Collating Your Own Data

Jump to bottom

Catherine Smith edited this page on Apr 10, 2019 · 1 revision

Collating your own data requires the data to be prepared in the correct JSON format and that JSON to be stored in the text repository. It also requires a project to be configured.

Each of your witnesses will need to be broken down into the units you want to collate. While collateX should be able to collate large chunks of data the collation editor is best suited to texts that can be broken down into shorter units. It was written for editing the New Testament so verse sized chunks are ideal, other projects have used poetic lines, sentences or other syntactic units.

Structuring the Text Repository

The text repostory is located at `collation/data/textrepo`.

In this location there is directory called `json`.

In this directory each document you want to collate needs its own directory. The directory name needs to be a unique reference to that document and should not use spaces in the name. The data for each document is stored in the relevant directory.

The Document/Transcription Metadata

Each document directory must contain a file called `metadata.json`.

This file must contain a JSON object with the following two keys

- **_id** *[string]* - must match the name of the directory for the document
- **siglum** *[string]* - the label to be used for this document in the collation editor interface and any generated apparatus

The values can be the same if your data allows this but both keys should still be provided. You can add other keys if they are useful to you for other putposes but the collation editor only requires, and will only use, these two.

metadata.json example

```
{
  "_id" : "NT_GRC_01_B04",
  "siglum" : "01"
}
```

The Collation Unit Data

The work you are editing needs to be divided into collatable chunks and each of these chunks or units must be given a label so you can refer to them. In the directory for each witness you need a JSON file for each of the units that is extant in that particular witness (lacunose and

omitted units are discussed later). The file name should be the label for the unit and the file extension should be `.json`.

The JSON structure requires the following keys (if the unit is omitted then the witnesses key is not required)

- **transcription** *[string]* - must match the name of the directory for the witnesses and the _id value in the metatdata.json file.
- **transcription_siglum** *[string]* - must match the siglum value in the metadata.json. file (this key will probably be phased out in later releases as it is a duplicate of siglum)
- **siglum** *[string]* - must match the siglum value in the metadata.json file.
- **witnesses** *[array/JSON]* - an array of JSON objects representating the reading/s of the text of this witness in this unit (see the subsection **Structure of Witnesses Data**).

Lacunose and Omitted Units

The collation editor can make a distinction between lacunose sections of text and omitted sections of text. Lacunose is used for text which is absent because it is missing due to physical damage to the manuscript, or missing pages. Omitted is used for text which is omitted from the text even though the material that it would have ben written on is present.

If a witness is Lacunose for an entire collation unit then you should not create a JSON file for the unit in that witness directory. The collation editor will interpret this as a lacunose passage in this witness.

If a witness omits an entire collation unit then the JSON file should be created for the unit unit but there should be no witnesses key in the JSON object. The collation editor will interpret the lack of the witnesses key as an omission.

Structure of Witnesses Data

The witness data is an array (list) of JSON objects. Each JSON object in the list represents a reading of the text in the witness. A list is used because it is possible for units to be repeated in the same witness or for corrections to a single instance of the unit to result in multiple 'readings'.

Each object in the list has two keys

- **id** *[string]* - this is the siglum to be used for this reading in this witness (this could include information about the hand or the instance number if the unit appears more than once in the text).
- **tokens** *[array]* - a list of JSON objects each of which represents a single word in this reading.

Each token object must include the following keys

- **index** *[string]* - the collation editor requires each reading in each witness to be numbered with sequential even numbers starting at 2. Data format though should be a string.
- **reading** *[string]* - this should be the same as the id for this reading.
- **t** *[string]* - a string representation of the word which will be sent to collateX. This could be normalised in some way such as always being lowercase. **NB:** this must not be an empty string or collateX will fail.
- **original** *[string]* - a string representation of the word in its original state in the witness. If you do no normalisation to t then this will be the same string. Having this value in addition allows the editor to go back to the original version before processing the display settings which are always applied to the original string (unless you have specified a setting that uses something else).
- **rule_match** *[array]* - a list of strings which should include all strings that would be considered a match for this token when applying rules. In simple cases this will just be a list of a single string equal to the same value as original. The most obvious use case when more than one token would be added is where there is an abbreviated form of a word in the text and an editor can choose to see either the expanded or abbreviated form in the collation editor. In this case a rule created for one form would need to apply to either and so both would appear in this list.

Any number of additional keys can be included in this list. If you are going to customise the settings then you may need to encode extra data in the token such as punctuation for example. You may also want to encode information about gaps in the text which is explained in the next section.

When dealing with corrected text the collation editor treats each hand as a completely separate witness to the text. For this reason it is advisable to provide a full representation of the reading and not just the corrected words.

Encoding Gaps within a Collation Unit

Within a collation unit the collation editor assumes text is omitted unless your witnesses data tells it otherwise.

