FieldWorks 7 XML model

Ken Zook

July 16, 2015

Contents

1 XML introduction 1
2 FieldWorks XML model 5
2.1 Basic properties 5
2.1.1 Strings 5
2.1.1.1 Unicode or BigUnicode 5
2.1.1.2 String or BigString 6
2.1.1.3 MultiUnicode or MultiBigUnicode 7
2.1.1.4 MultiString or MultiBigString 7
2.1.2 Other basic properties 8
2.2 Writing systems 8
2.3 Class instances 9
2.4 Owning properties 9
2.5 Reference properties 10
2.6 Top level XML 11
2.7 Custom fields 12
3 Source of FieldWorks XML files 14
4 FDOBrowser 15
5 Data protection, and recovering from a failure 16
5.1 Non-shared 16
5.2 Shared 17
6 Modifying data in FieldWorks XML files 17
6.1 Making spelling/orthographic changes in one writing system 17
6.2 Converting bar codes to formatted text 18

1 XML introduction

XML (Extensible Markup Language) is a standard file format for storing data along with its semantic markup. The official description is found at www.w3.org/TR/REC-xml. It provides numerous advantages over SFM including attributes and clear structural markers. Starting with a simple SFM dictionary entry:

\lx abal
\ps adj
\gl turbulent
\de Turbulent with rip tide.
\xv Maabal tuud ha tungud Duway Bullud.
\xe The current is very turbulent near Duway Bullud.
\ps v
\gl become turbulent
\de For the ocean (a rip tide) to become turbulent.
\xv Umabal tuud in dagat bang hunas.
\xe The sea becomes very turbulent at low tide.

```
\xv Magabal in dagat bang lumabay in kappal.
\xe There will be rip tide when a boat passes.
\dt 18/May/2004
```

This can be turned into a basic XML file:

```
<?xml version="1.0" encoding="UTF-8"?>
<SFMDatabase>
<lx>abal</lx>
<ps>adj</ps>
<gl>turbulent</gl>
<de>Turbulent with rip tide.</de>
<xv>Maabal tuud ha tungud Duway Bullud.</xv>
<xe>The current is very turbulent near Duway Bullud.</xe>
<ps>v</ps>
<gl>become turbulent</gl>
<de>For the ocean (a rip tide) to become turbulent.</de>
<xv>Umabal tuud in dagat bang hunas.</xv>
<xe>The sea becomes very turbulent at low tide.</xe>
<xv>Magabal in dagat bang lumabay in kappal.</xv>
<xe>There will be rip tide when a boat passes.</xe>
<dt>18/May/2004</dt>
</SFMDatabase>
```

The top line is a special format that identifies this as an XML file and identifies the encoding. XML files usually use Unicode UTF-8 data. Following this is a single top-level XML element required by XML. XML elements *must* have a “begin” tag (enclosed in wedges) and a matching “end” tag (enclosed in wedges with a slash after the opening wedge). Inside the element can be embedded elements or raw text. The SFMDatabase element contains a sequence of elements, while the remaining elements contain raw text. This example starts each element on a new line, but that is optional. An XML processor will allow any amount of white space between elements.

Additional information can be provided that makes the data more useful:

```
<?xml version="1.0" encoding="UTF-8"?>
<SFMDatabase>
 <entry>
  <lx lang="tsg">abal</lx>
  <sense>
   <ps lang="en">adj</ps>
   <gl lang="en">turbulent</gl>
   <de lang="en">Turbulent with rip tide.</de>
   <example>
    <xv lang="tsg">Maabal tuud ha tungud Duway Bullud.</xv>
    <xe lang="en">The current is very turbulent near Duway Bullud.</xe>
   </example>
  </sense>
  <sense>
   <ps lang="en">v</ps>
```

```
    <gl lang="en">become turbulent</gl>
    <de lang="en">For the ocean (a rip tide) to become turbulent.</de>
    <example>
     <xv lang="tsg">Umabal tuud in dagat bang hunas.</xv>
     <xe lang="en">The sea becomes very turbulent at low tide.</xe>
    </example>
    <example>
     <xv lang="tsg">Magabal in dagat bang lumabay in kappal.</xv>
     <xe lang="en">There will be rip tide when a boat passes.</xe>
    </example>
   </sense>
   <dt value="18/May/2004"/>
  </entry>
</SFMDatabase>
```

Some additional structure has now been added to the data. An Entry element surrounds the entry, a Sense element surrounds senses, and an Example element surrounds examples. Attributes have also been added to identify the language of each string. Attributes are placed inside the begin element after the element name. Elements may have 0 or more attributes. Each attribute has a name and the contents enclosed in double or single quotes. The date element has also been changed to include the date in a value attribute rather than inside the element. If the element does not have any content, an abbreviated form of the end tag can be used (a slash just before the closing wedge of the begin tag).

Within attribute data and embedded element data, certain characters *must* be escaped to avoid confusion with the markup around the data. The main characters you need to escape in data are as follows:

```
<   ⇨ &lt;
>   ⇨ &gt;
&   ⇨ &amp;
"   ⇨ &quot;
```

These are predefined entities. For example, to indicate "ISO-8859-1<>UNICODE" the XML data needs to be "ISO-8859-1<>UNICODE". Actually, any Unicode value can be escaped using the hexadecimal numeric character references, such as “ኣ”.

