FieldWorks database model

Ken Zook

December 10, 2008

Contents

1 Introduction

Metadata, stored procedures, and data for each FieldWorks project are stored in a SQL Server database. There are several methods for accessing data directly from the database.

- **SQL queries.** These can be executed directly through .NET SqlConnection, ODBC, OLEDB, or the FieldWorks OleDbEncap COM interface. They can also be executed from a file via the db program.
- **ISilDataAccess, IVwOleDbDa,** and **IVwCacheDa** COM interfaces that allow access in a more object-oriented way than pure SQL queries. See http://fieldworks.sil.org/objectweb for details on these interfaces.
- **FieldWorks Data Objects (FDO).** This is a .NET assembly of classes you can use to access and manipulate data in the FieldWorks database. It serves as an object-oriented layer between the relational database and the user application. FDO is discussed in Python database access.doc.

Microsoft SQL Server is a relational database made up of many tables, where each table has a set of columns or fields. Fields may contain actual data, or links to data in other tables. Key fields for each table are indexed for rapid access to the data.

SQL Server is a robust system that provides

- protection if power fails
- remote access
- triggers and constraints to limit invalid data
- backup and restore capabilities, and
- database maintenance programs.

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 queries 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.

2 FieldWorks database model

This section discusses the way the FieldWorks model is mapped to SQL Server relational database tables.

2.1 Metadata

Metadata tables are used to define FieldWorks object structure and hold other information that is not direct user data.

2.1.1 Module$

FieldWorks objects are defined in modules. The modules are defined in the Module$ table:

```
select * from Module$
```

Id	Name	Ver	VerBack
0	Cellar	1	1
3	Scripture	1	1
4	Notebk	1	1
5	Ling	1	1
6	LangProj	1	1

The Id and Name are the only relevant parts of this table. The Module Ids are incorporated into the Class Ids.

2.1.2 Class$

Classes are defined in the Class$ table. There are currently 173 classes defined in this table. Only a few classes are illustrated here:

```
select * from Class$
```

Id	Mod	Base	Abstract	Name
0	0	0	True	CmObject
5	0	0	True	CmMajorObject
7	0	0	False	CmPossibility
8	0	5	False	CmPossibilityList
5005	5	5	False	LexDb
5035	5	0	True	MoForm
5045	5	5035	False	MoStemAllomorph
5049	5	7	False	PartOfSpeech

- The Id column is the Class Id that is incorporated into property names and is often abbreviated “clsid” or “clid”.
- The module that defines this class is given in the Mod column.
- The Base column identifies the superclass for this class. For example, CmPossibilityList (class = 8) is a subclass of CmMajorObject (class = 5), which is a subclass of CmObject (class = 0).
- When Abstract is True, it means there are no instances of this class—only subclasses can exist.
- CmObject is an abstract class with an Id of 0 and is the only class that does not inherit from some other class.

2.1.3 ClassPar$

The ClassPar$ table provides a quick way to find all the superclasses for a given class:

```
select * from ClassPar$
```

Src	Dst	Depth
0	0	0
8	8	0
8	5	1
8	0	2
5045	5045	0

5045	5035	1
5045	0	2

Src and Dst are Class Ids. For example, this shows that CmPossibilityList (class = 8) is 2 inheritance levels from CmObject (class = 0) with CmMajorObject (class = 5) in between.

2.1.4 Field$

The Field$ table provides information for each FieldWorks property (field). There are currently 816 fields defined in this table.

```
select * from Field$
```

Id	Type	Class	DstCls	Name	Custom
5001	16	5		Name	0
8004	1	8		IsSorted	0
8008	27	8	7	Possibilities	0
8010	16	8		Abbreviation	0
8014	2	8		ItemClsid	0

- Built-in fields (Custom = 0) currently only use a subset of the columns in Field$.
- The Field Id is derived from the Class Id and is often abbreviated "flid".
- The Type field is described in Conceptual model overview.doc.

This is a summary of the types:

1	Boolean
2	Integer
3	Numeric
4	Float
5	Time
6	Guid
7	Image
8	GenDate
9	Binary
13	String
14	MultiString
15	Unicode
16	MultiUnicode
17	BigString
18	MultiBigString
19	BigUnicode
20	MultiBigUnicode
23	OwningAtom
24	ReferenceAtom
25	OwningCollection
26	ReferenceCollection
27	OwningSequence
28	ReferenceSequence

- In this sample, the name for the CmPossibilityList comes from the MultiUnicode (type = 16) Name field of the CmMajorObject superclass (class = 5001). The Possibilities field is an owning sequence (type = 27) that holds instances of CmPossibility (class = 7) or its subclasses.
- The Class column identifies the class to which this property belongs.
- If the property is an owning or reference property, DstCls lists the type of class (or subclass) that can be held in this property.

For custom fields, the remaining columns of Field$ are also used.

- CustomId: This is a GUID that is currently not used.
- Min: For integers, this can specify the minimum value.
- Max: For integers, this can specify the maximum value.
- Big: This may be unused, since the type already identifies big versions.
- UserLabel: This is the name users define, and what they see in the UI.
- HelpString: This is the description of the field.
- ListRootId: For reference properties, this holds the CmObject Id of the possibility list that provides choices for this reference property.
- WsSelector: This determines the default writing systems to display:

-1	first analysis writing system
-2	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

- XmlUI: This may be unused. The idea was to have a place to store XML code that would determine how this field was displayed, much like the code usually in *Parts.xml files under Language Explorer\Configuration.

This is one example of a Custom Field. The least significant 3 digits of the Field Id will always be 500 or more for custom fields, and Custom is always set to 1:

Id	5002500
Type	13
Class	5002
DstCls	
Name	custom
Custom	1
CustomId	d9923cb3-fcb0-4628-aa52-13cbcc7186e8
Min	
Max	
Big	False
UserLabel	Plural
HelpString	Plural form for this entry.
ListRootId	
WsSelector	-2
XmlUI	

2.1.5 Version$

This table has several columns relating to the database version. At this point, the only value used is DbVer that gives the database version for this database. FieldWorks programs check this number to verify that the database is the desired version.

2.1.6 Other tables

- Sync$: This table is used to synchronize changes to a database when more than one program is using the database.
- ObjInfoTbl$: This is a temporary table used during some stored procedures.
- ObjListTbl$: This is a temporary table used during some stored procedures.

2.2 Classes

2.2.1 CmObject

The database has a separate table for every class. CmObject is the base table that has a row for every FieldWorks object used in the database. All object ownership details are stored in CmObject. It has the following columns:

- Id: This is an integer holding the Object Id that is the main reference used within the database. Every object has a unique Id. In code, this is often referred to as HVO (handle to a viewable object). It is only used within a single database and is not saved to FieldWorks XML files.
- Guid$: This is a globally unique identifier for every object in the database.
- Class$: This is an integer holding the Class Id for this object, which matches an Id in Class$.
- Owner$: This is the CmObject Id that owns this object.
- OwnFlid$: This is an integer holding the Field Id on Owner$ that owns this object. This matches an Id in Field$.
- OwnOrd$: For sequence properties, this is an integer giving the order within the property. It is not required that these numbers be consecutive.
- UpdStmp: This is a “timestamp” data type that is guaranteed to be unique. It is changed automatically any time a property in the class is changed and is part of the strategy for identifying when some other program has changed this object while your program has it in its memory cache.
- UpdDttm: This is a “smalldatetime” that is updated automatically when changes are made in a class.

When an object is dumped to Fieldworks XML, the Id, UpdStmp, and UpdDttm columns are not dumped. Thus, when a file is loaded into the database from XML, there is no guarantee that the Id will be the same as the dumped file. In the database, the Field Id is used to reference related objects. In the XML file, the GUID is used for references. Class$, Owner$, OwnFlid$, and OwnOrd$ do not get dumped directly to the XML file, but they are inherent in the structure that gets dumped, so this information is not lost.

Here are a few examples with the GUID truncated to save space:

```
select * from CmObject

Id     Guid$        Class$  Owner$  OwnFlid$  OwnOrd$  UpdStmp  UpdDttm
1      2465f3c4…    6001                               0x2ECB3  10/21/2006 3:32:00 PM
4898   c924bfce…    8       1       6001049            0x2DC01  4/26/2006 2:23:00 PM
4899   63403699…    66      4898    8008      1        0x2E10A  4/26/2006 2:24:00 PM
5864   ba06de9e…    66      4898    8008      2        0x2E174  4/26/2006 2:24:00 PM
7329   f4491f9b…    66      4898    8008      3        0x2E1FD  4/26/2006 2:24:00 PM
4900   999581c4…    66      4899    7004      1        0x2E10B  4/26/2006 2:24:00 PM
5096   b47d2604…    66      4899    7004      2        0x2E11B  4/26/2006 2:24:00 PM
```

- In this example, object 1 is the root LangProject (class = 6001).
- Owner$, OwnFlid$, and OwnOrd$ are all null since the language project does not have an owner.

- Object 4898 is a CmPossibilityList (class = 8) that is owned by object 1 (LangProject) in the SemanticDomainList property (field = 6001049). It is an atomic property, so it does not have an OwnOrd$.
- Objects 4899, 5864, and 7329 are all CmSemanticDomain objects (class = 66) owned directly by the semantic domain list (id = 4898) in the Possibilities property (field = 8008) of CmPossibilityList.
- OwnOrd$ indicates these are the first three items in the list.
- Objects 4900 and 5096 are also CmSemanticDomain objects (class = 66) owned by the first CmSemanticDomain (id = 4899) in the semantic domain list. They are owned in the SubPossibilities property (field = 7004) of the CmSemanticDomain.

