution: With UTF-8 code points make sure you are using decomposed values. In ZEdit UTF-8 mode you can use Tools…Insert Characters to control the code points exactly.

5.1.2 Sample CC table for bulk edit processor

```
c Change prefixes, suffixes, and whole words in bulk edit
c Note input from FieldWorks is UTF-8 NFD

begin > utf8

group(first)
c Process string-initial prefixes
'n' u303 > 'XX' use(main) c Change initial ñ to XX
'pre2' > use(main) c Delete pre2 as a prefix (or a whole word)
```

```
c Process whole string
'n' u303 'uma' endfile > 'XYZ' endfile c change ñuma to XYZ
'word2' endfile > endfile c Delete word2
'' > use(main)

group(main) c Active in middle of string
c Process string-final suffixes
'na' endfile > 'ZZ' endfile c Changes final na to ZZ
'suf2' endfile > endfile c Delete suf2 at end of field (or whole word)
```

The two lines beginning with ‘n’ demonstrate dealing with UTF-8 NFD data. Even though the character in Flex appears as ‘ñ’, Flex stores this as an ‘n’ followed by a combining tilde diacritic. In your CC table, any non-ASCII character in either a match or output portion of a command must use the CC Unicode syntax with ‘u’ followed by the hex value of the Unicode code point.

To catch something at the beginning of the string (prefixes), include this in the first group of your CC table. Every command in this group must switch to another group when done, unless it actually closes the file. Also, you need a null match that does the same thing for strings that did not have one of the specified prefixes. Note: The table needs to change slightly if you want to distinguish between a prefix or an entire word with the same string.

To process whole strings (words), you also need to include these in the first group of your CC table, and the match string must end with ‘endfile’. In these cases, the replacement string should also include ‘endfile’ to close the process.

To catch something at the end of a string (suffixes), include these in some group other than the first group and each match string must include ‘endfile’. In these cases, the replacement string should also include ‘endfile’ to close the process.

Note: The table needs to change slightly if you want to distinguish between a suffix or an entire word with the same string.

The “Sample CC table designed for a bulk edit processor” table produces the following results:

ñuma ⇨ XYZ
ñum ⇨ XXum
word2 word3 ⇨ no change
word2 ⇨ deleted
pre2other ⇨ other
to cana ⇨ to caZZ
thesuf2 ⇨ the
natha ⇨ no change

5.1.3 Sample CC table to delete initial and final spaces and delete final punctuation

```
c Delete initial and final spaces and closing punctuation
c Note input from FieldWorks is UTF-8 NFD
```

```
begin > utf8 store(p) '.?!:;' endstore

group(first)
'  ' > ' ' back
' ' > use(main)
'' > use(main)

group(main)
'  ' > ' ' back
' ' endfile > endfile
any(p) endfile > endfile
```

5.1.4 Sample CC table that converts words into CV patterns

```
c Convert string into CV pattern
c Note input from FieldWorks is UTF-8 NFD

begin > utf8 store(v) 'aeiouAEIOU'
    store(c) 'bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ'
    store(d) u300 u301 u302 u303 u304 endstore

group(main)
any(v) > 'V'
any(c) > 'C'
'ch' > 'C' c Digraphs
any(d) > c Delete diacritics
```

This table converts vowels to “V” and consonants to “C”. Since Flex uses NFD, vowels with diacritics are two separate code points. Simply delete the combining diacritical marks.

5.1.5 Sample CC table that converts words to CVNS patterns

This table converts vowels to “V” and consonants to “C”, semivowels to S, and nasals to N.

```
c Convert string into CVNS pattern
c Note input from FieldWorks is UTF-8 NFD.

begin > utf8 store(v) 'aeiouAEIOU'
    store(c) 'bcdfghjklpqrstvwxyzBCDFGHJKLPQRSTVWXYZ'
    store(n) 'nmNM'
    store(s) 'wyWY'
    store(d) u300 u301 u302 u303 u304 endstore

group(main)
any(n) any(c) any(s) > 'NCS'
any(n) any(c) > 'NC'
any(c) any(s) > 'CS'
any(n) any(s) > 'CS'
any(v) > 'V'
any(n) > 'N'
any(c) > 'C'
'sh' > 'C'  c Digraphs
any(d) > c Delete diacritics
```

5.1.6 Sample CC table to extract \ge fields

This table returns the contents of \ge fields imported into the ImportResidue field. If they occur more than once, it appends the additional fields with a semicolon and space. The Source Field for this change is ImportResidue and the Target is the custom field or some other field that needs the contents of the \ge fields.