The XML sample above uses a nested structure, which is fairly easy for a user to understand. This information can also be stored in a flat structure which may work much faster for a computer to process. Here’s an example of the same data in a flat structure.

```
<?xml version="1.0" encoding="UTF-8"?>
<SFMDatabase>
 <obj class="entry" id="1">
  <lx lang="tsg">abal</lx>
  <dt value="18/May/2004"/>
  <ownobj id=10>
  <ownobj id=20>
 </obj>
 <obj class="sense" id=10>
```

```
    <ps lang="en">adj</ps>
    <gl lang="en">turbulent</gl>
    <de lang="en">Turbulent with rip tide.</de>
    <ownobj id=30>
   </obj>
   <obj class="sense" id=20>
    <ps lang="en">v</ps>
    <gl lang="en">become turbulent</gl>
    <de lang="en">For the ocean (a rip tide) to become turbulent.</de>
    <ownobj id=40>
    <ownobj id=50>
   </obj>
    <obj class="example" id=30>
     <xv lang="tsg">Maabal tuud ha tungud Duway Bullud.</xv>
     <xe lang="en">The current is very turbulent near Duway Bullud.</xe>
    </obj>
    <obj class="example" id=40>
     <xv lang="tsg">Umabal tuud in dagat bang hunas.</xv>
     <xe lang="en">The sea becomes very turbulent at low tide.</xe>
    </obj>
    <obj class="example" id=50>
     <xv lang="tsg">Magabal in dagat bang lumabay in kappal.</xv>
     <xe lang="en">There will be rip tide when a boat passes.</xe>
    </obj>
  </SFMDatabase>
```

This format stores the same data as the previous example, but it is arranged in a different format. Instead of having senses nested in entries, and examples nested in senses, we just have a sequence of objects; each object indicating the class of object it is, and having a unique identifier. Basic data such as strings and dates appear within the object. Instead of nesting owned (internal) objects, we use a ownobj reference to the object that would be owned within this object. So the entry owns object 10 followed by object 20. To find what this is, we have to search above or below the current object to find the object with an id of 10 or 20. Likewise, the examples are separate objects that are referenced within senses using their id numbers.

FieldWorks versions up through 6.0 used a nested format in the XML format for data. A few list files in FW 7.0 still use the nested format. FieldWorks 7.0 started using a flat structure. These files are given a file extension of *.fwdata. A database, such as db4o can also be used to store a sequence of objects as rows with each row storing an XML string representing the data. In FieldWorks 7.0 this type of db4o database file is given a *.fwdb extension. Transforming the data between an XML fwdata file and a db4o fwdb file is very fast because it is basically a dump to and from the database.

There are various kinds of XML editors that can make editing XML files easier. XMLmind is a free XML editor that can be customized in various ways. Andy Black provides customizations for working with linguistics papers called XLingPaper. He also provides a way in Flex to dump interlinear text to XML that works with these customizations. ZEdit (installed with FieldWorks) handles large XML files with ease, although it does not provide any special XML help.

Microsoft Internet Explorer has a limited ability to check the basic integrity of an XML file. By default, when you double-click an XML file, it opens in Internet Explorer and displays it with buttons to expand and contract element hierarchy. If something invalid is in the file, it usually gives a useful error message near the end at the point where it failed. Long files, such as FieldWorks projects open very slowly in Internet Explorer and many other editors.

Once your data is in XML, you can use processes such as XSLT to transform it in numerous ways. This can work in conjunction with style sheets to display the data in many interesting ways.

Comments in XML are ignored in processing and are enclosed in <!-- and -->:

```
<!-- this is a comment -->
```

2 FieldWorks XML model

FieldWorks provides a standard way to represent the conceptual model as XML data. This XML file uses the *.fwdata file extension. This is the native format for storing FieldWorks data. This format is zipped up along with other relevant information in a FieldWorks backup file with a *.fwbackup file extension. This section describes the FieldWorks XML model.

Note: The LinguaLinks Workshops and SFM import processes transform the input files into a hierarchical XML format that is similar to FieldWorks 6.0 and earlier data. It is then imported into existing FieldWorks Projects.

In a FieldWorks fwdata file, every object has a single element that includes any basic properties (strings, integers, etc.) and reference lists of any owned or referenced objects. The objects can be in any order since the file is flat rather than hierarchical.

2.1 Basic properties

2.1.1 Strings

All strings other than Unicode or BigUnicode are identified with a writing system code (e.g., ws="en"). This writing system code should match the file name of a writing system *.ldml file contained in the WritingSystemStore folder within the project folder. This ldml file stores information relevant to the writing system. (See FieldWorks Writing Systems.docx for details.) To see this code in a FieldWorks program, go to the General tab of the Writing System Properties dialog. The Internal Code is shown at the bottom of the dialog.

Unicode and String have special meaning in FieldWorks. All data is actually stored as Unicode code points. The difference in these two terms is that Unicode strings do not allow any embedding including specifying a language, while String strings always contain a language and may have other embedding.

In the XML representation of strings, there is no difference between the short and long versions as there used to be in the SQL Server database version. As a result, you need to consider four basic kinds of strings.

