LISTER: Life Science Experiment Metadata Parser

This repository contains a set of files to parse SOP from lab experiments.

Motivation

As a research group usually has its own set of SOP to conduct experiments, a tool to extract metadata from SOP-adapted experiment document (DOCX, eLabFTW entry or Markdown files) would be handy. The metadata extracted is helpful in documenting the research and hence improves the reproducibility of the conducted research. To enable the metadata extraction, the experiment documentation should follow some annotation rules (described later below).

Repository structure

- The base directory contains the metadata extraction script.
- *input* directory contains the *.DOCX/*.MD experiment documentation examples for the extraction.
- *output* directory contains extracted steps order – key – value in both JSON and XLSX format.

List of supported annotations

The parser extracts:

Extracted Items	Description	Representation	Example	Extracted order,key,value
Section	The section name	*<section|section name>*	*<section|Structure Preparation>*	• "-", *section*, *Structure Preparation*
Order	The *order* of the steps, based on the order of the paragraph in the docx SOP document	-	-	-
Key	The *key* for the metadata, based on the value represented in the curly bracket after the pipe character {value|key}.	{value|*key*}	{sequence alignment|*stage*}	• <order>, *stage*, sequence alignment
Comment	*Comments* are allowed within the key, represented within regular brackets after the pipe symbol. Comment can be placed both/either before and/or after key and/or value. **TBD**: How to serialize comments in the metadata parser output.	{value|(*comment*) key} or {value (*comment*)|key} or {value (*comment*)| (*comment*) key}	{receptor residue| (*minimization*) target}	• <order> target, receptor residue

Extracted Items	Description	Representation	Example	Extracted order,key,value
Value	The *value* of the metadata is based on the first value represented in the curly bracket before the pipe character {value|key}. Example: with 'sequence alignment' as the value.	{*value*|key}	{*sequence alignment*|stage}	• <order>, stage, *sequence alignment*
Control flow: `for each`	Extract multiple key value pairs related to `for each` iterations	< `for each` |iterated value>	< `for each` |*generated pose*>	• <order>, step type, *iteration* • <order>, flow type, *for each* • <order>, flow parameter, generated pose

Extracted Items	Description	Representation	Example	Extracted order,key,value
Control flow: `while`	Extract multiple key value pairs related to `while` iteration	< `while` |key|logical operator|value> ... <iteration operation|magnitude>	< `while` |pH|lte|7> ... <iterate|+|1>	• <order>, step type, *iteration* • <order>, flow type, *while* • <order>, flow parameter, pH • <order>, flow logical parameter, lte • <order>, flow compared value, 7 • <order> • flow type, iterate (after while) • flow operation, +, • <order>, flow magnitude, 1

Extracted Items	Description	Representation	Example	Extracted order,key,value
Control flow: `if`	Extract multiple key value pairs related to `if` iteration	<´if´|key|logical operator|value>	< `if` |pH|lte|7>	• <order>, step type, *conditional* • <order>, flow type, *if* • <order>, flow parameter, pH • <order>, flow logical parameter, lte • <order>, flow compared value, 7

Extracted Items	Description	Representation	Example	Extracted order,key,value
Control flow: `else if`	Extract multiple key value pairs related to `else if` iteration	< `else if` |key|logical operator|value>	< `else if` |pH|between|[8-12]>	• <order>, step type, *conditional* • <order>, flow type, *else if* • <order>, flow parameter, pH • <order>, flow logical parameter, between • <order>, flow range, [8-12] • <order>,start iteration value,8 • <order>,end iteration value,12
Control flow: `else`	Extract multiple key value pairs related to `else` iteration	< `else` >		• <order>, step type, *conditional* • <order>, flow type, *else*

Extracted Items	Description	Representation	Example	Extracted order,key,value
Control flow: `for`	Extract multiple key value pairs related to `for` iteration	< `for` |key|[range]|iteration operation|magnitude>	< `for` |pH|[1-7]|+|1>	• <order>, step type, *iteration* • <order>, flow type, *for* • <order>, flow parameter, pH • <order>, flow logical parameter, lte • <order>, flow flow range, [1-7] • <order>,start iteration value,1 • <order>,end iteration value,7 • <order>, flow operation, +, • <order>, flow magnitude, 1

The overall example of the SOP document is available in the */input* directory.

Supported operators

Logical operator

Logical operator is used to decide whether a particular condition is met during iteration/conditional block. It is available for `while`, `if` and `else if` control flow. The following logical operators are supported:

- `e` : equal
- `ne` : not equal
- `lt` : less than
- `lte` : less than equal
- `gt` : greater than
- `gte` : greater than equal
- `between` : between

Iteration operator

Iteration operator is used to change the value of compared variable during a loop. It is available for `while` and `for`. The following iteration operators are supported:

- `+` : iteration using addition
- `-` : iteration using subtraction
- `%` : iteration using modulo
- `*` : iteration using multiplication
- `/` : iteration using division

Document validation

LISTER checks the following problems upon parsing, and report accordingly:

- Orphaned brackets and indicates which line the error is located.
- Mismatched data types for conditionals and iterations.
- Mismatched argument numbers for conditionals and iterations.
- Invalid control flows.