Known issue: One potential problem with this process is that embedded writing systems and styles are lost with this approach.

```
c Return \ge fields, appending multiple ones together.
c Return null if the field doesn’t contain a \ge.

begin > utf8

group(main)
' \ge ' > next
'\ge ' > ifneq(field) ''
    begin
            '; '
    end
    append(field) use(copy)
endfile > ifneq(field) ''
    begin
            out(field)
    end
    endfile
'' > omit

group(copy)
endfile > dup back use(main)
' \' > '\' back
'\' > dup back use(main)
```

5.1.7 Sample CC table to delete \ge fields

After using the above table to copy the \ge contents to another field, you can use this table to delete the \ge fields from ImportResidue. With this table, the Source Field and Target Field are both ImportResidue.

```
c Delete \ge fields and return the rest of the original string.
```

```
begin > utf8 set(blank)
group(main)
' \ge ' > next
'\ge ' > store(field)
' \' > endstore clear(blank) dup
'\' > endstore clear(blank) dup
endfile > endstore endfile
```

5.2 ICU Unicode to Unicode transducer processors

ICU provides many transducers that may be useful. Many of these convert non-Roman orthographies to and from some other orthography, such as a Latin based orthography. For example, this is how you can set up a transducer that converts from Greek characters to a Romanized transcription.

1. In the Process tab, click Setup.
2. Click Add to add a new processor.
3. In the Processor Type combo, choose “ICU Unicode to Unicode transducer”.
4. In the Process options combo, choose “Greek to Latin”.
5. Enter a name such as Greek to Latin.
6. Click OK to close the dialog.

Now if you have a field with this Greek text

καὶ ἐξαλείψει ὁ θεὸς

and run the bulk edit process on this field, it will be converted to

kaì exaleípsei ho theòs

This is more readable to an English reader than the Greek characters. If you make another converter and pick “Latin to Greek” and run that on the converted text, it will return your text to the Greek orthography.

Here are some other useful ICU transducers.

- Any to Hex Escape/Unicode. If you set up this converter and preview it on the Greek text above, you will see
  U+03BAU+03B1U+03B9U+0300U+0020U+03B5U+0313U+03BEU+03B1U+03BBU+03B5U+03B9U+0301U+03C8U+03B5U+03B9U+0020U+03BFU+0314U+0020U+03B8U+03B5U+03BFU+0300U+03C2
  These are the Unicode values for each character. This transducer provides one way to determine the underlying Unicode values for a string.
- A corresponding ICU transducer is Hex Escape to Any/Unicode that converts this form back to the original text.
- FieldWorks feeds NFD data to processors. If you have a processor that requires NFC, and it cannot easily be rewritten, the ICU transducer Any to NFC forces the output to NFC. You can create a Compound converter that first calls Any to NFC then calls the main processor. See “Other processors” below for more information.
  Note: You cannot do this in two passes in bulk edit because FieldWorks will automatically change the text back to NFD after the first pass.
- The Any to Upper transducer forces any text to uppercase.
- The Any to Lower transducer forces any text to lowercase.

5.3 TECkit processor

It is possible to develop a TECkit converter that converts Unicode to Unicode. For example, NRSI has produced one of these to convert deprecated PUA characters to their new Unicode values. If you have a converter like this, you can use it in the bulk edit processor by selecting TECkit (compiled TEC file) or TECkit (source MAP file) in Setup Processor…Processor Type combo.

5.4 Regular Expression processor

You can use regular expressions in the Bulk Replace tab both in the Find and Replace edit boxes, but you need to type in the expressions each time. If you find a combination that you keep using over and over, add it as a processor in the Process tab:

1. In the Process tab, click Setup.
2. Click Add to add a new processor.
3. In the Processor Type combo, choose “Regular Expression (ICU)”.
4. In the Find->Replace edit box, enter the change you want to make.
   Tip: The match target comes first, followed by the two characters “->”, then the replacement string.
5. Enter a meaningful name.
6. Click OK to close the dialog.

Examples

- ^\s+-> removes leading spaces from a field.
  ^ matches the beginning of the string and \s+ matches 1 or more white space character(s). On the replace side is nothing, thus it deletes the matched text.
- \s+$-> removes trailing spaces from a field.
  $ matches the end of the string and \s+ matches 1 or more white space character(s). On the replace side is nothing, thus it deletes the matched text.
- [aeiou]\p{M}*->V converts all vowels with following diacritics to V.

5.5 Other processors

It is possible to use other processors, such as Python, Perl, and Compound converters. As long as they are set for Unicode-to-Unicode conversion, they show up in the Process combo and work with bulk edit.

5.5.1 Python ToUpper processor

The following steps demonstrate how to set up and use a Python processor in the Bulk Edit tools. This assumes that you have a version of Python installed on your machine. In the case of IronPython, create this registry key:

HKEY_LOCAL_MACHINE\SOFTWARE\Python\PythonCore\2.4

and add an InstallPath string variable that gives the path to your ipy.exe. (On development machines, PythonEC24.dll must be registered.)

1. Store the following Python code in a file, ToUpper.py.
   ```
   def ToUpper(s):
     return unicode.upper(s)
   ```

Note: When writing functions you need to use unicode functions (e.g., unicode.upper(s)) instead of str functions (e.g., str.upper(s)).

