EXMARaLDA Add-In for MS Excel - Version 0.9.8.1

1 Summary

The EXMARaLDA Add-In for MS Excel is a freely available (GPL) plugin written using the Office VBA API which allows users of Excel to import data from the EXMARaLDA XML format into a spreadsheet (preserving cells and spans, as well as most types of metadata – see below for details) and back again from a spreadsheet to XML. It can also export data to PAULA XML and from TreeTagger/CWB SGML (so-called TreeTagger format).

This add-in is compatible with Office XP, 2003, 2007, 2010 and 2013 under Windows XP, Vista, Windows 7 or 8, and comes with absolutely no warranty. The latest public version of the add-in can be found on http://exmaralda.org.

New in this release:

- Penn Treebank importer
- Hierarchical trees in PAULA exporter
- Multiple annotations for individual segment layers in PAULA exporter

2 Installing

To install the add-in follow these steps:

1. Copy the file *exmaralda_io_0.9.8.1.xla* to the directory that holds your Excel add-ins. On an English language Windows XP installation this is usually:

   ```
   C:\Documents and Settings\YOUR_USER_NAME\application
   data\Microsoft\AddIns
   ```

   Other languages may have slightly different paths (e.g. "Dokumente und Einstellungen" for "Documents and Settings" on German Windows).
2. Open Excel and choose *Tools -> Add-Ins…* (again, other languages may use different names, e.g. German *Extras* for *Tools* etc.)
3. Click *Browse…* and navigate to the file *exmaralda_io_0.9.8.1.xla*. Once the file is chosen, the add-in *Exmaralda_Io_0.9.8.1* should appear on the list of add-ins.
4. Check the box next to E*xmaralda_Io_0.9.8.1* and click OK. A new menu called *Exmaralda* should now appear in the menu bar above.

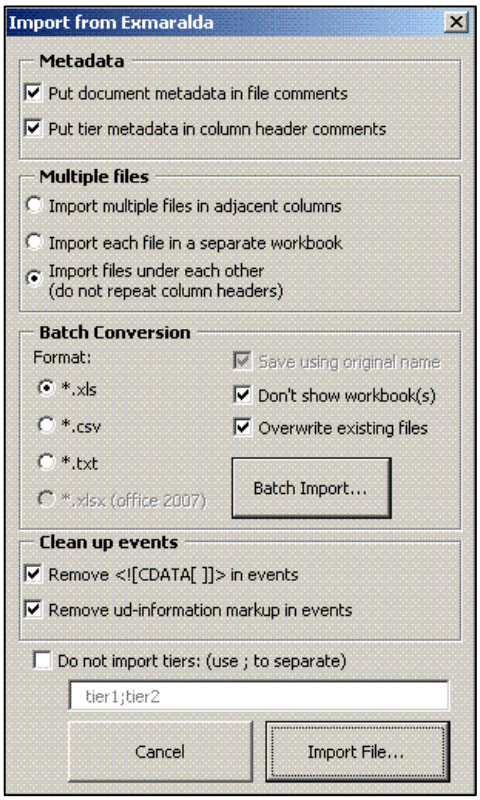
3 Uninstalling

To remove the add-in simply go to *Tools -> Add-Ins* and uncheck *Exmaralda_Io_0.9.8.1*. By pressing OK the Exmaralda menu will now be removed. You may then optionally delete the file *exmaralda_io_0.9.8.1.xla* if you wish.

4 Usage

The Exmaralda Add-In menu in Excel provides four basic functions as detailed below.

4.1 Import from Exmaralda



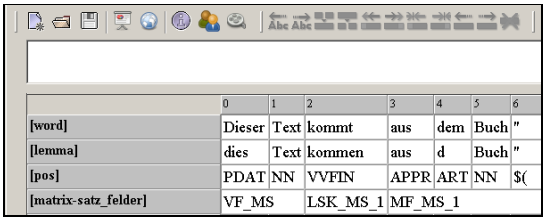
Choosing this function will open the Import Form.

