CollateX and XML

Computer-supported collation with CollateX

DH2015, Sydney, 2015-06-29

Overview

- Input documents may be in XML
- Desired output may be in XML
- CollateX does not support XML input or output natively
- CollateX does support JSON input and output natively
- The user can convert between XML and JSON

Work flow

- Input files are in XML
- Convert to JSON
  - Tokenize, normalize
- Input JSON into CollateX
- CollateX generates JSON output
- Convert JSON output to XML

Why JSON input

- CollateX tokenizes in a predefined way
  - Split on white space
  - Punctuation marks are tokens
- CollateX matches and aligns on literal string values of tokens
- You can override both of those with JSON input

Why JSON output

- CollateX can only produce plain-text alignment tables and JSON output from JSON input
- If you need JSON input and you want structured output, your only option is JSON output

Using XSLT within Python

- The Python lxml library supports XSLT …
- … but only XSLT 1.0 and XPath 1.0

What sort of JSON does CollateX expect

```
{
 "witnesses" : [
  {
   "id" : "A",
   "tokens" : [
     { "t" : "A", "ref" : 123 },
     { "t" : "black" , "adj" : true },
     { "t" : "cat", "id" : "xyz" }
   ]
  },
  {
   "id" : "B",
   "tokens" : [
     { "t" : "A" },
     { "t" : "white" , "adj" : true },
     { "t" : "kitten.", "n" : "cat" }
   ]
  }
 ]
}
```

JSON and Python dictionaries

- JSON is a hierarchical data structure with property names and values
- A Python dictionary is a hierarchical data structure with keys and values
- JSON can be expressed as a Python dictionary
- A Python dictionary can be serialized as a string
- The dictionary and string look alike to a human, but CollateX requires the dictionary

The XML input

- Divide large witnesses into smaller segments
- XML may have
  - No internal markup within a segment
    - <l id="13" n="13">Li salaus se torne al serain</l>
  - Internal markup constrained to a token
    - <l id="8" n="8">Ki maint <abbrev>et</abbrev> el pere et el fis</l>
  - Internal markup that crosses token boundaries
    - <l id="1116" n="1098">Nus cler<crease>s ne vos poroit</crease> desc<abbrev>ri</abbrev>re</l>

Producing JSON for CollateX input

- Replace whitespace *only in text() nodes* with empty <w/> milestone tags
- Convert milestones to <w> wrapper elements
- Retain markup within word tokens
- Flatten markup that crosses token boundaries

Tokenizing XML

- Replace whitespace *only in text() nodes* with empty <w/> milestone tags
- <l id="1116" n="1098">Nus cler<crease>s ne vos poroit</crease> desc<abbrev>ri</abbrev>re</l>
- <l id="1116" n="1098"><w/>Nus<w/>cler<crease>s<w/>ne<w/>vos<w/>poroit</crease><w/>desc<abbrev>ri</abbrev>re</l>

Insert milestones

- Convert milestones to <w> wrapper elements

<l id="1116" n="1098">
  <w>Nus</w>
  <w>cler<crease>s<w/>
  <w>ne</w>
  <w>vos</w>
  <w>poroit</crease></w>
  <w>desc<abbrev>ri</abbrev>re</w>
</l>

Retain markup within a single token

- Flatten markup that crosses token boundaries to empty start and end tags

```
<l id="1116" n="1098">
    <w>Nus</w>
    <w>cler<crease n="start"/>s<w/>
    <w>ne</w>
    <w>vos</w>
    <w>poroit<crease n="end"/></w>
    <w>desc<abbrev>ri</abbrev>re</w>
</l>
```

CollateX output


```
{
  "table": [
    [
      [{
        "n": "ki", "t": "Ki"
      }],
      [{
        "n": "maint", "t": "maint"
      }]
    ],
    [
      [{
        "n": "qui", "t": "Q<abbrev>ui<\/abbrev>"
      }],
      [{
        "n": "maint", "t": "mai<abbrev>n<\/abbrev>t"
      }]
    ],
  "witnesses": [
    "A",
    "B"
  ]
}
```


JSON to XML

- Convert JSON output to isomorphic XML
- Use XSLT to convert simple XML to desired output format (TEI, HTML, etc.)