FXT export options

Ken Zook

May 18, 2009

Contents

1 FXT introduction

In addition to various export options in Export options in Flex.doc, Flex provides a program that allows more sophisticated users or consultants to design custom options for exporting information from the FieldWorks database to XML, SFM, or plain text files.

You can run these custom exports outside of Flex or add them to the standard Flex export dialog.

These custom exports are defined by a FieldWorks XML template (FXT) XML file that instructs the FXT.exe (or FXT.dll) program what to extract from the database and how to format them in the output. FxtReference.doc is the initial documentation for this format. It is a good starting point to learn about FXT, although it is in a fairly early stage of development and does not cover all aspects of the program.

Note: In order to support the possibility of using the Firebird database engine in addition to Micrsoft SQL Server, and due to limited length of names in Firebird, some class, property, and procedure names were shortened in FieldWorks 5.4 compared to earlier versions. The spreadsheet, Model name changes.xls, lists the changes that were made. If you had custom FXT files for older versions you may need to make a few of these name changes for it to continue to work in FieldWorks 5.4 and later.

Note: It's best not to modify one of the installed template files. Instead, make a copy and work on the copy. The reason is that the installer will not uninstall a file that has been modified after it was installed. Then when a new installation is made, depending on dates, the new file may not replace the file you modified. This can lead to crashes related to mismatched data. Also, when you upgrade to a new version, you may need to adjust your custom FXT file so that it will be consistent with the new version.

2 FDO introduction

FXT is written on top of the FieldWorks Data Objects (FDO) layer, an object-oriented business layer that makes access to the database more intuitive. FDO is written in C# and

can be accessed from any program that can work with .NET 2.0 classes. The Translation Editor and Flex make heavy use of FDO internally. The freely available IronPython implementation of Python can also use this method to access the database (see Python database access.doc).

FDO provides generated classes for every class in the FieldWorks conceptual model, and generated methods on those classes for every property defined in the FieldWorks conceptual model. In addition, many hand-written methods on common classes provide additional functionality.

An introduction to FDO is given in fdoHelp.doc. The current programmer documentation for FDO is in FDO.chm, which was generated from the C# source code. FDO is discussed in more detail in Python database access.doc.

3 Basic FXT files

This section demonstrates minimal FXT files for exporting data in SFM, XML, and plain text formats.

Here is an example of a minimal FXT file you can use to dump the wordforms from the database to a standard format file:

```
<?xml version="1.0" encoding="UTF-8"?>
<template format="sf">
   <class name="LangProject">
      <group objProperty="WordformInventoryOA">
         <objVector objProperty="WordformsOC"/>
      </group>
   </class>
   <class name="WfiWordform">
      <multilingualStringElement name="w" simpleProperty="Form"/>
   </class>
</template>
```

An FXT file contains one template element with an optional format argument. When format is set to “sf”, the output will normally be in SFM as opposed to XML. Class elements are added within the template element. The first class element should be LangProject since that is the root owner of a FieldWorks project. A group element provides a way to change the current object while within the group. In this case it switches from using the language project to using the wordform inventory inside the WordformInventoryOA property. In FDO, owning and reference properties are suffixed to indicate the type of property:

- -OA ⇨ owning atomic
- -OC ⇨ owning collection
- -OS ⇨ owning sequence
- -RA ⇨ reference atomic
- -RC ⇨ reference collection
- -RS ⇨ reference sequence

WordformInventory is an atomic owning property that returns the WordformInventory. In that object, FW shows the object vector “Wordforms” which is an owning collection. This returns a collection of WfiWordforms.

The WfiWordform class element defines how FW should export wordforms. In this case, you export all writing systems of the Form property, using “\w” as the SFM marker.

Here is an example of the output from this template file when exported via FXT:

```
\wKal nihimbilira
\wKal tihindoksa
\wKal hiŋgabira
\wKal biliya
\wKalIPA biliya
```

FXT exports do not provide any sort or filter options. It exports all items in a given property, unless limited by a count argument. For sequences, the order will follow the sequence order. For collections, the order is random. Since the Wordforms property is a collection, the exported wordforms are not sorted. FXT output is UTF-8, using NFC normalization by default. An optional attribute for "template" allows NFD export, when desired (e.g., normalization="NFD"). When exporting strings, FXT returns the text portion of the string without any indication of embedded styles or writing systems.

