Importing complete interlinear glossed texts into SIL FLEx

Aleksandr Riaposov, Alexandre Arkhipov, Elena Lazarenko
University of Hamburg
{aleksandr.riaposov,alexandre.arkhipov,elena.lazarenko}@uni-hamburg.de

Abstract

This paper addresses the problem of data interchange involving SIL FLEx, a software widely used for morphological glossing of texts in underresourced languages. Interlinear glossed texts (IGT) created in FLEx can be exported in a well-defined XML format. However, IGT created in other programs (e.g. Toolbox or ELAN) or even in another version of the same FLEx project cannot be imported, which blocks the reuse of legacy data as well as roundtrips involving FLEx. We present a solution using Python and XSLT which was successfully used to merge a large collection of legacy texts in Forest and Tundra Enets (formerly glossed in Toolbox) with an existing FLEx project.

1 SIL FLEx and its import functionality

SIL FLEx (FieldWorks Language Explorer)[1] is a software for lexicon development and text analysis that is widely used in minority languages documentation. One of the key features of FLEx is keeping consistency between the interlinear glossed texts (IGT) and the lexicon. Any change in the gloss or grammatical properties in the lexical entry (typically corresponding to a morpheme, such as a root or an affix) is instantly reflected in the analyses of every wordform referring to that morpheme. On the other hand, as soon as a wordform has received a morpheme-by-morpheme analysis, this analysis is stored internally and will be suggested to the user if the same wordform is encountered elsewhere.

FLEx provides a range of import and export options to connect to other linguistic analysis software. In particular, roundtrips involving ELAN,[2] a multimedia annotation software, have become increasingly important.

A feature that has been much requested for years but is still not implemented in FLEx is the ability to import an IGT and preserve the analysis below the word level (morpheme breakdown, morpheme glosses, grammatical properties etc.). With the existing import functionality, the baseline text persists as well as sentence-level translations, but morpheme glossing has to be redone from scratch. As a consequence, an IGT can be exported from FLEx, but it is not possible to make changes to it externally and re-import it again into FLEx for further analysis, nor import it into another FLEx project. When a text leaves FLEx, it leaves FLEx for good.

This paper presents a way to circumvent this gap in interoperability with a set of scripts which translate the input IGTs into FLEx objects and append them to the main FLEx project XML data file (*.fwdata*). In the following sections, we present our first and most important use case (section 2), briefly describe the data formats and methods used (section 3), present the overall workflow and zoom in on the two core scripts (section 4).

2 Use case: importing Enets texts from Toolbox

Our motivation was the desire to merge existing FLEx projects with large collections of texts glossed in Field Linguist's Toolbox,[3] a previous generation text analysis software. We had a FLEx project for Forest Enets (Uralic; ISO 639-3: enf, Glottocode: fore1265) and a corresponding collection of 343 texts glossed in Toolbox at our disposal. The second FLEx project, as well as a smaller collection of 99 Toolbox texts, concerned the closely related Tundra Enets (ISO 639-3: enh,

[1] https://software.sil.org/fieldworks/
[2] https://archive.mpi.nl/tla/elan
[3] https://software.sil.org/toolbox/

Glottocode: tund1254). The total size of the Toolbox collections was ca. 140,000 tokens, so glossing all the texts from scratch would mean a few months' work.

The sets of transcription symbols, part of speech tags and morpheme glossing labels differed between the FLEx and Toolbox projects. Therefore, in order to minimize the discrepancies and the extent of manual corrections afterwards, the Toolbox data required additional adaptation prior to the import.

Our immediate goal – importing Enets texts – has been successfully attained. They are now published as part of the INEL Enets corpus (Shluinsky et al. 2024). However, we currently undertake further steps in order to make our solution generalizable and accessible to a wider community of FLEx users and for other workflows. In particular, it should be possible to apply our solution to IGTs exported by FLEx itself, performing an export-import roundtrip without the steps concerning Toolbox > ELAN > *.flextext* transformations.

3 Data formats and methods

First, the standard ELAN multi-files import function was used to convert legacy Toolbox analyses into ELAN *.eaf* format. ELAN is preferred as an intermediary stage since it uses an XML format (in contrast to the error-prone whitespace-separated plain text files in Toolbox) and offers more possibilities for preprocessing including powerful search and replace. A sample fragment of IGT in ELAN is shown in Figure 1.

