Python for FlexTools and FLEx 9.1

Ken Zook

June 26, 2024

Contents

1 Introduction

FieldWorks Language Explorer (FLEx) 9.1.25 is a program for developing dictionaries with any number of languages or writing systems, specifying grammar features, analyzing texts, parsing texts, and developing a research notebook (https://software.sil.org/fieldworks/).

FlexTools 2.3.1 is a program that allows Python scripts to read and/or modify data in a FLEx project using FLEx underlying code (https://github.com/cdfarrow/FLExTools/wiki), thus allowing skilled users to do tasks that are not feasible to do directly in FLEx. This program provides many shortcuts for doing common things that can't be done directly in FLEx.

This document gives an introduction to Python code that can be used in FlexTools or through the Python console to access and modify data beyond the numerous methods provided in FlexTools. It provides Python code to access FLEx data through the LCM (SIL Language & Culture Model) library. FlexTools uses Python 3.12 (https://www.python.org/downloads/).

This document assumes a basic understanding of the FLEx conceptual model as described in https://downloads.languagetechnology.org/fieldworks/Documentation/FLEx%209.1%20Conceptual%20Model.pdf.

2 Using Python in a FlexTools module

FlexTools will not allow you to open a project if FLEx, or any other program, already has that project open. There is an option in FLEx that allows you to use FlexTools with FLEx open. To set this mode in FLEx, go to File > Project Management > FieldWorks Project Properties > Sharing, and check "Share project contents with programs on this computer".

For the various examples given in remaining sections, the following imports are needed at the beginning of the module:

```
from flextoolslib import *
from SIL.LCModel import *
from SIL.LCModel.Core.KernelInterfaces import ITsString, ITsTextProps, ITsStrFactory,
ITsPropsFactory, ITsStrBldr, ITsIncStrBldr, ITsPropsBldr, ITsMultiString, FwTextPropType,
FwTextPropVar
from SIL.LCModel.Core.Text import TsStringUtils,TsIncStrBldr, TsStrBldr, TsStringComparer
from SIL.LCModel.Core.Cellar import CellarPropertyType, GenDate
from SIL.LCModel.Core.WritingSystems import CoreWritingSystemDefinition
from System import Guid, String, DateTime, Byte
import unicodedata
```

FlexTools typically uses 'project' to get to the FLEx project, and project.project to get to the LCM cache. The variable for the project is the first parameter to the module definition. My preference is defining some other names starting with project as the project name.

The following definitions are used in the rest of this document.

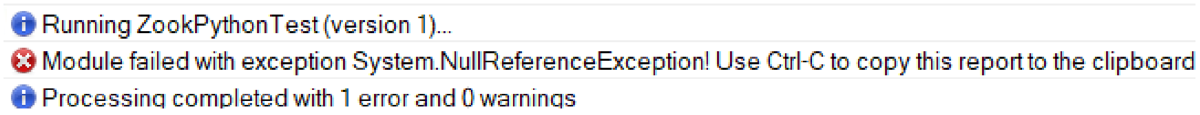
```
#----------------------------------------------------------------
def PythonTest(project, report, modifyAllowed):
# ----------------------------------------------------------------
cache = project.project
lp = cache.LangProject
sl = cache.ServiceLocator
```

```
lexicon = lp.LexDbOA
wsv = cache.DefaultVernWs
wsa = cache.DefaultAnalWs
```

These variables then represent these objects:

project	flexlibs.code.FLExProject.FLExProject
cache	the LCM cache, SIL.LCModel.LcmCache
lp	the LangProject object SIL.LCModel.ILangProject
sl	the Service Locator SIL.LCModel.ILcmServiceLocator
lexicon	the lexical database object SIL.LCModel.ILexDb
wsv	the top Vernacular writing system int
wsa	the top Analysis writing system int

Debugging a script can be somewhat challenging within FlexTools. When something is wrong, you’ll see something like this in the output window.



```
Running ZookPythonTest (version 1)...
Module failed with exception System.NullReferenceException! Use Ctrl-C to copy this report to the clipboard
Processing completed with 1 error and 0 warnings
```

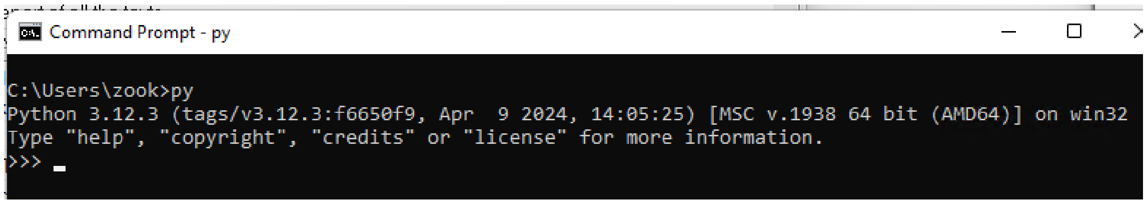
You can hover over the red line to get some error information, or use Ctrl-C and then paste it into an editor where you can see more detail. Sometimes the error report will lead you to an obvious error in your code that can be corrected to move on. Especially when starting out, it can be very challenging to know what to do about the error because there isn’t any way in FlexTools to get information on the underlying code. Using Python console in a Cmd window provides additional information that can often be helpful. This is described in the next section.

Note FlexTools has a Help > API Help menu which documents the FlexTools methods including where they are installed on your machine.

3 Using Python console in a Cmd Window

With FlexTools installed and working, it’s often helpful to run part of a Python script in a Windows Cmd window as this environment provides more help in debugging problems than you get in the FlexTools program. The Python console provides an excellent interactive way to access data from a FLEx project and allows you to make changes to the project directly from the console.

If you open a Cmd window and type py, you will get to the Python console with the >>> prompt.



```
C:\Users\zook>py
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

At this point you can type Python code, or paste a Python script. Ctrl+Z at the >>> prompt will close the Python console.

The minimal code to access FLEx code using FlexTools code is the following

```
import flexlibs
flexlibs.FLExInitialize()
project = flexlibs.FLExProject()
project.OpenProject('PythonTest')
# Additional code here
project.CloseProject()
flexlibs.FLExCleanup()
```

This works well if your code is not modifying the FLEx project. If your code needs to modify the FLEx project, the OpenProject method has an optional second parameter, that if set to True, will allow you to make changes to the FLEx project and save them when you CloseProject. The following code will open FLEx project PythonTest and add a minimal LexEntry to the project.

```
import flexlibs
cache = project.project
flexlibs.FLExInitialize()
from SIL.LCModel import *
project = flexlibs.FLExProject()
project.OpenProject('PythonTest',True)
entry = cache.ServiceLocator.GetInstance(ILexEntryFactory).Create()
project.CloseProject()
flexlibs.FLExCleanup()
```

In case you fail to include the True parameter and you want to make changes without restarting, this can be done with these steps. Before making changes you want to save, start a NonUndoable task.

```
cache.MainCacheAccessor.BeginNonUndoableTask()
```

Make whatever changes you want at this point, then give the following commands to save the changes to fwdata and close the project connection.

```
cache.MainCacheAccessor.EndNonUndoableTask()
cache.ServiceLocator.GetInstance(IUndoStackManager).Save()
project.CloseProject()
flexlibs.FLExCleanup()
```

For experimenting with the Python console, the following code can be executed after using py from the Cmd window to get into the Python console. This code will import all the modules needed for code in this document, open an existing ‘PythonTest’ project for updating FLEx, and setting common variables used in this document. If you don’t want to save any changes you were making, use Ctrl+Z to exit the console without executing CloseProject.

```
import flexlibs
from SIL.LCModel import *
from SIL.LCModel.Core.KernelInterfaces import ITsString, ITsTextProps, ITsStrFactory,
ITsPropsFactory, ITsStrBldr, ITsIncStrBldr, ITsPropsBldr, ITsMultiString, FwTextPropType,
FwTextPropVar
from SIL.LCModel.Core.Text import TsStringUtils,TsIncStrBldr, TsStrBldr, TsStringComparer
from SIL.LCModel.Core.Cellar import CellarPropertyType, GenDate
from SIL.LCModel.Core.WritingSystems import CoreWritingSystemDefinition
from System import Guid, String, DateTime, Byte
import unicodedata
flexlibs.FLExInitialize()
project = flexlibs.FLExProject()
project.OpenProject('PythonTest', True)
project.ProjectName()
```

```
cache = project.project
lp = cache.LangProject
lexicon = lp.LexDbOA
sl = cache.ServiceLocator
wsv = cache.DefaultVernWs
wsa = cache.DefaultAnalWs
```

3.1 Display a textual form of an object using print()

If you simply type a variable name and Enter, it may give useful information as shown in the first example below. At least it identifies it as a LexEntry. The print() method displays a textual form of an object, or an entire expression. In the second case below it is returning the HVO of the LexEntry.

