Big Data – Exercises

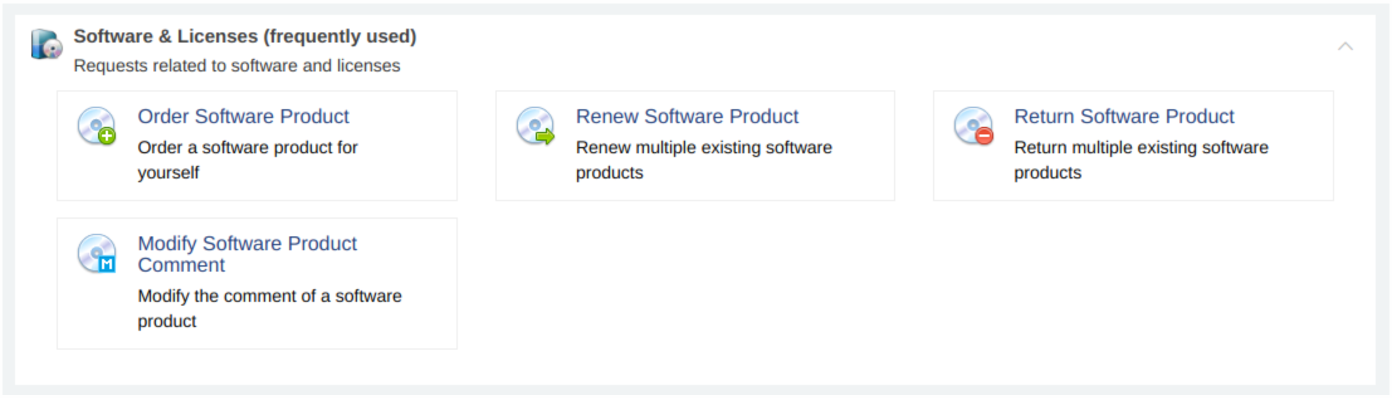
Fall 2021 – Week 4 – ETH Zurich

Introduction

This exercise will cover XML and JSON well-formedness.

For the next four weeks you will be using oXygen 23.1, an XML/JSON development IDE. Before starting, make sure oXygen is installed and working on your computer. You can download the required licence from the ETH IT shop:

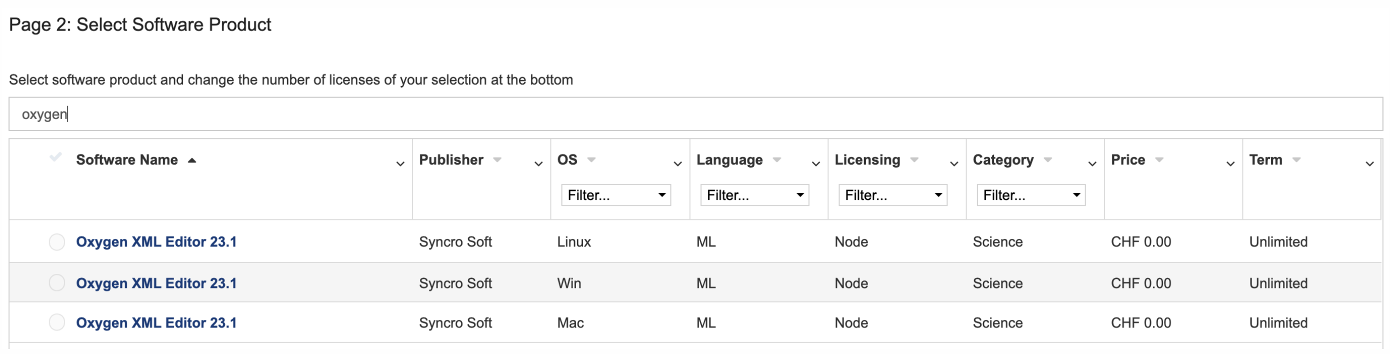
1. Login with your ETH credentials and go to **Software & Licenses** > **Order Software Product**.