2.2.2 Subclasses

Every class has its own table in the database with columns depicting fields that are defined on that class. To get full information for a particular object, you need to look at two or more tables since every class is a subclass of CmObject, and maybe other classes as well. For example, consider CmSemanticDomain:

```
select * from CmSemanticDomain where id = 4899
```

```
id    LouwNidaCodes          OcmCodes
4899  1A Universe, Creation; 14 Physical Events and States 772 Cosmology;…
```

This query shows the content of properties defined for object 4899 directly on the CmSemanticDomain class.

```
select id, DateCreated, ForeColor, Hidden, IsProtected from CmPossibility where id = 4899
```

```
id    DateCreated           ForeColor   Hidden IsProtected
4899  4/26/2006 2:23:30 PM  -1073741824 False  False
```

This query returns information for the same object 4899 that is defined on the CmPossibility class, which is the superclass of CmSemanticDomain. The select has been limited here to certain fields, since using “*” to get all fields would return Id, SortSpec, Confidence, Status, DateCreated, DateModified, HelpId, ForeColor, BackColor, UnderColor, UnderStyle, Hidden, and IsProtected. This is too much to display on one line in this document.

```
select * from CmObject where id = 4899
```

```
Id    Guid$       Class$ Owner$  OwnFlid$ OwnOrd$ UpdStmp  UpdDttm
4899  63403699…   66     4898    8008     1       0x2E10A  4/26/2006 2:24:00 PM
```

This query returns the information for the same object 4899 that is defined on the CmObject class, which is the superclass of CmPossibility and also a superclass of CmSemanticDomain.

To get all of the information for this object in one query, FieldWorks provides a view for every class that is defined in the database. The name of the view is the name of the class with underscore appended to the end. For CmSemanticDomain, the generated view is defined as:

```
select [CmPossibility_].*, [CmSemanticDomain].[LouwNidaCodes],
[CmSemanticDomain].[OcmCodes] from [CmPossibility_]
join [CmSemanticDomain] on [CmPossibility_].[Id] = [CmSemanticDomain].[Id]
```

In this query CmPossibility_ joins to CmObject, so using this view, you can get all information with this query:

```
select * from CmSemanticDomain_ where id = 4899
```

The result of the query returns columns Id, Guid$, Class$, Owner$, OwnFlid$, OwnOrd$, UpdStmp, UpdDttm, SortSpec, Confidence, Status, DateCreated, DateModified, HelpId, ForeColor, BackColor, UnderColor, UnderStyle, Hidden, IsProtected, LouwNidaCodes, and OcmCodes. These are too long to display in this document, but just as in regular tables, you can select specific fields from this view as well:

```
select id, owner$, DateCreated, LouwNidaCodes from CmSemanticDomain_ where id = 4899

id     owner$ DateCreated            LouwNidaCodes
4899   4898   4/26/2006 2:23:30 PM   1A Universe, Creation; 14 Physical Events and States
```

These queries on the actual tables only return some of the properties for a class. Properties that return more than one row (such as multistrings, reference collections, and sequences) require separate queries to return this information. This is discussed in the next section.

2.3 Properties

Properties for classes are either stored in columns directly on the class, or stored in separate tables that must be joined during retrieval.

2.3.1 Basic properties

2.3.1.1 Strings

SQL queries use “order by” clauses to sort returned values on one or more fields. When ordering by FieldWorks string properties, you can use the “order by” clause for Unicode, String, MultiUnicode, and MultiString values. SQL Server does not support the “order by” clause when using the ntext datatype that is used for BigUnicode, BigString, MultiBigUnicode, and MultiBigString. This means you cannot sort on these types of properties. Also, be aware that SQL sorting is independent of ICU, so it does not follow the sort order specified for your writing systems. Here are two examples:

```
select * from LexSense order by Source
select * from StTxtPara order by Contents – fails
```

The first query works fine since Source is a String property. The second query will fail with an error message because Contents is a BigString property.

SQL queries use “where” clauses to limit the number of returned rows. “Where” clauses are also limited when using “ntext” values:

```
select * from LexSense where source is not null
select * from LexSense where source = N'Source information'
select * from LexSense where source like N'%information%'
select * from LexSense where source > N'm'

select * from StTxtPara where contents is not null
select * from StTxtPara where contents = N'is not obligatory' -- fails
select * from StTxtPara where contents like N'%not%'
select * from StTxtPara where contents > N'b' -- fails
```

The first four queries above are all valid, because Source is a String property and can be equated or compared with other strings. Two of the last four queries fail with error messages because Contents is a BigString property. “ntext” strings cannot be equated or compared with other strings. They *can* be checked for being missing and can use the “like” clause. The “%” characters in string comparisons and “like” clauses are wildcards that can mean zero or more characters. Thus, the second-to-last query finds all strings containing “not” anywhere in the string.

When specifying strings, the N prefix indicates that the string is a Unicode string. Since all FieldWorks data is Unicode, it is best to use the N prefix for any string. Otherwise, SQL Server will convert the string to Unicode using the current code page, which may or may not be desirable.

When entering strings, you can use the “nchar” command (with decimal values) to embed single Unicode characters. You can also prepare the text in ZEdit in a UTF-8 window, Word, or some other Unicode editor and paste it in. These two strings are identical:

```
N'hi' + nchar(331) + N'xogabibi'
N'hiŋxogabibi'
```

When you get the results of a query, you can paste them into ZEdit in a UTF-8 window to explore the code points. You can also use SQL Unicode and substring commands. The following query returns 331, which is the decimal value of the “eng” character (U+014b):

```
declare @str nvarchar(400)
set @str = N'hiŋxogabibi'
select unicode(substring(@str,3,1))
```

2.3.1.1.1 Unicode/BigUnicode

Unicode and BigUnicode strings are stored as a column in the class table that defines the property and are accessed in the same way. Unicode is stored as “nvarchar(4000)” and BigUnicode strings are stored as “ntext”:

```
select id, HelpId from CmPossibility order by HelpId

id  HelpId
19  MaterialNotRelevant
22  Orientation
18  PVProjectVariables
```

This query returns the HelpId Unicode property on CmPossibility. These properties do *not* have any inherent writing system or formatting. They are just a sequence of Unicode characters.

The following query demonstrates setting a Unicode or BigUnicode string. Be sure to include a where statement or all possibilities will be changed at once.

```
update CmPossibility set HelpId = N'NewName' where id = 198
```

2.3.1.1.2 String/BigString

String and BigString strings have Unicode characters, but also have formatting associated with the string. Every String or BigString column in a class table also has a corresponding column with “_Fmt” appended to the property name. This format column contains compressed binary information that represents things such as writing system, styles, direct formatting, and embedded objects. SQL queries do not provide the power to decompress this binary information into something useful. FwKernel.dll contains TsString COM interfaces that are used to interpret this information. The format information contains offsets into the string, so changing the text without updating the formatting causes corrupted data. As a result, SQL queries should *not* attempt to change String or BigString fields. SQL queries will *not* be able to determine any embedded information on these strings.

Strings are stored as “nvarchar(4000)” and the associated format as “varbinary(8000)”. BigStrings are stored as “ntext” and the associated format as “image”:

```
select Id, Contents, Contents_Fmt from StTxtPara

Id    Contents                  Contents_Fmt
2933  Derivational operations   0x0100000000000000000000000010006E57E0000
3026  Example (English)         0x0100000000000000000000000010006E57E0000
```

This query returns the contents and formatting for StTxtPara. The writing system is embedded in “Contents_Fmt”.

Strings and BigStrings should not be modified using SQL code, because it cannot handle the compressed binary format. Also, the Contents_Fmt should never be left null because every String in FieldWorks should have a writing system, which is stored in the Contents_Fmt. There are some exceptions. If your string has no embedding, and there is already a Fmt field for a similar string with no embedding in the same writing system, the other Fmt could be used to set the new string. In the example above, the Fmt is the same for both strings because it just contains the overall writing system. There is no other embedding. If a string has any other embedding, you should not attempt to update it with SQL code.

2.3.1.1.3 MultiUnicode

MultiUnicode properties contain a collection of Unicode strings that are translations of the same string in different languages and/or writing systems. Each writing system can only occur *once* in a MultiUnicode property. You can have English and Spanish strings, but not two English strings. In order to increase efficiency, every MultiUnicode property is implemented as a separate table in the database with the class name and property name separated by an underscore. The table has 3 columns:

- Obj: This is the Object Id from CmObject for the object that owns this string.
- Ws: This is the Object Id for the LgWritingSystem.
- Txt: This is the Unicode string contents. It is stored as “nvarchar(4000)” (except WfiWordform_Form and MoForm_Form are limited to 300 because they are indexed):

```
select * from CmPossibility_Name

Obj  Ws     Txt
3    32490  Adverbio
3    32485  Adverb
3    32488  Adverbe
4    32490  Nombre
4    32485  Noun
4    32488  Nom
```

This query returns the names of CmPossibilty items including subclasses of CmPossibilty. In this case, the first CmPossibilty (Id = 3) has three translations for the name in Spanish (Ws = 32490), English (Ws = 32485), and French (Ws = 32488). To determine what writing system values mean, you can query the MultiUnicode name for LgWritingSystem:

```
select * from LgWritingSystem_Name
```

```
Obj     Ws      Txt
32485   32485   English
32488   32485   French
32490   32485   Spanish
32495   32485   Portuguese
32567   32485   Lela-Teli
```

Alternatively, you could join the two tables in a single query, changing the labels to make them more meaningful:

```
select cpn.obj Id, lwn.txt Language, cpn.txt Name from CmPossibility_Name cpn
join LgWritingSystem_Name lwn on lwn.obj = cpn.ws
```

```
Id  Language  Name
3   Spanish   Adverbio
3   English   Adverb
3   French    Adverbe
4   Spanish   Nombre
4   English   Noun
4   French    Nom
```

To modify a MultiUnicode string, use the SetMultiTxt$ stored procedure. The inputs should be the Field Id, Object Id of the owner, writing system, and the string:

```
exec SetMultiTxt$ 7001, 3, 40719, N'accessoires'
```

2.3.1.1.4 MultiString

All MultiString properties are stored in a single MultiStr$ table that has the following columns:

- Flid: This is the Field$ Id of the field that holds the string on Obj.
- Obj: This is the Object Id from CmObject for the object that owns this string.
- Ws: This is the Object Id for the LgWritingSystem.
- Txt: This is the String contents. It is stored as “nvarchar(4000)”.
- Fmt: This is the formatting for the string. It is stored as “varbinary(8000)”.

