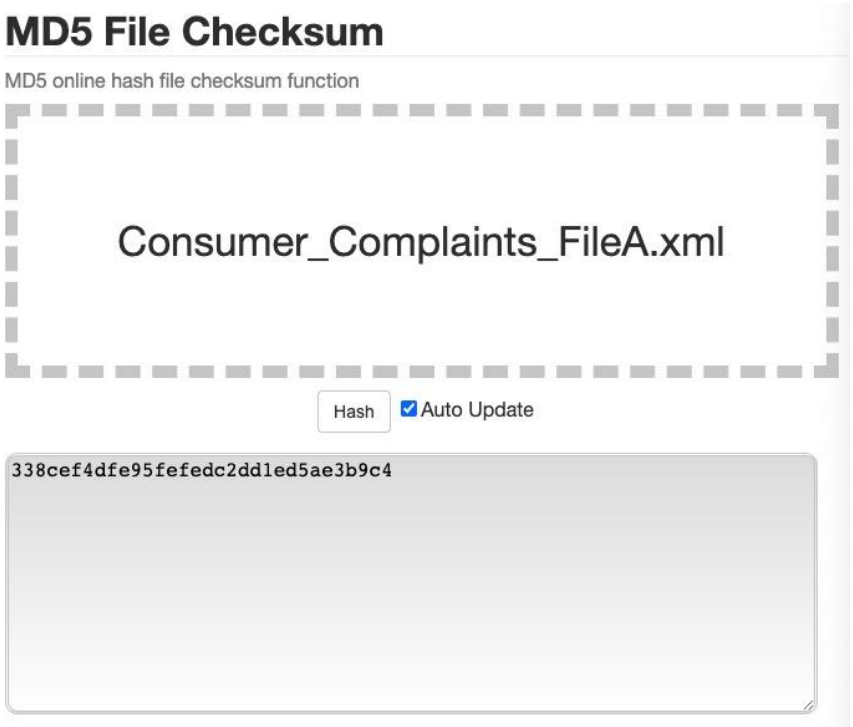
Assignment 4 Part 2: Data Canonicalization

Felicia Liu (liu318@illinois.edu)
Netid: liu318
CS598: Foundations of Data Curation

1. [12 points] Write a short profile of each file we have given you. Provide a narrative description of the file (50-100 words per file), including its format, properties, contents, and MD5 checksum. This website has a checksum calculator plus gives instructions for running a checksum in Unix and Linux systems: http://onlinemd5.com/; https://emn178.github.io/online-tools/md5_checksum.html

File A (Old System):

The MD5 file checksum for the original file A is 338cef4dfe95fefedc2dd1ed5ae3b9c4.



This file is in XML format and consistently indented by TAB SPACE. The parent element consumerComplaints contains eight consumer complaint records. Each record has a unique id attribute and seven sub-elements including information about:

- event: multiple events information is specified in type and date attribute.
- product: contains two child-elements productType and subproduct.
- issue: contains two child-elements issueType and sub-issue.
- consumerNarrative: an optional element.
- company: contains three child-elements companyName, companyState, companyZip.
- submitted: contains an attribute via describing how it was submitted.
- response: contains two child-elements publicResponse and responseType.

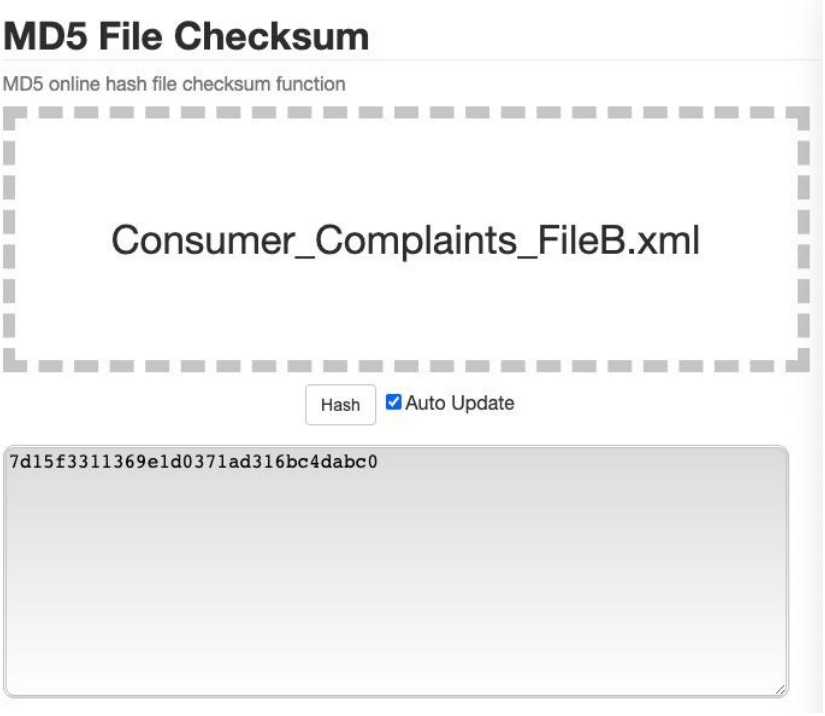
File A (DTD file):

It is designed to constrain files A's elements and attributes. It will be encoded using the UTF-8 in XML format under version 1.0.

- More than one complaint record can be stored in the root element consumerComplaints
- Within the complaint element, there are seven sub-elements defined including event, product, issue, consumerNarrative, company, submitted and response. They can be arranged in two ways.
- There are multiple events allowed, and the consumerNarrative may be optional.
- Sub-elements including sub-issue, sub-product and publicResponse are optional.
- In response, the ConsumerDisputed and Timely attribute can only have one of two values, N or Y.

File B (New System):

The MD5 file checksum for the original file B is 7d15f3311369e1d0371ad316bc4dabc0.



This file is encoded using the UTF-8 in XML format under version 1.0 and indented inconsistently by spaces. The parent element consumerComplaints contains eight consumer complaint records. Each record has its unique id and submissionType attributes. It also contains seven sub-elements including information about:

- event: multiple events information is specified in type and date attribute.
- product: contains two child-elements productType and subproduct.
- issue: contains two child-elements issueType and sub-issue.
- consumerNarrative: an optional element.
- company: contains three child-elements companyName, companyState, companyZip.
- submitted: mentioned but unused element.
- response: contains two attributes timely and consumerDisputed, two child-elements
- publicResponse and responseType.

File B (DTD file):

It is designed to constrain files B's elements and attributes. It will be encoded using the UTF-8 in XML format under version 1.0.