Similarly, the standard ELAN multi-files export function was used to convert the data into *.flextext* XML (see Figure 3 in Appendix A). This is an interchange format for IGT understood and produced by FLEx. However, while complete IGTs can be exported from FLEx, only a part of the data is read by the built-in FLEx import function. Crucially, all word- and morph-level analyses are ignored; hence the need for an external solution.

The main part of the import conversion is done by two Python scripts. The existing FlexTools Python package[4] was not used because it did not contain the necessary functions to access and create all the types of objects in the FLEx data model required for the interlinear import.

ref@unknown	AS100709_KRAS_065					
tx_lat_for_toolbox@unknown	sɔjza	to	tonin		oka?	
mb@unknown	sɔjza	to	toni	-xon	oka	-?
ge@unknown	good	lake	there(dir)	-LOC.SG	many	-PL
ps@unknown	adj	n	pronadv	-case-num	pronnum	-case-num
ger@unknown	хороший	озеро	туда	-LOC.SG	много	-PL
fl_r@unknown	Хороших озер там много.					
fl_e@unknown	There are many good lakes there.					
tx_lat_during_transcription@unknown	sɔjza to tɔnen oka					
fl_r_during_transcription@unknown	хороших озер там много					
TC	00:03:10.000 - 00:03:11.680					

Figure 1. A fragment of IGT in ELAN

In addition, two rounds of preprocessing were necessary: one on the ELAN files prior to ELAN > *.flextext* conversion and the other on the *.flextext* files prior to the main import stage. These were implemented with XSLT stylesheets.

4 Conversion workflow

4.1 Workflow overview

The general workflow is presented in Figure 2.

A1. Setup/use a FLEx project. In our case, FLEx projects already existed. Otherwise, a new FLEx project must be created. All the necessary languages and analysis tiers must be set up and respective writing systems codes known prior to the import. Also, some categories such as PartOfSpeech are currently presumed by the import scripts to be exhaustively listed in the *.fwdata* file.

A2. *parserFWdata* (Python) analyzes the *.fwdata* file to list objects (wordforms, morphemes, analyses etc.) already present in the FLEx project.

A3. *converter* (Python) creates an augmented copy of the original *.fwdata*. It reads the input IGTs from *.flextext* files and compares them to the lists of existing objects. New objects are appended to *.fwdata* in full, already existing objects are referenced with respective GUIDs.

B1. Toolbox > ELAN import using the ELAN multi-files import function.

B2. Preprocessing in ELAN (XSLT): tiers setup and data unification. Tiers are renamed to facilitate the assignment of the correct writing system code to each tier during the next step (B3). Also, replacements are made in transcription, part of speech tags, glosses, and speaker codes in order to bring legacy data to the same shape as the current FLEx project.

B3. ELAN > *.flextext* export using the ELAN multi-files export function.

B4. Preprocessing in *.flextext* (XSLT): punctuation. Texts glossed in Toolbox typically

[4] https://github.com/cdfarrow/flextools

have punctuation characters attached to adjacent words. At this step, they are moved to separate elements in the baseline (main transcription tier), as would have been done by the FLEx tokenizer during regular import.

Steps A2 and A3 are the core conversion steps and will be described in more detail in the next sections. Note that the number and the complexity of steps B1–B4 depend greatly on the input data. They were necessary in our main use case due to the necessity of merging legacy and current analyses. When importing into an empty FLEx project, only certain elements of preprocessing are needed like tier renaming in B2 and punctuation handling in B4. For an IGT exported from FLEx these steps might not be needed altogether.