2. In Bulk Edit select the Process tab, then click the Setup button.
3. Click Add to create a new converter, then click the More button to get to the advanced converters dialog.
4. In the 'Choose a Transduction Engine' dialog, choose Python Script and click Add.
5. In the Setup tab, locate your ToUpper.py file in the 'Python script file' box and make sure the Function Name is set to ToUpper. Also, make sure 'Unicode String' is selected in both radio buttons. If you want, you can use the Test tab to test the converter.
6. Click the 'Save in System Repository' button, giving it a name such as 'To Upper'.
7. Close OK to close the two dialogs.
8. You should now be able to select 'To Upper' in the Bulk Edit Process combo box. (This combo box lists all Unicode to Unicode converters even though they cannot be seen or modified in the current Setup dialog.)

In a similar way you can add Perl scripts, assuming you have a Perl installation, and you can add Compound converters that chain multiple converters into a single converter.

5.5.2 Fixing a bad UTF-16 import

Suppose you accidentally import some Toolbox UTF-8 fields using the Windows1252<>Unicode converter. Hopefully you'll catch this right away and start over and do a clean import choosing the <Already in Unicode> option. However, if you fail to notice this until it's no longer practical to do a clean import, you can create a compound converter that will undo the mess.

Here's a small sample that demonstrates what happens in this case. Suppose you start with the following data:

\de aàbßc

The Unicode data has code points 61 e0 62 df 63 but the actual bytes in the file are 61 c3 a0 62 c3 9f 63 since the file is in UTF-8. When you accidentally chose the Windows1252<>Unicode converter, the import converts each byte into Unicode using the CP1252 code page. The resulting Unicode values are 61 c3 a0 62 c3 178 63. Note that 9f was converted to 178 because this is the Unicode equivalent of the original character in code page 1252. Now what you see in Language Explorer is 'aÃ bÃŸc'. However, in the FieldWorks database, these Unicode values are actually stored as decomposed (NFD) Unicode values, 61 41 303 a0 62 41 303 59 308 63. So to convert the mess back into the original Unicode characters, you'll need to first convert back to NFC, then convert back to UTF-8 in a single step. Here's how to do this in the Language Explorer bulk edit mode. It involves creating a converter for each step, then a compound converter to combine the two together.

1. In Bulk Edit select the Process tab, then click the Setup button.
2. Click Add to create a new converter with the following settings:
   Name: Any to NFC
   Converter Type: ICU Unicode to Unicode transducer
   Mapping Name: Any to NFC
   Conversion Type: Unicode to and from Unicode

3. Click Add to create another new converter. This time you need to use a converter that is normally used to convert from Legacy to Unicode. In this dialog FieldWorks normally does not provide options for going from Unicode to Legacy data, but in this case we are actually going from Unicode to Unicode using a Legacy converter. To create this converter, click the More button.
4. In the Choose a Transduction Engine dialog, pick ICU Converter and click Add
5. In the ICU Converter dialog, select the Setup tab and choose UTF-8.
6. Click the Save in System Repository button, and type in the name 'Back to UTF-8', then click OK to close both dialogs.
7. Back in the Setup Processor dialog, click Add again to add another converter and click the More button.
8. In the Choose a Transduction Engine dialog, pick Compound and click Add
9. In the Compound Converter dialog, select the Setup tab and select 'Any to NFC' in the combo box, then click Add Step.
10. Now select 'Back to UTF-8' in the combo box and click Add Step.
11. Click OK, and enter the name 'Undo bad import' then click OK.
12. Click Close to close the Setup Processor
13. You can now use normal bulk edit operations with this 'Undo bad import' converter to fix any fields that were imported incorrectly.

I'm somewhat puzzled by how this works. It seems we should need another converter between the two converters above to convert the Unicode back to CP1252, but it did this automatically on my US-English machine. I suspect the Back to UTF-8 converter is doing the conversion automatically because my default code page is CP1252. If your default code page is something else, you may need to modify the above procedure.

6 Bulk Edit Bulk Replace tab

The Bulk Replace tab allows users to set up a Search and Replace to change strings in a given field. It provides regular expression Search and Replace capabilities. See "Normalization" and also "Regular expressions" for helpful information in using Find and Replace.

In the Bulk Replace tab you can set up a Find and Replace operation that changes a specific replacement string to a given writing system and/or style:

1. Highlight all or part of the replacement string.
2. Click Format and choose the writing system and/or style you want to apply.
   Result: Selects the "Match writing system" check box when you select a writing system.

Unfortunately, you *cannot* use wildcards when doing this. When writing systems are imported incorrectly, it would be nice to force an entire field to a given writing system. With regular expressions you *can* replace (.*) with $1 to effect a copy. But when regular expressions are enabled, Flex disables the ability to set writing systems or styles. So at this point you would need to do this in some other way, such as in an XML dump file.

CAUTION! FieldWorks 5.4 still has a bug in the replace dialog that can mess up your writing systems when the target field is non-English and a String or MultiString field. The replacement string defaults to English, unless you change it to something else. This means if you make a replacement in a French definition, the replacement will be an embedded English string. You can avoid this in the Find and Replace dialog by setting the correct writing system in the Format…Writing System option in the expanded mode of the dialog.