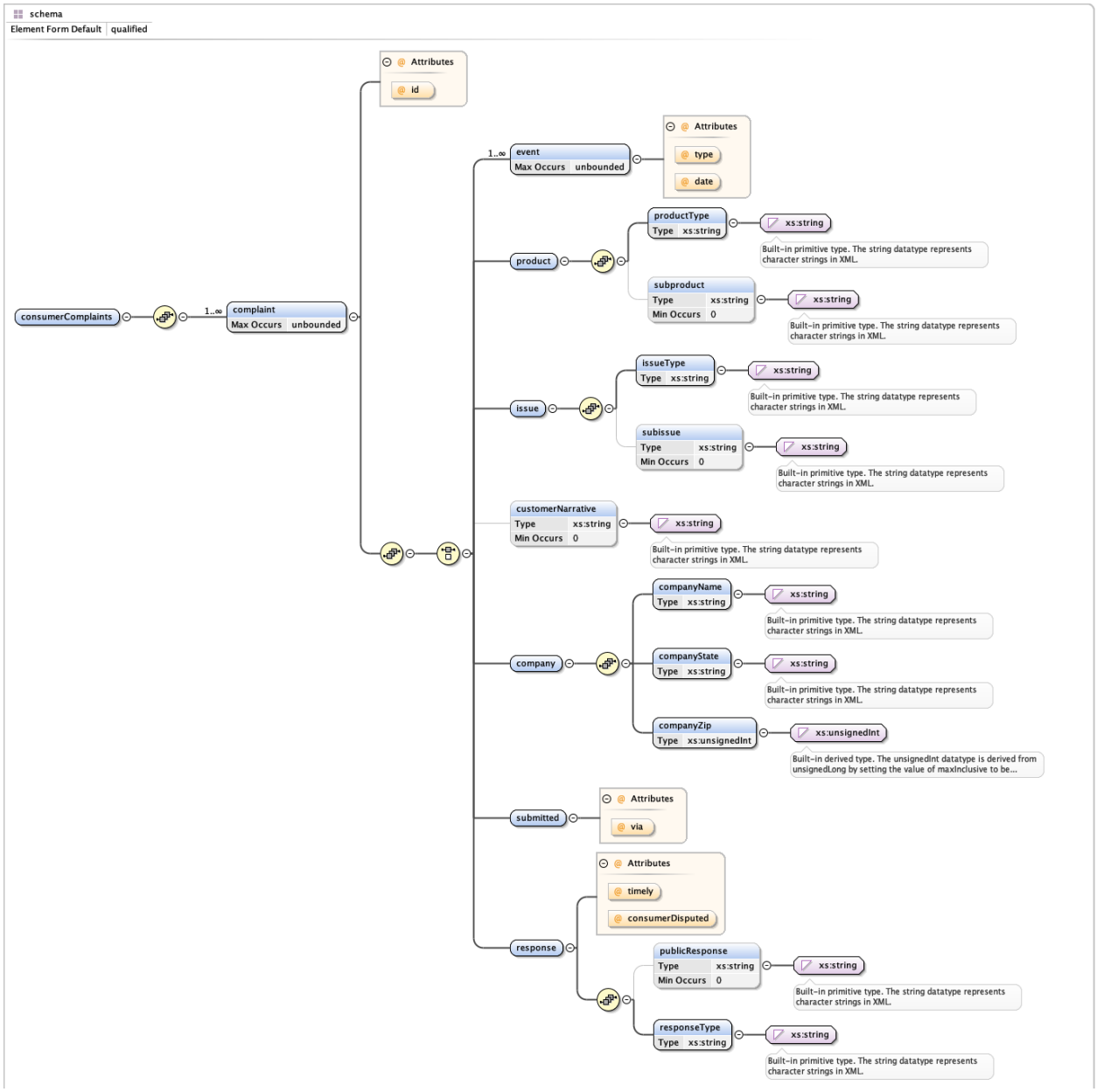
- More than one complaint record can be stored in the root element consumerComplaints
- Within the complaint element, there are seven sub-elements defined including event, product, issue, consumerNarrative, company, submitted and response. They can be arranged in two ways.
- There are multiple events allowed, and the consumerNarrative and submitted element may be optional.
- Sub-elements including sub-issue, sub-product and publicResponse are optional.
- In response, the ConsumerDisputed attribute can only have one of two values, N or Y. The Timely attribute can only have one of two values, yes or no.

2. **[5 points]** Create an XSD for each XML file and validate your XML file against the XSD. Compare the DTD of each file with its corresponding XSD and document the differences (if any) you observe. Your choices of attributes, constraints, and default values must be justified. You will lose points if we see useless attributes,unnecessary constraints, or unjustified default values in your XSD. Ensure that your XML documents validate against your XSDs. What purpose do you think each of these has?

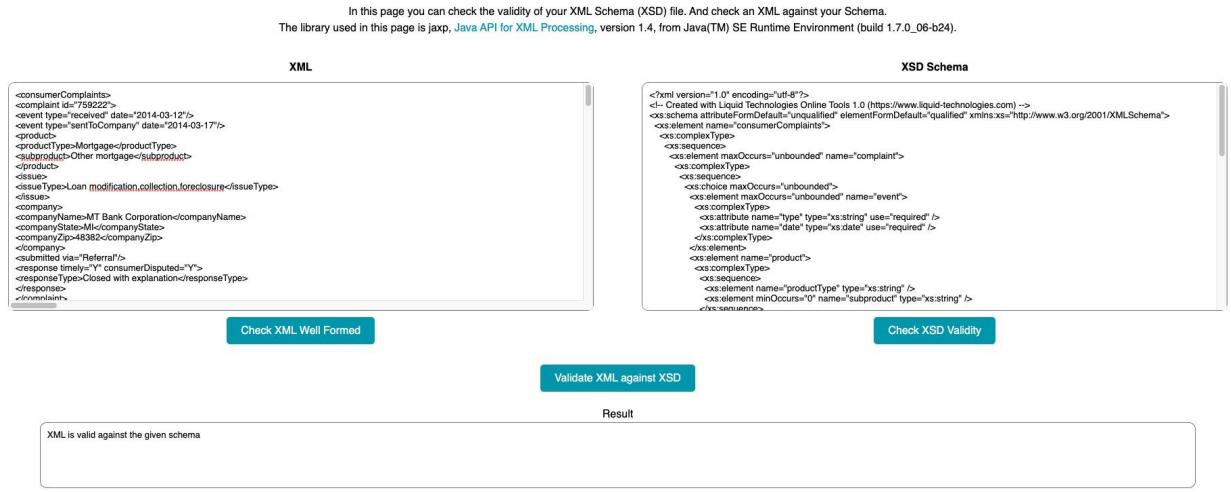
The XSD XML schema for file A is designed manually and presented as below (For further detail, refer to file “step2_file_a_xsd.xsd”):