2.1.1.1 Unicode or BigUnicode

The simplest type of string is a FieldWorks Unicode or BigUnicode string.

```
<EthnologueCode>
<Uni>XKAL</Uni>
</EthnologueCode>
```

This is the EthnologueCode property in LangProject. This kind of code and filenames do not have an inherent language, so they are specified as FieldWorks Unicode strings. There is a slight problem with this. FieldWorks needs to use a writing system in order to determine which font to use to display the string. At this point the FieldWorks display mechanism shows boxes for characters that are not in the current font rather than substituting other fonts as needed. FieldWorks generally chooses a writing system representing the system language on your computer, if the writing system exists, otherwise it defaults to English when displaying these strings.

2.1.1.2 String or BigString

Normally a string will have a single run, but to demonstrate some of the complexity you may encounter, this demonstrates how a complex FieldWorks String or BigString is represented:

```
<Contents>
<Str>
<Run ws="en">Went fishing with </Run>
<Run ws="de">Rolf</Run>
<Run ws="en"> at the </Run>
<Run ws="en" tags="I2121F090-4744-4BC0-AE46-3E0D48C071B5 I2BDD0E85-F9B2-
11D3-977B-00C04F186933">river</Run>
<Run ws="en" >. Within no time, he had caught about 10 </Run>
<Run ws="en" bold="invert" fontFamily="Adams Extended">fierce</Run>
<Run ws="en"> looking </Run>
<Run ws="en" namedStyle="Emphasized Text">pirana</Run>
<Run ws="en"> which we </Run>
<Run ws="en"
externalLink="silfw://localhost/link?app%3dflex%26database%3dC%3a%5cWork%5cFW7
Projects%5cTestLangProj%5cTestLangProj.fwdata%26tool%3dlexiconEdit%26guid%3d524
4672f-ee85-4b5e-ae2a-6c03ece57689%26tag%3d" namedStyle="External
Link">subsequently</Run>
<Run ws="en"> cleaned and ate for </Run>
<Run ws="en" externalLink="C:\Dropbox\Cooked pirana.jpg" namedStyle="External
Link">breakfast</Run>
<Run ws="en">.</Run>
<Run ws="en"
type="picture">424D262C00000000000036…04000028000000860000</Run>
</Str>
</Contents>
```

This is the Contents property of an StTxtPara object. It has many embeddings to demonstrate the rich capabilities of FieldWorks Strings. FieldWorks Strings contain a sequence of Runs inside a Str element, with each run specifying a writing system. All characters in a given run share the same attributes. Whenever a set of attributes changes, it requires a new Run. This FieldWorks String starts with a Run of English characters, then contains an embedded German word

(ws="de"). The word "river" has two overlay tags applied to the word. These were used in the data notebook to allow portions of strings to reference list items (not implemented in FieldWorks 7.0). The GUIDs in these cases are the FieldWorks IDs for two list items. The word "fierce" is hard-coded by inverting the bold toggle and setting a specific font. The word "pirana" is tagged with a style. The formatting associated with the Emphasized Text style is determined by an StStyle elsewhere in the system. The string "subsequently" is formatted with an External Link style, plus it holds a hot link to a FieldWorks lexical entry. The word "breakfast" also has an External Link style and references an external picture file. The final run represents an embedded object, which in this case is a picture, encoded as a long string of hex codes. FieldWorks does not use this way of inserting pictures anymore, but this demonstrates how other objects can be embedded in the string.

The value for the “ws” attribute in strings is the name of the associated LDML file in the WritingSystemStore directory inside the project directory.

2.1.1.3 MultiUnicode or MultiBigUnicode

Many properties (fields) in FieldWorks are designed to hold strings that will never have embedding, but may have translations in different languages or writing systems (e.g., lexeme form, gloss, and names and abbreviations for list items). Here is how these MultiUnicode or MultiBigUnicode properties are encoded.

```
<Name>
<AUni ws="en">adjunct</AUni>
<AUni ws="fr">accessoire</AUni>
<AUni ws="es">adjunto</AUni>
</Name>
```

This field is the Name for a CmPossibility object. In this case, there are three FieldWorks Unicode strings; English, French, and Spanish. Each FieldWorks Unicode string is stored in an AUni element. Even though the FieldWorks Unicode string itself does not have a writing system, FieldWorks knows the intended writing system by the “ws” attribute of the AUni element. It is an error to have more than one AUni element in a single property with the same writing system.

2.1.1.4 MultiString or MultiBigString

Other properties (fields) need the ability to store equivalent translations, but also need to allow things such as embedded writing systems and formatting. Here is how these MultiString or MultiBigString properties are encoded.

```
<Translation>
<AStr ws="en">
<Run ws="en">You may damage the computer.</Run>
</AStr>
<AStr ws="fr">
<Run ws="fr">Vous risquez d'endommager l'ordinateur.</Run>
</AStr>
</Translation>
```

This field is the Translation property on a CmTranslation object. This example has translations for English and French. This is a simple FieldWorks String in that it only contains one run, but

any number is possible. Each FieldWorks String is stored in an AStr element which identifies the overall writing system for the FieldWorks String. Normally this is the same writing system as the first run, but it does not have to be. You could have an English string that starts with a French word. It is an error to have more than one AStr element in a single property with the same writing system.