Metadata

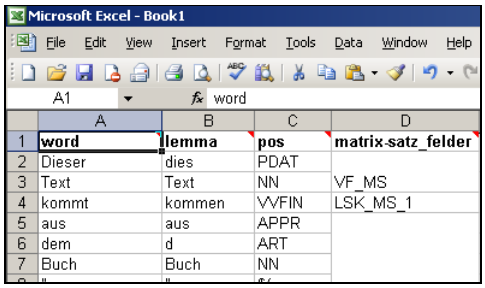
The form allows users to choose whether or not metadata applying to an entire EXMARaLDA document will be imported, and similarly for metadata applying to each tier within a document. Document metadata is stored in the Excel file's *Comments* property, which keeps the entire header of the original XML file. Tier metadata is stored in a comment to the header cell of each column. Note that at present **no metadata contained in <ud-tier-information> and <ud-information> tags within a <tier>** element is imported (support for this will probably be added in a future version. A further type of element which is ignored by the importer at present is the **<tierformat-table> of .exb files**, meaning that formatting information such as background colors for each tier etc. are not carried over.

Import Behavior and Multiple Files

Each tier is imported in a separate column in Excel, so that horizontal layers from EXMARaLDA are represented vertically as in the image below (this is because Excel can only accommodate 256 columns; see "Transposing" below for an EXMARaLDA style horizontal view). The user may determine where imported data will appear by selecting a cell or cells – data will then be imported starting from the top-leftmost selected cell. Each tier's *category* attribute is used as a column header in bold.



	0	1	2	3	4	5	6
[word]	Dieser	Text	kommt	aus	dem	Buch	"
[lemma]	dies	Text	kommen	aus	d	Buch	"
[pos]	PDAT	NN	VVFIN	APPR	ART	NN	$(
[matrix-satz_felder]	VF_MS		LSK_MS_1	MF_MS_1			



	A	B	C	D
1	word	lemma	pos	matrix-satz_felder
2	Dieser	dies	PDAT	
3	Text	Text	NN	VF_MS
4	kommt	kommen	VVFIN	LSK_MS_1
5	aus	aus	APPR	
6	dem	d	ART	
7	Buch	Buch	NN	

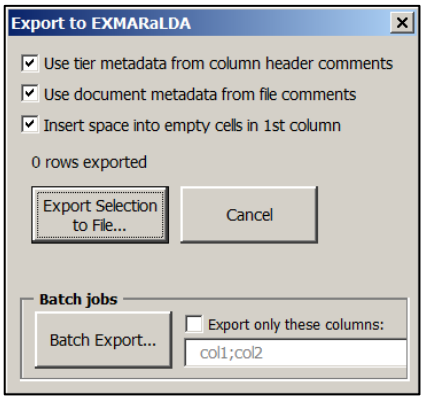
When importing multiple files, users can select whether the columns representing the annotation levels of each document should be imported side by side, each document in a separate Excel workbook, or continuously underneath each other (usually only suitable if all documents have precisely the same layers in the same order). When

documents are imported underneath each other, a header row is only generated for the first document, and the other documents are assumed to have exactly the same columns. If multiple files are imported into the same workbook and importing of document metadata is still selected, only the first document's header will be inserted into the *Comments* property.

Batch Conversion
To convert multiple files to Excel use the Batch Import option. Files can be saved as plain text (tab delimited), comma separated values (csv), Excel format (.xls), and in Excel 2007 or 2010, also in the new .xlsx format. Currently the file name is simply retained and the new extension is substituted for the original one. You can also choose not to show the result in Excel ("don't show workbooks") and whether or not existing files should be overwritten.