You can access this table directly. An easier way is to use built-in views for all MultiStrings that makes it look similar to a MultiUnicode table. This is the view definition for LexSense_Definition:

```
select [Obj], [Flid], [Ws], [Txt], [Fmt]
FROM [MultiStr$]
WHERE [Flid] = 5016005
```

Using this view, you do not have to figure out what Field Id to use since it is built into the view. Here is a query to list the definitions in senses using this view:

```
select * from LexSense_Definition
```

```
Obj   Flid     Ws     Txt           Fmt
6244  5016005  98507  的（表领属）  0x0100000000000000000000000010006CB800100
```

```
6244  5016005  98509  follows a pronoun…   0x010000000000000000000000010006CD800100
6306  5016005  98507  本                    0x010000000000000000000000010006CB800100
6306  5016005  98509  measure word for…    0x010000000000000000000000010006CD800100
```

This shows Chinese (Ws = 98507) and English (Ws = 98509) definitions for two different senses (6244 and 6306). The format strings are different because the writing systems are different.

Setting MultiStrings has the same cautions as discussed with String because SQL cannot create the compressed Fmt field. If a string has no embedding and you already have an Fmt field to use, update a MultiString with the SetMultiStr$ stored procedure. This takes a Field Id, Object Id of the owner, writing system, the string, and the string format:

```
exec SetMultiStr$ 5016005, 6049, 40733, N'hello world',
    0x0100000000000000000000000100061D9F0000
```

2.3.1.1.5 MultiBigString

MultiBigString works similarly to MultiString. The difference is that they use the MultiBigStr$ table which has the same columns as MultiStr$. The only difference in the tables is that the Txt column uses "ntext" and the Fmt column uses "image". Built-in views for MultiBigString properties work the same as MultiString views.

Setting MultiBigStrings has the same cautions as discussed with String because SQL cannot create the compressed Fmt field. If a string has no embedding and you already have a Fmt field to use, update a MultiBigString with the SetMultiBigStr$ stored procedure. This takes a Field Id, Object Id of the owner, writing system, the string, and the string format.

2.3.1.1.6 MultiBigUnicode

FieldWorks currently does not use MultiBigUnicode, but it is similar to MultiBigString. It uses MultiBigTxt$ to store strings for all properties. There is no Fmt column since this stores Unicode strings.

MultiBigUnicode strings can be altered using the SetMultiBigTxt$ stored procedure. This takes a Field Id, Object If of the owner, writing system, and the string.

2.3.1.2 Other basic properties

Outside of strings, all other basic properties are stored in columns in the class on which they were defined. You can use an "order by" clause in the select statement for any of these properties.

Booleans are stored as bit data types in the database:

```
select Id, ExcludeAsHeadword from LexEntry
```

Id	ExcludeAsHeadword
6324	False
6328	True

The following query is an example that updates a Boolean. (True = 1 and False = 0.) Include a "where" statement to avoid changing *all* entries at once:

```
update LexEntry set ExcludeAsHeadword = 1 where id = 6047
```

Integers are stored as “tinyint” (1 byte), “smallint” (2 bytes), or “int” (4 bytes), depending on the minimum and maximum settings. The default is 4 bytes:

```
select Id, HomographNumber from LexEntry

Id      HomographNumber
6268    1
6272    0
6289    2
```

The following query is an example that updates an integer. Include a “where” statement to avoid changing *all* entries at once:

```
update LexEntry set HomographNumber = 1 where id = 6047
```

Time is stored as a “datetime” in the database:

```
select Id, DateCreated from LexEntry

Id      DateCreated
6224    8/7/2003 8:42:42 AM
6228    8/13/2003 10:37:25 AM
```

The first query below is an example that updates a time to a specific time. The second query sets the time to the current time. Include a “where” statement to avoid changing *all* entries at once:

```
update LexEntry set DateCreated = '12/23/2004 9:20 PM' where id = 6047
update LexEntry set DateCreated = getdate() where id = 6047
```

GUIDs are stored as “uniqueidentifiers” in the database:

```
select Id, Guid$ from CmObject

Id  Guid$
1   2465f3c4-30ec-4b5b-bf0f-9aa0ba23634a
2   d7f7150c-e8cf-11d3-9764-00c04f186933
```

The following queries are examples that update a GUID. The first one sets it to a specific value. The second query creates a new GUID and sets the property to the new GUID. Include a “where” statement to avoid changing *all* objects at once:

```
update CmObject set Guid$ = '2465f3c4-30ec-4b5b-bf0f-9aa0ba23634b' where id = 1
update CmObject set Guid$ = newid() where id = 1
```

GenDates are stored as integers in the database:

```
select Id, DateOfEvent from RnEvent

Id      DateOfEvent
6842    193112111
6857    196000000
6860    0
```

The following query is an example that updates a GenDate. Include a “where” statement to avoid changing *all* events at once:

```
update RnEvent set DateOfEvent = 193112111 where id = 6860
```

Binary data is stored as “varbinary(8000)” in the database:

```
select Id, StyleRules from StPara
```

```
Id      StyleRules
6850    0x00018502064E006F0072006D0061006C00
7017    0x02016202E204008008018502064E006F0072006D0061006C00
```

Caution: StyleRules are compressed binary information that SQL cannot decipher, so it is highly unlikely that you would ever want to set StyleRules from SQL. The following query is an example that clears a binary field. Include a "where" statement to avoid changing *all* styles at once:

```
update StPara set StyleRules = null where id = 6850
```

2.3.2 Owning relationships

As discussed in the CmObject section, owning relationships are stored in Owner$, OwnFlid$, and OwnOrd$ fields in CmObject. These fields are left null for the few unowned objects in the database. For owning sequences, OwnOrd$ gives the order of the objects in the sequence. For owning collections, OwnOrd$ is null.

You can use CmObject directly to list ownership information. This is particularly useful if you do not know what object or property owns a given item. However, if you just want to find all the objects owned in a particular attribute of a class, a simpler way is to use a generated built-in view for all owning attributes (including atomic) that consists of the owning class name and property name, separated by an underscore.

This view is for LexEntry_Senses:

```
select [Owner$] as [Src], [Id] as [Dst], [OwnOrd$] as [Ord]
FROM [CmObject]
WHERE [OwnFlid$] = 5002011
```

When you use one of these views, you get a list of Src (owner) ids, Dst (owned) ids, and Ord (order) columns. If the property is atomic or is a collection rather than a sequence, it omits the Ord value:

```
select * from LexEntry_Senses order by Src, Ord
```

```
Src     Dst     Ord
6047    6049    1
6047    6050    2
```

This view states that LexEntry 6047 owns senses 6049 and 6050—in that order.

2.3.3 Reference relationships

2.3.3.1 Atomic references

Atomic references are CmObject Ids stored as an integer on the class that has that property:

```
select Id, SenseType from LexSense
```

```
Id      SenseType
6308    6336
```

In this case, LexSense (id = 6308) references a CmPossibility (id = 6336) in the Sense Types list.

2.3.3.2 Sequence and collection references

Collection and Sequence reference properties are implemented as separate tables in the database. The name is the class name and property name, separated with an underscore.

```
select * from LexReference_Targets
```

Src	Dst	Ord
6820	6308	1
6820	6099	2
6820	6270	3

Targets is a sequence reference property. In this example, LexReference (id = 6820) references three senses (6308, 6099, and 6270—in that order) via the Targets property.

Reference collections work the same way, except they omit the Ord values.

2.4 Views

SQL Server provides a way to simplify accessing data by taking common queries and packaging them into “views”. FieldWorks automatically builds many views based on class and property names. These were discussed earlier. Here is a summary:

- Class_: Returns a single row with all fields for a class and its superclasses (see section on Subclasses).
- Class_Property: Returns an ownership table for all owning properties (see section on Owning relationships).
- Class_Property: Returns a table for MultiStrings, MultiBigStrings, and MultiUnicode properties (see section on MultiStrings).

Another useful view is PropInfo$. This view combines the most useful information from Class$ and Field$ into a single table:

```
select * from PropInfo$ order by class, property
```

Class	Clid	Property	Flid	Type	Tid	Signature	Custom
ChkTerm	5125	Occurrences	512500	OwningSequence	27	ChkRef	false
ChkRef	5116	KeyWord	5116002	String	13		false

This table lists the Class Name and Id number, the Property Name and Field Id, the type of property and Id, the signature for owning/reference properties, custom flag, and Custom Id. FieldWorks classes and properties.xls is a spreadsheet that contains this information for ready access.

The following query lists the names of all views:

```
select * from sysobjects where type='V'
```

You can see the source code for a view using the following query:

```
select text from syscomments where id=object_id('LexSense_Senses')
```

Copy the results from the text field and paste it into ZEdit or some other editor to see more than the first line.