2.1.2 Other basic properties

Integer properties are represented as follows:

```
<HomographNumber val="1"/>
```

This is the HomographNumber property on LexEntry. The value of the integer is stored as a decimal "val" attribute. The default value for missing properties is false.

Boolean properties are represented as follows:

```
<IsSorted val="true"/>
```

This is the IsSorted property on CmPossibilityList. The value is stored in the "val" attribute as "true" or "false". The default value for missing properties is false.

GUID properties are represented as follows:

```
<ListVersion val="b41ff27f-5caf-4eac-92de-0f92acb0caa3"/>
```

This is the ListVersion property on CmPossibilityList. The GUID value is stored in the "val" attribute in this string format.

Binary properties are represented as follows:

```
<Details><Binary>6400000001000000</Binary></Details>
```

This is the Details property on UserView. Binary data is stored as a sequence of hexadecimal bytes.

Time properties are represented as follows:

```
<DateModified val="2009-9-23 20:53:23.390"/>
```

This is the DateModified property on CmMajorObject. The time is stored in the "val" attribute in the format YYYY-MM-DD HH:MM:SS.TTT.

GenDate properties are represented as follows:

```
<DateOfEvent val="193112111"/>
```

This is the DateOfEvent property on RnEvent. The generic date is stored as a decimal number in the "val" attribute.

2.2 Writing systems

Unlike previous versions of FieldWorks, Version 7.0 does not store writing system definitions in the data file. Writing systems for strings are still identified by a writing locale identifier, but the definition for that writing system (name, font, keyboard, collation, etc.) are now stored in a Unicode Locale Data Markup Language (LDML) file with the name being the locale identifier and an ldml extension. This file is an XML file that follows a Unicode standard defined in Unicode Technical Standard #35 (www.unicode.org/reports/tr35).

Each writing system used in the project data file has a corresponding LDML file in the WritingSystemStore directory directly under the project folder. In addition it is stored in the global WritingSystemStore directory located in C:\ProgramData\SIL (Vista and Windows 7), and C:\Documents and Settings\All Users\Application Data\SIL (Windows XP). More information on these files is available in FieldWorks 7 Writing systems.docx.

2.3 Class instances

Instances of classes are stored in an XML file as an rt element. Here is an example of the back translation CmPossibility item:

```
<rt class="CmPossibility" guid="80a0dddb-8b4b-4454-b872-88adec6f2aba"
ownerguid="d7f71649-e8cf-11d3-9764-00c04f186933">
<Abbreviation>
<AUni ws="en">BT</AUni>
</Abbreviation>
<DateCreated val="2004-9-11 2:17:31.153" />
<DateModified val="2004-9-11 2:19:0.0" />
<ForeColor val="-1073741824" />
<IsProtected val="true" />
<Name>
<AUni ws="en">Back translation</AUni>
</Name>
<Discussion>
<objsur t="o" guid="3aeef2e2-9466-41d5-afa7-d569f667fc79" />
</Discussion>
</rt>
```

Every instance of a class has an ‘rt’ element with at least class and guid attributes. The class attribute defines the class of the object. The guid attribute defines the unique identifier for this instance of the class. Most objects are owned by some other object. If so, they have an ownerguid attribute that references the guid of the owning instance. Unlike the older FieldWorks XML files, the guid attribute must be a valid GUID, not a unique string as before. If you need to generate guids outside of a programming environment, you can download a free GuidGen program from http://users.csc.calpoly.edu/~bfriesen/software/console.shtml. This can be run in a Cmd window to generate one or any number of guids. Type GuidGen -? for help.

The properties on the class are stored as nested elements. All of the properties in this example except Discussion are basic properties. Discussion is an owning property and will be discussed in the next section.

2.4 Owning properties

Owning properties are stored as an element that contains one or more object surrogates that link to the owned objects. The following example illustrates how this works.

```
<rt class="CmPossibility" guid="80a0dddb-8b4b-4454-b872-88adec6f2aba"
ownerguid="d7f71649-e8cf-11d3-9764-00c04f186933">
...
<Discussion>
```

```
<objsur t="o" guid="3aeef2e2-9466-41d5-afa7-d569f667fc79" />
</Discussion>
</rt>

<rt class="StText" guid="3aeef2e2-9466-41d5-afa7-d569f667fc79" ownerguid="80a0dddb-
8b4b-4454-b872-88adec6f2aba">
<Paragraphs>
<objsur t="o" guid="9555b62c-51f6-4b2a-ba51-944c5d69f4c1" />
</Paragraphs>
</rt>

<rt class="StTxtPara" guid="9555b62c-51f6-4b2a-ba51-944c5d69f4c1"
ownerguid="3aeef2e2-9466-41d5-afa7-d569f667fc79">
<ParseIsCurrent val="False" />
</rt>
```

The Discussion field in a CmPossibility owns an StText object which in turn owns one or more StTxtPara objects that store the actual text of the paragraph. The owning element stores one or more objsur elements that reference the guid of the owned object(s). In this example, note that the objsur guid in the Discussion element matches the StText guid field. Furthermore, the ownerguid of the StText field matches the guid field of the owning CmPossibility. Thus in an owning relationship, you can always trace up or down the ownership hierarchy using the guids. The same thing holds for the Paragraphs element that owns the StTxtPara object. The StTxtPara in this example does not have any Contents property, so it is a blank paragraph at this point.