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">

  <xs:element name="consumerComplaints">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="complaint" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:choice>
                <xs:element name="event" maxOccurs="unbounded">
                  <xs:complexType>
                    <xs:attribute name="type"/>
                    <xs:attribute name="date"/>
                  </xs:complexType>
                </xs:element>
                <xs:element name="product">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="productType" type="xs:string"/>
                      <xs:element name="subproduct" type="xs:string" minOccurs="0"/>
                    </xs:sequence>
                  </xs:complexType>
                </xs:element>
                <xs:element name="issue">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="issueType" type="xs:string"/>
                      <xs:element name="subissue" type="xs:string" minOccurs="0"/>
                    </xs:sequence>
                  </xs:complexType>
                </xs:element>
                <xs:element name="customerNarrative" type="xs:string" minOccurs="0"/>
                <xs:element name="company">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="companyName" type="xs:string"/>
                      <xs:element name="companyState" type="xs:string"/>
                      <xs:element name="companyZip" type="xs:unsignedInt"/>
                    </xs:sequence>
                  </xs:complexType>
                </xs:element>
                <xs:element name="submitted">
                  <xs:complexType>
                    <xs:attribute name="via"/>
                  </xs:complexType>
                </xs:element>
                <xs:element name="response">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="publicResponse" type="xs:string" minOccurs="0"/>
                      <xs:element name="responseType" type="xs:string"/>
                    </xs:sequence>
                    <xs:attribute name="timely"/>
                    <xs:attribute name="consumerDisputed"/>
                  </xs:complexType>
                </xs:element>
              </xs:choice>
            </xs:sequence>
            <xs:attribute name="id"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
```


schema
Element Form Default qualified
consumerComplaints
1..∞
complaint
Max Occurs unbounded
Attributes
id
1..∞
event
Max Occurs unbounded
Attributes
type
date
product
productType
Type xs:string
xs:string
Built-in primitive type. The string datatype represents character strings in XML.
subproduct
Type xs:string
Min Occurs 0
xs:string
Built-in primitive type. The string datatype represents character strings in XML.
issue
issueType
Type xs:string
xs:string
Built-in primitive type. The string datatype represents character strings in XML.
subissue
Type xs:string
Min Occurs 0
xs:string
Built-in primitive type. The string datatype represents character strings in XML.
customerNarrative
Type xs:string
Min Occurs 0
xs:string
Built-in primitive type. The string datatype represents character strings in XML.
company
companyName
Type xs:string
xs:string
Built-in primitive type. The string datatype represents character strings in XML.
companyState
Type xs:string
xs:string
Built-in primitive type. The string datatype represents character strings in XML.
companyZip
Type xs:unsignedInt
xs:unsignedInt
Built-in derived type. The unsignedInt datatype is derived from unsignedLong by setting the value of maxInclusive to be...
submitted
Attributes
via
response
Attributes
timely
consumerDisputed
publicResponse
Type xs:string
Min Occurs 0
xs:string
Built-in primitive type. The string datatype represents character strings in XML.
responseType
Type xs:string
xs:string
Built-in primitive type. The string datatype represents character strings in XML.

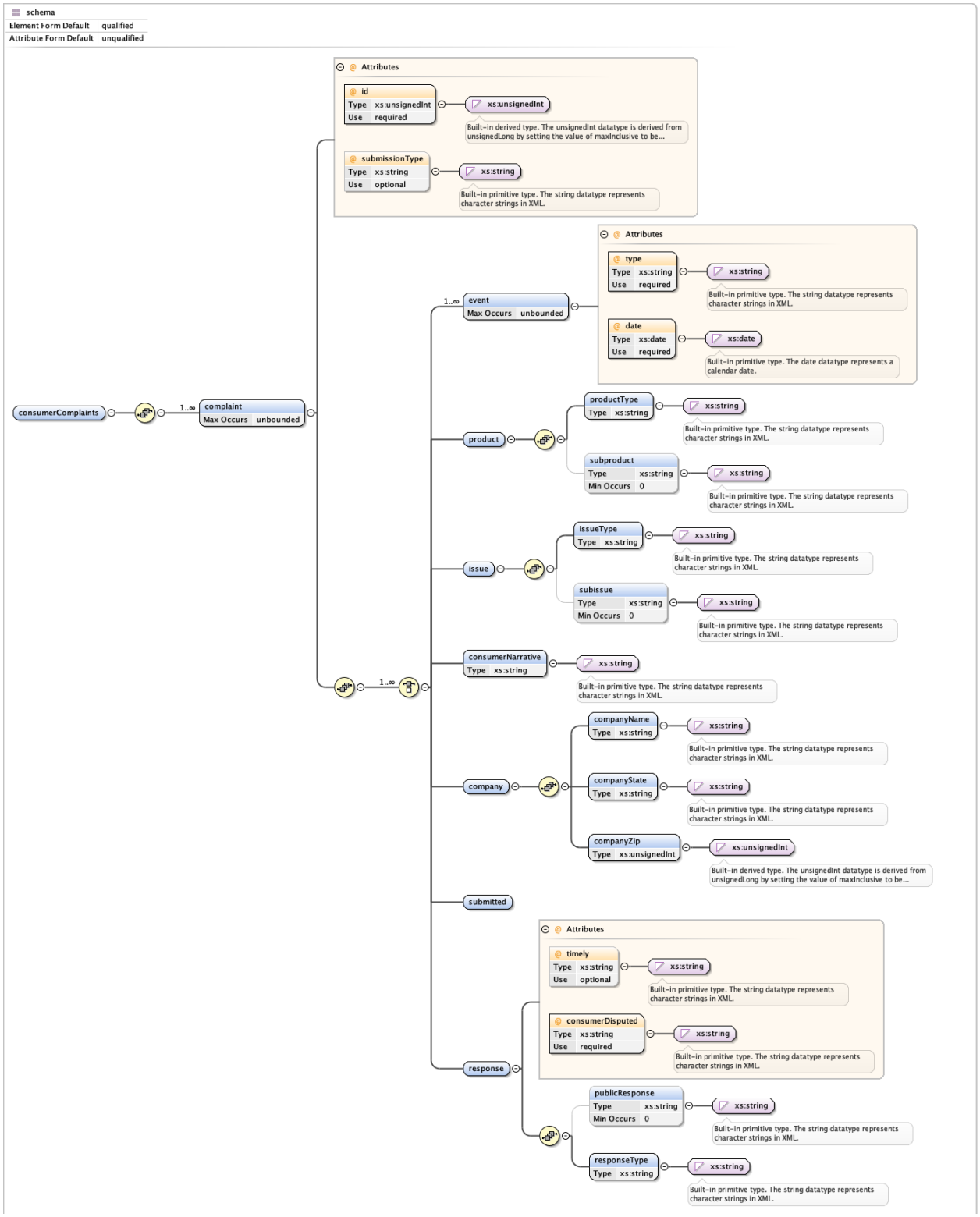
I used the W3C XML Schema (XSD) Validation online website: https://www.utilities-online.info/xsdvalidation to validate the XSD and XML documents for file A. The result is shown as “XML is valid against the given schema”.



Additionally, The XSD XML schema for file B is designed manually and presented as below (For further detail, refer to file “step2_file_b_xsd.xsd”):

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">

  <xs:element name="consumerComplaints">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="complaint" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:choice>
                <xs:element name="event" maxOccurs="unbounded">
                  <xs:complexType>
                    <xs:attribute name="type"/>
                    <xs:attribute name="date"/>