Since the template had multilingualStringElement, FXT appended the writing system abbreviation to the \w marker. In the last wordform the database had two strings: one in Kalaba and one in Kalaba IPA. These were both exported, but with different markers.

By removing the format attribute in the template element, FXT defaults to an XML output. Here is the same table with this change:

```
<?xml version="1.0" encoding="UTF-8"?>
<template>
   <class name="LangProject">
      <group objProperty="WordformInventoryOA">
         <objVector objProperty="WordformsOC"/>
      </group>
   </class>
   <class name="WfiWordform">
      <multilingualStringElement name="w" simpleProperty="Form"/>
   </class>
</template>
```

This is the output from this FXT file:

```
<?xml version="1.0" encoding="UTF-8"?>
<w ws="Kal">nihimbilira</w>
<w ws="Kal">tihindoksa</w>
<w ws="Kal">hiŋgabira</w>
<w ws="Kal">biliya</w>
<w ws="KalIPA">biliya</w>
```

In this case, the element name is "w" and it includes a "ws" attribute identifying the writing system of the string.

An FXT file can also put out explicit text surrounding the information dumped from the database:

```
<?xml version="1.0" encoding="UTF-8"?>
<template format="plain">
   <class name="LangProject">
      <group objProperty="WordformInventoryOA">
         <objVector objProperty="WordformsOC"/>
      </group>
   </class>
   <class name="WfiWordform">
      <string simpleProperty="Form" ws="vernacular"/>
      <newLine/>
   </class>
</template>
```

In this case, instead of using multilingualStringElement, the template has the "string" element which takes a "simpleProperty" attribute giving the property name, and a "ws" attribute identifying the first vernacular writing system.

The "plain" format argument in the template prevents FXT from dumping the XML header line. The "newLine" element inserts a CR-LF following each wordform. FXT also provides "tab", "space", and "text" elements that can insert a tab, space, or the specified text at the desired location.

This is the output from this FXT file:

nihimbilira
tihindoksa
hiŋgabira
biliya

Since this example requested the vernacular writing system, it did not give the IPA form of "biliya".

4 FXT elements

This table summarizes valid elements in an FXT file.

Element	Required Attrs	Optional Attrs	Description
attribute	simpleProperty	name, value, optional, before, after	Adds a property as an XML attribute value. SimpleProperty specifies the property of the object. Name is the attribute name. If missing, uses the property name as the name.
attributeIndirect	target, simpleProperty	name, value, optional, before, after	Make an attribute that has a GUID, hvo, or other simple property of a referenced or owned atomic object.
booleanElement	name, simpleProperty	optional, writeAsTrait	Dumps a Boolean using the specified format.
call	name	flags	Subclasses can use this to call a superclass for generic processing. Name is the name of the superclass. Flags is a comma-separated list of flags that will cause an element to hide if a hideFlag element matches one of the flags.
comment			Anything enclosed in a comment element is ignored.
customMultilingualStringElement	name	field, custom, ws = {all, all analysis, all vernacular, every}	Allows export of custom multilingual properties. Name is used for the xml element or SFM marker. Field is the internal name of the custom field. Custom is the user name of the custom field. Must specify either field or custom. Ws is the writing system(s) to use. Defaults to "all". "every" includes all writing systems regardless of checks. Other options use checked writing systems.
customStringElement	name	field, custom	Allows export of custom properties. Name is used for the xml element or SFM marker. Field is the internal name of the custom field. Custom is the user name of the custom field. Must specify either field or custom.
dateAttribute	name, format,		Dumps a date using the specified format. Name