In addition to a guid atttribute, the objsur element also has a ‘t’ attribute. This attribute is ‘o’ for an owning link and ‘r’ for a reference link. Reference properties are discussed in the next section.

Atomic owning elements have zero or one objsur element. Collection owning elements have any number of objsur elements and the order is irrelevant. Sequence owning elements have any number of objsur elements and the order is significant.

Not all objects have ownerguid attributes. Certain objects such as LexEntry and WfiWordform are unowned. Since all LexEntry objects are virtually owned by the LexDb, there is only one LexDb in a project, and the order is not stored explicitly in the data, we can assume that all entries are essentially owned by the LexDb without explicitly recording all of the ownerguids. All WfiWordforms were originally owned by a WordformInventory, but since the WordformInventory had no other purpose, it was removed from the model leaving the WfiWordforms unowned.

2.5 Reference properties

Reference properties are similar to owning properties. Reference properties are stored as an element that contains one or more object surrogates that link to the referenced objects. The following example illustrates how this works.

```
<rt class="LexSense" guid="e79c8d25-eb3d-43ec-94b7-ccb4fead1d40"
ownerguid="d6e17340-dc0f-458a-9fd6-c8de368918d5">
...
<SemanticDomains>
<objsur t="r" guid="ba06de9e-63e1-43e6-ae94-77bea498379a" />
```

```
<objsur t="r" guid="191ca5a5-0a67-426e-adfc-6fdf7c2aaa2c" />
</SemanticDomains>
</rt>

<rt guid="191ca5a5-0a67-426e-adfc-6fdf7c2aaa2c" class="CmSemanticDomain"
ownerguid="dc177f3c-d0fd-4232-adf1-a77b339cdbb2">
<Name>
<AUni ws="en">House</AUni>
</Name>
<Abbreviation>
<AUni ws="en">6.5.1.1</AUni>
</Abbreviation>
...
</rt>

<rt guid="ba06de9e-63e1-43e6-ae94-77bea498379a" class="CmSemanticDomain"
ownerguid="c924bfce-beed-4382-95e8-62b54951c83d">
<Name>
<AUni ws="en">Person</AUni>
</Name>
<Abbreviation>
<AUni ws="en">2</AUni>
</Abbreviation>
...
</rt>
```

In this example, a sense references two semantic domains. The SemanticDomains element of the sense contains two objsur objects. Each one has a ‘t’ attribute set to ‘r’ to indicate it is a reference link. It also contains a guid attribute that matches the corresponding guid attribute of a CmSemanticDomain object. Thus the first semantic domain link is to the Person CmSemanticDomain and the second is to the House CmSemanticDomain.

Unlike an owning property, there is no corresponding link from the CmSemanticDomain back to the LexSense that references it. A reference link is a one way link. If you want to find all of the senses that reference a particular CmSemanticDomain, you would need to search all SemanticDomain elements looking for ones that have an objsur with a matching guid.

Atomic reference elements have zero or one objsur element. Collection reference elements have any number of objsur elements and the order is irrelevant. Sequence reference elements have any number of objsur elements and the order is significant.

2.6 Top level XML

This illustrates the top level of a FieldWorks *.fwdata XML file:

```
<?xml version="1.0" encoding="utf-8"?>
<languageproject version="7000028">
...
</languageproject>
```

The top element is languageproject which has a version attribute that gives the version of the database. If this version is lower than the current program, the file will be migrated to the program version when the file is opened. If the file version is newer than the current program, the file cannot be opened without upgrading the program to the same version as the file. The contents of the languageproject element is an optional AdditionalFields element described in the next section, and a collection of 'rt' elements representing the objects in the database.

2.7 Custom fields

When custom fields are defined, an AdditionalFields element is added to the languageproject element. AdditionalFields holds CustomField elements, each one defining a custom field defined by the user that is unique to this database.

```
<AdditionalFields>
<CustomField name="Weather" class="RnGenericRec" destclass="7" type="RC"
wsSelector="-3" helpString="originally a standard part of Data Notebook records"
listRoot="2bdd1159-f9b2-11d3-977b-00c04f186933" />
<CustomField name=" My Complex String" class="LexEntry" type="String" wsSelector="-
1" />
<CustomField name=" My Simple Strings" class="LexEntry" type="MultiUnicode"
wsSelector="-3" />
<CustomField name=" My Paragraphs" class="LexEntry" destclass="14" type="OA" />
<CustomField name=" My List Items " class="LexEntry" destclass="7" type="RC"
listRoot="d7f713a0-e8cf-11d3-9764-00c04f186933" />
<CustomField name="My Date" class="LexEntry" type="GenDate" />
<CustomField name="My Number" class="LexEntry" type="Integer" />
</AdditionalFields>
```