                  </xs:complexType>
                </xs:element>
                <xs:element name="product">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="productType" type="xs:string"/>
                      <xs:element name="subproduct" type="xs:string" minOccurs="0"/>
                    </xs:sequence>
                  </xs:complexType>
                </xs:element>
                <xs:element name="issue">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="issueType" type="xs:string"/>
                      <xs:element name="subissue" type="xs:string" minOccurs="0"/>
                    </xs:sequence>
                  </xs:complexType>
                </xs:element>
                <xs:element name="customerNarrative" type="xs:string" minOccurs="0"/>
                <xs:element name="company">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="companyName" type="xs:string"/>
                      <xs:element name="companyState" type="xs:string"/>
                      <xs:element name="companyZip" type="xs:unsignedInt"/>
                    </xs:sequence>
                  </xs:complexType>
                </xs:element>
                <xs:element name="submitted" minOccurs="0"> </xs:element>
                <xs:element name="response">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="publicResponse" type="xs:string" minOccurs="0"/>
                      <xs:element name="responseType" type="xs:string"/>
                    </xs:sequence>
                    <xs:attribute name="timely"/>
                    <xs:attribute name="consumerDisputed"/>
                  </xs:complexType>
                </xs:element>
              </xs:choice>
            </xs:sequence>
            <xs:attribute name="id"/>
            <xs:attribute name="submissionType"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```


schema
Element Form Default qualified
Attribute Form Default unqualified
consumerComplaints
1..∞
complaint
Max Occurs unbounded
@ Attributes
@ id
Type xs:unsignedInt
Use required
xs:unsignedInt
Built-in derived type. The unsignedInt datatype is derived from unsignedLong by setting the value of maxInclusive to be...
@ submissionType
Type xs:string
Use optional
xs:string
Built-in primitive type. The string datatype represents character strings in XML.
1..∞
event
Max Occurs unbounded
@ Attributes
@ type
Type xs:string
Use required
xs:string
Built-in primitive type. The string datatype represents character strings in XML.
@ date
Type xs:date
Use required
xs:date
Built-in primitive type. The date datatype represents a calendar date.
product
productType
Type xs:string
xs:string
Built-in primitive type. The string datatype represents character strings in XML.
subproduct
Type xs:string
Min Occurs 0
xs:string
Built-in primitive type. The string datatype represents character strings in XML.
issue
issueType
Type xs:string
xs:string
Built-in primitive type. The string datatype represents character strings in XML.
subissue
Type xs:string
Min Occurs 0
xs:string
Built-in primitive type. The string datatype represents character strings in XML.
consumerNarrative
Type xs:string
xs:string
Built-in primitive type. The string datatype represents character strings in XML.
company
companyName
Type xs:string
xs:string
Built-in primitive type. The string datatype represents character strings in XML.
companyState
Type xs:string
xs:string
Built-in primitive type. The string datatype represents character strings in XML.
companyZip
Type xs:unsignedInt
xs:unsignedInt
Built-in derived type. The unsignedInt datatype is derived from unsignedLong by setting the value of maxInclusive to be...
submitted
response
@ Attributes
@ timely
Type xs:string
Use optional
xs:string
Built-in primitive type. The string datatype represents character strings in XML.
@ consumerDisputed
Type xs:string
Use required
xs:string
Built-in primitive type. The string datatype represents character strings in XML.
publicResponse
Type xs:string
Min Occurs 0
xs:string
Built-in primitive type. The string datatype represents character strings in XML.
responseType
Type xs:string
xs:string
Built-in primitive type. The string datatype represents character strings in XML.

I used the W3C XML Schema (XSD) Validation online website: https://www.utilities-online.info/xsdvalidation to validate the XSD and XML documents for file B. The result is shown as “XML is valid against the given schema”.

Comparing the DTD of each file with its corresponding XSD, there exist several differences:

File A:

- DTD requires the attribute date and type in event element.
- In response of DTD file, the ConsumerDisputed and Timely attribute can only have one of two values, N or Y

File B:

- The seven elements in DTD can be arranged in two ways.