Clean Up
Users can also select to clean up the input XML by removing <![CDATA[ ]]> tags and ud-information markup within events (formatting within individual EXMARaLDA cells). Finally, it is also possible to omit certain layers that should not be imported. To do so, select *do not import tiers* and input the names of the tier categories that should be ignored, separated by a semicolon, e.g. *lemma;pos*

4.2 Export to Exmaralda



A selected range of cells may be exported to EXMARaLDA XML using the *Export to EXMARaLDA* menu. The first row of the exported data is assumed to contain the category names for each annotation layers, and all other rows are treated as data proper. If the *Comments* field of the Excel workbook contains an EXMARaLDA header (usually generated by the importer), this may be used for the exported file, otherwise a generic header is generated automatically. Similarly, tier attributes stored by the importer in comments to the header cell of each layer may be exported, with the exception of user defined tier metadata in **<ud-tier-information> and <ud-information> tags** within tiers. If no tier attributes are found, default values are inserted, which assume that the first column is a transcription layer (type="t") and all other layers are annotation layers (type="a"), and that the display name of the layer is the same as the category name.

You can also choose to insert a space into empty cells in the first column, which can be useful if you're using that column for tokens and exporting the data to a program that does not tolerate empty tokens. Finally, the Batch Export works similarly to the Batch Import – you can select multiple files and export them all using the above settings. Here it is also possible to specify a subset of columns to be exported from all files, separated by semicolons.

4.3 Transpose Selection in a New Sheet

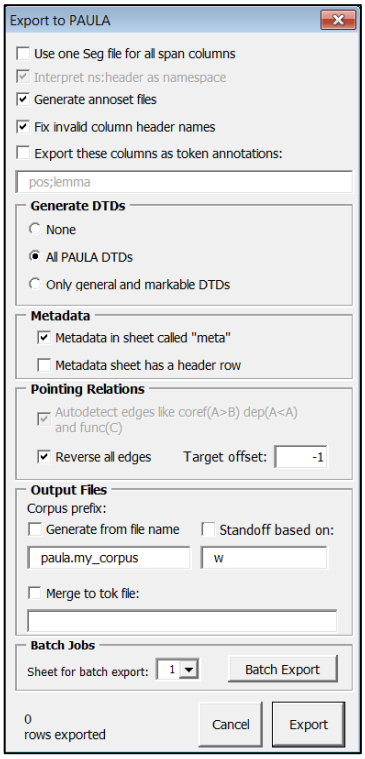
Since an Excel 2003 spreadsheet can only contain a maximum of 256 columns but as many as 65536 rows, imported data is presented vertically by default. It is however possible to transpose a selection into a horizontal format similar to the EXMARaLDA editor using this command. The first row is assumed to contain layer headers and is rendered in bold face. If the selected data contains more than 256 rows, the rows are broken up into groups of up to 256 rows each and the headers are repeated for each block to make navigation easier. Note that some spans may cross the border of a block of 256 lines. In such cases the add-in will attempt to find a cutoff point starting at a 200 row block. Data that cannot be split up in this way (or in any way, if there is a span of more than 256 rows), will not be able to be transposed.

4.4 Extract Metadata from Comments

When importing data from EXMARaLDA, metadata is saved in the files *comments* property as unparsed XML. It is possible to use the menu item "extract metadata to new sheet" to parse this XML into a new sheet called "meta" to get a tabular view of the imported metadata. The table contains three columns, one for the metadata name, one for the value, and a third column specifying whether this metadata was applied to the entire file (*project-name* or *transcription-name*), to a speaker (e.g. *sex* and *language* annotations) or a user-defined item (*ud-information*).

If a sheet named "meta" already exists, the extraction will fail. Also, if no metadata was imported from EXMARaLDA, the function will exit without creating the metadata sheet.