```
>>> entry = cache.ServiceLocator.GetInstance(ILexEntryFactory).Create()
>>> entry
<SIL.LCModel.ILexEntry object at 0x000002EFF68511C0>
>>> print(entry)
LexEntry : 11330
```

The following print statement would display the headword.

```
print(entry.HeadWord)
```

3.2 Class of object using __class__

The __class__ method on a variable will display the class of the variable. For example, after creating a new entry we can verify that it worked with this command

```
>>> entry = cache.ServiceLocator.GetInstance(ILexEntryFactory).Create()
>>> entry.__class__
<class 'SIL.LCModel.ILexEntry'>
```

This confirms that the entry object we received from this method is actually an interface for a LexEntry object.

3.3 Methods available on an object using dir()

The dir() command will give the methods that are available on an object. To find out what's available on the entry object, we can use this command.

```
>>> entry = cache.ServiceLocator.GetInstance(ILexEntryFactory).Create()
>>> dir(entry)
['AddComponent', 'AllAllomorphs', 'AllOwnedObjects', 'AllReferencedObjects', 'AllSenses',
'AlternateFormsOS', 'Bibliography', 'Cache', 'CanDelete', 'ChangeRootToStem', 'CheckConstraints',
'ChooserNameTS', 'CitationForm', 'CitationFormWithAffixType', 'ClassID', 'ClassName', 'Comment',
'ComplexFormEntries', 'ComplexFormEntryRefs', 'ComplexFormsNotSubentries',
'CreateVariantEntryAndBackRef', 'DateCreated', 'DateModified', 'Delete', 'DeletionTextTSS',
'DialectLabelsRS', 'DoNotPublishInRC', 'DoNotShowMainEntryInRC', 'DoNotUseForParsing',
'EntryRefsOS', 'Equals', 'EtymologyOS', 'FindMatchingVariantEntryBackRef',
'FindMatchingVariantEntryRef', 'FindOrCreateDefaultMsa', 'GetDefaultClassForNewAllomorph',
'GetHashCode', 'GetObject', 'GetType', 'Guid', 'HasMoreThanOneSense', 'HeadWord',
'HeadWordForWs', 'HeadWordRef', 'HeadWordRefForWs', 'HeadWordReversalForWs',
'HomographForm', 'HomographFormKey', 'HomographNumber', 'Hvo', 'Id', 'ImportResidue',
'IndexInOwner', 'IsCircumfix', 'IsComponent', 'IsFieldRelevant', 'IsFieldRequired',
'IsMorphTypesMixed', 'IsOwnedBy', 'IsValidObject', 'IsVariantOfSenseOrOwnerEntry', 'LIFTid',
'LexEntryReferences', 'LexemeFormOA', 'LiftResidue', 'LiteralMeaning', 'LookupComplexEntryType',
'MLHeadWord', 'MainEntriesOrSensesRS', 'MakeVariantOf', 'MergeObject', 'MinimalLexReferences',
'MorphTypes', 'MorphoSyntaxAnalysesOC', 'MoveSenseToCopy', 'NumberOfSensesForEntry',
'ObjectIdName', 'OwnOrd', 'OwnedObjects', 'Owner', 'OwnerOfClass', 'OwningFlid',
'PicturesOfSenses', 'PostClone', 'PrimaryMorphType', 'PronunciationsOS', 'PublishAsMinorEntry',
'PublishIn', 'ReferenceTargetCandidates', 'ReferenceTargetOwner', 'ReferringObjects',
'ReplaceMoForm', 'ReplaceObsoleteMsas', 'Restrictions', 'Self', 'SenseWithMsa', 'SensesOS',
'Services', 'SetLexemeFormAlt', 'ShortName', 'ShortNameTSS', 'ShowMainEntryIn', 'SortKey',
'SortKey2', 'SortKey2Alpha', 'SortKeyWs', 'Subentries', 'SummaryDefinition',
'SupportsInflectionClasses', 'ToString', 'VariantEntryRefs', 'VariantFormEntries',
'VariantFormEntryBackRefs', 'VisibleComplexFormBackRefs', 'VisibleComplexFormEntries',
'VisibleVariantEntryRefs', '__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__',
'__ge__', '__getattribute__', '__getstate__', '__gt__', '__hash__', '__init__', '__init_subclass__',
'__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'get_AllAllomorphs', 'get_AllOwnedObjects',
'get_AllSenses', 'get_AlternateFormsOS', 'get_Bibliography', 'get_Cache', 'get_CanDelete',
'get_ChooserNameTS', 'get_CitationForm', 'get_CitationFormWithAffixType', 'get_ClassID',
'get_ClassName', 'get_Comment', 'get_ComplexFormEntries', 'get_ComplexFormEntryRefs',
'get_ComplexFormsNotSubentries', 'get_DateCreated', 'get_DateModified', 'get_DeletionTextTSS',
'get_DialectLabelsRS', 'get_DoNotPublishInRC', 'get_DoNotShowMainEntryInRC',
'get_DoNotUseForParsing', 'get_EntryRefsOS', 'get_EtymologyOS', 'get_Guid',
'get_HasMoreThanOneSense', 'get_HeadWord', 'get_HeadWordRef', 'get_HomographForm',
'get_HomographFormKey', 'get_HomographNumber', 'get_Hvo', 'get_Id', 'get_ImportResidue',
'get_IndexInOwner', 'get_IsMorphTypesMixed', 'get_IsValidObject', 'get_LIFTid',
'get_LexEntryReferences', 'get_LexemeFormOA', 'get_LiftResidue', 'get_LiteralMeaning',
'get_LookupComplexEntryType', 'get_MLHeadWord', 'get_MainEntriesOrSensesRS',
'get_MinimalLexReferences', 'get_MorphTypes', 'get_MorphoSyntaxAnalysesOC',
'get_NumberOfSensesForEntry', 'get_ObjectIdName', 'get_OwnOrd', 'get_OwnedObjects',
'get_Owner', 'get_OwningFlid', 'get_PicturesOfSenses', 'get_PrimaryMorphType',
'get_PronunciationsOS', 'get_PublishAsMinorEntry', 'get_PublishIn', 'get_ReferringObjects',
'get_Restrictions', 'get_Self', 'get_SensesOS', 'get_Services', 'get_ShortName', 'get_ShortNameTSS',
'get_ShowMainEntryIn', 'get_SortKey', 'get_SortKey2', 'get_SortKey2Alpha', 'get_SortKeyWs',
'get_Subentries', 'get_SummaryDefinition', 'get_VariantEntryRefs', 'get_VariantFormEntries',
'get_VariantFormEntryBackRefs', 'get_VisibleComplexFormBackRefs',
'get_VisibleComplexFormEntries', 'get_VisibleVariantEntryRefs', 'set_DateCreated',
'set_DateModified', 'set_DoNotUseForParsing', 'set_HomographNumber', 'set_ImportResidue',
'set_LexemeFormOA', 'set_LiftResidue']
```

There are obviously a lot of methods available on LexEntry.

3.4 Details on a method using __doc__

Suppose you see “HeadWordForWs” and would like to know what parameters are needed. The __doc__ method gives minimal information about the methods. The following example shows that the HeadWordForWs method takes an Int32 as an argument.

```
>>> entry = cache.ServiceLocator.GetInstance(ILexEntryFactory).Create()
>>> entry.HeadWordForWs.__doc__
'SIL.LCModel.Core.KernelInterfaces.ITsString HeadWordForWs(Int32)'
```

It doesn’t tell you what the Int32 represents, but it’s at least a clue. In this case the Int32 is a writing system identifier integer.

Here’s another example checking on the IsOwnedBy method, which tells us the method expects a CmObject, or subclass as a parameter.

```
>>> entry = cache.ServiceLocator.GetInstance(ILexEntryFactory).Create()
>>> entry.IsOwnedBy.__doc__
'Boolean IsOwnedBy(SIL.LCModel.ICmObject)'
```

This shows some of the responses you might get when trying to use unfamiliar methods. The first attempt fails because it is a bound method, which means it needs arguments. The second attempt with null arguments shows that it is expecting some argument, but it doesn't tell you what it's looking for. The __doc__ method answers that question that it is looking for a CmObject.

```
>>> entry = cache.ServiceLocator.GetInstance(ILexEntryFactory).Create()
>>> entry.IsOwnedBy
<bound method 'IsOwnedBy'>
>>> entry.IsOwnedBy()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: No method matches given arguments for ICmObject.IsOwnedBy: ()
>>> entry.IsOwnedBy.__doc__
'Boolean IsOwnedBy(SIL.LCModel.ICmObject)'
```

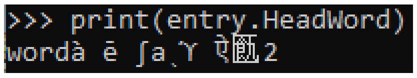
If you have an HVO and want to get information on the object, you can use code similar to this.