- The attribute submissionType of the event can only have one of five values, Referral, Web, Phone, InPerson or Mail.
- In response to the DTD file, the ConsumerDisputed attribute can only have one of two values, N or Y. The Timely attribute can only have one of two values, yes or no.

Purpose of designing DTD and XML XSD schema:

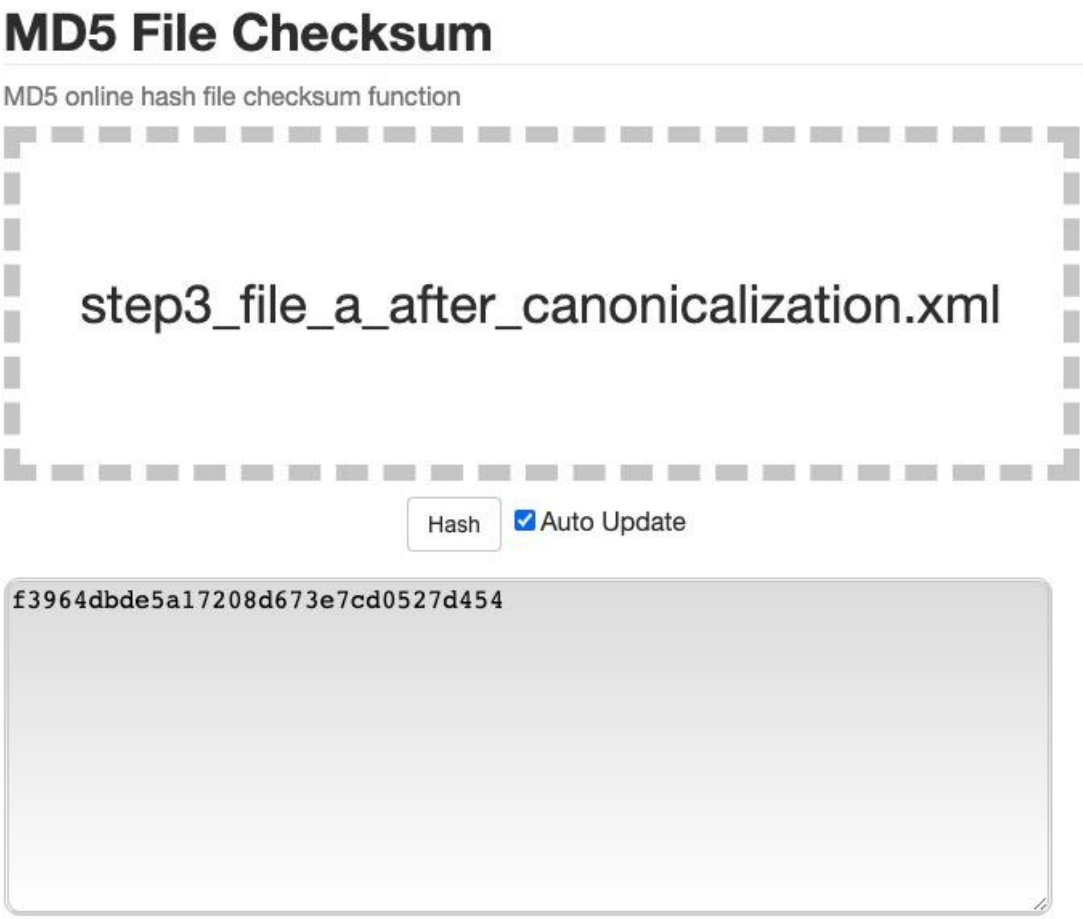
- A DTD document can assist in defining the legal building blocks of an XML document. In addition, it also defines the XML document structure with a list of elements and attributes.
- An XSD XML schema is also used to define the XML document structure with a list of elements and attributes, what is more, XSD XML schema can also define the data type that are included in the XML document. On the contrary, DTD document does not support data type.
- An XSD XML schema supports the number and order of child elements, while the DTD document cannot support it.
- By designing the XSD XML schema, we can easily derive new elements from the existing elements. However, it is difficult to derive new child elements in a DTD document.

3. Evidence of understanding of canonicalization (successful canonicalization) [step 3]

[25 points] Canonicalize the two data files and run checksums again to check for equivalence. Please make sure to document your checksum results (which you will report in step 6). Optionally, you may write and employ a script to conduct the canonicalization process; if you do, please make sure to submit a well-documented, readable text file of the script. For an example of canonicalization, see the videos for week 10.

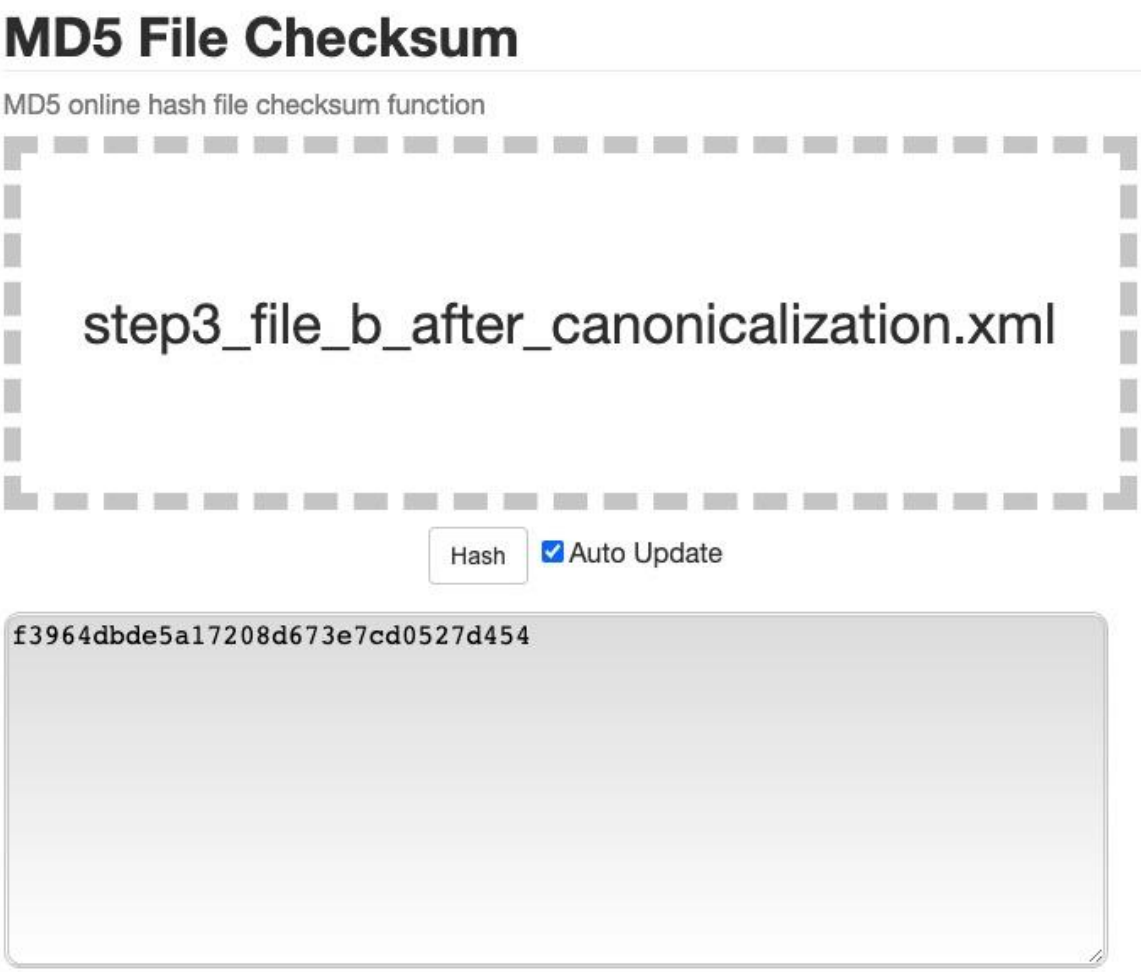
For further details of the canonicalized file A, refer to:
file “step3_file_a_after_canonicalization.xml” in the attachment.

The MD5 file checksum result for canonicalized file A is
f3964dbde5a17208d673e7cd0527d454.



For further details of the canonicalized file B, refer to:
file “step3_file_b_after_canonicalization.xml” in the attachment.

The MD5 file checksum result for canonicalized file B is
f3964dbde5a17208d673e7cd0527d454.



For further information regarding the canonicalization process, refer to the answer to the question “Describe your process for canonicalization (i.e., decisions, actions, representation selection, attribute issues, provenance decisions). Report the checksum values after canonicalization” in session 6.

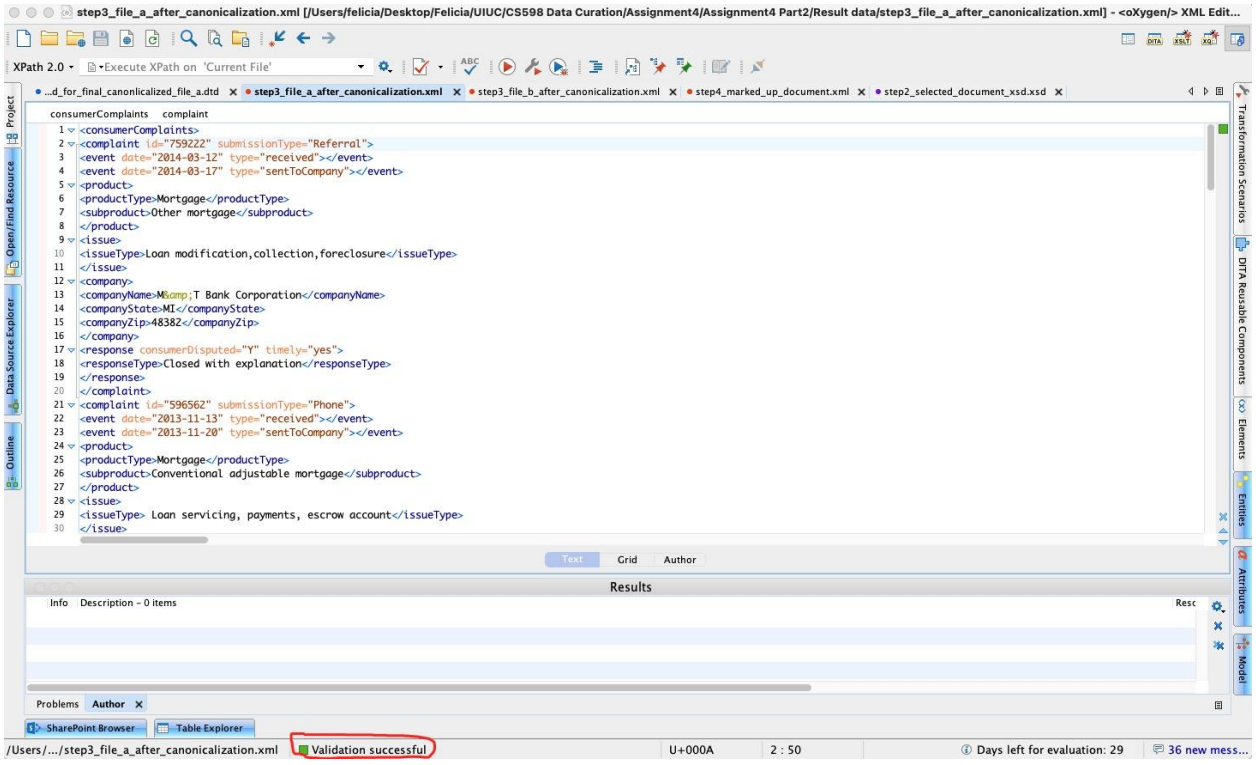
4. **[10 points]** Create and document the DTD of the final, canonicalized data file from step 3 above.

For further details of the external DTD file of the canonicalized XML file A, refer to: file “step4_dtd_for_final_canonlicalized_file_a.dtd” in the attachment.
For further details of the external DTD file of the canonicalized XML file B, refer to: file “step4_dtd_for_final_canonlicalized_file_b.dtd” in the attachment.

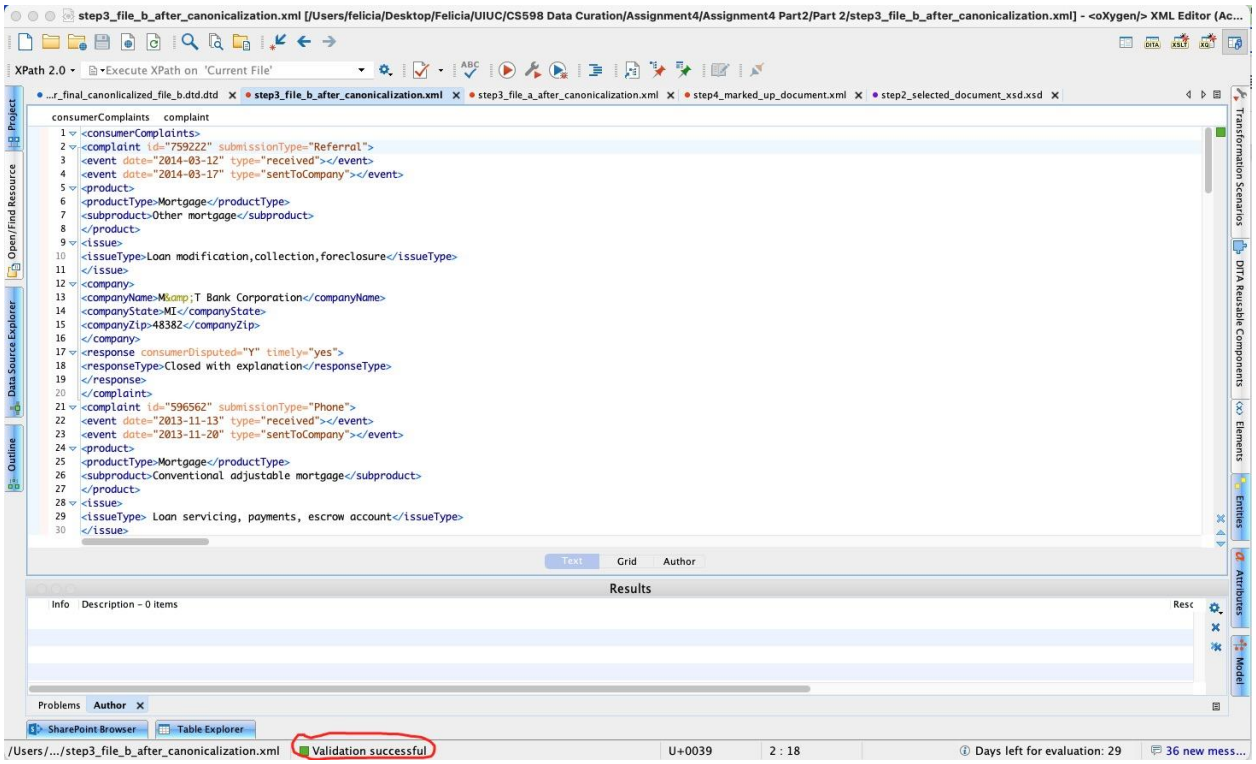
5. **[5 points]** Be sure that all of your files are validated as they will be assessed for compliance to their DTDs. You must test whether your document will validate against your DTD.

*Validation tools **may** require an internal DTD, which means that you must paste your DTD into your XML document, below the XML declaration and above the root element of your XML document. Note also that the syntax for internal vs. external DTDs is slightly different.*

For the DTD design and XML document of canonicalized file A, I used the Oxygen XML editor to validate the file “step4_dtd_for_final_canonlicalized_file_a.dtd”and file “step3_file_a_after_canonicalization.xml”. The result is shown as “Validation successfully”.



For the DTD design and XML document of canonicalized file A, I used the Oxygen XML editor to validate the file “step4_dtd_for_final_canonlicalized_file_a.dtd”and file “step3_file_a_after_canonicalization.xml”. The result is shown as “Validation successfully”.



6. **[10 points]** In a separate document, answer the following reflection prompts:

Q:

How may your canonicalization support the overarching goals of data curation (revisit objectives and activities of Week 1)? Discuss at least **10** activities/objectives, how many of you were able to achieve, and how many of them were not able to.

A:

The canonicalization between file A and file B is able to support the following overarching goals of data curation:

- **Collection**: Support the collection and acquisition of data.

  Explanation: While migrating the data sets from the old system to the new system, conducting data canonicalization provides support to collect and acquisite the data accurately.

- **Organization**: Employ an appropriate data model and use appropriate standards