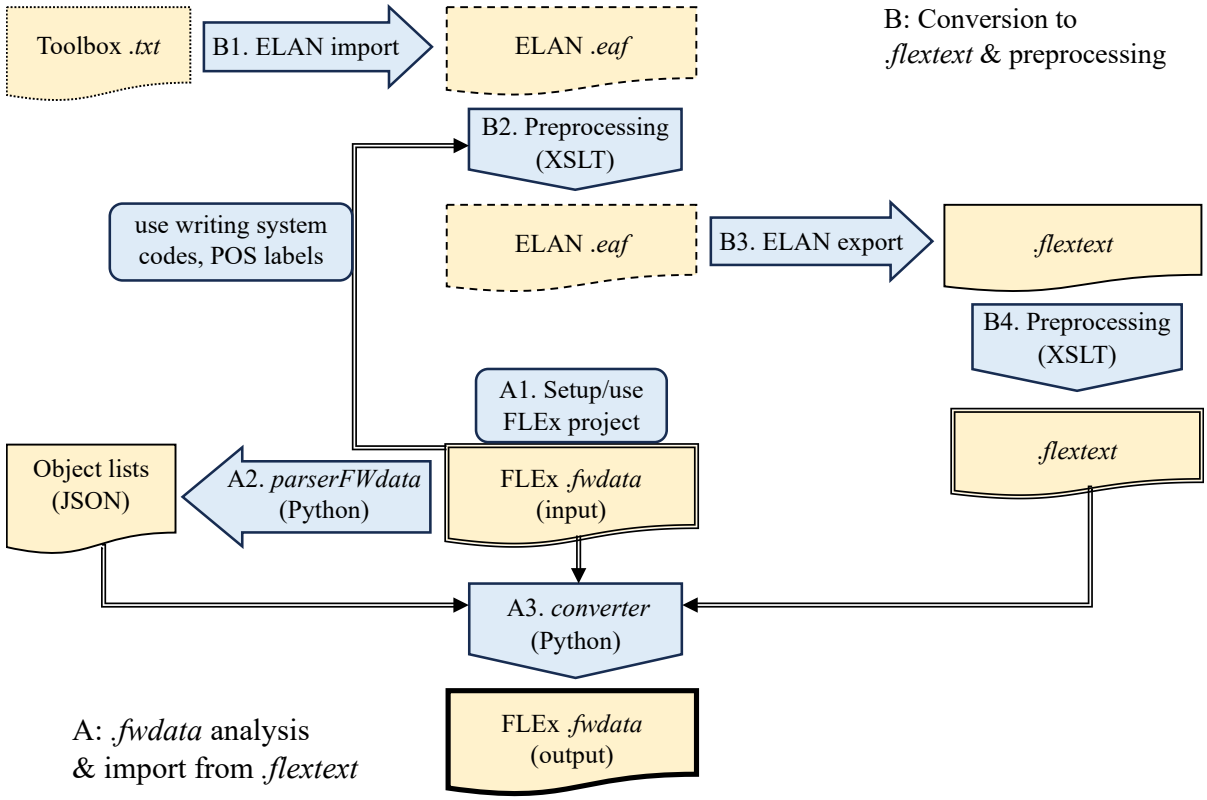

Figure 2. Conversion workflow

4.2 FLEx data analysis script: *parserFWdata*

FLEx stores most data related to one language/project an XML *.fwdata* file. The goal of the *parserFWdata* script is to read the *.fwdata* file, extract objects of certain types and store them in respective JSON files. This step is necessary because of the way the *.fwdata* file is structured: a single instance of preliminary parsing is advantageous over iterative on-the-fly parsing for performance reasons.

The *.fwdata* file has a rather flat structure, documented in (Zook 2024); parts related to IGT are explained in more detail in (Zook 2022). Objects of various kinds (including lexical entries, allomorphs, grammatical analyses, glosses, etc.) are represented by `<rt>` elements that are all immediate children of the root element `<languageproject>` and are interlinked via unique identifiers (GUIDs). As illustrated in Figure 5 in Appendix A, an `<rt>` element has obligatory attributes `"guid"`, whose value is an identifier, and `"class"`, which determines the kind of object represented by the element, in this case WfiWordform. The internal structure of each class is different. As an example, WfiWordform contains the surface form of a wordform (`"oka?"`) and identifiers of its existing analyses, pointing to `<rt>` elements of the class WfiAnalysis.

The *parserFWdata* script writes the relevant information from the select `<rt>` elements, including their GUID relations, into JSON files (see Figure 4 in Appendix A).