Possible attributes of the CustomField element are

- name—Specifies the user-defined name of the custom field.
- class—Specifies the class name to which the custom field applies.
- destclass—(optional) Specifies the class number of the object owned or referenced by this field.
- type—Specifies the kind of field. Not all of these possibilities are currently supported in the UI. Possibilities are
  - Binary—The field holds binary data stored as a sequence of hexadecimal bytes.
  - Boolean—The field holds a Boolean value (e.g., true, false)
  - GenDate—The field holds a generic date stored as an integer. The format is described in Conceptual model overview.doc.
  - Guid—The field holds a unique identifier for an object.
  - Integer—The field holds an integer value
  - String—The field holds a complex string
  - MultiString—The field holds a number of complex strings, each for a different writing system.
  - Time—The field holds a time value (e.g., 2006-09-28 14:15:34.507)
  - Unicode—The field holds a simple string
  - MultiUnicode—The field holds a number of simple strings, each for a different writing system.

  - OA— The field owns a single object (atomic)
  - OC— The field owns an unordered collection of objects
  - OS— The field owns an ordered sequence of objects
  - RA— The field holds a reference to a single object (atomic)
  - RC— The field holds references to an unordered collection of objects
  - RS— The field holds references to an ordered sequence of objects
- wsSelector—(optional) This number determines the default writing system(s) for this field. Possibilities are
  - -1—The first analysis writing system.
  - -2—The first vernacular writing system.
  - -3—All analysis writing systems.
  - -4—All vernacular writing systems.
  - -5—All analysis then all vernacular writing systems.
  - -6—All vernacular then all analysis writing systems.
- listRoot—(optional for reference properties) If the dstclass is a CmPossibility (7) or subclass, this attribute holds the guid of the CmPossibilityList that owns the items that can be held in this reference property.
- helpString—(optional) This is a user-defined description of how the field is used.

The following examples show how data is stored in custom fields. The name attribute matches the name attribute of the custom field.

This is an example of a custom Single-line Text field set to First Analysis or First Vernacular writing system. The underlying type is a MultiString field, although the UI does not enable more than one writing system. The string itself can have embedded information.

```
<Custom name="My Complex String">
<AStr ws="en">
<Run ws="en">This is the field content.</Run>
</AStr>
</Custom>
```

This is an example of a custom Single-line Text field set to All ... writing systems. The underlying type is a MultiUnicode field, so it cannot have embedded information. The UI follows normal writing system rules for MultiUnicode fields.

```
<Custom name="My Simple Strings">
<AUni ws="en">house</AUni>
<AUni ws="fr">maison</AUni>
</Custom>
```

This is an example of a Multiparagraph Text custom field. This is always an owning atomic property that holds one StText. This example only shows the StText guid. The rest of the objects would be defined as normal <rt> elements.

```
<Custom name="My Paragraphs">
<objsur guid="c2ac5444-26cf-4318-9e08-1f12d64e173e" t="o" />
</Custom>
```

This is an example of a List Reference (multiple items) custom field. The underlying type is a reference collection field which holds references to objects. The (single item) type would be the same except the UI only allows it to hold one item.

```
<Custom name="My List Items">
<objsur guid="d7f713a5-e8cf-11d3-9764-00c04f186933" t="r" />
<objsur guid="d7f713a6-e8cf-11d3-9764-00c04f186933" t="r" />
</Custom>
```

This is an example of a date custom field. The underlying type is a GenDate field.

```
<Custom name="My Date" val="201011171" />
```

This is an example of a number custom field. The underlying type is an Integer field.

```
<Custom name="Number" val="45" />
```

Note: FieldWorks versions between 7.0 and 7.2.5 had a bug when importing new custom fields from a LIFT file created by WeSay. This resulted in a custom field definition with type="MultiBigString". For example:

```
<CustomField name="SILCAWL" class="LexSense" type="MultiBigString" wsSelector="-
5" />
```

The data for the field was stored as an AStr, with multiple analysis and vernacular writing systems. Currently Flex only supports custom fields as AUni with multiple writing systems or AStr with one writing system. So this buggy custom field is a combination between the two.

FieldWorks 8.0 will not accept these custom fields and refuses to open. Assuming the user only used a single writing system, the simplest solution is to edit the CustomField definition in the FwData file (using the ZEdit editor in the FieldWorks directory) to type="String" and change the wsSelector from -5 to -1 if the field is the first analysis writing system, or to -2 if it's the first vernacular writing system. If multiple writing systems are used in this field, it should be changed to type="MultiUnicode", but then you'll have to convert the data from AStr to AUni, and if the user added embeddings, these have to be removed. Contact LsDev in Dallas for help in this situation.

3 Source of FieldWorks XML files

The native format for storing project data uses the flat XML file described above. When you are not sharing your data with other users, this is the format that is saved in the project folder, and has an extension of fwdata. When you are sharing data with other users, the data is stored in a db4o database with an extension of fwdb. This file is described in more detail in FieldWorks 7 database model.docx, but it basically stores a record for each object. The content of the record is the 'rt' XML element from the fwdata file. A FieldWorks backup file (fwbackup extension) is a zip file that always stores a fwdata file, even if the current data is a fwdb file.

When creating a new FieldWorks project, C:\Program Files\SIL\FieldWorks\Templates\NewLangProj.xml is used as the beginning template. This file is also a fwdata file.