  Explanation: While performing the data canonicalization between file A and file B, appropriate data models like DTD documents, XSD documents are designed to validate the XML files. For example, the DTD files for file A (old system) and file B (new system) define the legal building blocks of the XML files, in addition, the DTD files also define the constraint of the elements to ensure the standards of the data transformation.

- ***Storage****:* Support reliable and effective storage

  Explanation: While migrating the data to a new system, the business needs to ensure the completeness and sufficiency of data transformation, and needs to ensure that high-quality data is delivered to the new platform. Canonicalizing file

A and file B will benefit the data consistency so that data reliability and data security are ensured.

- **Preservation**: Ensure that the data can be understandable and usable in the future.

  Explanation: Data canonicalization plays an important role in maintaining a documented strategy, which includes not just bit sequence preservation and syntax documentation, but also the documentation of semantics for data elements and the generation and preservation of all metadata needed to ensure that the data is useable and understandable, and can be authenticated and audited for provenance.

- **Discoverability**: Support the ability to search for and locate relevant data

  Explanation: While conducting data canonicalization between file A (old system) and file B (new system), I have revisited the metadata to support searching for and finding relevant data in relevant formats.

- **Access**: Support the ability to retrieve and distribute data
  Explanation: While conducting data canonicalization between file A (old system) and file B (new system), the XSD XML schemas and DTD documents are designed, which clearly show the structure of the databases. Therefore, it is beneficial to maintain systems, tools and metadata that support the efficient and reliable retrieval and distribution of data.

- **Workflow**: Support the ability to systematize data workflows

  Explanation: While conducting data canonicalization between file A (old system) and file B (new system), relevant XSD schema, DTD documents and XML documents have been designed to support systematizing the data workflows.

- **Reproducibility:** Support ability to reproduce results, ensuring scientific validity and reliability

Explanation: The detailed explanation regarding how data canonicalization will benefit reproducibility will be answered in the question “How does the way data is represented impact reproducibility?” below.

The process to perform data canonicalization between file A (old system) and file B (new system) is also a quality control and improvement process, which ensures the data consistency for data transformation. Consequently, the following curation activities can also be achieved during this process.

- **Identification**: Support the ability to identify, authenticate, and validate data
- **Integration**: Support integration of data from different sources using different data model
- **Reformatting:** Support reformatting for use by different tools or to match new format standards
- **Sharing:** Support sharing data between researchers, teams, and institutions
- **Communication:** Support representation, publishing, and visualizations that provide insight
- **Provenance:** Support identifying what inputs, processes, and calculations are responsible for data values
- **Modification:** Support management of corrections and updates

Even though carrying out the data canonicalization between file A (old system) and file B (new system) is advantageous in multiple aspects, the following aspects should still be improved and implemented since the data canonicalization is unable to achieve these goals.

- **Compliance:** Ensure compliance to legal, regulatory, and local policy requirements
- **Security:** Ensure that data is secure from tampering or inappropriate access and distribution

Q:

Which additional curation activities would you recommend to enhance the data set for future discovery and use?

A:

- The dataset A and dataset B are both related to financial institutions, however, while carrying out the data curation process, the compliance requirements and relevant policies are not considered. Nevertheless, in the real-world scenario, the data curation should take the **compliance** into consideration and ensure that the datasets fulfill the legal, regulatory and local policy requirements.
- There is no restriction for **Security** requirements in the customer complaint case. In the real-world scenario, the business needs to consider who should have the access to the database and how to set up the limitation. In addition, while developing the data curation in an organization, the company should involve methods for controlling access and determining user identity and privileges, as well as data identity, authentication, and validation.