2.5 Stored procedures

FieldWorks provides many stored procedures for special purposes, particularly updating the database.

Caution: Be *extremely* cautious with any updates! It is easy to damage your data to the point where FieldWorks programs will fail.

The following query lists the names of all stored procedures:

```
select * from sysobjects where type='P'
```

You can see the source code for a stored procedure using the following query:

```
select text from syscomments where id=object_id('MakeObj_WfiWordform')
```

Copy the results from the text fields and paste them into ZEdit or some other editor to see more than the first line. If it is too long, it will come in several parts with an extra Return between each section.

The StoredProcs.htm file documents many of the FieldWorks stored procedures. It is fairly accurate, although not totally up-to-date. If there is any doubt, check the source code.

A number of stored procedures and functions have a “grfcpt” argument. This is an integer with a bit for each kind of owning and reference property desired:

Owning Atomic	8388608
Reference Atomic	16777216
Owning Collection	33554432
Reference Collection	67108864
Owning Sequence	134217728
Reference Sequence	268435456
All Owning	176160768
All Reference	352321536
All Owning & Reference	528482304

Some stored procedures and functions also have a “riid” argument. This is a Class Id that can be used to filter the returned results. It will return any instances of the specified class or any of its subclasses.

2.6 Functions

FieldWorks provides about 24 functions for special purposes. These can be used in various places in SQL queries to return a single value or data as though it were a table, even though it is not reading a literal table.

The following query lists the names of all functions:

```
select * from sysobjects where type = 'TF' or type = 'FN'
```

You can see the source code for a function using the following query:

```
select text from syscomments where id=object_id('fnGetOwnedObjects$')
```

Copy the results from the text fields and paste them into ZEdit or some other editor to see more than the first line. If it is too long, it will come in several parts with an extra Return between each section.

The file, StoredProcs.htm documents some of the FieldWorks functions. It is fairly accurate, although it is missing some details and has some errors. If there is any doubt, check the source code.

For an example using a function, see the section on “Display information from possibility list”.

3 Database Management Tool (dbmt)

Microsoft provides SQL Server Management Studio when you have full versions of SQL Server. This provides the ability to execute SQL queries, view tables, and views, and make basically any change you need to a database. This program cannot be redistributed.

In your C:\Program Files\SIL\FieldWorks directory, FieldWorks provides dbmt.exe. This gives the ability to execute SQL queries on the database and return the results in a table. This is very useful if you need to investigate something directly in the database, or make updates.

Caution: Be extremely cautious with any updates! It is easy to damage your data to the point where FieldWorks programs will fail.

When you start the program, it brings up a “Connect to SQL Server” dialog. The default server is “.\SILFW”, which will open your FieldWorks instance of SQL Server 2005 Express. You can also type in paths to open databases on other machines as well (e.g., ls-zook\SILFW). On your local machine, log on with “Windows authentication”. For remote machines, use “SQL Server authentication”. For the logon, type “FwDeveloper” and for the password, type “Careful”. You also need to use SQL Server authentication if you are not logged on as a system administrator.

If you are running Vista, you need to run dbmt as an administrator. Here’s how to start it:

1. Click the Start button
2. In the edit box above the Start button type dbmt
3. dbmt should show up in the program list above the edit box. Right-click this and choose Run as Administrator.
4. If a dialog comes up asking if you want to run the program, click Allow.
5. At this point you should be able to run dbmt normally.

After you are logged on, make sure the combo at the top of the window is set to the desired database. (**CAUTION!** Never modify the master table or you may need to uninstall SQL Server 2005 and reinstall it to get things to work again.) With the desired database selected, type or paste in queries and execute them by using F5 or clicking the green triangle on the toolbar. The results from the query show up in the bottom pane. You can select one row or one column and copy it to the clipboard to save the results. Click the open box to the left of the column headings to select the entire table and save it. Use Ctrl+C to copy it to the clipboard and Ctrl+V to paste it into Excel, ZEdit, Word, or some other program. Any data returned in this way is Unicode data. If you use ZEdit, make sure it is open to either UTF-8 or UTF-16 mode.

Dbmt does not parse a query ahead of time, so it will not execute multiple batches in a single query as SQL Server Management Studio can do. Multiple batches are separated by GO statements. The following example demonstrates one of these.

```
USE master
GO
Xp_readerrorlog
```

In order to execute these queries in SQL Server, you need to execute the code consecutively between each GO statement. In this case, highlight the first line and press F5, then highlight the third line and press F5.

Unlike SQL Server Management Studio, "dbmt" does not provide an object browser window that allows you to explore things such as tables, views, and stored procedures. Use this query to list their names:

```
select * from sysobjects where type ='x'
The "x" in this query can be one of the following:
    S: system tables
    U: user tables
    P: stored procedures
    V: views
    C: constraints
    PK: primary key
    TR: trigger
```

Use the following queries to get information about the columns and constraints of a table:

```
exec sp_MShelpcolumns 'LexEntry', @orderby = 'id'
exec sp_MStablekeys 'LexEntry'
exec sp_MStablechecks 'LexEntry'
```

The first query below lists triggers in the database, and the second displays the source code for a particular trigger.

```
select * from sysobjects where type = 'TR' order by name
select text from syscomments where id=object_id(N'TR_LgWritingSystem_ObjDel_Del')
```

4 Working with SQL

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 XML file. Any time you plan to do this, make sure you *first* back up your project and check carefully what you did before going on.

Several things *can* be done safely. It never hurts the database to execute "select" queries because they do not change the database. Thus, you can extract any information from the database without harm. The potential problems come with any "update", "insert", or "delete" SQL commands, or any stored procedure that modifies the database. Some of the issues are discussed in this section. Never change the structure of any of the tables.

For any questions on Microsoft Transact-SQL syntax, refer to SQL Server Books Online. This can be freely downloaded from www.microsoft.com/downloads/details.aspx?FamilyID=A6F79CB1-A420-445F-8A4B-BD77A7DA194B&displaylang=en and installed on your machine. The installation file is sqlbolsetup.msi (34.4Mb).

There are two ways to include comments in SQL code:

- Enclose the comments in /* comment */.
- Anything following two hyphens on a line is a comment.

4.1 Creating objects

Never create new objects by inserting them directly into CmObject or any of the class tables. This always involves updating CmObject as well as subclass tables. You should *only* insert new objects with one of the CreateObject stored procedures.

Caution! When some objects are created by a FieldWorks program, the C#/C++ code automatically creates additional objects or presets certain values. If this additional information is not set properly, it could cause the program to crash due to malformed data. As a result, unless you have a good understanding of what should happen, you should not attempt to add new objects.

The basic stored procedure for creating new objects is CreateOwnedObject$. See StoredProcs.htm for details. This example appends a new LexSense to an entry with an Id of 6047 and returns the Id and GUID of the new sense:

```
declare @newId int, @newGuid uniqueidentifier, @clid int, @flid int,
    @entryId int, @ownSeq int
set @ownSeq = 27
set @clid = 5016 -- LexEntry
set @entryId = 6047
set @flid = 5002011 -- LexEntry_Senses
exec CreateOwnedObject$ @clid, @newId output, @newGuid output, @entryId,
    @flid, @ownSeq, null
select @newId, @newGuid
```

There is a generated stored procedure for each class of the form MakeObj_LexEntry. These procedures have arguments for all of the basic attributes for the class and its superclasses. The arguments for the procedures are modified as you add or remove custom fields. You can get a list of the arguments by looking at the source code (see the "Stored procedures and functions" section). If objects contain *any* form of FieldWorks String (as opposed to FieldWorks Unicode), do *not* use SQL to add these objects because it cannot properly construct the Fmt portion of these strings. You can still use one of these stored procedures to create an object if you let the txt and "fmt" arguments null. If you need to add these kinds of objects, it is best done in XML (see FieldWorks XML model.doc).

WfiWordform is one object that does not contain *any* String properties. Check out the header for the source for this stored procedure to get the following arguments for the method:

```
@WfiWordform_Form_ws int = null, @WfiWordform_Form_txt nvarchar(4000) = null,
@WfiWordform_SpellingStatus int = 0,
@WfiWordform_Checksum int = 0,
@Owner int = null,
@OwnFlid int = null,
@StartObj int = null,
@NewObjId int output,
@NewObjGuid uniqueidentifier output,
@fReturnTimestamp tinyint = 0,
@NewObjTimestamp int = null output
```

The following method will create a new WfiWordform, setting the name to “new wordform” in the first vernacular writing system. It sets the SpellingStatus and Checksum to 0. The new wordform is appended to the WordformInventory_Wordforms property and the query returns an error code (0 = no errors), which is the Object Id and GUID of the new object. Note the N prefix added to the form to indicate this is a Unicode string. In this case, do not specify the new Id or GUID, but allow the program to generate these values. Also ignore the last two arguments:

```
declare @errRet int, @vern int, @owner int, @clid int, @ownflid int,
      @newId int, @newGuid uniqueidentifier
select top 1 @vern = dst from LangProject_CurVernWss
select @owner = dst from LangProject_WordformInventory
select @clid = id from Class$ where name = 'WordformInventory'
select @ownflid = id from Field$ where name = 'Wordforms' and class = @clid
exec MakeObj_WfiWordform @vern, N'new wordform', 0, 0, @owner, @ownflid, null,
      @newId output, @newGuid output
set @errRet = @@error
select @errRet, @newId, @newGuid
```

Because of the restrictions on adding new objects via SQL, it is better to add them via XML.

4.2 Deleting objects

Sometimes it is helpful to delete an object from the database via SQL.

Warning: *Never* do this by directly deleting rows from any class table!