Image extraction

Images are extracted from the experiment documents, but as for now there is no metadata or naming scheme from the extracted images.

Constraints and recommendations

Constraints

- Subprocess/substep is not currently supported to simplify data storage process (or let me (Fathoni) know if this is really necessary, providing suggestion on how to store/serialize it would be great). Any subprocess will be stored simply as a next step from its parent process.
- Comments are currently not yet extracted in the parser's output but it is already parsed in the background - it is mainly intended to provide more detail about a particular key/value for SOP adopters.

Recommendations

- Avoid the use of reference without explicit KV-pair (avoid e.g., "*Repeat step 1 with similar parameters*"), as this will make the metadata for that particular implicit step unextracted.
- To minimize confusion regarding units of measurement (e.g., `fs` vs `ps`), please explicitly state the units within the value portion of the KV-pair, e.g., `{0.01 ps|gamma_ln}`.

Running the parser

Using a *.docx file as an input

Requirement: a *.docx file adhering to annotation rules.

1. Create an experiment entry according to the above annotation rules, save it into a *.docx file.
2. Ensure that the code blocks under `PARSING FROM ELABFTW CONTENT`, `PARSING FROM MARKDOWN`, `FROM ELAB` and `FROM MARKDOWN` block are commented.
3. Uncomment the code blocks under `PARSING FROM DOCX DOCUMENT` and `FROM MARKDOWN`.
4. Change the output directory/filename.
5. Change the input directory/file name in the python script .
6. Run the script.

Using an eLabFTW experiment entry via API

Requirement: 1) an eLabFTW experiment entry adhering to annotation rules, 2) the experiment number from that experiment and 3) eLabFTW API Token from your user account.

1. Create SOP according to the above annotation rules, save it into an experiment.
2. Ensure that the code blocks under `PARSING FROM DOCX DOCUMENT`, `PARSING FROM MARKDOWN`, `FROM DOCX` and `FROM MARKDOWN` block are commented.
3. Uncomment the code blocks under `PARSING FROM ELAB DOCUMENT` and `ROM MARKDOWN`.
4. Change the output directory/filename.
5. Change the API token, API endpoint URL, and experiment ID in the python script.

6. Run the script.

Using a Markdown file as an input

Requirement: 1) an experiment entry adhering to annotation rules in Markdown format.

1. Create SOP according to the above annotation rules, save it into an experiment in *.MD format.
2. Ensure that the code blocks under `PARSING FROM DOCX DOCUMENT` , `PARSING FROM ELABFTW CONTENT`, `FROM ELAB` and `FROM DOCX` block str commented.
3. Uncomment the code block under `PARSING FROM DOCX DOCUMENT` and `FROM MARKDOWN`.
4. Change the output directory/filename.
5. Change the input directory/file name in the python script.
6. Run the script.

Open for discussion

1. Comments serialization, as well as representing and structuring comments in the output file.
2. Maintaining information in the form of images, tables, figures etc.
3. Integration with eLabFTW.
4. Whether integration with SWATE and ARC is feasible.
5. Creating ontology terms from collected SOPs, and linking the keys with the ontologies.
6. How step number should be counted, e.g., should it be restarted from 1 after a new section.
7. **How can we accommodate the paragraph with similar key having several values?.** -- currently allowed with warnings shown in the logs.

Miscellanous

Associated conditionals

Writing conditionals can be exhausting when there are many else clauses involved. To simplify the writing of conditionals, the authors can choose to write it in a concise manner as an associated conditionals using the /{number} notations after the key-value pairs, and a comment notated by regular bracket after the intended value. Here is an example:

Example

*"The top five templates identified by TopDomain were {3N25_A (a), 4YJ5_A (b), 3GR4_A (c), 1A49_A (d), 6DU6_B (e)|template_pdbs}**/1** with sequence identities of {99% (a), 93% (b), 93% (c), 100% (d), 63%*

*(e)|template_identities}**/1**, coverages of {95% (a), 97% (b), 97% (c), 97% (d), 96%(e)|template_coverages}**/1**, and predicted TM-Score of {0.96 (a), 0.96 (b), 0.96 (c), 0.96 (d), 0.93(e)| template_confidences}**/1**, respectively.*"

Explanation

From the SOP above, associated key-values are marked with the number after the "/" symbol, so keys with a similar number after the "'/" are grouped together. In the example above, *template_pdbs, template_identities, template_coverages,* and *template_confidences* belong to the same association. The values on each key are then associated according to the comment in the regular bracket. So, if template_pdbs = 3N25_A, then template_identities = 99%, template_coverages = 95%, and template_confidences = 0.96. On a table, the correlation is grouped as shown below.

<style> </style>

association set	1	1	1	1	1
mapping	a	b	c	d	e
template_pdbs	4YJ5_A	3N25_A	3GR4_A	1A49_A	6DU6_B
template_identities	0.99	0.93	0.93	1	0.63
template_coverages	0.95	0.97	0.97	0.97	0.96
template_confidences	0.96	0.96	0.96	0.96	0.93

Note: the SOP users need to adapt this themselves by removing irrelevant values and /{number} annotations when adapting for their experiment. SOP is not going to be parsed, only the experiments will.