```
>>> entry = sl.GetObject(9409)
>>> print(entry.Guid)
ded6e683-0e6d-4569-9941-83dd203cafcd
>>> entry
<SIL.LCModel.ICmObject object at 0x0000023F7F6814C0>
>>> entry.ClassName
'LexEntry'
>>> entry = ILexEntry(entry)
>>> entry
<SIL.LCModel.ILexEntry object at 0x0000023F7F680D80>
```

You can get the CmObject using the service locator GetObject method using the HVO or a guid. If you want the guid in order to find it in fwdata or for some other purpose, you can use the Guid property. If you just type entry without anything else, it gives some basic information showing that the entry is not really an entry, but an instance of CmObject. So, at this point it only provides methods that are available on CmObject. ClassName is one method that will show what kind of CmObject it is. In this case it shows that it is a LexEntry. Knowing the class, you can cast it to ILexEntry to get an actual instance of LexEntry which then allows you to use any methods on LexEntry.

3.5 Using non-Roman scripts in Python console

A Cmd window is not an ideal environment for working with non-Roman scripts. For example, if a citation form has "wordàēʃa͕ϓऐ匭" when you print this out in the Python console, it looks like this:



```
>>> print(entry.HeadWord)
wordà ē ʃa͕ϓ ऐ匭2
```

If you copy it from the console, it actually has the correct characters: wordàēʃa͕ϓऐ匭2.

From the Python console, you can also paste in Unicode characters and it will work properly even though the display is limited.

```
>>> entry.CitationForm.set_String(wsv, "൭ꝏ 💗 wordàēʃạ́Ƴऐ匭")
>>> print(entry.HeadWord)
൭ꝏ 💗 wordàēʃạ́Ƴऐ匭
```

When entering data in the console you can also use the Unicode values (e.g., a\u03b2b will result in aβb. \u is used for 4-hex-digit Unicode values, and \U is used for 8-hex-digit values for the upper planes of Unicode.

4 Working with basic FLEx properties

4.1 Strings

FieldWorks uses various methods for storing strings on objects, depending on the amount of information it needs to store. See section 2.3.1 in https://downloads.languagetechnology.org/fieldworks/Documentation/FLEx%209.1%20Conceptual%20Model.pdf for more detail on this.

Note that internally, FieldWorks stores strings in Unicode NFD (normalization decomposed) format. When you get a string from the database using FlexTools, it will be in NFD, so your Python program will need to handle this properly. When storing strings from FlexTools, it would be best to convert it to NFD. FieldWorks normally converts all data to NFC when it goes into the clipboard or is saved to disk, but when it is loaded from fwdata, imported from other files, or a person is typing or pasting, it converts the data to NFD to be consistent with what's in memory. But FlexTools is working directly on the internal data. If you write NFC data, it will stay that way until FLEx closes the project and reopens it. When reopened, it will be changed to NFD internally along with all other strings.

You can convert a string to NFD in Python using this code which converts NFC (61 e0 62) to NFD (61 61 300 62). The NFC a with grave accent (e0) is changed to NFD 'a' with a combining acute accent (61 300).

```
s = unicodedata.normalize('NFD', 'aàb')
```

4.1.1 FieldWorks strings

There are two basic types of strings in FieldWorks. The names are somewhat misleading because all data is in Unicode. But in this context, Unicode means a Unicode string without any embedding, and String is a Unicode string that allows embedding. Then both of these types can be in a multi property that allows multiple strings in different writing system alternatives.

4.1.2 Unicode

The LiftResidue property on LexEntry is an example of a Unicode string. Assuming 'entry' is a LexEntry object, this code retrieves the string from LiftResidue:

```
lres = entry.LiftResidue
```

This sets LiftResidue to ‘abc’:

```
entry.LiftResidue = 'abc'
```

4.1.3 String

The ImportResidue property on LexEntry is an example of a String string. Assuming ‘entry’ is a LexEntry object, this code retrieves the string from ImportResidue:

```
ires = entry.ImportResidue
```

In this case, ires is really an ITsString. If you need a simple string, you can use this command:

```
ires = entry.ImportResidue.Text
```

To set a String property, you need to have a TsString that includes text and a writing system. This is a simple way to set ImportResidue to ‘Simple string’ in the first analysis writing system.

```
entry.ImportResidue = TsStringUtils.MakeString('Simple string.', wsa)
```

4.1.4 MultiUnicode

The Form property of the MoForm owned by the LexemeForm property of LexEntry is an example of a MultiUnicode string. Assuming ‘entry’ is an entry object and ‘wsv’ is the first vernacular writing system, this command retrieves the string from the lexeme form:

```
lf = entry.LexemeFormOA.Form.get_String(wsv)
```

Note that get_String returns an ITsString. To get a plain string from this, you may need to append the .Text method. This is needed when you are searching for a string. For example,

```
if entry.LexemeFormOA.Form.get_String(wsv).Text == "maison":
```

MultiUnicode and MultiString provide an optional way to specify a writing system other than specifying an actual writing system. These options are available for a property that may have multiple writing systems with data.

Method name	Description
BestAnalysisAlternative	The first analysis writing system, or if not present, any other analysis alternative.
BestVernacularAlternative	The first vernacular writing system, or if not present, any other vernacular alternative.
BestAnalysisVernacularAlternative	The best analysis writing system, or if not present, the best vernacular writing system
BestVernacularAnalysisAlternative	The best vernacular writing system, or if not present, the best analysis writing system

This is an example using one of these methods to return a string for the lexeme form.

```
lf = entry.LexemeFormOA.Form.BestVernacularAlternative
```

CitationForm on LexEntry is also MultiUnicode. This sets the CitationForm to ‘manger’.

```
entry.CitationForm.set_String(wsv, 'manger')
```

4.1.5 MultiString

The Comment property on LexEntry is an example of a MultiString property. Assuming ‘entry’ is an entry object and ‘wsa’ is the analysis writing system, this command retrieves an ITsString from the specified alternative of the property:

```
note = entry.Comment.get_String(wsa)
```

This will set the Comment to ‘comment’ in the wsa alternative.

```
entry.Comment.set_String(wsa, 'comment')
```

4.1.6 TsString

This section gives an introduction for working with TsStrings. There are additional methods that are not discussed here.

4.1.6.1 TsString class

TsString is the class used for a FieldWorks String that allows embedding. To work with these strings, you will need to import appropriate classes and methods if you don’t use the default imports listed in section 2.

```
from SIL.LCModel.Core.KernelInterfaces import ITsString, ITsTextProps, ITsStrFactory,
ITsPropsFactory, ITsStrBldr, ITsIncStrBldr, ITsPropsBldr, ITsMultiString, FwTextPropType,
FwTextPropVar
from SIL.LCModel.Core.Text import TsStringUtils,TsIncStrBldr, TsStrBldr
```

If you just want a simple TsString string in a single writing system, you can use this method

```
tsString = TsStringUtils.MakeString('Simple string.', wsa)
```

If you need to get a simple Unicode string from a TsString ignoring writing systems, styles, etc., you can use the Text method.

```
str = tsString.Text
```

The Length method gives the number of characters in the TsString.

```
chars = tsString.Length
```

The number of runs can be returned with RunCount.

```
runs = tsString.RunCount
```

The GetSubstring method on TsString will return a TsString using a begin and end offset into the source string. Using the above tsString, this command would set sub to a TsString containing ‘string’.

```
sub = tsString.GetSubstring(7, 13)
```

4.1.6.2 Incremental string builder

Here is an example using an incremental string builder to make a more complex string, “A small round object. *mots français*. The rest of the string.” where the italic portion is in the vernacular writing system and uses the Emphasized Text style. In FLEx fwdata, this will result in

```
<Bibliography>
<AStr ws="en">
<Run ws="en">A small round object. </Run>
```

```
<Run namedStyle="Emphasized Text" ws="fr">mots français.</Run>
<Run ws="en"> The rest of the string.</Run>
</AStr>
</Bibliography>
```

Use TsStringUtils to get a TsIncStrBldr. Then use these steps to build the string.

1. Use the SetIntPropValues method on the builder to set a ktptWs type with a value of the wsa (en) writing system.
2. Append the portion of the string that is in English.
3. Use the SetIntPropValues method to set a ktptWs type with a value of the wsv (fr) writing system
4. Use the SetStrPropValue method to set type ktptNamedStyle and value "Emphasized Text", which is the name of a style that will display italic.
5. Append the French string that will be in the French writing system and italic style.
6. Use SetIntPropValues to set the writing system back to analysis.
7. SetStrPropValue to set the style to None to cancel the Emphasized Text style
8. Append the rest of the English string.
9. Use GetString to get the TsString from the builder and store it in the Bibliography property of entry.