4.3 Appending new data: *converter*

The *converter* script reads *.flextext* files from a folder one at a time, assuming that each *.flextext* file contains one interlinear text. It checks the input

against the JSONs generated by *parserFWdata* for existing objects and appends new objects to the *.fwdata* file. It takes the top-down approach: starting with bigger elements such as the text itself, paragraphs, and sentences (presumably unique and always getting appended to the database), the script gradually descends to the word and morpheme level. In case of a full match, the script inserts a link containing the GUID of the matched element. Otherwise, a new element of the relevant class has to be created. The converter stores the identifiers of the new elements it generated so that in case it finds the same element again, a link to it will be provided instead of yet another new object.

After the list of *.flextext* files is exhausted, the script writes out an updated *.fwdata* file.

5 Discussion

In order to make the scripts more approachable by the wider community of FLEx users, we are providing a simple graphical solution, *launcher.py*, that launches *parserFWdata.py* and *converter.py*. The user needs to specify the codes for the target language ("vernacular") writing system, as well as these used for the glossing ("analysis"), provide the paths to the *.fwdata* file and to the folder with the *.flextext* files to be imported, and then run *parserFWdata.py* ("Parse") and *converter.py* ("Convert") in that order.

It bears mention that the scripts – and the entire workflow – have been initially created with a rather specific use case in mind. While their adaptation to an unsupervised general use may require certain efforts, we have already successfully applied the scripts to an unrelated collection of Toolbox texts in Kalmyk (Mongolic, ISO 639-3: xal, Glottocode: kalm1244).

The adaptation of the entire workflow to a new language took a few hours. However, in the future the procedure should become faster due to already implemented changes. It is also planned to integrate the commands to run the XSL stylesheets used in preprocessing steps B2 and B4 into the launcher. The lists of replacements to perform will be stored entirely in external settings files, so that the user could easily edit them without the need to modify the stylesheets.

The duration of the import process itself depends on the initial size of the FLEx project, i.e. principally on the number of distinct objects of each class (wordforms, morphs, etc.) which must be checked for each appended item. A comparison of processing times for two collections on a mid-range laptop is given in Table 1 below.

	For. Enets	**Kalmyk**
Original FLEx project		
Lexicon: morphs	1,662	11
Wordforms (tokens)	27,996	9
Wordforms (types)	8,236	9
Imported IGTs		
Wordforms (tokens)	111,028	8,548
Wordforms (types)	16,014	3,268
Resulting FLEx project		
Lexicon: morphs	3,921	1,916
Wordforms (tokens)	139,024	8,557
Wordforms (types)	21,631	3,268
Processing times		
parserFWdata	33 m 55 s	3 s
converter	5 h 22 m	1 m 11 s

Table 1. Processing times for two IGT collections

Acknowledgments

This publication has been produced in the context of the joint research funding of the German Federal Government and Federal States in the Academies' Programme, with funding from the Federal Ministry of Education and Research and the Free and Hanseatic City of Hamburg. The Academies' Programme is coordinated by the Union of the German Academies of Sciences and Humanities.

We would like to thank Andrey Shluinsky for his meticulous work with the Enets data, and Sarah Moeller and PT Anderson for a stimulating discussion of the conversion tool. We also thank the anonymous reviewers for their valuable comments.

References

Andrey Shluinsky, Olesya Khanina, and Beáta Wagner-Nagy. 2024. INEL Enets Corpus. Version 1.0. Publication date 2024-11-30. https://hdl.handle.net/11022/0000-0007-FE1D-C. Archived at Universität Hamburg. In: The INEL corpora of indigenous Northern Eurasian languages. https://hdl.handle.net/11022/0000-0007-F45A-1

Ken Zook. 2022. Interlinear text implementation in FLEx.https://downloads.languagetechnology.org/fieldworks/Documentation/Interlinear%20text%20in%20Flex.pdf

Ken Zook. 2024. FLEx 9.1 Conceptual Model. https://downloads.languagetechnology.org/fieldworks/Documentation/FLEx%209.1%20Conceptual%20Model.pdf