4.5 Export to PAULA



Export to PAULA works similarly to EXMARaLDA export, but with several additional options. It is assumed that the first selected column represents the tokens of the document and the first selected row contains annotation level names, which should be valid XML attribute names. It is possible to select automatic correction of annotation level names, which at the moment replaces spaces and '@' signs with underscores, umlauts and *ß* ligatures with plain vowel followed by *e* and *ss* respectively, and quotation marks and apostrophes with _quot_ and _apos_. The token column is also used to generate the base text of the document, which is generated with spaces to separate the contents in each row. This column may not contain spans. Alternatively, it is possible to specify an already existing token file in the text field at the bottom of the form. If this is done, all references to a token file are replaced with this string, and no raw text and token files are created. Another option is to specify that the standoff annotations should treat some other column as the base segmentation file, ignoring the tokens. This is useful if the PAULA annotations are supposed to point at a different format (e.g. TEI XML), or are supposed

to be merged with a different project. In this case, all annotation spans are generated to match the spans of the column specified in "standoff based on". Annotations encompassing fewer lines than those in the spans of that column are then no longer possible.

Pointing Relations

Pointing relations can be exported if column names are used of the type *name(A>B)*, where *A* and *B* stand for any column letters. Using *name(A>A)* creates pointing relations from and to the same level of annotation markables. This can be used to create dependencies or coreference annotations. For a more robust annotation that is not disrupted by the addition/removal of columns, columns may also be referred to by name, e.g. *coref(entity>entity)* produces a pointing relation of type *coref* between elements of the column labeled *entity*. If you use namespaces, these must be included: *coref(ner:entity>ner:entity)*.

The contents of the cells in a pointing relation column may be either empty or numerical. Numerical values give the target token line in the Excel sheet (not counting the header). For example, a value *1* in the fourth line of column B after the header, with the header *coref(A>A)*, means "create a pointing relation of the type 'coref' from the element in the fourth annotation line of column A to the element in the first annotation line of column A". If you are annotating directly in Excel and would like to use real Excel line numbers (i.e. counting the header line), use the "target offset" option to adjust the target line. If you would like the pointing relations to point not from 4 to 1 in the example above, but the other way around, select "reverse edges". It is also possible to reverse only individual columns, e.g. if you'd like reversed dependency annotations and non-reversed coreference. In this case, mark reversed columns with a left angle bracket: *name(A<A)* and do not select reverse edges (this reverses *all* edges).

Finally, in order to annotate pointing relations with key-value pairs, use column headers like *anno(B)* where *anno* is the annotation name and *B* is the letter of the column containing the pointing relations to be annotated. For example, to annotate the coref relation in the example above as type="anaphoric" use *type(B)* as a header for a new column and the value *anaphoric* in the cell corresponding to the row of the pointing relation. The example above could be applied to the table below:

	A	B	C
1	**token**	**coref(A>A)**	**type(B)**
2	Bob		
3	said		
4	so		
5	himself	1	anaphoric

Note that when using full column names as references, instructions in parentheses should be removed, i.e. in the example above *type(B)* can be replaced by *type(coref)*, but **not** by *type(coref(A>A))*.

Annosets, DTDs and Multiple Annotations for one Segment or Token
Users can also choose whether or not an annoset will be generated for the data (a PAULA XML manifest of the annotation levels present in the document), what DTDs should be generated (all PAULA DTD's, only those relevant to span annotations, or none) and whether or not all span annotations are equally granular, in which case only one markable (Seg) file can be generated to define all spans, with each annotation level creating only a single feature file referring to the joint Seg file. This last option relies on the identical granularity of the annotation levels and will fail if this assumption is violated. A further text field lets users choose annotation names (separated by semi-colon) which will be imported as feature attributes of the tokens, without a markable Seg file. To use this option, the corresponding level should contain no spans, or else unexpected results may follow.

Assigning Individual Annotations to the Same Markable
Beyond the possibility to assign all annotations to the same set of markables as described above, it is also possible to set individual annotation columns to only generate annotation features and attach these to the markables of some specific column. This will only work if the column defining the markables has as at least as the same cells filled as the feature annotation column. It is possible for the markable column to have filled cells that are empty in the added feature annotation, but not vice versa.