```
tisb = TsStringUtils.MakeIncStrBldr()
tisb.SetIntPropValues(FwTextPropType.ktptWs.value__, FwTextPropVar.ktpvDefault.value__, wsa)
tisb.Append("A small round object. ")
tisb.SetIntPropValues(FwTextPropType.ktptWs.value__, FwTextPropVar.ktpvDefault.value__, wsv)
tisb.SetStrPropValue(FwTextPropType.ktptNamedStyle.value__, "Emphasized Text")
tisb.Append("mots français.")
tisb.SetIntPropValues(FwTextPropType.ktptWs.value__, FwTextPropVar.ktpvDefault.value__, wsa)
tisb.SetStrPropValue(FwTextPropType.ktptNamedStyle.value__, None)
tisb.Append(" The rest of the string.")
tisbout = tisb.GetString()
entry.Bibliography.set_String(wsa, tisbout)
```

The FwTextPropType enums are defined in https://github.com/sillsdev/liblcm/blob/master/src/SIL.LCModel.Core/KernelInterfaces/TextServ.idh. The two types described above are typical ones needed in FLEx data.

TsIncStrBldr also provides an AppendTsString() method that appends an existing TsString to the current string. The following method would add a second copy of the string to tisb.

```
tisb.AppendTsString(tisbout)
```

4.1.6.3 String builder

There is also a TsStrBldr that allows you to modify an existing TsString. You can get a TsStrBldr for an existing TsString using GetBldr(). Any changes you make will not affect the input string, but will be available as a new TsString using the GetString() method.

```
tsb = tisbout.GetBldr()
```

TsStrBldr has a Clear() method that will remove anything from the builder. Methods on TsStrBldr are defined in TextServ.idh in liblcm\src\SIL.LCModel.Core\KernelInterfaces. There are methods for mapping between character indexes and run indexes, methods to fetch characters, methods to get properties, and methods to modify the current state. An example is ReplaceTsString method taking 3 parameters, ichMin, ichLim, and a TsString. It will replace the characters from index ichMin up to, but not including the character at index ichLim with the contents of the TsString. Here's an example that replaces French 'mots' (words) with French 'plusieurs mots' (many words) from the tisbout string from above to tsbout:

```
newfr = TsStringUtils.MakeString('plusieurs mots', wsv)
tsb.ReplaceTsString(22, 26, newfr)
tsbout = tsb.GetString()
```

If the third parameter is None, it deletes the characters between this range.

4.1.6.4 Comparing TsStrings

The Equals method will return whether two TsStrings are equal or not. The following method sets eq to False since these two strings are not equal.

```
eq = tsbout.Equals(tisbout)
```

Normal Python compare methods may not give the correct results because they are ignoring the collation on the FLEx writing system. In order to compare two TsStrings, we need to use the Compare method on a TsStringComparer. The Compare method takes two TsString parameters. If the first string is less than the second string, it returns -1. If the strings are equal, it returns 0. If the first string is greater than the second string, it returns 1.

There are two ways to get a TsStringComparer. The first approach uses a .NET comparer that does not pay attention to language.

```
netCmpr = TsStringComparer()
netCmpr.Compare(tsstring1, tsstring2)
```

To pay attention to language, TsStringComparer needs to have a CoreWritingSystemDefinition argument. To get a CoreWritingSystemDefinition for a standard language, you can use a language id as in this example

```
en = CoreWritingSystemDefinition('en')
encmp = TsStringComparer(en)
encmp.Compare(tsstring1, tsstring2)
```

In order to compare using a custom collation from a FLEx writing system in the project, you need to use this approach to get a CoreWritingSystemDefinition.

```
daws = sl.WritingSystems.DefaultAnalysisWritingSystem
dawscmp = TsStringComparer(daws)
dawscmp.Compare(tsstring1, tsstring2)
```

DefaultVernacularWritingSystem will also work. sl.WritingSystems.AllWritingSystems will return a list of CoreWritingSystemDefinition for all writing systems in the project.

4.1.6.5 Finding substrings in TsStrings

TsStringUtils.FindTextInString provides a way to find the first substring in a target string. FindTextInString has 6 parameters. The first is the TsString to search for. The second is the TsString source string to search. The third is a WritingSystemFactory. The fourth is a Boolean MatchWholeWord. If True, it will only find the search string if it is a complete word. If False, it will find the search string anywhere. The fifth and sixth parameters are an output integer for returning the begin and end index of the location in the source string. However, with Python, it doesn't use this, so use 0 for both of these.

The method returns an array with 3 values. The first is True if the search text was found in the source. In this case, the second is the index where the search test started, and the third is the index of the first letter following the search text. The first is False if a match was not found, and the second digits are not significant. Case is significant, but writing system is not significant.

This example from the Python console shows the results with different sources and settings. It demonstrates that case is significant, writing system is not significant, when MatchWholeWord is true, it only matches complete words, when false, it matches any location in the source.

```
>>> round = TsStringUtils.MakeString('round', wsa)
>>> wsf = cache.WritingSystemFactory
>>> source = TsStringUtils.MakeString('round rounding around boy', wsv)
>>> TsStringUtils.FindTextInString(round, source, wsf, True, 0, 0)
(True, 0, 5)
>>> source = TsStringUtils.MakeString('round rounding around boy', wsa)
>>> TsStringUtils.FindTextInString(round, source, wsf, True, 0, 0)
(True, 0, 5)
>>> source = TsStringUtils.MakeString('round rounding around boy', wsa)
>>> TsStringUtils.FindTextInString(round, source, wsf, False, 0, 0)
(True, 0, 5)
>>> source = TsStringUtils.MakeString('Round round boy', wsa)
>>> TsStringUtils.FindTextInString(round, source, wsf, True, 0, 0)
(True, 6, 11)
>>> source = TsStringUtils.MakeString('Round round boy', wsa)
>>> TsStringUtils.FindTextInString(round, source, wsf, False, 0, 0)
(True, 6, 11)
>>> source = TsStringUtils.MakeString('around boy', wsa)
>>> TsStringUtils.FindTextInString(round, source, wsf, True, 0, 0)
(False, 0, 0)
>>> source = TsStringUtils.MakeString('around boy', wsa)
>>> TsStringUtils.FindTextInString(round, source, wsf, False, 0, 0)
(True, 1, 6)
>>> source = TsStringUtils.MakeString('rounding boy', wsa)
>>> TsStringUtils.FindTextInString(round, source, wsf, True, 0, 0)
(False, 0, 0)
>>> source = TsStringUtils.MakeString('rounding boy', wsa)
>>> TsStringUtils.FindTextInString(round, source, wsf, False, 0, 0)
(True, 0, 5)
>>> source = TsStringUtils.MakeString('carerounding boy', wsa)
>>> TsStringUtils.FindTextInString(round, source, wsf, True, 0, 0)
(False, 0, 0)
>>> source = TsStringUtils.MakeString('carerounding boy', wsa)
>>> TsStringUtils.FindTextInString(round, source, wsf, False, 0, 0)
(True, 4, 9)
```

4.2 Other basic properties

4.2.1 Booleans

The DoNotUseForParsing property on LexEntry is an example of a Boolean property. Assuming ‘entry’ is an entry object, this command retrieves the Boolean result:

```
bool = entry.DoNotUseForParsing
```

This sets the Boolean:

```
entry.DoNotUseForParsing = True
```

4.2.2 Integers

The HomographNumber property on LexEntry is an example of an integer property. Assuming ‘entry’ is an entry object, this command retrieves the homograph number:

```
hom = entry.HomographNumber
```

This sets the homograph number:

```
entry.HomographNumber = 2
```

4.2.3 Time

The DateCreated property on LexEntry is an example of a time property. Assuming ‘entry’ is an entry object, this command retrieves the create time:

```
dt = entry.DateCreated
```

The time returned here is a .NET DateTime object. Its methods are available when you execute the Python ‘from System import *’ command included in section 2.

This sets the create date: the first to the current time and the second to any specified time:

```
entry.DateCreated = DateTime.Now
entry.DateModified = DateTime.Parse("11/7/2006 12:45:58 PM")
```

4.2.4 GUIDs

The Guid property on LexEntry is an example of a GUID property. Assuming ‘entry’ is an entry object, this command retrieves the GUID:

```
guid = entry.Guid
```

The GUID returned here is a .NET GUID object. Its methods are available when you execute the Python ‘from System import *’ command included in section 2.

This sets the GUID: the first to a new GUID and the second to any specified instance:

```
entry.Guid = Guid.NewGuid()
entry.Guid = Guid("edef982a-f69a-4793-95fb-f4398e4a2ddf")
```

4.2.5 GenDates

The DateOfEvent property on RnGenericRec is an example of a GenDate. Assuming rnRec is a generic record object, this command retrieves the GenDate:

```
genDate = rnRec.DateOfEvent
```

This sets genDate to “About June 4, 2024” and then sets the DateOfEvent to that value.