A Illustrative data fragments (simplified)

```
<word
guid="d5b46d83-23f4-4864-ad22-2bed35eaea2c">
   <item lang="enf" type="txt">oka?</item>
   <morphemes>
      <morph
   guid="3a41081e-bf6a-47b4-9b26-8a16f7d9d382" >
         <item lang="enf" type="cf">oka</item>
         <item lang="en" type="gls">many</item>
         <item lang="en" type="msa">quant</item>
         <item lang="ru" type="gls">много</item>
      </morph>
      <morph
   guid="08d6ff35-0d3b-4eab-a7c6-4575f756ea86" >
         <item lang="enf" type="cf">-ʔ</item>
         <item lang="en" type="gls">-3PL.S</item>
         <item lang="en" type="msa">-n:case</item>
         <item lang="ru" type="gls">-3PL.S</item>
      </morph>
   </morphemes>
</word>
```

Figure 3. *.flextext* IGT fragment with a wordform "okaʔ" as prepared for import (steps B3–B4)

```
{
"WfiWordform-GUID":
   "0019844c-c714-4a45-a157-6d899142318f",
"WfiWordform": "okaʔ",
"WfiAnalysis-GUID":
   "bb58d46e-5fb8-4dc4-98ae-38f010c87978",
"Category-GUID":
   "a4a759b4-5a10-4d7a-80a3-accf5bd840b1",
"senses": [
   {"Morph-GUID":
   "c64e2792-0170-4845-b3e5-931d01520eb9",
   "Morph": "oka",
   "Sense-Guid":
   "3d9bcf53-8afb-48d2-8bb7-9662001f733a"},
   {"Morph-GUID":
   "d31395c8-d543-41ae-b890-0a166cbe28f0",
   "Morph": "ʔ",
   "Sense-Guid":
   "31f13a26-8398-48aa-9edb-8eecae933dec"} ]
}
```

Figure 4. JSON representation of the wordform "okaʔ" in the FLEx project as created by *parserFWdata*

```
<rt class="WfiWordform" guid="0019844c-c714-4a45-a157-6d899142318f">
   <Analyses>
      <objsur guid="bb58d46e-5fb8-4dc4-98ae-38f010c87978" t="o" />
      <objsur guid="fcc35025-c6a4-4fd0-871d-89fff023512c" t="o" />
   </Analyses>
   <Form><AUni ws="enf">oka2</AUni></Form> …
</rt>

<rt class="WfiAnalysis" guid="bb58d46e-5fb8-4dc4-98ae-38f010c87978">
   <Category><objsur guid="a4a759b4-5a10-4d7a-80a3-accf5bd840b1" t="r" /></Category>
   <MorphBundles>
      <objsur guid="53d08c84-64b1-4417-bff8-6907421fe03e" t="o" />
      <objsur guid="829a06b6-566c-44b9-8a0d-72891c1a65d7" t="o" />
   </MorphBundles> …
</rt>

<rt class="PartOfSpeech" guid="a4a759b4-5a10-4d7a-80a3-accf5bd840b1">
   <Abbreviation><AUni ws="en">quant</AUni></Abbreviation>
   <Name><AUni ws="en">Quantifier</AUni></Name> …
</rt>

<rt class="MoStemAllomorph" guid="c64e2792-0170-4845-b3e5-931d01520eb9">
   <Form><AUni ws="enf">oka</AUni><AUni ws="enf-x-source">ôka</AUni></Form>
   <MorphType><objsur guid="d7f713e8-e8cf-11d3-9764-00c04f186933" t="r" /></MorphType> …
</rt>

<rt class="LexSense" guid="3d9bcf53-8afb-48d2-8bb7-9662001f733a">
   <Gloss><AUni ws="en">many</AUni><AUni ws="ru">много</AUni></Gloss> …
</rt>

<rt class="MoAffixAllomorph" guid="d31395c8-d543-41ae-b890-0a166cbe28f0">
   <Form><AUni ws="enf">2</AUni></Form>
   <MorphType><objsur guid="d7f713dd-e8cf-11d3-9764-00c04f186933" t="r" /></MorphType> …
</rt>

<rt class="LexSense" guid="31f13a26-8398-48aa-9edb-8eecae933dec">
   <Gloss><AUni ws="en">3PL.S</AUni><AUni ws="ru">3PL.S</AUni></Gloss> …
</rt>
```

Figure 5. Representation of the wordform "okaʔ" and related objects as <rt> elements in *.fwdata*