4 FDOBrowser

FieldWorks includes a FDOBrowser program that opens with a FDO Model tab to give class and property information on the current FieldWorks data model. There are no installed shortcuts for FDOBrowser.exe, so it needs to be run directly from c:\Program Files\SIL\FielodWorks 7.

You can open any FieldWorks fwdata or fwdb file to investigate the contents of the file. The LangProj tab lets you browse through the ownership hierarchy of objects in the database. The Repositories tab lets you look at all of the objects of a certain class. The program provides some initial editing capabilities that can be used to modify the data. The hope is that this program can be improved in time to provide more powerful capabilities to search and edit data. Of course, care should be taken any time you make changes at this low level since it bypasses some of the data integrity code that is built into the other FieldWorks programs. As a result, you could edit the file in a way that would make it impossible to load the file into Flex or TE. Always make a backup copy of your data first so that you will not lose important work if you do something wrong.

If a file is damaged too badly, it may not even open in the FDOBrowser program. In that case, you'll need to edit the fwdata file using ZEdit (installed with FieldWorks) or some XML editor that is capable of opening large files. There are several ways you can validate a damaged fwdata file to determine what is wrong.

One validation method is to use a DTD (XML Document Type Definition) file which describes the structure of the data. Every released version of FieldWorks comes with a fwdata.dtd file that can be used to validate the data. In a Cmd window open on your FieldWorks program directory, you can use the following command to validate data with a DTD file.

```
rxp -s -V -D languageproject fwdata.dtd "/work/ww/distfiles/Data/Turkish Romans.xml"

bin\rxp -s -V -D languageproject /work/ww/distfiles/fwdata.dtd
"/work/ww/distfiles/Data/Turkish Romans.xml"
```

Another validation method is to use a RNG (Relax NG Full Syntax Schema) file which describes the structure of the data. Every released version of FieldWorks comes with a fwdata.rng file that can be used to validate the data. In a Cmd window open on your FieldWorks program directory, you can use the following command to validate data with a RNG file.

```
RngValidate fwdata.rng "Data\Turkish Romans.xml"

DistFiles>RngValidate fwdata.rng "Data\Turkish Romans.xml"
```

FieldWorks also comes with FixFwData.exe in the FieldWorks program directory. This program checks for incorrect linkages and some writing system problems in a fwdata file and automatically cleans up your data if any problems are found. Sometimes if a fwdata file cannot be opened in FieldWorks, running this program will correct the problem enough to allow it to be opened. You can run FixFwData from a Cmd window by specifying the full path of the fwdata file. If you can open another project in Flex, you can also run this program on a different file by choosing Tools...Utilities, click the "Find and fix errors in a FieldWorks data (XML) file" checkbox and click "Run Checked Utilities Now". You can then select the project you want to fix.

If you have a fwdb file that cannot be opened due to some kind of corrupted data, FDOBrowser may be able to convert the data from fwdb to fwdata format. To do this, start FDOBrowser, then choose Tools...Extract db4o file contents to XML. Choose the db4o (fwdb) file, then specify the XML (fwdata) file, then click Extract. The resulting fwdata file can then be checked and fixed with the above tools.

5 Data protection, and recovering from a failure

As always, you should be making frequent backups of your data using File…Project Management…Back up this Project. These .fwbackup files should be copied safely to backup media. If all else fails in our attempt to protect your data, you should be able to restore your project from the last .fwbackup file with minimal loss. Working long periods of time without making backups is asking for trouble.

5.1 Non-shared

The project data is stored in an .fwdata file. Various attempts are made by Flex to ensure that this file does not become corrupted. When the project is open in a FieldWorks program, a .lock file is created. When that file exists, if another FieldWorks program from a separate process attempts to open the file, it will be blocked and will give the user an error message that the file is already in use.

As the user works, Flex frequently saves the data in a separate thread which doesn't block continuing work during this process:

1. The full data with changes is written to an entirely new .tmp file without modifying the .fwdata file.
2. When the write is complete, the original .fwdata file is renamed to .bak, replacing the file that was there.
3. The .tmp file is then renamed to .fwdata.

Under very rare circumstances, you might find that there is no .fwdata file. In this case, the project will not appear in the File…Open dialog. But if you look in the project file, you will likely see a .tmp or .bak file. In this case, the safest thing to do is to make a backup copy of these files in the same directory, so that if anything else goes wrong in recovery, you at least have the best chance of recovering. Then rename the .tmp (most recent) or .bak (second most recent) file to .fwdata. Now Flex should open without losing any data, or at most, everything except the last edits.

When you stop Flex on a large project, you'll probably see a small dialog with a progress bar showing progress on completing the save. Don't turn off the computer during this time.

If something happens to corrupt the .fwdata file so that it can't be opened, Flex will try to warn the user of the problem and ask if they want to try to use the .bak file instead. I think this works under some circumstances and not under others. In any case, if you can't open Flex, but you have a .fwdata and .bak file, the best thing is to make a copy of both files in the project directory, then delete the .fwdata file and rename the .bak file to .fwdata. If that solves the problem, great. If not, you will probably need help, and the project directory should be sent with your request for help.