Software & Licenses (frequently used)
Requests related to software and licenses

Order Software Product Order a software product for yourself	Renew Software Product Renew multiple existing software products	Return Software Product Return multiple existing software products
Modify Software Product Comment Modify the comment of a software product		

2. Look for "oxygen" and select the version that fits your local setup.



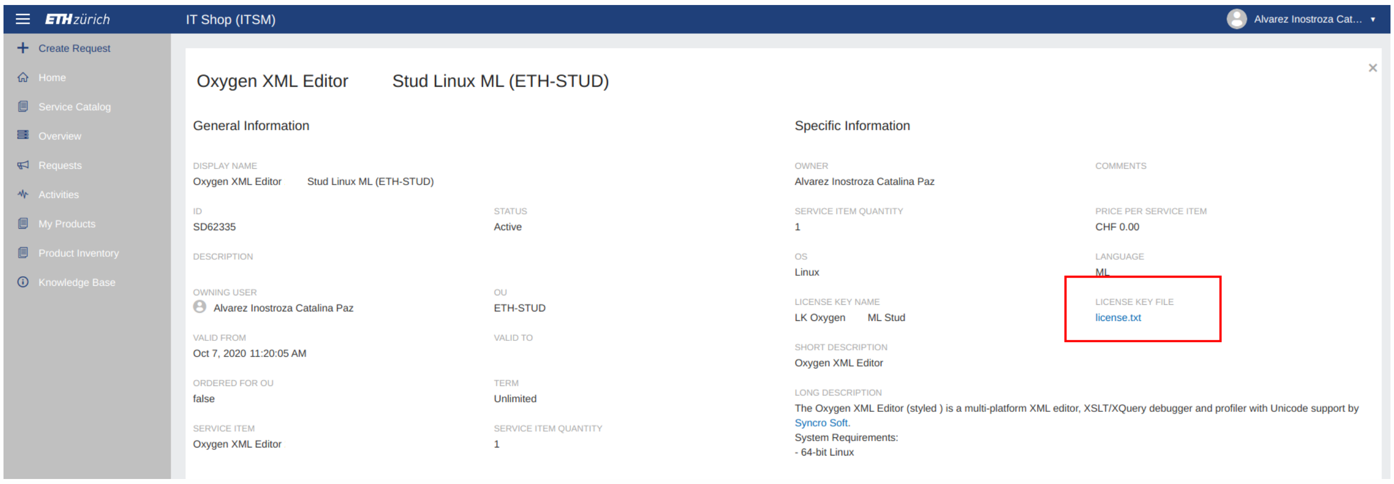
Page 2: Select Software Product

Select software product and change the number of licenses of your selection at the bottom

oxygen

Software Name	Publisher	OS	Language	Licensing	Category	Price	Term
		Filter...	Filter...	Filter...	Filter...		
Oxygen XML Editor 23.1	Syncro Soft	Linux	ML	Node	Science	CHF 0.00	Unlimited
Oxygen XML Editor 23.1	Syncro Soft	Win	ML	Node	Science	CHF 0.00	Unlimited
Oxygen XML Editor 23.1	Syncro Soft	Mac	ML	Node	Science	CHF 0.00	Unlimited

3. Click **Next step** at the bottom, and accept the terms of services.
4. Wait until you get the confirmation email (it should take a couple of minutes). Follow the instructions, and download the **license file**:



ETH zürich · IT Shop (ITSM) · Alvarez Inostroza Cat...

Create Request
Home
Service Catalog
Overview
Requests
Activities
My Products
Product Inventory
Knowledge Base

Oxygen XML Editor Stud Linux ML (ETH-STUD)

General Information

DISPLAY NAME Oxygen XML Editor Stud Linux ML (ETH-STUD)	
ID SD62335	STATUS Active
DESCRIPTION	
OWNING USER Alvarez Inostroza Catalina Paz	OU ETH-STUD
VALID FROM Oct 7, 2020 11:20:05 AM	VALID TO
ORDERED FOR OU false	TERM Unlimited
SERVICE ITEM Oxygen XML Editor	SERVICE ITEM QUANTITY 1

Specific Information

OWNER Alvarez Inostroza Catalina Paz	COMMENTS
SERVICE ITEM QUANTITY 1	PRICE PER SERVICE ITEM CHF 0.00
OS Linux	LANGUAGE ML
LICENSE KEY NAME LK Oxygen ML Stud	LICENSE KEY FILE license.txt
SHORT DESCRIPTION Oxygen XML Editor	

LONG DESCRIPTION
The Oxygen XML Editor (styled) is a multi-platform XML editor, XSLT/XQuery debugger and profiler with Unicode support by Syncro Soft.
System Requirements:
- 64-bit Linux

1 XML

1.1 Well-formedness

Correct the following XML documents to be well-formed. Try first to "parse" it in mind, the use oXygen to check.