	property		is used for the xml element or SFM marker. Property specifies the date property.
element	name	hideFlag	Dumps nested elements using an outside xml or SFM marker specified by name. If hideFlag matches one of the flags fed in from a call element, it skips this property.
FxtDocumentDescription	dataLabel, formatLabel, defaultExtension, filter		Causes the FXT file to show up in the Flex File…Export Lexicon dialog, using the attributes and contents to fill in the dialogs.
generateCustom	class	fieldType	Dumps information from a custom field: class is LexEntry or LexSense. fieldType is mlstring or simplestring.
group	objProperty	preload	Repeats everything in the group for each object in objProperty. Preload gives an FDO method that preloads information into the cache to speed up the dump process. If the preload value is not null, it assumes the required data is already in the cache, so does not load again.
if		class, field, intequals	Includes the element contents if the integer property equals the value specified in intequals. If class is missing, assumes the field is on the current object. Field is a sequence of property names, separated by slash, leading to the integer property (e.g., EntryType/Type gets the EntryType object on the entry, and then gets the Type property from the LexEntryType).
ifnot		class, field, intequals	Includes the element contents if the integer property does not equal the value specified in intequals. If class is missing, assumes the field is on the current object. Field is a sequence of property names, separated by slash, leading to the integer property (e.g., EntryType/Type gets the EntryType object on the entry, and then gets the Type property from the LexEntryType).
multilingualStringElement	name, simpleProperty	ws = {all, all analysis, all vernacular}	Dumps the multilingual string(s). Uses name for the xml element or SFM marker. SimpleProperty specifies the property of the object. Ws defaults to all.
newLine			Inserts a CR-LF in the output file
numberElement	name, simpleProperty	ifnotequal, ifless, ifgreater	Dumps a number. Uses name for the xml element or SFM marker. SimpleProperty specifies the property of the object. Can use Ifnotequal, ifless, and ifgreater to control when the number gets dumped.
objAtomic	objProperty		Dumps the object inside an owning or reference atomic property specified by objProperty.
objVector	objProperty	itemLabel, virtualclass, count	Dumps the objects in the owning vector specified by objProperty. Virtualclass identifies the virtual class for objects in this property. Uses ItemLabel for the xml element or SFM marker (defaults to “object”). If you specify count, it only dumps that number of objects.
progress		progressIncre	Updates a progress bar

		ment	
refAtomic	simpleProperty		Use objAtomic instead of this.
refObjVector	field	ordered, itemLabel, virtual = {t, f, true, false, y, n, yes, no}, itemProperty, itemWsProp, classtag	Dumps the objects from the specified field. Is only supported for sf output. Uses ItemLabel for the xml element or SFM marker (defaults to "subobject"). If virtual is true, it looks for a virtual field with the specified field name. Displays the ItemProperty name for the referenced objects (defaults to "ShortName"). ItemWsProp can specify a property that holds the writing system to use in the label. If you give classtag, it is appended to the current class name with a hyphen. This combined class name is then used in the FXT file to display this class. If ordered = "true", includes the ord value as an attribute in the output.
refVector	field	ordered, itemLabel, virtual = {t, f, true, false, y, n, yes, no}, itemProperty, itemWsProp	Dumps the objects from the specified field. Uses ItemLabel for the xml element or SFM marker (defaults to "subobject"). If virtual is true, it looks for a virtual field with the specified field name. Displays the ItemProperty name for the referenced objects (defaults to "ShortName"). ItemWsProp can specify a property that holds the writing system to use in the label. If ordered = "true", includes the ord value as an attribute in the output.
space			Inserts a space in the output file.
string	simpleProperty	before, after, ws = {analysis, vernacular}	Dumps the simpleProperty as a string. If it is a multilingual property, requires ws to choose the correct alternative. Includes the before and after values before and after the property value.
stringElement	name	writeAsField, wrappingElementName, internalElementName, before, after	Dumps a string.
tab			Inserts a tab in the output file.
template		format = {xml, sf, plain}, type, datatype, xslt	The top level element in an FXT file. xslt causes the specified transform(s) to be run on the exported file.
text			Inserts the contents in the output file.
xmlstring	simpleProperty	ws = {analysis, vernacular}	Dumps the simpleProperty as a string. If it is a multilingual property, requires ws to choose the correct alternative
anything else			Inserts anything else in the file as a literal element in the output file.

5 Running FXT from the command line

You can run FXT files external to FieldWorks programs using the following syntax in a DOS box:

```
fxt databaseName fxtFile outputFile
```

FXT.exe is in the c:\Program Files\SIL\FieldWorks directory, so it works best to open a DOS box on this directory when using this program:

```
fxt "Sena 3" wordforms.xml wf.sfm
```

This example uses the wordforms.xml FXT template file on the Sena 3 database with the exported information going to wf.sfm.

6 Adding FXT export options to Flex

FXT files can also be run from within Flex. To do this, place the FXT file in c:\Program Files\SIL\FieldWorks\Language Explorer\Export Templates and use an .xml extension. (This is intended for lexicon exports, but for now you can put anything else in there to have it show up automatically in the UI.)