5.2 Shared

When you are in sharing mode, your data is stored in an .fwdb Db4o database file. FieldWorks programs can use this data from different processes or across the network. As changes are made the changed data is written to this file and then other programs are updated with the changes.

If the .fwdb file ever becomes corrupted, the best chance of recovery would be to convert this to a .fwdata file and then use FixFwData or other processes to try to repair the file. To convert a .fwdb file to .fwdata open FDOBrowser, then choose Tools...Extract db4o file contents to XML. Choose the db4o (fwdb) file, then specify the XML (fwdata) file, then click Extract.

6 Modifying data in FieldWorks XML files

There are times when you may need to make some consistent changes to your data that are not possible in bulk edit. If there are only a few places, it is obviously best to do it manually. However, if hundreds or thousands of consistent changes are needed, and you have the ability to use CC, XSLT, Python, or some other program that can deal with XML data, it may well be worth the time to make the automated changes in the fwdata file.

Most examples here use CC tables. If you don't have SIL Consistent Changes program installed, it can be downloaded from www.sil.org/computing/catalog/index.asp.

Caution: As with any method for modifying the database outside of a FieldWorks program, if you do not know what you are doing, you can inadvertently damage the data. FieldWorks applications may no longer run or it could do damage in a way that will not show up until later. Be *extremely* cautious about making *any* changes to the fwdata file. Any time you plan to do this, make sure you *first* back up your project and then after the changes are made, check your changes carefully before going on to other work.

The sections below give several examples to illustrate some of the techniques for using the XML file to do something that is not possible to do within the current programs. These examples illustrate how CC can be used for some tasks, but it imposes severe limitations on operations involving more than one object since it is difficult to follow links. Also, CC is not capable of inserting multiple new objects since each one requires generating a GUID and CC does not provide this capability. XSLT is an XML transformation language that can be useful for working with these files, except many XML processors require loading the entire file into a DOM in memory. This is typically quite slow and can run out of memory. Perl and Python are probably better choices to use if you are familiar with these languages, or you need to do a lot of work in XML files.

6.1 Making spelling/orthographic changes in one writing system

Suppose you need to make some spelling or orthographic changes, e.g., to all the Tausug strings in the database, whether they are embedded or not. With bulk edit, you can do this on a field-by-field basis (although not when embedding is involved). There may be many other strings in the database (such as Scripture, data notebook, and lists) that also use this writing system that should also be changed. At this point there is no way to do this inside any FieldWorks program. This task is quite simple when working with a fwdata file.

Every string for a given writing system is in *one* of these two environments:

- <Run ... ws="tsg" ... >a string in Tausug</Run>

- <AUni ws="tsg">a string in Tausug</AUni>

There could be additional attributes in the Run element, but each Run or AUni that has the “tsg” writing system will contain Tausug text.

This CC table will make this change:

```
c  Change in writing system.cc
c  To set up, set the ws in the begin statement and put your changes in changes group

begin > store(ws) 'tsg' endstore use(main)

group(changes)
'f' > 'v'
'F' > 'V'
'c' > 'k'
'C' > 'K'

group(main)

'<Run ' > next
'<AUni ' > dup use(run)

group(run)
'ws="' cont(ws) '" />' > dup use(main)
'ws="' cont(ws) '"' > dup set(change)
'>' > dup if(change) use(changes,inRun) else use(main) endif

group(inRun)
'/>' > next
'</AUni>' > next
'</Run>' > dup clear(change) use(main)
```

Note: If the changes you make could affect interlinear baselines (or interlinearized scripture), after making the changes, you should use Tools…Utilities, “Force Rechecking Word Breaks”. Otherwise your baseline will be out of sync with wordforms which can cause problems in the concordance view and can mess up interlinear spelling changes.

Note: If you use LIFT to transfer data to and from Flex it’s possible you may have some LiftResidue elements that contain strings used in WeSay but not in Flex. These strings would either start with lang="xxx" or lang="xxx". The above table would need to be modified if you want to also convert these strings.

6.2 Converting bar codes to formatted text

When importing data from standard format (SFM) into Flex you can convert bar codes such as |bxyz|r to emphasized text or some other formatting. However, this conversion isn’t available when importing from LinguaLinks, and it’s possible you missed these conversions when importing data into LinguaLinks. If you have these kinds of in-line markers in FieldWorks and would like to convert these to something meaningful, you can use a CC table such as the

following one to make these conversions. This particular table converts any sequence of |b…|r to use the Emphasized Text style.

```
c This converts text between |b and |r to emphasized text.

group(main)
'<Run ' > store(run) dup use(startRun)

group(startRun)
'>' > out(run) dup use(inRun)

group(inRun)
'</Run>' > dup use(main)
'|b' > '</Run>' nl out(run) ' namedStyle="Emphasized Text">'
'|r' > '</Run>' nl out(run) '>'
```

This table would convert

```
<Run ws="es">This is a |btest|r with |bemphasized text</Run>
```

to

```
<Run ws="es">This is a </Run>
<Run ws="es" namedStyle="Emphasized Text">test</Run>
<Run ws="es"> with </Run>
<Run ws="es" namedStyle="Emphasized Text">emphasized text</Run>
```

You could use similar changes to convert the bar code to some other writing system if that were desirable.