Instead, use the stored procedure, “DeleteObjects”, which takes a string argument listing one or more comma-separated object Ids. When an object is deleted by this procedure, all objects owned by this object are also deleted. This includes all basic properties plus strings for all of the deleted objects. It also removes any of the deleted object Ids from any other objects that reference these objects.

This example uses a stored procedure to delete one object:

```
exec DeleteObjects '41189'
```

Caution! *Never* use this method to delete LgWritingSystems if there is *any* chance that the writing system is being used by any data! This method does *not* clean up multistring fields and has *no* way to clean up embedded strings. This will lead to certain crashes due to defective data. The only *safe* way to delete writing systems is the XML approach. In FieldWorks XML model.doc, see “Removing a writing system”.

4.3 Rearranging objects

There are several stored procedures discussed in StoredProcs.htm that enable moving objects or references. If you need to do this, check the documentation for MoveOwnedObject$, MoveToOwnedAtom$, MoveToOwnedColl$, MoveToOwnedSeq$, ReplaceRefColl…, and ReplaceRefSeq....

4.4 Display information from possibility list

Use this query to get a list of top-level possibility items from a list (Semantic Domains list here), and give the English name and abbreviation for each:

```
declare @ws int, @list int
select @ws = id from LgWritingSystem where ICULocale = 'en'
select @list = dst from LangProject_SemanticDomainList
select plp.dst Id, @ws Ws, pn.txt Name, pa.txt Abbreviation from
CmPossibilityList_Possibilities plp
left outer join CmPossibility_Name pn on pn.obj = plp.dst and pn.ws = @ws
left outer join CmPossibility_Abbreviation pa on pa.obj = plp.dst and pa.ws = @ws
where plp.src = @list
```

```
Id      Ws     Name                    Abbreviation
9944    40716  Universe, creation      1
10909   40716  Person                  2
12374   40716  Language and thought    3
14263   40716  Social behavior         4
16541   40716  Daily life              5
17272   40716  Work and occupation     6
18779   40716  Physical actions        7
20009   40716  States                  8
22259   40716  Grammar                 9
```

This is just a small subset of the semantic domains. The rest are nested under these, up to several levels. SQL does not handle recursion very well. In this case, since the semantic domain list is the only list that uses CmSemanticDomain items, it is possible to get around the recursion limitation by simply using the CmSemanticDomain table. Use the outline number for sorting. To get a full list of all 1,792 domains, use this query:

```
declare @ws int
select @ws = id from LgWritingSystem where ICULocale = 'en'
select sd.id Id, @ws Ws, pa.txt Abbr, pn.txt Name from CmSemanticDomain sd
left outer join CmPossibility_Name pn on pn.obj = sd.id and pn.ws = @ws
left outer join CmPossibility_Abbreviation pa on pa.obj = sd.id and pa.ws = @ws
order by pa.txt
```

```
Id      Ws     Abbr      Name
9944    40716  1         Universe, creation
9945    40716  1.1       Sky
9946    40716  1.1.1     Sun
9947    40716  1.1.1.1   Moon
9963    40716  1.1.1.2   Star
9975    40716  1.1.1.3   Planet
10007   40716  1.1.2     Air
```

If you have a hierarchical list that does not have a unique class name, a stored function, fnGetOwnedObjects$, can help. It returns a temporary table with the desired information. The third argument to the function is a mask (176160768) that indicates owned objects. The 7th argument (missing in StoredProcs.htm) indicates it should only return CmPossibility (class = 7) items and subclasses:

```
declare @ws int, @list int
select @ws = id from LgWritingSystem where ICULocale = 'en'
select @list = dst from LangProject_SemanticDomainList
select oi.ObjId Id, @ws Ws, pn.txt Name, pa.txt Abbr
```

```
        from fnGetOwnedObjects$(@list, null, 176160768, 0, 0, 1, 7, 1) oi
    left outer join CmPossibility_Name pn on pn.obj = oi.ObjId and pn.ws = @ws
    left outer join CmPossibility_Abbreviation pa on pa.obj = oi.ObjId and pa.ws = @ws
    order by oi.OrdKey
```

```
Id      Ws      Name               Abbr
9944    40716   Universe, creation 1
9945    40716   Sky                1.1
9946    40716   Sun                1.1.1
9947    40716   Moon               1.1.1.1
9963    40716   Star               1.1.1.2
9975    40716   Planet             1.1.1.3
10007   40716   Air                1.1.2
10008   40716   Blow air           1.1.2.1
10024   40716   Weather            1.1.3
```

4.5 Display headwords from dictionary

A dictionary headword displays the citation form, if it exists, otherwise the lexeme form. It also adds affix markers from the MorphType of the MoForm and displays the homograph number if not 0. This query will return headwords for all entries in the first vernacular writing system:

```
declare @vern int
select top 1 @vern = dst from LangProject_CurVernWss
select coalesce(t.Prefix collate SQL_Latin1_General_CP1_CI_AS, '') +
        coalesce(cf.Txt, f.txt) +
        coalesce(t.Postfix collate SQL_Latin1_General_CP1_CI_AS, '') +
        case le.HomographNumber
                when 0 then ''
                else cast(le.HomographNumber as varchar(3))
        end Headword
--select f.Txt lexeme, cf.Txt citation, le.HomographNumber homograph,
--      t.Postfix postfix, t.Prefix prefix
from LexEntry le
left outer join LexEntry_CitationForm cf on cf.Obj = le.id and cf.ws = @vern
left outer join LexEntry_LexemeForm lf on lf.Src=le.id
left outer join MoForm_Form f on f.Obj=lf.Dst and f.ws = @vern
left outer join MoForm mf on mf.Id=lf.Dst
left outer join MoMorphType t on t.Id=mf.MorphType
```

```
Headword
*himbilira1
nadra
ke=
=lo
-ul-
dok2
```

If you comment out the first select and uncomment the second, you get a table with lexeme form, citation form, homograph number, postfix, and prefix. The two collate clauses are necessary when concatenating strings with different collations.

4.6 Add translations to lists

To add translations for a large list of items such as names on the Semantic Domain list, do the following:

1. Dump out the original names by using one of the techniques above.
2. Translate these names.
3. Massage this data into this format:
   exec SetSemanticDomainName @anal, N'Sky', @vern, N'天空'
   where the first name is the original English name and the second name is the translation.
4. Execute the following query. This creates a temporary stored procedure that will accept the step 3 commands. This stored procedure looks up a name in the analysis writing system. It then modifies or inserts the vernacular name for the same item. Here is the query to create the stored procedure:

```
CREATE  proc [SetSemanticDomainName]
   @WsAnal int,
   @AnalTxt nvarchar(1000),
   @WsVern int,
   @VernTxt nvarchar(1000)
as
   declare @id int, @tmp int
   set @id = null
   select @id=sd.id from CmSemanticDomain sd
      join CmPossibility_Name pn on pn.obj = sd.id
      where pn.ws = @WsAnal and pn.txt = @AnalTxt
   if @id is not null begin
      set @tmp = null
      select @tmp=obj from CmPossibility_Name where obj = @id and ws = @WsVern
      if @tmp is null begin
         insert into CmPossibility_Name (obj, ws, txt) values (@id, @WsVern, @VernTxt)
      end else begin
         update CmPossibility_Name set txt = @VernTxt where obj = @id and ws = @WsVern
      end
   end
```

5. Execute a query with the massaged data, setting the writing systems accordingly. Summary:

```
declare @anal int, @vern int
select @anal=id from LgWritingSystem where IcuLocale = 'en'
select @vern=id from LgWritingSystem where IcuLocale = 'zh'
exec SetSemanticDomainName @anal, N'The physical universe', @vern, N'物质世界'
exec SetSemanticDomainName @anal, N'Sky', @vern, N'天空'
……
```

6. Execute the following query to remove the temporary stored procedure:

```
if object_id('SetSemanticDomainName') is not null begin
      print 'removing proc SetSemanticDomainName'
      drop proc [SetSemanticDomainName]
      end
end
```

These three steps can all be stored in a single file with a GO between each section. It can then be run in one step by using the “db exec” command, but it must be stored as UTF-16 for db exec to work properly (unless you can simply use ANSI).

4.7 Reload anthropology list

If you want to change your anthropology list for some reason, and you do not mind losing any links that you have set up in FieldWorks to the existing anthropolofy codes (in Flex or Data Notebook), you can use the following process to reload the list. If you have a lot of links that you would not want to lose, there is another way using the FieldWorks XML dump file that may work, depending on the situation.

To reload the anthro list:

1. Use File...Project Management...Backup and Restore to back up your project in case anything goes wrong, then close all FieldWorks programs.
2. Start the dbmt program in your c:\Program Files\SIL\FieldWorks directory.
3. Click OK to the initial dialog, then select Ibwe (or your FieldWorks project) in the combo box at the top.
4. Paste the following query into dbmt and press F5 to execute it.
   ```
   select * from LangProject_AnthroList
   ```
5. Note the number in the Dst column after executing the above query. Paste the following query into dbmt, replacing 159 with the number in your Dst column, and then execute it with F5.
   ```
   exec deleteObjects '159'
   ```
   This step took 5 minutes on my fast machine, so it will probably take 2-3 times longer on your machine. Just wait until it completes, even if it doesn't appear to be doing anything.
6. Paste the following query into dbmt and execut it with F5
   ```
   declare @id int, @lp int, @guid uniqueidentifier
   select @lp = id from LangProject
   exec CreateOwnedObject$ 8, @id output, @guid output, @lp, 6001012, 23, null
   ```
7. Start Flex or Data Notebook on your FieldWorks project. It should come up with a "Choose a List of Anthropology Categories" dialog (which may be partially hidden by the splash screen). Pick the list you want to load (usually the top one) and click OK. If it happens to crash at the end of this, just restart the program and it should work OK.