```
genDate = GenDate(202406042)
rnRec.DateOfEvent = genDate
```

4.2.6 Binary

Sid on UserConfigAcct is an example of a Binary property, although it is not used in current versions of FLEx. Assuming uca is a UserConfigAcct object, this command retrieves the Sid Binary data

```
bytes = uca.Sid
```

This returns a System.Byte[] array, however, this needs further research as I was unable to get valid results when testing.

5 Using Factories

Objects in FLEx are created using Factory classes from the ServiceLocator in the LCM Cache. FlexTools typically uses project.project.ServiceLocator to get access to the service locator. In this document, I use sl defined in section 2.

The ServiceLocator has a GetInstance method that will return a factory for each FieldWorks class. Most factories have a Create method without any arguments that can be used to create an instance of the class. This code will create a new LexEntry object

```
entry = sl.GetInstance(ILexEntryFactory).Create()
```

Most FieldWorks objects are in an ownership hierarchy. After creating an object, it needs to be added to the desired field on the owner. LexEntry is one of the few classes that do not have owners. Senses are owned by entries, so if we want to add a sense to the entry, we first create the LexSense object using this code

```
sense = sl.GetInstance(ILexSenseFactory).Create()
```

Before filling in properties on a newly created object, it is best to immediately attach the new object to its owner. If you don't do this, some properties cannot be set since they depend on information from their owner. Setting the PartOfSpeech on MoStemMsa is an example of this. This is the code that adds the sense to the Senses owning sequence property of LexEntry.

```
entry.SensesOS.Add(sense)
```

The full list of classes that do not have owners is: LangProject, LexEntry, PunctuationForm, ScrRefSystem, Text, VirtualOrdering, and WfiWordform. CmPicture and CmPossibilityList may or may not have owners.

There are several class factories that do not have a Create() method without any arguments. In the model spreadsheet (https://downloads.languagetechnology.org/fieldworks/Documentation/MasterFieldWorksModel%20classes%20and%20fields%207000072.xlsx), these classes have 'false' in the GenCreate column. This includes these classes:

```
CmTranslation
Scripture
ScrBook
ScrRefSystem
ScrDraft
ScrTxtPara
```

For these classes you need to check parameters in overrides in both the factory and the class overrides code in the LCM library. An example of this is the CmTranslation object that is owned in Translations properties on LexExampleSentence and StTxtPara. The Create method for CmTranslation needs to include the owning object as well as the item from the Translations Type list when it is created. In this process the Create method will both add the CmTranslation to the owner and set the Type property on the CmTranslation.

6 Using Repositories

FLEx has a repository for each class of object that has a list of all of the instances of that class in the project. The repositories can be accessed from the ServiceLocator in the LCM Cache. FlexTools typically uses project.project.ServiceLocator to get access to the service locator. In this document, I use sl defined in section 2.

The ServiceLocator has a GetInstance method that will return a repository for each FieldWorks class. If you want to get a single object with a known guid or hvo, you can use the class repository to get the object. This example uses the ILexEntry repository to get an entry based on the guid or the hvo.

```
lexRepo = sl.GetInstance(ILexEntryRepository)
entry = lexRepo.GetObject(Guid("7b184e6c-3f51-4c9e-8a2c-501d7bb973d8"))
entry = lexRepo.GetObject(5142)
```

You can get the number of objects in the repository by using the Count method

```
num = lexRepo.Count
```

This is one way to get the first item from the repository using a for loop:

```
for e in lexRepo.AllInstances():
    entry = e
    break
```

If you want to print the first 5 entries, you could use this code:

```
i = 0
for e in lexRepo.AllInstances():
    i += 1
    print(e)
    if i >= 5:
        break
```

Here is one line to get one object from any repository starting with the ServiceLocator. This returns a LexEntry, but by changing the class name for the repository, you could get one for any class.

```
entry = next(iter(sl.GetInstance(ILexEntryRepository).AllInstances()))
```

If you want to get several instances you could use this:

```
it = iter(sl.GetInstance(ILexEntryRepository).AllInstances())
e1 = next(it)
e2 = next(it)
e3 = next(it)
```

Here is one way to get a possibility item (MorphType) with a given name using a repository. This assumes the list does not have duplicate names. If the item is not found, ‘stem’ will remain ‘None’.

```
morphTypeRepo = sl.GetInstance(IMorphTypeRepository)
stem = None
for m in morphTypeRepo.AllInstances():
   if m.Name.get_String(wsa).Text == "stem":
      stem = m
      break
```

There is a simpler way to do this using the FindPossibilityByName method on CmPossibilityList.

```
plist = lexicon.MorphTypesOA
stem = plist.FindPossibilityByName(plist.PossibilitiesOS, 'stem', wsa)
```

7 Using Atomic owning and reference properties

To add an object to an atomic owning property we use this syntax. In this case, LexemeForm on LexEntry is an atomic owning property. This code creates a MoStemAllomorph object and adds it to the LexemeForm of the entry.

```
stemAllo = sl.GetInstance(IMoStemAllomorphFactory).Create()
entry.LexemeFormOA = stemAllo
```

To access an object from an atomic property, you use this syntax. This would get the MoForm object owned in the atomic LexemeForm property of LexEntry.

```
allo = entry.LexemeFormOA
```

In the Python console you can see how this works. It initially sets allo to an IMoForm as demonstrated with the __class__ method. You can get the exact class by using the ClassName method. Once you know the class name, you can cast the object to an instance of that class. At that point all of the methods on MoStemAllomorph would be available.

```
allo = entry.LexemeFormOA
>>> allo.__class__
<class 'SIL.LCModel.IMoForm'>
>>> allo.ClassName
‘MoStemAllomorph’
>>> allo = IMoStemAllomorph(allo)
>>> allo.__class__
<class 'SIL.LCModel.IMoStemAllomorph'>
```

If you just want to get the LexemeForm string, you can use this code

```
lexform = entry.LexemeFormOA.Form.get_String(wsv).Text
```

Atomic reference properties are accessed in the same way, but the property would have RA in the property name.

The following would get the object from the MorphoSyntaxAnalysis atomic reference property of LexSense.

```
stemMsa = sense.MorphoSyntaxAnalysisRA
```

The following would set the MorphoSyntaxAnalysis atomic reference property on LexSense to the stemMsa object.

```
sense.MorphoSyntaxAnalysisRA = stemMsa
```

The following line would remove the MoStemAllomorph from the LexemeForm atomic owning property. This would delete the MoStemAllomorph from the project.

```
entry.LexemeFormOA = None
```

8 Using Sequence owning and reference properties

The Add method will add an object to a sequence owning property. The following example creates a LexSense object and adds it to the Senses owning sequence property of the entry.

```
sense = sl.GetInstance(ILexSenseFactory).Create()
entry.SensesOS.Add(sense)
```

For sequence properties, the Insert method will let you add an object at a specific location in a sequence. It has two parameters: the first is an integer offset into the sequence (0 inserts before the first item, etc.) The second parameter is the object you are inserting.

In this example the entry already has 2 senses. This creates a new sense and inserts it between the original senses and gives it a gloss of “new 2nd sense”.

```
sense = sl.GetInstance(ILexSenseFactory).Create()
entry.SensesOS.Insert(1, sense)
sense.Gloss.set_String(wsa, 'new 2nd sense')
```

The MoveTo method allows you to move one or more items in a sequence to a new location. The first parameter is the index of the first item in the list to move. The second parameter is the index of the last item in the list to move. The third parameter is the target sequence. The fourth parameter is the starting index in the target list to which you want to move the item(s).

For an entry with 3 senses, this code would switch the order of the second and third senses (e.g., original 1, 2, 3, and result 1, 3, 2).

```
entry.SensesOS.MoveTo(1,1,entry.SensesOS,3)
```

In sequence properties, there are methods that allow you to remove items from the sequences.

The Remove method will remove a specific object from the sequence. It has one parameter which is the object you want to remove from the sequence.

Given an entry with 2 senses, this code will remove the first sense. It returns True because it succeeded in removing the sense.

```
sense = entry.SensesOS[0]
success = entry.SensesOS.Remove(sense)
print(success)
True
```

The RemoveAt method will remove one item from the sequence. It has one parameter which is an integer offset into the sequence (0 would remove the first item).

Given an entry with 3 senses, this line will remove the middle entry.

```
entry.SensesOS.RemoveAt(1)
```

You can remove everything from a sequence account using the Clear method.

```
entry.SensesOS.Clear()
```

When items are removed from an owning sequence, they are deleted from the project.

One way to get a single entry would be to use the Entries method on LexDb. This is virtual since dictionary entries do not have owners, so it doesn’t work the same as an actual sequence owning property.

```
for e in lexicon.Entries:
   entry = e
   break
```

You can get one entry without the for loop using this command.

```
entry = next(iter(lexicon.Entries))
```

To get one entry at a time, you could use this:

```
iter = iter(lexicon.Entries)
e1 = next(iter)
e2 = next(iter)
```

Possibility lists have a Possibilities owning sequence property. The following code will get the Parts Of Speech possibility list from LangProject and gets the number of PartOfSpeech objects at the top level. Then, since this is an owning sequence, we can use an index to access the first PartOfSpeech object.

