**Part 1 – Step 3: Canonicalization**

Simple pipeline: In this step, we employ a python script to perform the work. The same canonicalization rules are applied against both file A and file B. At the end, we can compare their checksum to confirm whether they are identical.

The following process is to be applied against both documents

*Step 1 - Read document using an XML Parser*: By using an XML parser, we effectively obtain the document intrinsic content. The XML parser has a few built-int features to easily choose (unify) UTF-8 encoding, expand internal DTDs if any (file B), and omit comment elements. We also exclude DTD when loading the document because final DTD will be determined after canonicalization.

|  |
| --- |
| from lxml import etree  parser = etree.XMLParser(  encoding='utf8',   attribute\_defaults=True,   remove\_comments=True,   load\_dtd=False) complaintsRoot = etree.parse(  file\_path, parser=parser).getroot() |

ComplaintRoot is a root node which represents the entire dataset as a tree. From here, we have an easy logical interface to work with. This is advantageous than working directly with content string

*Step 2 – Attribute string values cleanup*: Here we trim off excessive whitespace surrounding a node text. For paragraph-like field such as `consumerNarrative` and `publicResponse`, we normalize line-ends so that long text comparison is not affected by insignificant spaces.

|  |
| --- |
| def normalize\_line\_ends(txt: str) -> str:  pts = txt.split('\n')  pts = [ss.strip() for ss in pts]  return ' '.join(pts)  for elem in complaintsRoot.iter('\*'):  if elem.text:  #print(elem.text)  elem.text = elem.text.strip()  for elem in complaintsRoot.iter('consumerNarrative'):  elem.text = normalize\_line\_ends(elem.text) for elem in complaintsRoot.iter('publicResponse'):  elem.text = normalize\_line\_ends(elem.text) |

*Step 3 – Attributes sort and format*: For all elements, sort their attributes by tag name, and trim attribute values. By doing this, corresponding attributes are lined up appropriately for cross-reference elements comparison.

|  |
| --- |
| def trim\_and\_sort\_attrs(tt: etree) -> None:  for k in tt.attrib:  tt.attrib[k] = tt.attrib[k].strip()  dd = {}  for k in tt.attrib:  v = tt.attrib.pop(k)  dd[k] = v  for k in sorted(dd):  tt.attrib[k] = dd[k]  def sort\_child\_elements(tt: etree, key\_func) -> None:  if not tt[:]:  return  tt[:] = sorted(tt, key=key\_func)  for child in tt[:]:  sort\_child\_elements(child, key\_func)  pass  for anyTag in complaintsRoot.iter('\*'):  trim\_and\_sort\_attrs(anyTag) # sort elements at all levels by tag name alphabetically sort\_child\_elements(  complaintsRoot[:],   lambda x: x.tag) |

*Final indentation* - The last step is to conventionalize indentation. Doing this makes sure parent-child relationships are distinguished deterministically as the content is printed

|  |
| --- |
| # normalize indentation etree.indent(complaintsRoot) |

And obviously, write the XML content as string to an output file

|  |
| --- |
| # step: single encoding: UTF8 str = etree.tostring(  complaintsRoot,   encoding='utf8',   pretty\_print=True,   xml\_declaration=True) # return final xml outcome return str  f = open("final-file.xml", "w") f.write(strA.decode('utf8')) f.close() |

At this point, we may attempt to compare the 2 processed documents. We use <https://www.diffchecker.com/diff>, an online diff tool to check line by line difference between 2 arbitrary texts. The outcome is quite interesting. Here are observations over 2 datasets:

*Observation 1*: In the old system (file A), a complaint has a child element named `submitted` which tells the channel (Web, Referral) via which customer feedback was sent. This is effectively the `submissionType` attribute in the new system (file B).