To mark a column as referring to the markables of another column, use one of the two syntax types illustrated in the following figure:

	A	B	C	D
1	**tok**	**entity**	**type(=B)**	**np_type(=entity)**
2	This			
3	is			
4	John	entity	person	proper_name
5	.			

The column-markable identity is indicated with the '=' sign in parentheses. The column being referred to can be addressed just as for pointing relations, either by its capitalized letter (=B) or by its name (=entity). These work exactly the same, but (=entity) is more robust if columns are added to the left of column B. Note that as always, namespaces need to be retained when referring by name, so if B is called mmax:entity, then C can have the header type(=mmax:entity), but **not** type(=entity).

Hierarchical Syntax Trees
There is now partial support for constituent syntax trees or other hierarchical 'struct' annotations in Excel. To use these, call multiple columns by the same name and add a suffix in **square** brackets indicating which layer the current column is anchored to. The deepest layer will always be anchored to a different named layer (typically the tokens, 'tok'), while higher layers are numbered to indicate the order of the hierarchy. The exporter assumes that larger spans dominate contained smaller spans, and prefers to dominate the span at the next nearest depth. If this is empty, spans at a lower depth are looked for, until the lowest layer is reached. The following image illustrates the structure:

	A	B	C	D	E	F	G	H
1	**tok**	**pos**	**cat[tok]**	**cat[1]**	**cat[2]**	**cat[3]**	**cat[4]**	**cat[5]**
2	Die	ART						
3	Jugendlich	NN	NX					
4	in	APPR						
5	Zossen	NE	NX	PX	NX	VF		
6	wollen	VMFIN	VXFIN	LK				
7	ein	ART						
8	Musikcafé	NN	NX	MF			SIMPX	
9	.	$.						PSEUDO

The layer cat[tok] gives the entry point for the syntax tree: it dominates 'tok' directly. It is dominated by cat[1], which is dominated by cat[2], and so on. Note however that dominance can skip layer depths when there are gaps in the spans. For example, in row 4, cat[1] has a span 'PX', but cat[tok] is empty. Therefore cat[1] will attempt to dominate the next layer in the tree hierarchy, which is 'tok'. Since a 'tok' is not empty on row 4, PX will dominates the 'tok' on row 4 (the word 'in'), but also the 'NX' from column cat[tok] in row 5, since cat[tok] is not empty on that row. In this way, spans always imply maximal dominance of elements under them, at the first available level of depth. Similarly cat[5] dominates the large span 'SIMPX' from cat[4], but also the tok on row 9 (the period).

Note also that this syntax tree format can be generated in Excel by the PTB importer plugin from a text file in the Penn Treebank bracketing format.

Corpus Prefixes and Namespaces

Users may specify a corpus prefix, which will be used at the beginning of all standoff file names in the PAULA document. Alternatively the prefix can be generated automatically from the Excel file name (minus the extension .xls or .xlsx). It is also possible to use namespaces in column headers with an intervening colon (namespace:annoname), in which case the namespace is prefixed to the respective file names with a period as a separator, to form a PAULA namespace.

Metadata

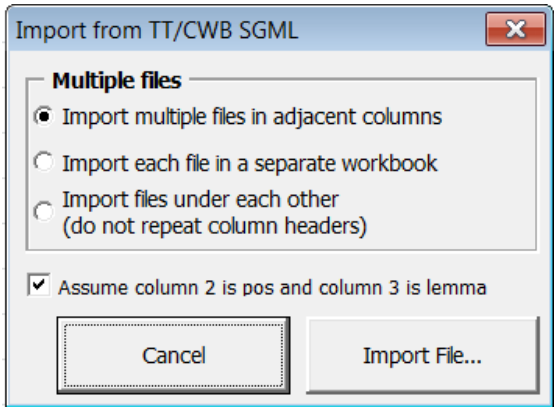
Metadata may be stored in a separate worksheet, which must be called "meta". Names of metadata attributed should be in column A of the sheet and the values in column B. The sheet may optionally have a header row (e.g. column headers like "name" and "value"); if so, select the appropriate option in the metadata section of the export form.