```
poslist = lp.PartsOfSpeechOA
num = poslist.PossibilitiesOS.Count
pos = poslist.PossibilitiesOS[0]
```

This code is one way to print the names of the items

```
for p in poslist.PossibilitiesOS:
  print(p)
```

PossibilityList has a method that will return all of the possibilities including nested ones. This code will print all of these:

```
for p in poslist.ReallyReallyAllPossibilities:
   print(p)
```

PossibilityList has a FindPossibilityByName method that will return an item from the list in a specified writing system with a specified name or abbreviation where. case is significant. If an item is not found, the result will be None. In addition to flat lists, this will recurse to subitems as well.

This example sets stem to the ‘stem’ IMorphType item from the Morph Types list using the default analysis writing system.

```
plist = lexicon.MorphTypesOA
stem = plist.FindPossibilityByName(plist.PossibilitiesOS, 'stem', wsa)
```

For a sequence property, you can use an index to access an object in the sequence. This example returns the second sense in entry.

```
sense = entry.SensesOS[1]
```

9 Using Collection owning and reference properties

The Add method will add an object to a collection owning property. The following example creates a MoStemMsa object and adds it to the MorphoSyntaxAnalyses owning collection property of the entry.

```
msa = sl.GetInstance(IMoStemMsaFactory).Create()
entry.MorphoSyntaxAnalysesOC.Add(msa)
```

There are no Insert methods for collections since these properties have no inherent order.

The following code will get the number of styles in an owning collection

```
num = lp.StylesOC.Count
```

The StylesOC property on LangProject is a collection property, so it doesn't accept an index to get a specific item. However, it provides a ToArray method that can be indexed. So this code will pick up the 'first' style, but keep in mind there really isn't a first style since they are in a collection, although in fwdata, there is an order that the objsur elements are stored, so this method will use that order.

```
style = lp.StylesOC.ToArray()[0]
```

Collections can still be processed with a 'for' loop. The following code will print all of the styles.

```
for s in lp.StylesOC:
   print(s)
```

This method would just print the first style:

```
for s in lp.StylesOC:
    print(s)
    break
```

You can remove everything from a collection property using the Clear method.

```
entry.MorphoSyntaxAnalysesOC.Clear()
```

You can remove a single object from a collection property using the Remove method. It has a single argument, which is the object you are wanting to remove. This method returns True if it found and removed the object, or False, otherwise. The following removes the msa MoMorphoSyntaxAnalysis object from the MorphoSyntaxAnalyses reference property of LexEntry.

```
entry.MorphoSyntaxAnalysesOC.Remove(msa)
True
```

When removing an object from a reference property, the object is not deleted from the project since this is only a reference, and not an owning property.

Adding to reference properties is similar to owning properties. When adding, you need to add the actual object to the property, not just a guid or hvo. For example, the following code shows two ways to add a semantic domain (1.6 – Animal) to a sense. The first sample uses the FindPossibilityByName method to find the semantic domain with abbreviation 1.6 ('Animal' would also work) and then it adds the item to the SemanticDomains reference collection of LexSense. The second sample gets a SemanticDomain object using the GetInstance on the

ServiceLocator and then using the GetObject method to get an item using the guid for the item, then it can add that to the sense. For most factory lists, the guids are identical across projects. Both of these samples would produce the same result.

```
plist = lp.SemanticDomainListOA
sense.SemanticDomainsRC.Add(plist.FindPossibilityByName(plist.PossibilitiesOS, '1.6', wsa))

semdomRepo = sl.GetInstance(ICmSemanticDomainRepository)
semdom = semdomRepo.GetObject(Guid("944cf5af-469e-4b03-878f-a05d34b0d9f6"))
sense.SemanticDomainsRC.Add(semdom)
```

10 Additional object methods

10.1 Merging objects

There is a generic MergeObject method on CmObject which may work on most classes. It works on LexEntry and LexSense, but not on RnGenericRec.

```
void MergeObject(ICmObject objSrc, bool fLoseNoStringData);
```

MergeObject takes two parameters. The first is the source object that will be deleted after the merge. The second is a Boolean fLoseNoStringData. This merges the source object into this destination object. If fLoseNoStringData is false: For atomic properties, if this object has something in the property, the source property is ignored. For sequence properties, the objects in the source will be moved and appended to the properties in this object. Any references to the source object will be transferred to this object. The source object is deleted at the end of this method. String properties are copied from the source if the destination (this) has no value and the source has a value. if fLoseNoStringData is true, the above is modified as follows: 1. If a string property has a value in both source and destination, and the values are different, append the source onto the destination. 2. If an atomic object property has a value in both source and destination, recursively merge the value in the source with the value in the destination.

Given two ILexEntry objects, this method will merge en2 (source) into en1 (destination), and en2 is deleted. True means it will try not to lose any strings, which may result in appended strings.

```
en1.MergeObject(en2, True)
```

10.2 Equal objects

The Equals method can be used between any two objects. It will only return True if the two objects are the same object (e.g., the Hvo's match).

```
en1.Equals(en2)
```

This would return False, assuming en1 and en2 are not the same object. This method would presumably be useful for iterating through a sequence to find the object in a sequence.

10.3 Deleting objects

To delete an object, use the Delete method

```
entry.Delete()
```

This deletes the entry, along with everything it owns and removes references to any of the deleted objects. In most cases this is sufficient. For some things when you delete one object, some other object(s) should also be deleted since they are no longer needed.

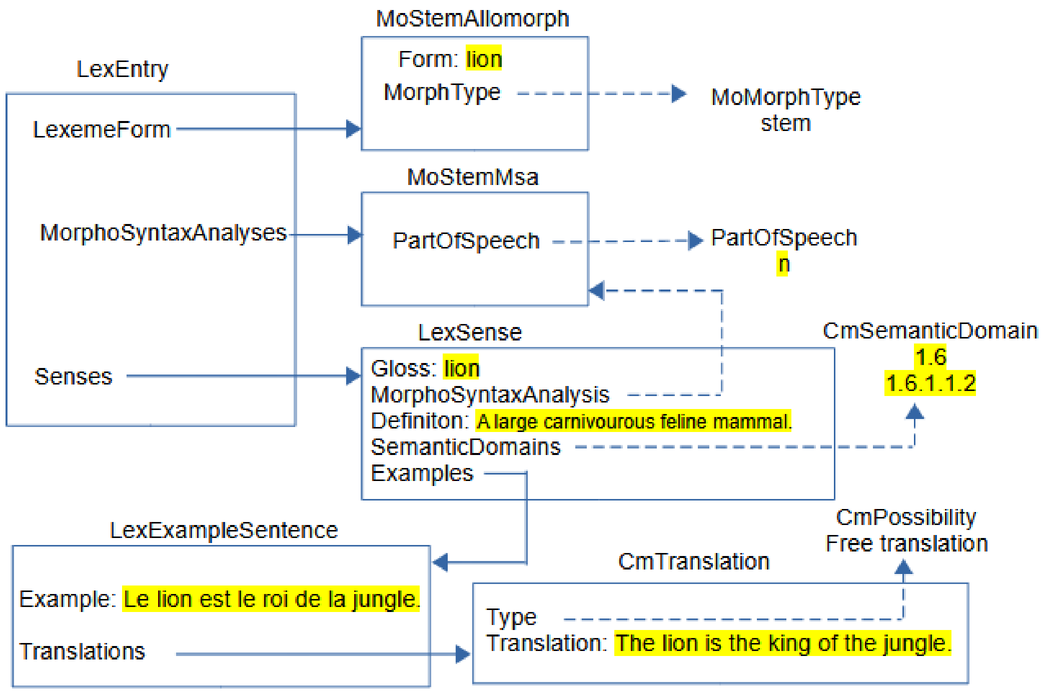
When you use a method to remove an object from an owning property, the removed object will be deleted from the project.

11 Building a FLEx entry using Python

This example demonstrates how you can add the following French entry to FLEx.

> **lion** *n* A large carnivorous feline mammal. *Le lion est le roi de la jungle.* The lion is the king of the jungle. (sem. domains: 1.6.1.1.2 - Carnivore, 1.6 - Animal.)

This diagram illustrates the classes and properties that will be created in this section.



The following Python code is one way to implement the entry represented by these classes.

This assumes that sl has been set to the ServiceLocator in the LCM Cache, lp for the LangProject class, and lexicon for the LexDb class as in section 2

When a class is created, it’s best to place it in the owning property of its owner before adding other properties to the new class. Some of the properties (e.g., the PartOfSpeech for MoStemMsa) will give an error if you attempt to assign a value to it before it has an owner, presumably because it needs to check something in the owner as part of the setting process.