1.

```xml
<?xml version="1.0"?>
<catalog>
    <!-- Start book list --to be defined -->
   <Book id=`bk101`>
      <author>&cright; Gambardella, Matthew</author>
      <title>XML Developer's Guide</title>
      <genre>Computer</genre>
      <price>44.95€</price>
      <publish_date version='hard' version='soft'>2000-10-01</publish_date>
      <_description lang=en>An `in-depth look` at creating applications
      with XML <for dummies>.</_description>
      <xml_parse>true</xml_parse>
   </book>
</>
```

2.

```xml
<?xml version="1.0" encoding="utf-16"?>
<h:library xmlns:xdc="http://www.xml.com/books" xmlns:h="http://xml.com/library">
    <head><h:title>Book Review</title></head>
    <body/>
        <_xdc:bookreview>
            <xdc:title>XML: A Primer</xdc:title>
            <_table _style='container'>
                <h:tr align="#center">
                    <h:td>Author<h:span>St. Laurent & Tom Faron</h:td></h:span>
                </h:tr>
                <h:tr align="#left">
                    <h:td><xdc:author>Simon St. Laurent</xdc:author></h:td>
                    <h:td><xdc:price>31.98</xdc:price></h:td>
                    <h:td><xdc:#pages>352</xdc:#pages></h:td>
                    <h:td><xdc:_date>1998/01</xdc:_date></h:td>
                    <h:td><xdc:-comment>Love it</xdc:-comment></h:td>
                </h:tr>
            </_table>
        </_xdc:bookreview>
    </body>
</h:library>
```

1.2 Create your own XML

1. Copy the text of the introduction above (including the title until 'your computer') and paste it into oXygen as plain text. Create a possible XML document, having the same context and including formatting (title, sections, style, links, etc.). Make sure your XML is well-formed and save it as `doc1.xml`.
2. Copy the same text into Microsoft Word or OpenOffice and save it XML (both programs allow to export as `.xml` if you use *Save as...*).

Questions

1. Compare the two XML. What differences do you notice?
2. Is this data structured, unstructured, or semi-structured?

1.3 XML Names

Which of the following are well-formed XML tags (i.e. which tag contain a conform XML name)?

1. `<_bar/>`
2. `<123foo/>`
3. `<Foo/>`
4. `<foo 123>`
5. `<foo_123/>`
6. `<foo#123/>`
7. `<foo-123/>`
8. `<foo.123/>`

1.4 Predefined entities

XML has only 5 predefined entities. Connect each escape code to the corresponding value.

1.	`&lt;`	>
2.	`&amp;`	"
3.	`&gt;`	'
4.	`&quot;`	&
5.	`&apos;`	<

2. JSON

2.1 Well-formedness

Correct the following JSON documents to be well-formed. Try first to "parse" them in mind, the use oXygen to check.

1.

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  age: 25,
  "isRetired",
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100",
    'is verified' : "true"
  }
  'phoneNumbers': [
    {
      "type": [["home"]],
      "@number": "212 555-1234"
    },
    {
      "type": [["office"]],
      "@number": "646 555-4567"
    },
    {
      "type": [["mobile"[],
      "@number": "123 456-7890"
    }
  ],
  "children": [],
  "settings": {},
  "spouse": Null
}
```

2.

```
[
    1: {
      "name": 'John'
      "lastname": 'Smith',
      "account": "jsmith"
      "phonenumbers" [{
           "type": "home",
           "1phone": 212-3242,
           "2phone": "545-4568"
       }]
    },
    2: {
      "name": "Jane"
      "lastname": 'Doe',
      "account": "jdoe"
      "phonenumbers" [
      {
           "type": "home",
           "phone": "8989 7685"
      },
      "phone": "545-4568"
      ],
      "account": "janedoe"
    }
]
```

3 Conversions from a relational database

Messages from conversations between users are stored in a SQL table. Translate this table into XML and JSON.

conversation_id	**people**	**sender**	**content**	**timestamp**	**is_read**	**attachment_id**
42	charlie,ari,jesse	charlie	hey, here's the doc ><	1510410193	TRUE	NULL
42	charlie,ari,jesse	charlie	NULL	1510410244	TRUE	doc_6492
42	charlie,ari,jesse	ari	thanks! \o/	1510432987	FALSE	NULL
17	rudy,sage	rudy	look at this cute "bat-cat"! 😻	1500897189	TRUE	img_91847
17	rudy,sage	NULL	aww ♥	1506610190	TRUE	NULL

4 More XML

4.1 HTML vs XHTML

As mentioned during the lectures, HTML has a similar structure to XML, but it's more lenient, meaning, that valid HTML documents are not necessarily valid XML documents. XHTML is an extension to HTML that conforms to the XML standard. Is the following correct HTML? Is it correct XML? Is it XHTML?