4.8 Delete orphaned entries

Probably due to a problem with importing incorrectly, one lexical database had thousands of lexical entries that basically had a lexeme form and nothing else. These entries were missing senses, but minor entries also typically do not have senses. The following query lists all of the entries that are missing senses and do not have MainEntriesOrSenses set.

```
select le.id from LexEntry_ le
left outer join LexEntry_Senses ls on ls.src = le.id
left outer join LexEntry_MainEntriesOrSenses ms on ms.src = le.id
where ls.src is null and ms.src is null
order by DateCreated
```

This query can be converted into a query that deletes these entries from the database. Large deletions are typically slow, so this query may take around 20 minutes to delete several thousand entries.

```
declare @hvo int, @nvchvo nvarchar(20)
declare mycursor cursor local static forward_only read_only for
   select le.id from LexEntry_ le
   left outer join LexEntry_Senses ls on ls.src = le.id
   left outer join LexEntry_MainEntriesOrSenses ms on ms.src = le.id
   where ls.src is null and ms.src is null
open mycursor
fetch next from mycursor into @hvo
while @@fetch_status = 0
begin
   set @nvchvo = @hvo
   exec deleteobjects @nvchvo
   fetch next from mycursor into @hvo
end
close mycursor
deallocate mycursor
```

4.9 Delete Scripture from a FieldWorks project

This query will delete everything related to Scripture from a FieldWorks project. The nocount sections eliminate a long series of reporting that is basically useless. This is especially helpful when running this using the db program as described below.

```
-- Turn off reporting
declare @fIsNocountOn int
set @fIsNocountOn = @@options & 512
if @fIsNocountOn = 0 set nocount on

declare @hvo int, @nvchvo nvarchar(20)
-- Delete Scripture
select @hvo = dst from LangProject_TranslatedScripture
set @nvchvo = @hvo
exec deleteobjects @nvchvo
-- Delete Scripture UserViews
declare mycursor cursor local static forward_only read_only for
   select id from UserView where App = 'A7D421E1-1DD3-11D5-B720-0010A4B54856'
open mycursor
fetch next from mycursor into @hvo
while @@fetch_status = 0
begin
   set @nvchvo = @hvo
   exec deleteobjects @nvchvo
   fetch next from mycursor into @hvo
end
close mycursor
deallocate mycursor

Restore reporting to original state
if @fIsNocountOn = 0 set nocount off
```

Note, this query could be placed in a text file (e.g., DeleteScripture.sql) in your %ALLUSERSPROFILE%\Application Data\SIL\FieldWorks\Data directory (**Note**: %ALLUSERSPROFILE%\Application Data is c:\Documents and Settings\All Users\Application Data on Windows XP and c:\ProgramData on Vista.) and run from a command line using the db program. This could also be placed in a batch file and connected to a desktop icon for simple execution, if you wanted to do something like this frequently. The batch file might have:

```
db delete MyDictProject
db copy MyProject MyDictProject
db exec DeleteScripture.sql MyDictProject
```

4.10 Delete all items from a possibility list

This query will delete all items in the DomainTypes (Academic Domains) list. This might be useful if you want to import your own list of local semantic domains, or some other list you’ve created. By changing the initial select, you could clear out any other list as well.

```
declare @hvo int, @nvchvo nvarchar(20)
select @hvo = dst from LexDb_DomainTypes
declare mycursor cursor local static forward_only read_only for
   select id from CmPossibility_ where owner$ = @hvo
open mycursor
fetch next from mycursor into @hvo
while @@fetch_status = 0
begin
   set @nvchvo = @hvo
   exec deleteobjects @nvchvo
   fetch next from mycursor into @hvo
end
close mycursor
deallocate mycursor
```

4.11 Exploring interlinear text

The following query provides basic information on interlinear texts including Ids for significant objects in the structure and text for titles, paragraph baselines, and translations and notes. If your titles are primarily in some writing system other than English, change the ‘en’ on the second line to the writing system IcuLocale of the desired writing system. See Conceptual model overview.doc section on Interlinear Text for further details.

```
declare @titleWs int, @seg int
select @titleWs = id from LgWritingSystem where IcuLocale = 'en'
select @seg = cad.id from CmAnnotationDefn cad
join CmPossibility_Name cpn on cpn.obj = cad.id
where cpn.txt = 'Text Segment'
select tx.id Text, cn.txt Title, sp.src StText, sp.dst Para, sp.ord Pos, par.contents BaseLine,
      cba.id Seg, cba.BeginOffset BegOff, at.src IndAnn, ws.IcuLocale ws, co.txt Trans
from Text tx
left outer join CmMajorObject_Name cn on cn.obj = tx.id and cn.ws = @titleWs
left outer join Text_Contents tc on tc.src = tx.id
left outer join StText_Paragraphs sp on sp.src = tc.dst
join StTxtPara par on par.id = sp.dst
left outer join CmBaseAnnotation_ cba on cba.BeginObject = sp.dst and cba.AnnotationType
= @seg
left outer join CmIndirectAnnotation_AppliesTo at on at.dst = cba.id
left outer join CmAnnotation_Comment co on co.obj = at.src
left outer join LgWritingSystem ws on ws.id = co.ws
order by cn.txt, tx.id, sp.ord, cba.BeginOffset
```

This is a truncated example of the output for Sena 3.

```
Text  Title         StText  Para  Pos  BaseLine        Seg    BegOff IndAnn ws  Trans
228   Canoe trip    229     230   1    Wapakila m'm... 17811  0      17812  en  "He embarked in a canoe…
228   Canoe trip    229     231   2    Na masiku mb... 49167  0
225   Have courage  226     227   1    Pisapha, mbw... 17719  0      17720  en  These things hurt but wh…
225   Have courage  226     227   1    Pisapha, mbw... 17719  0      17720  pt  Estas coisas doem mas…
232   In the garden 233     234   1    Babanga ana...  17806  0      17807  en  My father has a large…
232   In the garden 233     235   2    Mwenemo mu...
232   In the garden 233     236   3    Muna milara...
232   In the garden 233     237   4    Ikhalipo minga...
232   In the garden 233     238   5    Baba aipisa.
232   In the garden 233     239   6    Mwezi wa kh...
232   In the garden 233     240   7    Mpunga wab...
```