```
#### Create entry
entry = sl.GetInstance(ILexEntryFactory).Create()
#### Create MoStemAllomorph and add to entry LexemeForm
stemAllo = sl.GetInstance(IMoStemAllomorphFactory).Create()
entry.LexemeFormOA = stemAllo
stemAllo.Form.set_String(wsv, 'lion')
```

```
plist = lexicon.MorphTypesOA
stemAllo.MorphTypeRA = plist.FindPossibilityByName(plist.PossibilitiesOS, 'stem', wsa)
#### Create MoStemMsa and add to entry MorphoSyntaxAnalyses
stemMsa = sl.GetInstance(IMoStemMsaFactory).Create()
entry.MorphoSyntaxAnalysesOC.Add(stemMsa)
plist = lp.PartsOfSpeechOA
stemMsa.PartOfSpeechRA = plist.FindPossibilityByName(plist.PossibilitiesOS, 'Noun', wsa)
#### Create sense and add to entry Senses
sense = sl.GetInstance(ILexSenseFactory).Create()
entry.SensesOS.Add(sense)
sense.Gloss.set_String(wsa, 'lion')
sense.Definition.set_String(wsa, 'A large carnivorous feline mammal.')
sense.MorphoSyntaxAnalysisRA = stemMsa
plist = lp.SemanticDomainListOA
sense.SemanticDomainsRC.Add(plist.FindPossibilityByName(plist.PossibilitiesOS, '1.6', wsa))
sense.SemanticDomainsRC.Add(plist.FindPossibilityByName(plist.PossibilitiesOS, '1.6.1.1.2', wsa))
#### Create example sentence and add to sense Examples
example = sl.GetInstance(ILexExampleSentenceFactory).Create()
sense.ExamplesOS.Add(example)
example.Example.set_String(wsv, 'Le lion est le roi de la jungle.')
#### Create example translation  and add to example Translations
plist = lp.TranslationTagsOA
freeTrans = plist.FindPossibilityByName(plist.PossibilitiesOS, 'Free translation', wsa)
translation = sl.GetInstance(ICmTranslationFactory).Create(example, freeTrans)
translation.Translation.set_String(wsa, 'The lion is the king of the jungle.')
```

This summarizes what happens in each section of this code and shows the resulting data in the FLEx fwdata file.

The first step is to create the LexEntry which is the top of the ownership hierarchy for the entry. We use the ILexEntryFactory to create this LexEntry.

```
entry = sl.GetInstance(ILexEntryFactory).Create()
```

At this point we just have the LexEntry class without the three owned objects that will be added in the following steps.

```
<rt class="LexEntry" guid="58ec354f-9013-4aca-b20f-9e3f999e1b96">
<DateCreated val="2024-05-23 14:54:36.285" />
<DateModified val="2024-05-23 14:54:36.285" />
<DoNotUseForParsing val="False" />
<HomographNumber val="0" />
<LexemeForm>
<objsur guid="856fd1a9-644d-4b60-9611-35dcb3d64d18" t="o" />
</LexemeForm>
<MorphoSyntaxAnalyses>
<objsur guid="c710bbdc-11f7-46be-96c2-e81674138eed" t="o" />
</MorphoSyntaxAnalyses>
<Senses>
<objsur guid="933e76a4-6ded-47df-81ca-581f4937c08a" t="o" />
</Senses>
</rt>
```

The next step is to add a MoStemAllomorph to the entry. We use the IMoStemAllomorphFactory to create the object, then store it in the LexemeForm owning property of the LexEntry. Then we can store the lexeme form text in the vernacular alternative of the Form property of the MoStemAllomorph. To set the MorphType, we use the

FindPossibilityByName method on the Morph Types list to get the ‘stem’ MorphType, and then store it in the MorphType atomic reference property.

```
stemAllo = sl.GetInstance(IMoStemAllomorphFactory).Create()
entry.LexemeFormOA = stemAllo
stemAllo.Form.set_String(wsv, 'lion')
plist = lexicon.MorphTypesOA
stemAllo.MorphTypeRA = plist.FindPossibilityByName(plist.PossibilitiesOS, 'stem', wsa)
```

```
<rt class="MoStemAllomorph" guid="856fd1a9-644d-4b60-9611-35dcb3d64d18"
ownerguid="58ec354f-9013-4aca-b20f-9e3f999e1b96">
<Form>
<AUni ws="fr">lion</AUni>
</Form>
<IsAbstract val="False" />
<MorphType>
<objsur guid="d7f713e8-e8cf-11d3-9764-00c04f186933" t="r" />
</MorphType>
</rt>
```

The next step is to add a MoStemMsa object to the entry. We use the IMoStemMsaFactory to create the object, and then add it to the MorphoSyntaxAnalyses owning property of LexEntry. To set the PartOfSpeech reference property we use the FindPossibilityByName method on the Parts of Speech list to get the ‘Noun’ item, then store a reference to the item in the PartOfSpeech reference atomic property.

```
stemMsa = sl.GetInstance(IMoStemMsaFactory).Create()
entry.MorphoSyntaxAnalysesOC.Add(stemMsa)
plist = lp.PartsOfSpeechOA
stemMsa.PartOfSpeechRA = plist.FindPossibilityByName(plist.PossibilitiesOS, 'Noun', wsa)
```

```
<rt class="MoStemMsa" guid="c710bbdc-11f7-46be-96c2-e81674138eed" ownerguid="58ec354f-
9013-4aca-b20f-9e3f999e1b96">
<PartOfSpeech>
<objsur guid="a8e41fd3-e343-4c7c-aa05-01ea3dd5cfb5" t="r" />
</PartOfSpeech>
</rt>
```

The next step is to add a sense to the entry. We use the ILexSenseFactory to create the LexSense object, and then store it in the Senses owning sequence property of LexEntry using the Add method. Next, we fill in the analysis alternatives of the Gloss and Definition properties. Next, we store a reference to the MoStemMsa in the MorphoSyntaxAnalysis atomic reference property of LexEntry Next, we add a couple semantic domains to the sense. In this case, we use the FindPossibilityByName method on the semantic domain list with Abbreviations of 1.6, and 1.6.1.1.2 to get the items, and then store a reference to the items in the SemanticDomains reference collection property of LexSense.

```
sense = sl.GetInstance(ILexSenseFactory).Create()
entry.SensesOS.Add(sense)
sense.Gloss.set_String(wsa, 'lion')
sense.Definition.set_String(wsa, 'A large carnivorous feline mammal.')
sense.MorphoSyntaxAnalysisRA = stemMsa
plist = lp.SemanticDomainListOA
sense.SemanticDomainsRC.Add(plist.FindPossibilityByName(plist.PossibilitiesOS, '1.6', wsa))
sense.SemanticDomainsRC.Add(plist.FindPossibilityByName(plist.PossibilitiesOS, '1.6.1.1.2', wsa))
```

At this point the Examples property is empty. It gets set in the next section.

```
<rt class="LexSense" guid="933e76a4-6ded-47df-81ca-581f4937c08a" ownerguid="58ec354f-9013-4aca-b20f-9e3f999e1b96">
<Definition>
<AStr ws="en">
<Run ws="en">A large carnivorous feline mammal.</Run>
</AStr>
</Definition>
<Examples>
<objsur guid="807ee5ce-9312-4761-bc6c-089c0e6b34ac" t="o" />
</Examples>
<Gloss>
<AUni ws="en">lion</AUni>
</Gloss>
<MorphoSyntaxAnalysis>
<objsur guid="c710bbdc-11f7-46be-96c2-e81674138eed" t="r" />
</MorphoSyntaxAnalysis>
<SemanticDomains>
<objsur guid="56ef3f06-7fb9-462e-a7d0-517f3ce1623f" t="r" />
<objsur guid="944cf5af-469e-4b03-878f-a05d34b0d9f6" t="r" />
</SemanticDomains>
</rt>
```

Next, we'll add an example sentence to the sense. We use the ILexExampleSentenceFactory to create an instance of the class and then store it in the Examples owning sequence property of LexSense using the Add method. Then we can add the sentence text in the vernacular alternative of the Example property on ExampleSentence.

```
example = sl.GetInstance(ILexExampleSentenceFactory).Create()
sense.ExamplesOS.Add(example)
example.Example.set_String(wsv, 'Le lion est le roi de la jungle.')
```

At this point the Translations property is empty. It gets set in the next section.

