LISTER: Life Science Protocol Metadata Parser

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

Motivation

As research group usually has its own set of SOP to conduct experiments, a tool to extract metadata from an editable document (e.g., DOCX) would be handy. The metadata is helpful in documenting the research and hence improves the reproducibility of the conducted research. To enable the metadata extraction, the SOP should follow some annotation rules (described later below).

Repository structure

- The base directory contains the metadata extraction script.
- *input* directory contains the docx SOPs example for 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
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*

Extracted Items	Description	Representation	Example	Extracted order,key,value
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> ... <+|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 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/sop2.docx* file. The color in the *sop2.docx* does not play any role in the order/key/value extraction.

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.

Constraints and recommendations

Constraint

- Only one unique key per step is supported. If there are similar keys within one step, only the last pair of that key will be saved and the previous ones will be overridden (**TODO**: create a warning in the validator if a non-empty KV already existed).
- Subprocess/substep is not currently supported to simplify data storage process (or let me 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.
- Some of the keys are not officially on the Amber's output parameter list (e.g., No. of atom in the used PDB molecules), but please feel free to suggest a new key. These suggested keys will be analyzed for further inclusion in metadata standard in the specific domain.
- Avoid 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.
- Comments are currently not yet extracted in the parser's output but it is already parsed in the background - still need to find a way how to simplify the data serialization.

Recommendations

- To minimize confusion regarding units of measurement (e.g., `fs` vs `ps`), please explicitly state the units as a comment within the value portion of the KV-pair, e.g., `{0.01 (ps)|gamma_ln}`.
- Default measurement unit should be used (e.g., `ps` instead of `fs` in e.g., AmberMD's `gamma_ln` variable).
- Avoid superfluous blank lines (**TODO**: Discard step numbering on an empty line/section - or implement specific step numbering functionality).

Running the parser

1. Create SOP according to the above annotation rules.
2. Change the input directory/file name in the python script (2nd last line). - check ´# ADJUST INPUT/OUTPUT FILE HERE´ in the code.
3. Change the output directory/filename (last line). - check ´# ADJUST INPUT/OUTPUT FILE HERE´ in the code.
4. Run the script.

Open for discussion

1. Comments serialization, as well as representing and structuring comments in the output file.
2. The necessity of supporting substeps parsing and serialization.

3. Support for plain text format.
4. Support for markdown:
    1. This will require redesign on how the text should be annotated, as the bracketing method will break (or requires a lot of metacharacters).
    2. On the plus side, MD is supported by eLabFTW, and since it is a non binary file, the changes can be tracked on version control system.
5. Maintaining information in the form of images, tables, figures etc.
6. Integration with eLabFTW.
7. Whether integration with SWATE and ARC is feasible.
8. Creating ontology terms from collected SOPs, and linking the keys with the ontologies.
9. How step number should be counted, e.g., should it be restarted from 1 after a new section.

Further plans

1. Implement TODOs in this readme, and bug fixes.
2. Gather feedback from LISTER users.
3. Consult CAi and Biochemistry1 for LISTER's implementability on other labs.
4. Align the used keys with terms from an ontology, or if the term does not exist, create a new term by extending an ontology or creating a term within a new ontology.