The following query can be used to give information on the interlinear annotations for an interlinear text. The title of the interlinear text should be entered on the 3rd line. In versions after FieldWorks 5.0 a stored function fnGetTextAnnotations will be available to give this information. This method uses the top analysis language for the glosses and the top vernacular writing system for the wordforms. (Thanks to Steve Miller for this example.)

```
DECLARE @nAnnotationDefnPIC INT, @nAnalysisWS INT, @nVernacularWS INT,
@nvcTextName NVARCHAR(4000)
SELECT @nvcTextName = N'Canoe trip'

SELECT @nAnnotationDefnPIC = Obj
FROM CmPossibility_Name
WHERE Txt = 'Punctuation In Context'

SELECT TOP 1 @nVernacularWS = dst FROM LangProject_CurVernWss
SELECT TOP 1 @nAnalysisWS = dst FROM LangProject_CurAnalysisWss

DECLARE @tblTextAnnotations TABLE (
        TextId INT,
        TextName NVARCHAR(4000),
        Paragraph INT,
        StTxtParaId INT,
        BeginOffset INT,
        EndOffset INT,
        AnnotationId INT,
        WordFormId INT,
        Wordform NVARCHAR(4000),
        AnalysisId INT,
        GlossId INT,
        Gloss NVARCHAR(4000))

IF @nAnalysisWS IS NULL
        SELECT TOP 1 @nAnalysisWS = Dst
        FROM LangProject_CurAnalysisWss ORDER BY Ord
IF @nVernacularWS IS NULL
        SELECT TOP 1 @nVernacularWS = dst
        FROM LangProject_CurVernWss ORDER BY Ord

--== Annotation is not an InstanceOf anything ==--
INSERT INTO @tblTextAnnotations
SELECT
        cmon.Obj AS TextId,
	cmon.Txt AS TextName,
	tp.Ord AS Paragraph,
	stp.Id AS StTxtParaId,
	cba.BeginOffset,
	cba.EndOffset,
	cba.Id AS AnnotationId,
	NULL AS WordFormId,
	SUBSTRING(stp.Contents, cba.BeginOffset + 1, cba.EndOffset - cba.BeginOffset)
		COLLATE SQL_Latin1_General_CP1_CI_AS AS WordForm, --( avoids collate
mismatch
	NULL AS AnalysisId,
	NULL AS GlossId,
	NULL AS Gloss
FROM CmMajorObject_Name cmon
JOIN Text_Contents tc ON tc.Src = cmon.Obj
JOIN StText st ON st.Id = tc.Dst
JOIN StText_Paragraphs tp ON tp.Src = st.Id
JOIN StTxtPara stp ON stp.Id = tp.Dst
JOIN CmBaseAnnotation cba ON cba.BeginObject = stp.Id
JOIN CmAnnotation ca ON ca.Id = cba.Id
WHERE ca.InstanceOf IS NULL
	AND cmon.Txt = @nvcTextName
	AND ca.AnnotationType = @nAnnotationDefnPIC
--== Annotation is an InstanceOf Wordform ==--
UNION
SELECT
	cmon.Obj AS TextId,
	cmon.Txt AS TextName,
	tp.Ord AS Paragraph,
	stp.Id AS StTxtParaId,
	cba.BeginOffset,
	cba.EndOffset,
	cba.Id AS AnnotationId,
	wwff.Obj AS WordFormId,
	wwff.Txt AS WordForm,
	NULL AS AnalysisId,
	NULL AS GlossId,
	NULL AS Gloss
FROM CmMajorObject_Name cmon
JOIN Text_Contents tc ON tc.Src = cmon.Obj
JOIN StText st ON st.Id = tc.Dst
JOIN StText_Paragraphs tp ON tp.Src = st.Id
JOIN StTxtPara stp ON stp.Id = tp.Dst
JOIN CmBaseAnnotation cba ON cba.BeginObject = stp.Id
JOIN CmAnnotation ca ON ca.Id = cba.Id
JOIN WfiWordForm_Form wwff ON wwff.Obj = ca.InstanceOf AND wwff.WS =
@nVernacularWS
WHERE cmon.Txt = @nvcTextName
--== Annotation is an InstanceOf Annotation ==--
UNION
SELECT
	cmon.Obj AS TextId,
	cmon.Txt AS TextName,
	tp.Ord AS Paragraph,
	stp.Id AS StTxtParaId,
	cba.BeginOffset,
        cba.EndOffset,
        cba.Id AS AnnotationId,
        wwff.Obj AS WordFormId,
        wwff.Txt AS WordForm,
        wa.Id AS AnalysisId,
        NULL AS GlossId,
        NULL AS Gloss
    FROM CmMajorObject_Name cmon
    JOIN Text_Contents tc ON tc.Src = cmon.Obj
    JOIN StText st ON st.Id = tc.Dst
    JOIN StText_Paragraphs tp ON tp.Src = st.Id
    JOIN StTxtPara stp ON stp.Id = tp.Dst
    JOIN CmBaseAnnotation cba ON cba.BeginObject = stp.Id
    JOIN CmAnnotation ca ON ca.Id = cba.Id
    JOIN WfiAnalysis wa ON wa.Id = ca.InstanceOf
    LEFT OUTER JOIN WfiWordForm_Analyses wwfa ON wwfa.Dst = wa.Id
    LEFT OUTER JOIN WfiWordForm_Form wwff ON wwff.Obj = wwfa.Src AND wwff.WS =
    @nVernacularWS
    WHERE cmon.Txt = @nvcTextName
    --== Annotation is an InstanceOf Gloss ==--
    UNION
    SELECT
        cmon.Obj AS TextId,
        cmon.Txt AS TextName,
        tp.Ord AS Paragraph,
        stp.Id AS StTxtParaId,
        cba.BeginOffset,
        cba.EndOffset,
        cba.Id AS AnnotationId,
        wwff.Obj AS WordFormId,
        wwff.Txt AS WordForm,
        wa.Id AS AnalysisId,
        wgf.Obj AS GlossId,
        wgf.Txt AS Gloss
    FROM CmMajorObject_Name cmon
    JOIN Text_Contents tc ON tc.Src = cmon.Obj
    JOIN StText st ON st.Id = tc.Dst
    JOIN StText_Paragraphs tp ON tp.Src = st.Id
    JOIN StTxtPara stp ON stp.Id = tp.Dst
    JOIN CmBaseAnnotation cba ON cba.BeginObject = stp.Id
    JOIN CmAnnotation ca ON ca.Id = cba.Id
    JOIN WfiGloss_Form wgf ON wgf.Obj = ca.InstanceOf AND wgf.WS = @nAnalysisWS
    LEFT OUTER JOIN WfiAnalysis_Meanings wam ON wam.Dst = wgf.Obj
    LEFT OUTER JOIN WfiAnalysis wa ON wa.Id = wam.Src
    LEFT OUTER JOIN WfiWordForm_Analyses wwfa ON wwfa.Dst = wa.Id
    LEFT OUTER JOIN WfiWordForm_Form wwff ON wwff.Obj = wwfa.Src AND wwff.WS =
    @nVernacularWS
    WHERE cmon.Txt = @nvcTextName
    ORDER BY tp.Ord, cba.BeginOffset

    SELECT * FROM @tblTextAnnotations
```

Here's a truncated sample of the output from Sena 3.

```
Text  TextName    Para ParaId  BOff  EOff  AnnId  WFId  Wordform   AnalId  GlossId  Gloss
228   Canoe trip  1    230     0     8     17769  3037  wapakila   3038    3039     he embarked
228   Canoe trip  1    230     9     17    17759  2962  m'mwadia   2967    2968     in canoe
```

```
228  Canoe trip  1   230   19   27   17813  3200  mbakwira     3201   3202   go up
228  Canoe trip  1   230   28   30   17760  2973  pa           2974   2975   ASSOC
228  Canoe trip  1   230   31   35   17775  2978  mudi         2979
228  Canoe trip  1   230   39   44   17774  2981  maulo        2982   2983   afternoon
228  Canoe trip  1   230   46   51   17810  3196  dzuwa        3197   3198   sun
228  Canoe trip  1   230   52   61   17761  2990  mbidadoka
228  Canoe trip  1   230   63   71   17762  2991  mbatsama     2992
228  Canoe trip  2   231   0    2    17758  2856  na           2857
228  Canoe trip  2   231   3    9    17814  2997  masiku       2998   2999   evening
228  Canoe trip  2   231   10   18   17763  3002  mbazidza     3003   3004   come
228  Canoe trip  2   231   19   27   17808  3188  nkhalamu     3189   3190   lion
228  Canoe trip  2   231   28   34   17815  3206  ziwiri       3207   3208   two
228  Canoe trip  2   231   36   45   17764  3014  mbazilila    3015
228  Canoe trip  2   231   47   54   17765  3018  mbagopa      3019   3020   afraid
228  Canoe trip  2   231   56   65   17766  3024  mbapakila    3025
228  Canoe trip  2   231   66   72   17809  3192  pontho       3193   3194   again
228  Canoe trip  2   231   73   81   17767  2962  m'mwadia     2963
228  Canoe trip  2   231   83   93   17768  3029  mbawambuka   3030   3031   cross
```

4.12 Baseline and Translations from interlinear text

The following query can be used to gather information about your interlinear text. This lists the interlinear text id and title, the StText id, along with each Paragraph id, position, and baseline text, then the segment ids within each paragraph, and the type and translation/note for each segment.

```
declare @titleWs int, @segDef int, @analWs int
select @titleWs = id from LgWritingSystem where IcuLocale = 'en'
select @segDef = id from CmObject where Guid$ = 'B63F0702-32F7-4ABB-B005-
C1D2265636AD'
select top 1 @analWs = dst from LangProject_CurAnalysisWss order by ord
select tx.id Text, cn.txt Title, sp.src StText, sp.dst Para, sp.ord Pos, par.contents BaseLine,
     seg.id Seg, abb.txt Type, com.txt Trans
from Text tx
left outer join CmMajorObject_Name cn on cn.obj = tx.id and cn.ws = @titleWs
left outer join Text_Contents tc on tc.src = tx.id
left outer join StText_Paragraphs sp on sp.src = tc.dst
join StTxtPara par on par.id = sp.dst
left outer join CmBaseAnnotation_ seg on seg.BeginObject = par.id and seg.AnnotationType =
@segDef
left outer join CmIndirectAnnotation_AppliesTo apto on apto.dst = seg.id
left outer join CmAnnotation ann on ann.id = apto.src
left outer join CmPossibility_Abbreviation abb on abb.obj = ann.AnnotationType and abb.ws =
@analWs
left outer join CmAnnotation_Comment com on com.obj = ann.id and com.ws = @analWs
order by cn.txt, tx.id, sp.ord, seg.BeginOffset
```

Here’s a truncated sample of the output from Sena 3 (with the first analysis writing system set to English).

```
Text  Title          StText  Para  Pos  BaseLine                  Seg    Type  Trans
246   Canoe trip     247     248   1    Wapakila m'mwadia…        23032  FT    He embarked in a canoe...
246   Canoe trip     247     249   2    Na masiku mbazidza…       23055  FT    At night two lions came and…
243   Have courage   244     245   1    Pisapha, mbwenye…         22965  FT    These things hurt but what...
243   Have courage   244     245   1    Pisapha, mbwenye...       23057  FT    For all of us will die.
250   In the Garden  251     252   1    Babanga ana munda…        23027  FT    My father has a large garden.
250   In the Garden  251     253   2    Mwenemo muna miti...      23043  FT    There he has many trees.
250   In the Garden  251     254   3    Muna milaranja, mifigu... 23045  FT    He has orange trees, banana...
250   In the Garden  251     255   4    Ikhalipo minga m'munda.   23047  FT    There were thorns in the garden.
250   In the Garden  251     256   5    Baba aipisa.              23049  FT    Father burned them.
250   In the Garden  251     257   6    Mwezi wa khumi na...      23051  FT    In November he planted a lot...
250   In the Garden  251     258   7    Mpunga wabuluka...        23053  FT    The rice produced well.
```

4.13 Splitting an interlinear text

Suppose you have an interlinear text and want to move some of the paragraphs to another text, or into a new text. There is no way to do this in the current program. You can cut some paragraphs from the baseline text and past them into a new text, but in the process all the interlinearization, free translations, and notes get lost. However, moving paragraphs is fairly easy to do via SQL. The object is to move the desired StTxtPara elements to another StText, making sure the id of the StTxtPara is not changed. The interlinearization, free translations, and notes will then move with the paragraph since they are references to the paragraph.