```
<rt class="LexExampleSentence" guid="807ee5ce-9312-4761-bc6c-089c0e6b34ac"
ownerguid="933e76a4-6ded-47df-81ca-581f4937c08a">
<Example>
<AStr ws="fr">
<Run ws="fr">Le lion est le roi de la jungle.</Run>
</AStr>
</Example>
<Translations>
<objsur guid="6252d6ff-b0f3-4129-9650-77826569d7d4" t="o" />
</Translations>
</rt>
```

Finally, we can add a translation for the example sentence. We use ICmTranslationFactory to create the object. However, this is one of the classes in the model that has the GenCreate (in the spreadsheet) property set to false. This means there isn't a Create() method to create it. Instead, there is an override for Create that takes two parameters; the owning LexExampleSentence, and the item from the Translation Types list. In this case, we use the FindPossibilityByName method on the Translation Types list to get the 'Free Translation' list item, and include it as the second parameter to the Create method. The first parameter is the LexExampleSentence object. This method automatically adds the CmTranslation object to the Translations owning property of

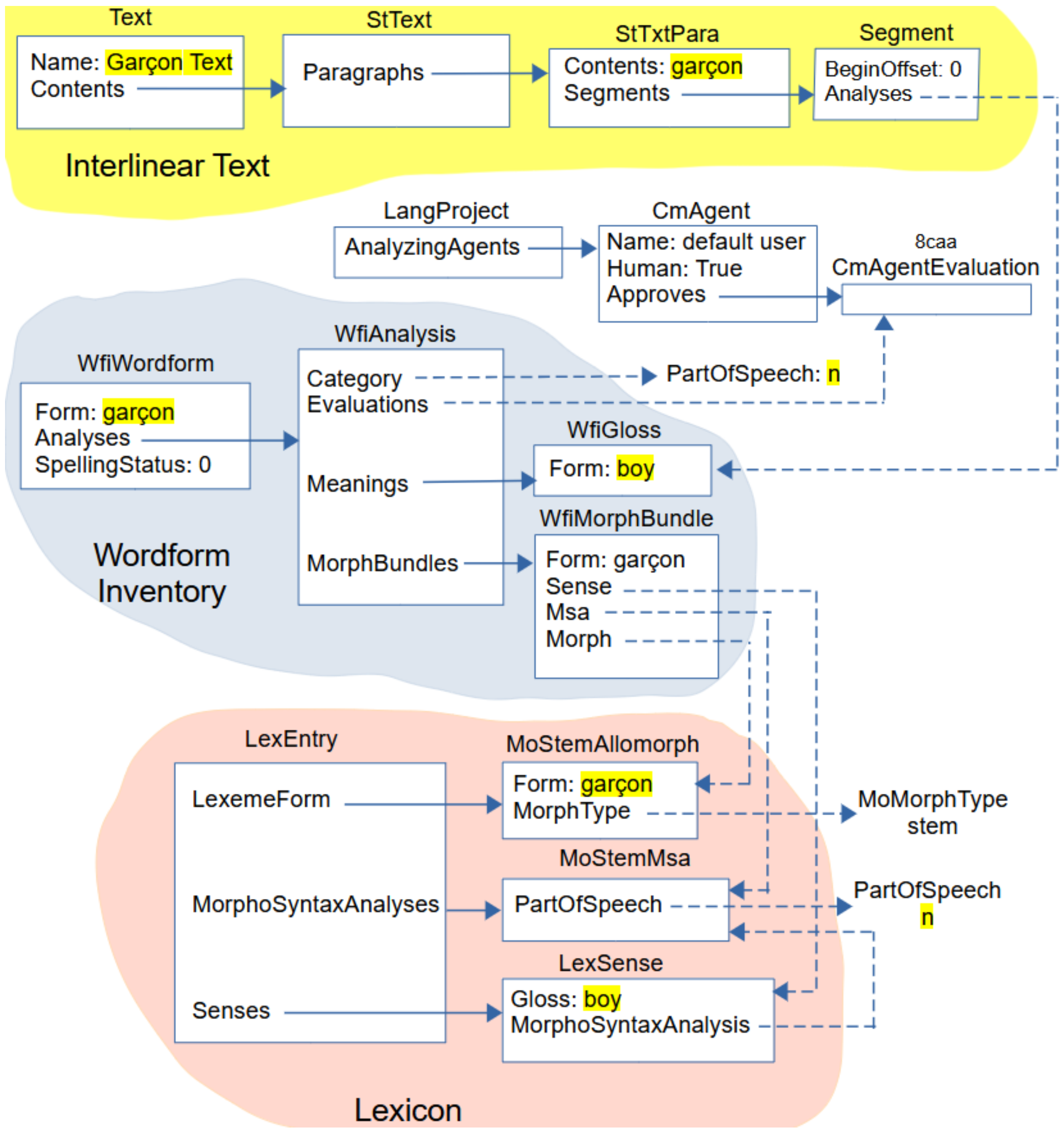
LexExampleSentence. Next we add the translation text in the analysis alternative of the Translation property of the CmTranslation.

```
plist = lp.TranslationTagsOA
freeTrans = plist.FindPossibilityByName(plist.PossibilitiesOS, 'Free translation', wsa)
translation = sl.GetInstance(ICmTranslationFactory).Create(example, freeTrans)
translation.Translation.set_String(wsa, 'The lion is the king of the jungle.')
```

```
<rt class="CmTranslation" guid="6252d6ff-b0f3-4129-9650-77826569d7d4" ownerguid="807ee5ce-9312-4761-bc6c-089c0e6b34ac">
<Translation>
<AStr ws="en">
<Run ws="en">The lion is the king of the jungle.</Run>
</AStr>
</Translation>
<Type>
<objsur guid="d7f7164a-e8cf-11d3-9764-00c04f186933" t="r" />
</Type>
</rt>
```

12 Creating an entry, analysis, and text using Python

Here is an example of creating an entry, an interlinear text and a complete analysis approved by the user. This diagram shows the structure needed for this. It assumes that the text, entry, and wordform is not currently in the project.



This code assumes that sl is set to the ServiceLocator.

```
#### Create entry
entry = sl.GetInstance(ILexEntryFactory).Create()
#### Create MoStemAllomorph and add to entry LexemeForm
stemAllo = sl.GetInstance(IMoStemAllomorphFactory).Create()
entry.LexemeFormOA = stemAllo
stemAllo.Form.set_String(wsv, 'garçon')
plist = lexicon.MorphTypesOA
```

```
stemAllo.MorphTypeRA = plist.FindPossibilityByName(plist.PossibilitiesOS, 'stem', wsa)
#### Create MoStemMsa and add to entry MorphoSyntaxAnalyses
stemMsa = sl.GetInstance(IMoStemMsaFactory).Create()
entry.MorphoSyntaxAnalysesOC.Add(stemMsa)
plist = lp.PartsOfSpeechOA
stemMsa.PartOfSpeechRA = plist.FindPossibilityByName(plist.PossibilitiesOS, 'Noun', wsa)
#### Create sense and add to entry Senses
sense = sl.GetInstance(ILexSenseFactory).Create()
entry.SensesOS.Add(sense)
sense.Gloss.set_String(wsa, 'boy')
sense.MorphoSyntaxAnalysisRA = stemMsa

#### Create WfiWordform
wordform = sl.GetInstance(IWfiWordformFactory).Create()
wordform.Form.set_String(wsv, 'garçon')
### Create WfiAnalysis
analysis = sl.GetInstance(IWfiAnalysisFactory).Create()
wordform.AnalysesOC.Add(analysis)
plist = lp.PartsOfSpeechOA
analysis.CategoryRA = plist.FindPossibilityByName(plist.PossibilitiesOS, 'Noun', wsa)
agevalRepo = sl.GetInstance(ICmAgentEvaluationRepository)
apprAgent = agevalRepo.GetObject(Guid("8caa11bb-cac4-4836-a081-1666245106b9"))
analysis.EvaluationsRC.Add(apprAgent)
### Create WfiGloss
wgloss = sl.GetInstance(IWfiGlossFactory).Create()
analysis.MeaningsOC.Add(wgloss)
wgloss.Form.set_String(wsa, 'boy')
### Create WfiMorphBundle
wmb = sl.GetInstance(IWfiMorphBundleFactory).Create()
analysis.MorphBundlesOS.Add(wmb)
wmb.Form.set_String(wsv, 'garçon')
wmb.SenseRA = sense
wmb.MsaRA = stemMsa
wmb.MorphRA = stemAllo

#### Create Text
itext = sl.GetInstance(ITextFactory).Create()
itext.Name.set_String(wsv, 'Garçon Text')
### Create StText
sttext = sl.GetInstance(IStTextFactory).Create()
itext.ContentsOA = sttext
### Create StTxtPara
stpara = sl.GetInstance(IStTxtParaFactory).Create()
sttext.ParagraphsOS.Add(stpara)
stpara.Contents = TsStringUtils.MakeString('garçon.', wsv)
### Create Segment
seg = sl.GetInstance(ISegmentFactory).Create()
stpara.SegmentsOS.Add(seg)
seg.AnalysesRS.Add(wgloss)
```

Note that the guid for the approved human agent should be the same in all projects.

Note that BeginOffset on Segment is a read-only attribute, so it can’t be set. Since this code does not set ParseIsCurrent on StTxtPara, when Flex opens that text, it will parse the paragraph, setting ParseIsCurrent to True, and setting BeginOffset appropriately, which is 0.