Q:

How does the way data is represented impact reproducibility?

A:

Data curation for reproducibility includes documenting not only data collection and management, but also documenting processing and analysis. On the one hand, in order to document analysis, we need to determine needs, and develop relevant data models and *metadata,* and reformat, correct, or update data. On the other hand, to document processing, we need to ensure success and efficiency by managing the development of appropriate organizational units and roles, providing training, advocating for change, and managing curatorial activities.

Before carrying out the data canonicalization, a detailed analysis has been performed and documented to identify the similarities and differences between file A (old system) and file B (new system) in step 1.

Before canonicalizing the two files, the XSD XML schema for both file A and file B are generated in step 2 to better understand the structure of the two XML files, and the data models and workflows for file A and file B are presented respectively. Based on the XSD interpretation, we are able to further analyze and document the differences between the old dataset and new dataset. What is more, validation between XSD and XML files is also performed to make sure that the files are validated against each other.

To ensure data consistency and sufficiency, it is of utmost importance to deal with the differences between the two datasets, which is also an indispensable part of data transformation and ensuring data reproducibility. In step 3, the data is cleaned including correcting spelling errors, finding missing values or numbers and identifying incorrect data entries. After cleaning the two datasets, the corresponding XML files are also created and the MD5 file checksums towards the XML files are carried out to check the canonicalization.

Additionally, external DTD files are also generated respectively in step 4 and validated against the corresponding XML files in step 5. The DTD files allow us to define the structure of the XML documents, and control how to mark up the XML documents.

After finishing all the steps mentioned above, it leads to final and canonicalized XML and DTD documents, which are well-formed and validated and are capable of ensuring data integrity, data consistency and no information loss. As a consequence, a readable, understandable, reliable and well-structured dataset can be reproduced during this procedure.

Q:

Describe your process for canonicalization (i.e., decisions, actions, representation selection, attribute issues, provenance decisions). Report the checksum values after canonicalization.

A:

In the first step of the data canonicalization, I attempted to identify all the differences between the original file A and file B. To successfully achieve it, I leveraged a code diff tool: https://www.w3docs.com/tools/code-diff/ and also checked the differences manually, and have successfully identified a few differences. There are some difference comparison examples shown as below:

Data inconsistency 1:

File A	File B
<consumerComplaints> <complaint id="759222"> <event type="received" date="2014-03-12"/> <event type="sentToCompany" date="2014-03-17"/> <product> <productType>Mortgage</productType> <subproduct>Other mortgage</subproduct> </product> <issue> <issueType>Loan modification,collection,foreclosure</issueType> </issue> <company>	<consumerComplaints> <complaint id="759222" submissionType="Referral"> <event type="received" date="2014-03-12"/> <event type="sentToCompany" date="2014-03-17"/> <product> <productType>Mortgage</productType> <subproduct>Other mortgage</subproduct> </product> <issue> <issueType>Loan modification,collection,foreclosure</issueType> </issue> <company>

<companyName>MT Bank Corporation</companyName> <companyState>MI</companyState> <companyZip>48382</companyZip> </company> <submitted via="Referral"/> <response timely="Y" consumerDisputed="Y">	<companyName>MT Bank Corporation</companyName> <companyState>MI</companyState> <companyZip>48382</companyZip> </company> <response timely="yes" consumerDisputed="Y">

Data inconsistency 2:

File A	File B
<complaint id="596562"> <event type="received" date="2013-11-13"/> <event type="sentToCompany" date="2013-11-20"/> <product> <productType>Mortgage</productType> <subproduct>Conventional adjustable mortgage</subproduct> </product>	<complaint id="596562" submissionType="Phone"> <event date="2013-11-13" type="received"/> <event type="sentToCompany" date="2013-11-20"/> <product> <productType>Mortgage</productType> <subproduct>Conventional adjustable mortgage</subproduct> </product>