The following query can be used to gather information about your interlinear text. If you include the ‘where’ clause to limit the output based on the baseline text, and the baseline text contains an apostrophe, it must be quoted with a leading apostrophe. The % at the beginning and end of the string is a wildcard meaning anything can go before or after the string.

```
declare @titleWs int
select @titleWs = id from LgWritingSystem where IcuLocale = 'en'
select tx.id Text, cn.txt Title, sp.src StText, sp.dst Para, sp.ord Pos, par.contents BaseLine
from Text tx
left outer join CmMajorObject_Name cn on cn.obj = tx.id and cn.ws = @titleWs
left outer join Text_Contents tc on tc.src = tx.id
left outer join StText_Paragraphs sp on sp.src = tc.dst
join StTxtPara par on par.id = sp.dst
--where par.contents like N'%Wapakila m''mwadia%'
order by cn.txt, tx.id, sp.ord
```

Here’s a sample of the output for Sena 3 after creating a new interlinear text.

```
Text   Title          StText  Para   Pos  BaseLine
228    Canoe trip     229     230    1    Wapakila m'mwadia, mbakwira pa mudi na…
228    Canoe trip     229     231    2    Na masiku mbazidza nkhalamu ziwiri…
225    Have courage   226     227    1    Pisapha, mbwenye pinafunika n'khuphata …
232    In the garden  233     234    1    Babanga ana munda ukulu.
232    In the garden  233     235    2    Mwenemo muna miti mizinji.
232    In the garden  233     236    3    Muna milaranja, mifigu, mindimu na mimanga.
232    In the garden  233     237    4    Ikhalipo minga m'munda.
232    In the garden  233     238    5    Baba aipisa.
232    In the garden  233     239    6    Mwezi wa khumi na ubodzi abzwala maningi…
232    In the garden  233     240    7    Mpunga wabuluka maningi.
49168 New Text        49169   49170  1
```

Suppose the goal is to move the first paragraph from the ‘Canoe trip’ to the end of ‘New Text’. In this case we want to move the 230 paragraph from the 229 StText to the 49169 StText following the 49170 paragraph. The MoveOwnedObject$ stored procedure for moving owned objects in a vector is the best way to accomplish this move. It can move one or more paragraphs. Here’s a summary of the parameters to MoveOwnedObject$:

@SrcObjId int, – The ID of the object that owns the source object(s)

@SrcFlid int, – The FLID (field ID) of the object attribute that owns the object(s)

@ListStmp int, – The timestamp value of the object(s) to be moved. Unused.

@StartObj int = null, – The ID of the first object to be moved

@EndObj int = null, – The ID of the last object to be moved

@DstObjId int, – The ID of the object which will own the object(s) moved
@DstFlid int, – The FLID (field ID) of the object attribute that will own the object(s)
@DstStartObj int = null – the ID of the object before which the object(s) will
be moved. If null, the objects will be appended

The FLID for StText_Paragraphs is 14001. Thus the following query will move the single paragraph to the end of the destination StText.

```
exec MoveOwnedObject$ 229, 14001, null, 230, 230, 49169, 14001, null
```

After executing this query, the results from the first query verify that the paragraph moved as desired. If you open the database in Flex, you'd see that the interlinearization and free translations moved with the paragraph.

```
Text  Title        StText Para  Pos  BaseLine
228   Canoe trip   229    231   2    Na masiku mbazidza nkhalamu ziwiri…
225   Have courage 226    227   1    Pisapha, mbwenye pinafunika n'khuphata...
232   In the garden 233   234   1    Babanga ana munda ukulu.
232   In the garden 233   235   2    Mwenemo muna miti mizinji.
232   In the garden 233   236   3    Muna milaranja, mifigu, mindimu na mimanga.
232   In the garden 233   237   4    Ikhalipo minga m'munda.
232   In the garden 233   238   5    Baba aipisa.
232   In the garden 233   239   6    Mwezi wa khumi na ubodzi abzwala maningi…
232   In the garden 233   240   7    Mpunga wabuluka maningi.
49168 New Text     49169  49170 1
49168 New Text     49169  230   2    Wapakila m'mwadia, mbakwira pa mudi na…
```

4.14 Correcting interlinear analyses

One team had hundreds of interlinear analyses made where the morpheme line was set to the suffix lexeme form ‘se_1’. Later they realized a better analysis would be to use the ‘e’ allomorph of the ‘se_1’ entry in the environment where the preceding morph ended in r, l, m, n, s, or S. There were at least three complicating factors that made it impractical to do this within the Flex UI. First, there were two homographs of ‘se’, but the morph line showed ‘se’ for both. Only ‘se_1’ had the ‘e’ allomorph. Second, there isn't any way to filter on the environment, and this suffix was very common. Third, the UI doesn't provide a way to change just the morph line for a given analysis. In this case, all the program needs to do is change the WfiMorphBundle for the appropriate wordform analyses from pointing to the lexeme form to point to the allomorph instead. The following query displays information on the analyses that need to be changed.

```
declare @se1 int, @e int
select top 1 @se1 = mf.obj, @e = mfa.obj from MoForm_Form mf
join LexEntry_LexemeForm cf on cf.dst = mf.obj
join LexEntry le on le.id = cf.src
left outer join LexEntry_AlternateForms af on af.src = le.id
left outer join MoForm_Form mfa on mfa.txt = 'e'
where mf.txt = 'se' and le.HomographNumber = 1
select wf.txt Word, wa.owner$ WordId, mf1.txt Morph1, mb1.id Morph1Id,
   mf2.txt Morph2, mb2.id Morph2Id from WfiMorphBundle_ mb2
join WfiMorphBundle_ mb1 on mb1.owner$ = mb2.owner$
    and mb1.ownord$ = mb2.ownord$ - 1
left outer join MoForm_Form mf1 on mf1.obj = mb1.morph
left outer join MoForm_Form mf2 on mf2.obj = mb2.morph
join WfiAnalysis_MorphBundles mb on mb.dst = mb1.id
join WfiAnalysis_ wa on wa.id = mb.src
left outer join WfiWordform_Form wf on wf.obj = wa.owner$
where mb2.morph = @se1 and substring(mf1.txt, len(mf1.txt), 1) in ('r', 'l', 'm', 'n', 's', 'S')
```

Here's a partial list of the results from this query showing the suffix as Morph2 and the preceding root as Morph1 and the full wordform on the left.

```
Word        WordId   Morph1   Morph1Id  Morph2  Morph2Id
aanane      22994    aanan    22997     se      22998
ahhase      23282    ahhes    23285     se      23286
ahhaye      23287    ahhes    23290     se      23291
ammane      24379    n        24383     se      24384
aNise       25020    aNNis    25023     se      25024
aroose      25735    aros     25738     se      25739
ekTeSmine   30420    Smin     30424     se      30425
hinTiSi     37853    hinTis   37856     se      37857
saatare     70214    saatar   70217     se      70218
```

The following query will then make the desired switch to use the 'e' allomorph in place of the 'se' lexeme form.

```
declare @se1 int, @e int
select top 1 @se1 = mf.obj, @e = mfa.obj from MoForm_Form mf
join LexEntry_LexemeForm cf on cf.dst = mf.obj
join LexEntry le on le.id = cf.src
left outer join LexEntry_AlternateForms af on af.src = le.id
left outer join MoForm_Form mfa on mfa.txt = 'e'
where mf.txt = 'se' and le.HomographNumber = 1
update WfiMorphBundle set morph = @e
where id in (
  select mb2.id from WfiMorphBundle_ mb2
  join WfiMorphBundle_ mb1 on mb1.owner$ = mb2.owner$
       and mb1.ownord$ = mb2.ownord$ - 1
  left outer join MoForm_Form mf1 on mf1.obj = mb1.morph
  where mb2.morph = @se1 and substring(mf1.txt, len(mf1.txt), 1) in ('r', 'l', 'm', 'n', 's', 'S'))
```

4.15 Merging Lexical Relations

Flex currently does not provide a convenient way to merge relations from one lexical relation type to another. As long as they have the same reference set type, this can be done easily using a SQL query. For example, suppose you had two specific-generic relations, one with capitalized names and the other with lowercase names. (One way this can happen is via a LinguaLinks import where names differ.)

You can tell if a lexical relation is being used by going to the Lexical Relations view in the Lists area. Click on the relation and then click the X deletion button in the toolbar. In the box that comes up, if there are any senses or entries using this relation, a paragraph will come up saying how many items are linked to this relation.

Another way to tell is using a SQL query to investigate a given relation. In this example we will look for links to a relation with generic as the reverse name and generic as a referse name. We'll transfer items from the generic relation to the Generic relation. The following two queries will show the LexReference items that will be moved.

```
-- Show relations in source
select lrm.src, lrm.dst from LexRefType lrt
```

```
  left outer join LexRefType_ReverseName lrn on lrn.obj = lrt.id
  join LexRefType_Members lrm on lrm.src = lrt.id
  where lrn.txt = 'generic' -- source Reverse Name

-- Show relations in destination
select lrm.src, lrm.dst from LexRefType lrt
  left outer join LexRefType_ReverseName lrn on lrn.obj = lrt.id
  join LexRefType_Members lrm on lrm.src = lrt.id
  where lrn.txt = 'Generic' -- destination Reverse Name
```

The following query will move the LexReferences from the source LexRefType to the destination LexRefType.

```
-- Merge lexical relations from 'generic' to 'Generic'
declare @srcRefType int, @dstRefType int
select @srcRefType=lrt.id from LexRefType lrt
  left outer join LexRefType_ReverseName lrn on lrn.obj = lrt.id
  where lrn.txt = 'generic' -- source Reverse Name
select @dstRefType=lrt.id from LexRefType lrt
  left outer join LexRefType_ReverseName lrn on lrn.obj = lrt.id
  where lrn.txt = 'Generic' -- destination Reverse Name
select * from LexRefType_Members where src = @srcRefType -- show source relations
select * from LexRefType_Members where src = @dstRefType -- show destination relations
update LexRefType_Members set src = @dstRefType where src = @srcRefType
```