The FXT file also needs a FxtDocumentDescription element inside the template element. This element has the following arguments:

- **dataLabel:** The value shows up in the Data column of the Export dialog.
- **formatLabel:** The value shows up in the Format column of the Export dialog.
- **defaultExtension:** The value gives the default extension of the output file in the "Save as type" combo in the "Export to…" dialog.
- **filter:** The value gives the options for the "Save as type" combo in the "Export to…" dialog. Each part has the name that is displayed in the dialog and the file filter designation for that name.

The body of the element is text that appears in the "About the selected export method" window in the Export dialog.

Here is an example from the mdf.xml file:

```
<FxtDocumentDescription dataLabel="Lexicon" formatLabel="MDF
Standard Format" defaultExtension="db" filter="Standard Format files
(*.db)|*.db|All files (*.*)|*.*">
Export using the Multi-Dictionary Formatter standard.  This can be
imported into Lexique Pro for publishing dictionaries, either on the
Web or in print. (Note the exported file actually includes writing
system designators as part of the SFM code, so it is not pure MDF.)
</FxtDocumentDescription>
```

When a user chooses File…Export Lexicon from Flex, the program

- looks for all .xml files in c:\Program Files\SIL\FieldWorks\Language Explorer\Export Templates with FxtDocumentDescription elements, and
- lists those in the dialog as options for the user to select.

The FXT file is not limited to exporting dictionaries, but can export whatever you want.

If the XML file has invalid syntax, it will fail to show up in the Export dialog. A quick way to check the syntax is to open the XML file in Internet Explorer. If there are any errors, IE usually gives a helpful error message near the end explaining the problem.

Users may want to add additional XSLT processing to a file dumped from FXT. For example, they may want to dump the data as XML and then apply a transform to convert that to SFM, RTF, or some other format by adding the xslt attribute to the template element. The value is an ordered list of *.xsl file names. After dumping the file using FXT, Flex runs the transforms to produce the final output.

7 Examples of FXT exports

One of the best ways to learn to use FXT is to look at existing examples installed with FieldWorks. One source of these files is in c:\Program Files\SIL\FieldWorks\Language Explorer\Export Templates. These show up in the File…Export Lexicon dialog. Another source is c:\Program Files\SIL\FieldWorks\Language Explorer\Configuration\Grammar\FXTs. The grammar sketch uses these files and they are also part of the parser. The ZEdit Tools…Find In Files menu option provides a convenient way to search all files under these directories for examples of a particular element to investigate.

When developing an FXT dump, if you use the File…Export Lexicon dialog in Flex, you may encounter an error. You will get a rather useless "An error occurred while exporting your data. This is probably a bug in the FieldWorks code" error message. If you call FXT from the command line, you usually get a much more useful error message that will help you identify the problem. This error message also occurs if the output file is locked for some reason (such as if it is open in Excel, or if a large file is open in ZEdit).

7.1 Dumping parts of speech as a tab-delimited file

Here is an FXT file you can use from the File…Export Lexicon dialog. It dumps the abbreviation and names of the Category (Parts of Speech) list in a tab-delimited format that you can import into Excel:

```
<?xml version="1.0" encoding="UTF-8"?>
<template format="plain">
  <FxtDocumentDescription dataLabel="Categories (Parts of Speech)" formatLabel="Tab-delimited"
defaultExtension="txt" filter="Tab-delimited files (*.txt)|*.txt|All files (*.*)|*.*">Export the Category (Parts of
Speech) list to a tab-delimited file.</FxtDocumentDescription>

  <class name="LangProject">
    <group objProperty="PartsOfSpeechOA">
      <call name="CmPossibilityList"/>
      <newLine/>
    </group>
  </class>

  <class name="CmPossibilityList">
    <string simpleProperty="Name" ws="analysis"/><newLine/>
    <text>Abbr.</text><tab/><text>Name</text><newLine/>
    <objVector objProperty="PossibilitiesOS"/>
  </class>

  <class name="CmPossibility">
    <string simpleProperty="Abbreviation" ws="analysis"/>
    <tab/>
    <string simpleProperty="Name" ws="analysis"/>
    <newLine/>
    <objVector objProperty="SubPossibilitiesOS"/>
  </class>

  <class name="PartOfSpeech">
    <call name="CmPossibility"/>
  </class>

</template>
```

This obtains the parts of speech list from the PartsOfSpeech atomic owning property on LangProject, and then calls the CmPossibilityList class to process the list. This class puts

out some header information, then dumps the Possibilities owning sequence. The objects at this level are PartOfSpeech classes, but since the properties to display are in CmPossibility, and may be common to other classes, call this superclass to do the work. The CmPossibility class formats the abbreviation and name, and then dumps all subpossibilities.