Batch Export

When exporting multiple Excel files, use the batch export command, which first asks for (multiple) Excel files and then for a location to generate the PAULA corpus. A subfolder with the same name as the respective Excel file name will be generated at the target location for each PAULA location. By default the first sheet of each chosen Excel file is exported, though you may choose a different sheet number from the drop-down list instead.

4.6 Import from TreeTagger/CWB SGML

The TreeTagger / CWB SGML format uses SGML elements in single lines to encompass spans of tokens and gives any number of attribute value annotations within these elements. Rows without markup elements are interpreted as tokens, and further strings separated by tabs are annotations of those tokens, usually part-of-speech (pos) and lemmas.



The TreeTagger importer allows multiple files to be imported next to each other, below each other, or in separate workbooks, much like the EXMARaLDA importer (see Section 4.1 for details). A further option is given whether it should be assumed that the first two token annotations after each token are called 'pos' and 'lemma', otherwise these, and in any case any annotations after these, are sequentially names ExtraAnno# (where # is the column number).

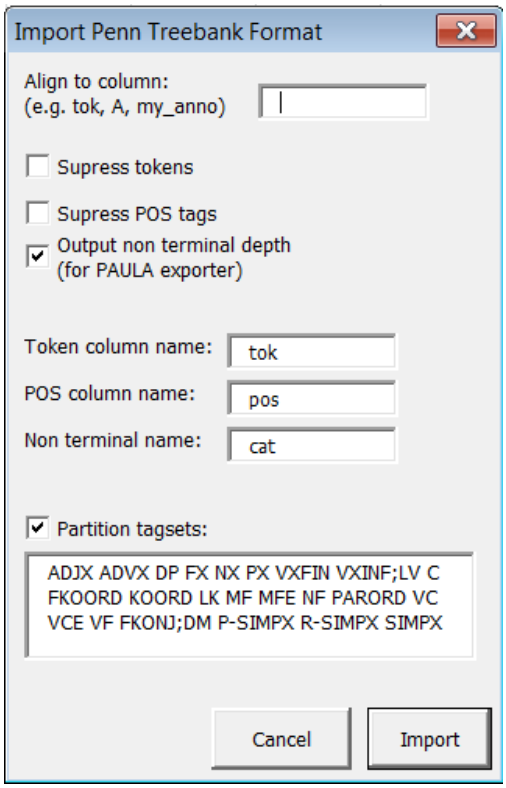
The behavior of the importer is as follows. Consider the following example input. The sentence "This is a test" is encompassed by a <sent> element with no attributes and contains two <chunk> elements with the attribute type="NP". Additionally, the verb is encompassed by a tag <verb> with an attribute verb="auxiliary".

```
<sent>
<chunk type="NP">
This DT0  this
</chunk>
<verb verb="auxiliary">
is   VBZ  be
</verb>
<chunk type="NP">
a    AT0  a
test NN1  test
</chunk>
</sent>
```

The importer will produce the following annotation columns: *sent*, filled with the value "sent", since this annotation contains no other information; *chunk@type* filled with the value "NP" (two spans); and *verb* with the value "auxiliary". Note that in the last case, the column *verb@verb* is not generated. Additionally, the importer will put the tokens into a column named *tok* and further generate the columns *pos* and *lemma*, or ExtraAnno1 and ExtraAnno2 if the option to assume pos and lemma is turned off.

Also note that conflicting hierarchies are no problem for SGML, and the importer accepts these. However, as dictated by the TreeTagger format, elements and tokens must be on individual lines, no whitespace may precede them, and nesting of elements of the same type if forbidden.

4.7 Import from Penn Treebank Bracket Format



The Penn Treebank importer can import data in the Penn Treebank bracket format, which represents syntax trees as bracket structures. The import form offers several options.

Align to column
Instead of importing the tokens from the Penn syntax tree, users can specify a column that already contains the tokens. This can be a column letter or header label. Filling this box causes tokens not to be imported, and the next lowest level of the syntax tree will refer to this layer as a token source.