Data inconsistency 3:

File A	File B
<response timely="Y" consumerDisputed="N"> <responseType>Closed with monetary relief</responseType> </response> </complaint> <complaint id="2364257"> <event type="received" date="2017-02-28"/> <event type="sentToCompany" date="2017-02-28"/> <product> <productType>Credit card</productType> </product>	<response consumerDisputed="N" timely="yes"> <responseType>Closed with monetary relief</responseType> </response> </complaint> <complaint id="2364257"> <event date="2017-02-28" type="received"/> <event type="sentToCompany" date="2017-02-28"/> <product> <productType>Credit card</productType> </product>

After figuring out all the differences between file A and file B, the following steps are performed to canonicalize the two files. The canonicalization steps are mainly done manually in ATOM by leveraging some online diff checker tool including https://www.diffchecker.com/diff/, https://www.w3docs.com/tools/code-diff/, https://www.editpad.org/tool/diff-checker

The steps performed to canonicalize the two datasets can be referred to the following websites:

https://www.w3.org/TR/xml-c14n/

https://obj-sys.com/docs/xbv20/JavaCSUsersGuide/ch07s01.html

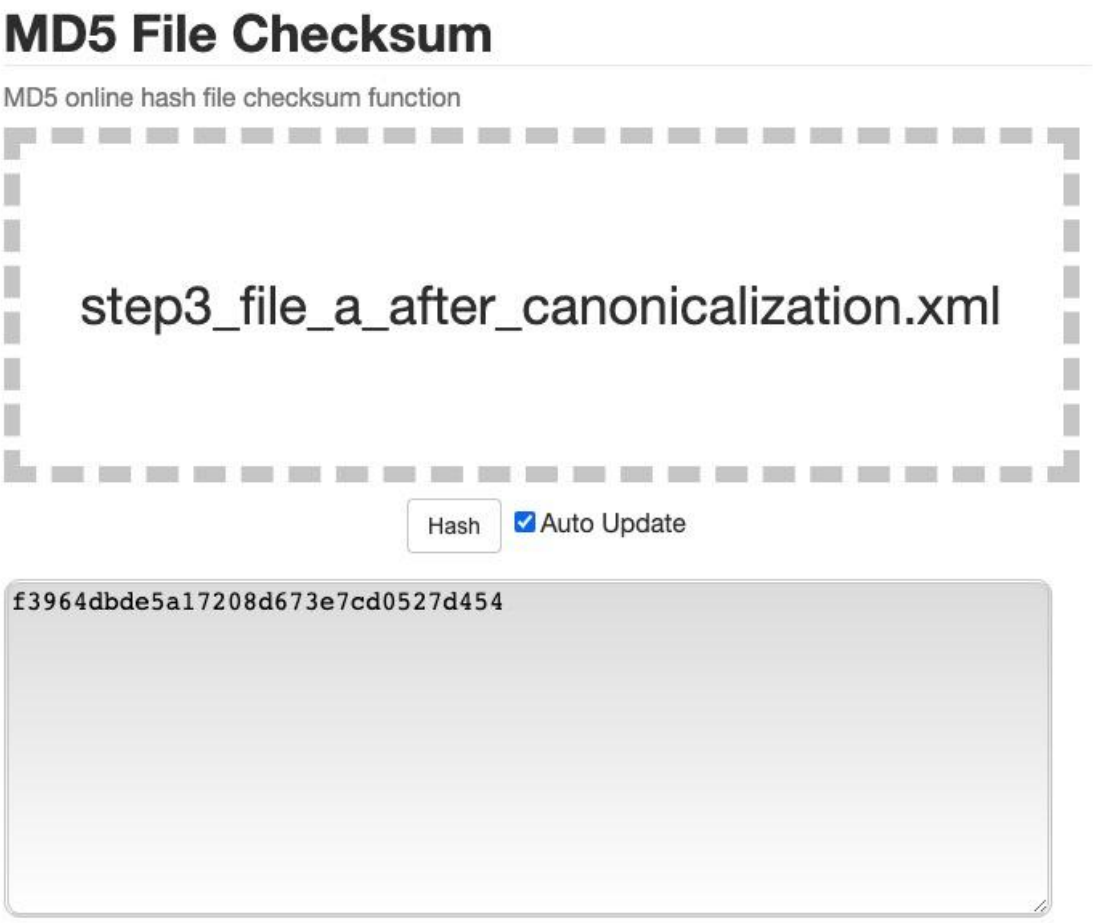
https://www.w3.org/TR/xml-exc-c14n/

1. Convert the two files to single character encoding to ensure that the two files are encoded as “utf-8”
2. Before parsing, normalize the line breaks to #xA. For example, for elements like “customerNarrative” and “publicResponse”, normalized line-ends are applied to ensure that the lines are in one line

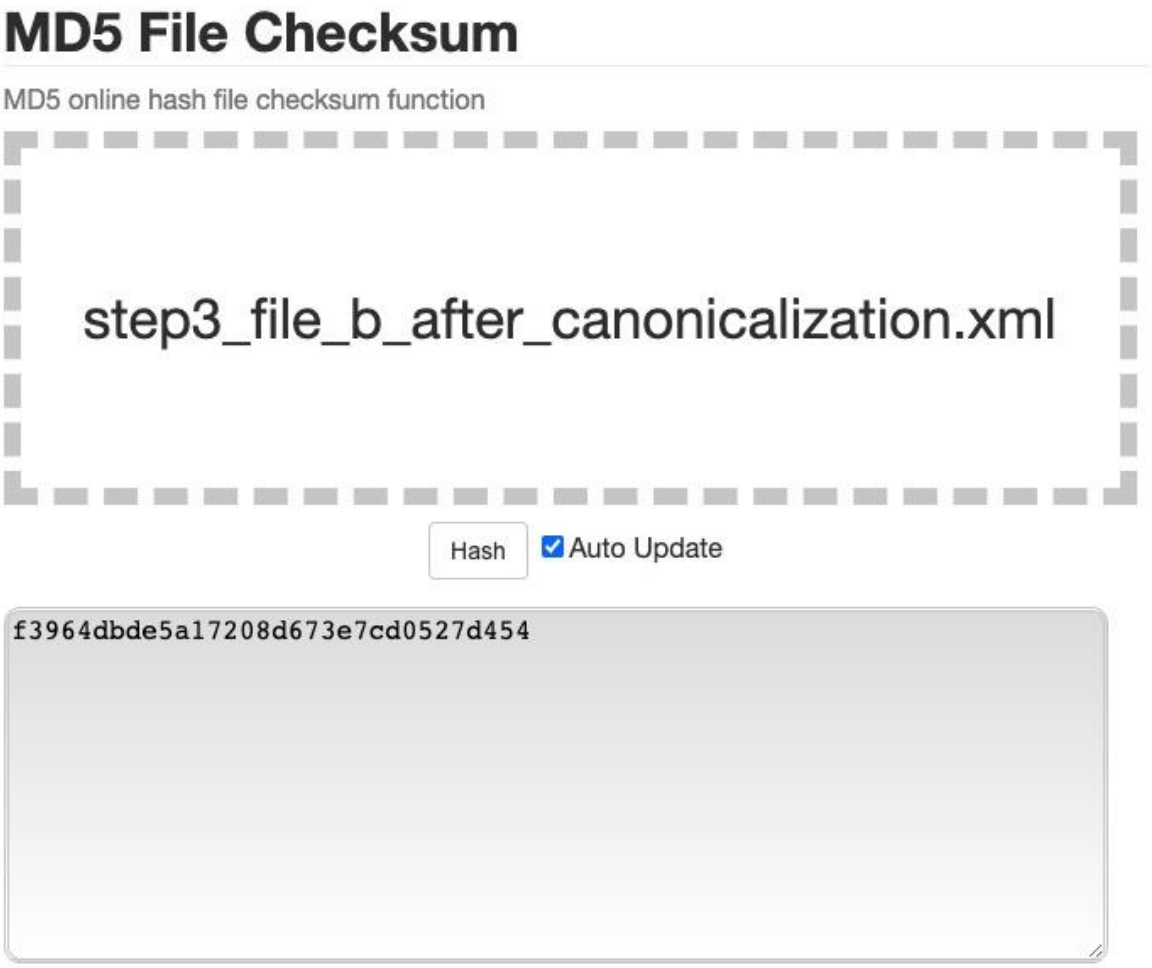
3. Check if there exists CDATA section, if it is, substitute the CDATA section with corresponding character content
4. Remove XML declaration and document type declaration if applicable
5. Check if there exist empty elements, if it is, switch to start-end tag pairs
6. Trim off excessive and redundant white spaces in the text. For example, for the elements like “customerNarrative” and “publicResponse”, unnecessary white spaces are removed
7. Check if there are whitespaces outside of the elements, if it is, normalize the whitespaces and ensure that the line is one line
8. Check if the white spaces between the text characters remain unchanged
9. Check if the attribute value delimiters are set to double quotation marks
10. Check if there are special characters like @#$%^&* in attribute values, if it is, remove the special characters. For example, for element “companyName”, there exist special characters, which should be removed
11. Check if there exist superfluous namespace declarations, if it is, it should be removed from the elements
12. Check if all the elements contain default attributes, no issue identified for this step
13. Check if we are able to convert the attributes to elements since attributes are difficult to expand in the future
14. The namespace declarations and attributes of each element are rearranged in lexicographic order
15. After applying the above steps to file A and file B, I still noticed that there are differences between file A and file B. Hence, I determined to process with canonicalized file B to sync it up with canonicalized file A
16. Since there is no attribute “submitVia” in the original file B design, I decided to get rid of this attribute. As is shown in the comparison table, in canonicalized file B, under element “complaint”, some contains attribute “submissionType” and some did not, hence, I grasped all the attribute “Via” under element “submitted” from file A and added this part to attribute “submissionType” in file B if it is missing.

17. By comparing the original DTD design of file A and file B, I noticed that for attribute “response timely”, file A used “Y|N” while file B used “yes|no”. According to the curatorial activities, data must be frequently reformatted in order to support new tools, new versions of existing tools, or to meet new format standards. I determined to keep the “yes|no” format, which adhered to “Reformatting” curatorial activity.
18. Applied similar steps to identify and eliminate all the differences
19. Create a new version of file A XML document and file B XML document, generate corresponding DTD files and validate the two files respectively
20. Check the MD5 file checksum results, which are displayed as below

The MD5 file checksum result for canonicalized file A is f3964dbde5a17208d673e7cd0527d454.



The MD5 file checksum result for canonicalized file B is f3964dbde5a17208d673e7cd0527d454.



The MD5 file checksum results for canonicalized file A and canonicalized file B are consistent.