Here is a small example of the output of this FXT dump

Parts Of Speech

Abbr.	Name
adjunct	adjunct
art	article
def	definite article
indef	indefinite article
disc	disc
interj	interjection

7.2 Dumping XML information from the wordform inventory

Here is an FXT file you can use from the File…Export Lexicon dialog to dump some of the information from the wordform inventory in an XML format:

```
<?xml version="1.0" encoding="UTF-8"?>
<template>
  <FxtDocumentDescription dataLabel="Wordforms" formatLabel="XML" defaultExtension="xml"
filter="XML files (*.xml)|*.xml|All files (*.*)|*.*">
  Export the wordforms and associated information to an XML file.
  </FxtDocumentDescription>

  <class name="LangProject">
    <group objProperty="WordformInventoryOA">
      <objVector objProperty="WordformsOC"/>
    </group>
  </class>

  <class name="WfiWordform">
    <element name="wf">
      <attribute name="UserCount" simpleProperty="UserCount"/>
      <attribute name="ParserCount" simpleProperty="ParserCount"/>
      <newLine/>
      <element name="form">
        <string simpleProperty="Form" ws="vernacular"/>
      </element>
      <newLine/>
      <objVector objProperty="AnalysesOC"/>
    </element>
    <newLine/>
  </class>

  <class name="WfiAnalysis">
    <element name="wa">
      <objVector objProperty="MeaningsOC"/>
      <objAtomic objProperty="CategoryRA"/>
    </element>
    <newLine/>
  </class>

  <class name="WfiGloss">
    <element name = "gl">
      <string simpleProperty="Form" ws="analysis"/>
    </element>
  </class>
```

```
  <class name="PartOfSpeech">
    <element name="pos">
      <string simpleProperty="Name" ws="analysis"/>
    </element>
  </class>

</template>
```

Without the newline elements, the output is scrunched into one big paragraph. This is hard to read in normal text editors. The UserCount and ParserCount properties are virtual properties. They work fine in this context, although they make the export much slower. It exports these properties as attributes on the wf element. The remaining objects are dumped as contents of appropriate elements.

Here is a small example of the output of this FXT dump:

```
<?xml version="1.0" encoding="UTF-8"?>
<wf UserCount='1' ParserCount='1'>
<form>nihimbilira</form>
<wa><gl>I see</gl><gl>I perceive</gl><gl>I understand</gl><pos>verb</pos></wa>
<wa></wa>
</wf>
<wf UserCount='0' ParserCount='1'>
<form>tihindoksa</form>
<wa></wa>
</wf>
<wf UserCount='1' ParserCount='4'>
<form>pus</form>
<wa></wa>
<wa><gl>green</gl><pos>modifier</pos></wa>
<wa></wa>
<wa></wa>
</wf>
```

7.3 Adding a custom field to mdf.xml or RootBasedMDF.xml

You may add a custom Plural string field using the first vernacular writing system to LexEntry and want to include that in your MDF export. You can do this easily by adding the following line inside the LexEntry class element:

```
<customStringElement name="pl" custom="plural"/>
```

This exports the plural forms with a \pl SFM code.

7.4 Adding guids to mdf.xml

In some cases you may want to export the globally unique identifier (GUID) FieldWorks uses to identify each object (e.g., entry and sense). You can do this in a copy of mdf.xml by adding a new element to extract a Guid field from each object of interest. You can do this easily by adding the following line inside the LexEntry and LexSense class elements:

```
<class name="LexEntry">
   <element name="lx" progressIncrement="true">
      <string  simpleProperty="LexemeFormWithAffixType"/>
   </element>
   <element name="gde">
      <string  simpleProperty="Guid"/>
   </element>

<class name="LexSense">
   <element name="sn">
```

```
    <string  simpleProperty="SenseNumber"/>
</element>
<element name="gds">
    <string  simpleProperty="Guid"/>
</element>
```

The gde element will export a \gde field containing the Guid for each entry. The gds element will export a \gds field containing the Guid for each sense.