Suppress Tokens/POS Tags
These checkboxes simply cause tokens and their POS tag annotations not to be outputted if desired.

Output Non-Terminal Depth
Selecting this option will append the layer depth in the tress in square brackets to each column. This is necessary for export to the PAULA format (see PAULA Exporter above and the figure below).

Token, POS and Non-Terminal Names
These boxes determine the column header given to the token column, the POS column and alls of the non-terminal columns (defaults are tok, pos and cat).

Partition Tagsets
If the tree contains multiple hierarchies (e.g. for German, topological fields intertwined with syntactic phrase categories), it can be useful to split up the PTB tree into multiple separate trees. To do so, input multiple lists of space-separated tags, and separate the lists with a semi-colon, as shown in the form above. As a result, the syntax tree for each tagset will be imported while ignoring nodes with the labels from other tagsets. For example, if you have a dominance SIMPX > VF > NX, as in columns E-G below, and the middle node VF belongs to a different tagset than the other two, then one syntax tree can be generated with SIMPX dominating NX directly and another tree with a separate VF node. To give the second tree a separate node name, use multiple node names in the node name box, separated by semicolons. Compare the two figures below for a fused tree and one separating VF, MF and LK (topological field annotations) into a separate tree labeled ‘field’.

	A	B	C	D	E	F	G	H
1	**tok**	**pos**	**cat[tok]**	**cat[1]**	**cat[2]**	**cat[3]**	**cat[4]**	**cat[5]**
2	Die	ART						
3	Jugendlich	NN	NX					
4	in	APPR						
5	Zossen	NE	NX	PX	NX	VF		
6	wollen	VMFIN	VXFIN	LK				
7	ein	ART						
8	Musikcafé	NN	NX	MF			SIMPX	
9	.	$.						PSEUDO

Annotation with one unpartitioned syntax tree

	A	B	C	D	E	F	G
1	**tok**	**pos**	**cat[tok]**	**cat[1]**	**cat[2]**	**cat[3]**	**field[tok]**
2	Die						
3	Jugendlich	NX					
4	in						
5	Zossen	NX	PX	NX			VF
6	wollen	VXFIN					LK
7	ein						
8	Musikcafé	NX			SIMPX		MF
9	.					PSEUDO	

Multiple trees, with the field VF, MF and LK partitioned into a separate hierarchy

5 Limitations and Assumptions about the Data

5.1 EXMARaLDA XML

For the sake of simplicity and speed in handling large files, the add-in does not actually parse EXMARaLDA XML on import, but rather reads each file line by line, making certain assumptions about the order of elements and attributes. The add-in supports the following variations:

- Time line events' start and end attributes may stand in any order
- Time line elements may have labels and may be numbered arbitrarily
- For each tier, a category attribute is expected, which may appear anywhere

In all other cases it should be assumed that the importer expects data that looks like the data produced by the exporter. That said, the importer is somewhat robust in that it ignores anything it does not understand: this means it will happily import invalid XML if it also contains the elements it expects. If you run into problems caused by hidden assumptions made by the importer, please file a bug report, ideally with a sample of the data which caused the problem.

5.2 PAULA XML

As noted above, the following assumptions are made about the Excel data to be exported:

- The tokens are in the leftmost selected column, which contains no spans.
- Column headers contain only alphanumeric ascii strings beginning with a letter and without spaces or special characters (Umlaut, accents etc.; but see automatic header correction in 4.5 above)
- When merging to one markable (Seg) file, conflicting spans will lead to errors (the exporter does not validate for such errors at the moment).

For more information on PAULA XML see:

http://www.sfb632.uni-potsdam.de/en/paula-en.html

6 Contact

For questions, bug reports or feature requests regarding the add-in please contact Amir Zeldes (amir {dot} zeldes {at} rz {dot} hu-berlin {dot} de).