To encode lacunose text in addition to the required keys in the token object you will need to add additional keys and details about the lacunose section. When the gap follows a word (as in is not before the first word of the context unit). This is done by adding two extra keys to the token object.

- **gap_after** *[boolean]* - should always be true.
- **gap_details** *[string]* - the details of the gap which will appear between <> in the editor eg. lac 2 char

If this is a gap before the very first extant word in the given unit then you must add the following two keys to the first token.

- **gap_before** *[boolean]* - should always be true.
- **gap_before_details** *[string]* - the details of the gap which will appear between <> in the editor eg. lac 2 char

Examples

Simple collation unit JSON example

Document siglum: 01
Text: A simple example sentence

```
[
  {
    "id": "01",
    "tokens": [
        {
          "index": 2,
          "reading": "01",
          "original": "A",
          "t": "a",
          "rule_match": ["a"]
        },
        {
          "index": 4,
          "reading": "01",
          "original": "simple",
          "t": "simple",
          "rule_match": ["simple"]
        },
        {
          "index": 6,
          "reading": "01",
          "original": "example",
          "t": "example",
          "rule_match": ["example"]
        },
        {
          "index": 8,
          "reading": "01",
          "original": "sentence",
          "t": "sentence",
          "rule_match": ["sentence"]
        }
    ]
  }
]
```

Complex collation unit JSON example

Document siglum: 02
Text: A ~~complex~~ corrected example [lac 7-8 char] with damage

02* will be used for the first hand and 02C for the correction


```
[
  {
    "id": "02*",
    "tokens": [
        {
          "index": 2,
          "reading": "02*",
          "original": "A",
          "t": "a",
          "rule_match": ["a"]
        },
        {
          "index": 4,
          "reading": "02*",
          "original": "complex",
          "t": "complex",
          "rule_match": ["complex"]
        },
        {
          "index": 6,
          "reading": "02*",
          "original": "example",
          "t": "example",
          "rule_match": ["example"],
          "gap_after": true,
          "gap_details": "lac 7-8 char"
        },
        {
          "index": 8,
          "reading": "02*",
          "original": "with",
          "t": "with",
          "rule_match": ["with"]
        },
        {
          "index": 10,
          "reading": "02*",
          "original": "damage",
          "t": "damage",
          "rule_match": ["damage"]
        }
    ]
  },
  {
    "id": "02C",
```

```
      "tokens": [
          {
            "index": 2,
            "reading": "02C",
            "original": "A",
            "t": "a",
            "rule_match": ["a"]
          },
          {
            "index": 4,
            "reading": "02C",
            "original": "corrected",
            "t": "corrected",
            "rule_match": ["corrected"]
          },
          {
            "index": 6,
            "reading": "02C",
            "original": "example",
            "t": "example",
            "rule_match": ["example"],
            "gap_after": true,
            "gap_details": "lac 7-8 char"
          },
          {
            "index": 8,
            "reading": "02C",
            "original": "with",
            "t": "with",
            "rule_match": ["with"]
          },
          {
            "index": 10,
            "reading": "02C",
            "original": "damage",
            "t": "damage",
            "rule_match": ["damage"]
          }
      ]
    }
  ]
```

Project Configuration

The project configurations are located at `collation/data/project`.

In this directory there can be a number of different project directories each must contain a `config.json` file. The project supplied for the example is called `default`. You can replace this configuration with your own or add a new directory for a second project. If you add a new project then you will need to edit the file at `collation/static/js/local_services.js` to

change line 361 which should read

```
_current_project = 'default';
```

and replace 'default' with the string that matched your project directory. This means you can have lots of different project configurations stored and select the one that you want to run by editing this file.

The minimum information required in the project configuration is

- **id** *[string]* - this must agree with the name of the project directory.
- **name** *[string]* - the name of the project to be displayed in the collation editor.
- **managing_editor** *[string]* - this is the id of the user in charge of this project. For the standalone version leave this as 'default'.
- **editors** *[array]* - a list of all users with permission to edit this project. For the standalone version leave this as a list with just 'default' in it.
- **witnesses** *[array]* - a list of all of the documents whose text is to be collated. The string used should be the one you chose for the directory name and the _id value in the metadata.json file.
- **base_text** *[string]* - the id of the document you want to use as a base text. This must be one of the documents in the list of witnesses for the project.

There are many other settings that can be added to the project configuration and customisations of the collation editor are also specified in this file. These will be documented in the collation_editor_core readme file.

Simple project configuration example

```
{
  "id": "default",
  "name": "example project",
  "managing_editor": "default",
  "editors": ["editors"],
  "witnesses": [
                  "document_1",
                  "document_2",
                  "document_3",
                  "document_4",
                  "document_5"
              ],
  "base_text": "document_3"
}
```

Pages 3

Clone this wiki locally

```
https://github.com/itsee-birmingham/standalone_collation_editor.wiki.git
```