|  |  |
| --- | --- |
| **File A (canonicalized)** | **File B (canonicalized)** |
| <?xml version='1.0' encoding='utf8'?>  <consumerComplaints>  <complaint id="14038">  <company>  <companyName>U.S. BANCORP</companyName>  <companyState>AZ</companyState>  <companyZip>85008</companyZip>  </company>  <event date="2017-01-22" type="sentToCompany"/>  <event date="2017-01-17" type="received"/>  <issue>  <issueType>Loan servicing, payments, escrow account</issueType>  </issue>  <product>  <productType>Mortgage</productType>  <subproduct>Other mortgage</subproduct>  </product>  <response consumerDisputed="Y" timely="Y">  <responseType>Closed without relief</responseType>  </response>  <submitted via="Referral"/>  </complaint> | <?xml version='1.0' encoding='utf8'?>  <consumerComplaints>  <complaint id="14038" submissionType="Referral">  <company>  <companyName>U.S. BANCORP</companyName>  <companyState>AZ</companyState>  <companyZip>85008</companyZip>  </company>  <event date="2017-01-22" type="sentToCompany"/>  <event date="2017-01-17" type="received"/>  <issue>  <issueType>Loan servicing, payments, escrow account</issueType>  </issue>  <product>  <productType>Mortgage</productType>  <subproduct>Other mortgage</subproduct>  </product>  <response consumerDisputed="Y" timely="Y">  <responseType>Closed without relief</responseType>  </response>  </complaint> |

With these kinds of variation in place, identity comparison would still fail by doing byte by byte matchup. Resolution for this issue requires knowledge on the global data. Here we assume that in the old system, a complaint should have only 1 `submitted` child element. This makes practical sense. We validate this as part of the canonicalization procedure. Then, we need to resort to a single representation. Let’s choose to present it as an attribute named `submissionType` instead of having a child element. So, this overall step consists of:

* Removing `submitted` child element if there’s any
* Retain the `submitted` `via` value and set it as `submissionType` attribute on the complaint element

This step is performed just before the final indentation:

|  |
| --- |
| for complaint in complaintsRoot.iter('complaint'):  subs = [sub for sub in complaint.iter('submitted')]  if subs:  assert 1 == len(subs)  if 'via' in subs[0].attrib:  submType = subs[0].attrib['via']  complaint.attrib['submissionType'] = submType  # remove the <submitted /> element  complaint.remove(subs[0]) |

*Observation 2*: It seemed there’re inconsistent representation of binary answer (N, No for No; Y, Yes for Yes).

|  |  |
| --- | --- |
| <product>  <productType>Debt collection</productType>  <subproduct>Medical</subproduct>  </product>  <response consumerDisputed="N" timely="N">  <responseType>Untimely response</responseType>  </response> | <product>  <productType>Debt collection</productType>  <subproduct>Medical</subproduct>  </product>  <response consumerDisputed="N" timely="no">  <responseType>Untimely response</responseType>  </response> |

We resolve this issue by unifying the attribute values to a single form: “N” for No and “Y” for Yes. This step is also run right before the final indentation:

|  |
| --- |
| def uniform\_yes\_no(tt: etree) -> None:  for k in tt.attrib:  v = tt.attrib[k]  if v.lower() == 'y' or v.lower() == 'yes':  tt.attrib[k] = 'Y'  elif v.lower() == 'n' or v.lower() == 'no':  tt.attrib[k] = 'N'  for complaint in complaintsRoot.iter('complaint'):  resp = [resp for resp in complaint.iter('response')]  assert 1 == len(resp)  resp = resp[0]  uniform\_yes\_no(resp) |

To summarize, the whole workflow consists of the following steps:

* Step 1 - Read document using an XML parser
* Step 2 – Attribute string values cleanup
* Step 3 – Attribute sort and format
* Observation 1 – Resolve `submissionType` variation
* Observation 2 – Resolve Yes/No binary answer inconsistency
* Final indentation
* Write result to new XML file

We employ a small utility function to compare checksum of 2 documents after cleansing. We asserted that the outcome checksums are equal

|  |
| --- |
| def get\_checksum(ss: bytearray) -> str:  hash\_object = hashlib.md5(ss)  return hash\_object.hexdigest()  fA\_path = 'Consumer\_Complaints\_FileA.xml' fB\_path = 'Consumer\_Complaints\_FileB.xml'  strA = canonicalize\_utf8\_encoded(fA\_path) strB = canonicalize\_utf8\_encoded(fB\_path)  assert get\_checksum(strA) == get\_checksum(strB) |

We even went further by comparing their full binary contents. The results are also confirmed to be exact equal

|  |
| --- |
| def binary\_compare(a: bytearray, b: bytearray) -> bool:  if len(a) != len(b):  return False  idx = 0  buf = 8  while idx < len(a):  ba = str(a[idx: buf])  bb = str(b[idx: buf])  if ba != bb:  return False  idx += buf  return True  is\_same = binary\_compare(strA, strB) assert is\_same |

Therefore, we can conclude that the 2 contents are indeed identical.