```html
<html>
  <head>
    <title>Untitled</title>
  </head>
  Dear jane <br>
  <p>You are invited at the weekly meeting
  <p>Yours sincerely, <br>
  John
</html>
```

4.2 XML Namespaces

1. Is the following XML file well-formed?
2. What are the namespaces of each attribute and each element?
3. What's wrong with this file? Fix it so it is well-formed, follows best practices, and each element uses the correct namespace.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<foo
xmlns="http://xmlrepo.test/foo.xml"
xmlns:foo="http://xmlrepo.test/foo.xml"
xmlns:math="http://xmlrepo.test/math.xml">
    <bar:baz xmlns:bar="http://xmlrepo.test/bar.xml" bar:attr="some attribute" lalala="some other attribute">
        <svg xmlns="http://xmlrepo.test/svg.xml">
            <textbox>
                <math:msup>42</math:msup>
                <foo:plus/>
                <math:msub>17</math:msub>
            </textbox>
            <foo_value id="748">some value</foo_value>
        </svg>
        <svg xmlns:svg="http://xmlrepo.test/svg.xml">
            <svg:textbox>
                <math:msup>42</math:msup>
                <foo:plus/>
                <msub>17</msub>
            </svg:textbox>
            <bar_value id="867">some other value</bar_value>
        </svg>
        <math:othermath/>
    </bar:baz>
</foo>

```

5. From XML to JSON - back to the REST API request result from previous exercise sessions.

In this exercise you are asked to translate the following XML document into a JSON document.
Remember the REST API call we used during the tutorial about Azure Blob Storage. The result was an XML file. Below you can find the result of the request (with some elements removed for simplicity and a second fake blob added to the response). Now that you can parse it, transform it as a JSON file.

```xml
<EnumerationResults ContainerName="https://melaniestorage.blob.core.windows.net/exercise02">
    <Blobs>
        <Blob>
            <Name>picture</Name>
            <Url>https://melaniestorage.blob.core.windows.net/exercise02/picture</Url>
            <Properties>
                <Last-Modified>Wed, 03 Oct 2018 07:22:16 GMT</Last-Modified>
                <Content-Length>136356</Content-Length>
                <Content-Encoding />
                <BlobType>BlockBlob</BlobType>
            </Properties>
        </Blob>
        <Blob>
            <Name>music</Name>
            <Url>https://melaniestorage.blob.core.windows.net/exercise02/music</Url>
            <Properties>
                <Last-Modified>Wed, 03 Oct 2018 07:23:16 GMT</Last-Modified>
                <Content-Length>222222</Content-Length>
                <Content-Encoding />
                <BlobType>BlockBlob</BlobType>
            </Properties>
        </Blob>
    </Blobs>
</EnumerationResults>
```

6. JSON to XML - exploring an open API

In this exercise you can use any open API that answers with a JSON. One such API is: the Star Wars API. Below you can find an (slightly modified) example of the response to the request: https://swapi.dev/api/people/1/. Parse it and transform it to XML.

```json
{
  "name": "Luke Skywalker",
  "height": "172",
  "mass": "77",
  "homeworld": "http://swapi.dev/api/planets/1/",
  "films": [
    "http://swapi.dev/api/films/1/",
    "http://swapi.dev/api/films/2/",
    "http://swapi.dev/api/films/3/",
    "http://swapi.dev/api/films/6/"
  ],
  "starships": [],
  "vehicles": [
    "http://swapi.dev/api/vehicles/14/",
    "http://swapi.dev/api/vehicles/30/"
  ]
}
```

7. XML vs CSV - the limits of tables for heterogeneous data

If your document consists of a collection of heterogeneous objects with different attributes, XML/JSON turns out to be more suited than a comma-separated format to store the data. In this exercise we want to show that denormalization is a good idea in this setting.

You are given the following XML document representing a collection of products available in an online shop selling all kinds of products. In this product catalog each product has different attributes. You are asked to turn this data into a CSV file.

```xml
<productscatalog>
    <product>
        <id> 1 </id>
        <category> BBQ </category>
        <type> Gas </type>
        <height> 120cm </height>
    </product>
    <product>
        <id> 2 </id>
        <category> notebook </category>
        <brand> Apple </brand>
        <specs>
            <RAM> 16Gb </RAM>
            <storage> 128Gb </storage>
        </specs>
    </product>
    <product>
        <id> 3 </id>
        <category> shoes </category>
        <size> 39 </size>
        <model> Heels </model>
    </product>
</productscatalog>
```

1. Write the documents in a CSV format (i.e. in a table).
2. What are the disadvantages of the CSV format compared to the XML format in this case?
3. Give an example of one use case where the CSV format would be more appropriate than the XML format.