EAST WEST UNIVERSITY

CSE479
Web Programming

Nishat Tasnim Niloy

Lecturer

Department of Computer Science and Engineering

Faculty of Science and Engineering

Topic 10

XML and JSON files

XML

eXtensible Markup Language

What is XML?

- eXtensible Markup Language
- Markup language for documents containing structured information
- XML has no mechanism to specify the format for presenting data to the user
- An XML document resides in its own file with an ‘.xml’ extension
- HTML tags tell a browser how to display the document.
- XML tags give a reader some idea what some of the data means.

What is XML Used For?

- XML documents are used to transfer data from one place to another often over the Internet.
- XML is a data transfer technology, not a data storage technology.
- XML files are just text files stored on your machine or another one; they need to be read, parsed and written to, and only your program can do that.

Advantages of XML

- XML is text (Unicode) based.
  - Takes up less space.
  - Can be transmitted efficiently.
- One XML document can be displayed differently in different media.
- XML documents can be modularized. Parts can be reused.

Why XML (1) ?

- Electronic data interchange is critical in today's networked world and needs to be standardized
  - Examples:
    - Banking: funds transfer
    - Education: e-learning contents
    - Scientific data
      - Chemistry: ChemML, …
      - Genetics: BSML (Bio-Sequence Markup Language), …
- Each application area has its own set of standards for representing information
- XML has become the basis for all new generation data interchange formats (markups)

Why XML (2)

- Earlier electronic formats were based on plain text with line headers indicating the meaning of fields
  - Does not allow for nested structures, no standard “type” language
  - Tied too closely to low level document structure (lines, spaces, etc)
- Each XML based standard defines what are valid elements, using XML type specification languages (i.e. grammars) to specify the syntax
  - E.g. DTD (Document Type Descriptors) or XML Schema
  - Plus textual descriptions of the semantics
- XML allows new tags to be defined as required
- A wide variety of tools is available for parsing, browsing and querying XML documents/data

Why XML (3)

- SGML (Standardized General Markup Language) is more difficult
  - XML implements a subset of its features
- HTML will not do it …
  - HTML is very limited in scope, it's a language (vocabulary) for delivering (interactive) web pages
  - XML is Extensible, unlike HTML
  - Users can add new tags, and separately specify how the tag should be handled for display
  - XML is a formalism for defining vocabularies (i.e. a meta-language), HTML is just a SGML vocabulary

2 ways to look at the XML universe

(1) XML as formalism to define vocabularies (also called applications)

Example DTD :

```
<!ELEMENT page  (title, content, comment?)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT content (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
```

Exemple of an XML document:

```
<page>
 <title>Hello XML friend</title>
 <content>
      Here is some content :)
 </content>
 <comment>
      Written by DKS/Tecfa,
 </comment>
</page>
```

2 ways to look at the XML universe

(2) XML as a set of languages for defining:

- Contents
- Graphics
- Style
- Transformations and queries
- Data exchange protocols
- …..

Example of an HTML Document

```
<html>
        <head><title>Example</title></head.
<body>
        <h1>This is an example of a page.</h1>
        <h2>Some information goes here.</h2>
</body>
</html>
```

Example of an XML Document

```
<?xml version="1.0"/>
<address>
   <name>Alice Lee</name>
   <email>alee@aol.com</email>
   <phone>212-346-1234</phone>
   <birthday>1985-03-22</birthday>
</address>
```

XML - Encoding

- Encoding is the process of converting unicode characters into their equivalent binary representation. When the XML processor reads an XML document, it encodes the document depending on the type of encoding. Hence, we need to specify the type of encoding in the XML declaration.

XML - Encoding Types

- There are mainly two types of encoding – UTF-8, UTF-16
- UTF stands for UCS Transformation Format, and UCS itself means Universal Character Set. The number 8 or 16 refers to the number of bits used to represent a character. They are either 8(1 to 4 bytes) or 16(2 or 4 bytes). For the documents without encoding information, UTF-8 is set by default.

Syntax

- Encoding type is included in the prolog section of the XML document. The syntax for UTF-8 encoding is as follows −
- <?xml version = "1.0" encoding = "UTF-8“>
- In the above example encoding="UTF-8", specifies that 8-bits are used to represent the characters. To represent 16-bit characters, UTF-16 encoding can be used.
- The XML files encoded with UTF-8 tend to be smaller in size than those encoded with UTF-16 format.

Comparisons

XML	HTML
• Extensible set of tags • Content orientated • Standard Data infrastructure • Independent hardware tool used to transport & store data • Dynamic Type • No strict rules for processing • Customs Tags	• Fixed set of tags • Presentation oriented • No data validation capabilities • Well known mark up language used to develop web pages • Static Type • Strict rules must be followed when processing • Predefined tags

Authoring XML Elements

- An XML element is made up of a start tag, an end tag, and data in between.
- Example:

  <director> Matthew Dunn </director>

- Example of another element with the same value:

  <actor> Matthew Dunn </actor>

- XML tags are case-sensitive:

  <CITY> <City> <city>

- Elements must be properly nested
- XML declaration is the first statement
- Every document must contain a root element
- Certain characters are reserved for parsing

Authoring XML Elements (cont'd)

- An attribute is a name-value pair separated by an equal sign (=).
- Example:

  <City ZIP="94608"> Emeryville </City>

- Attributes are used to attach additional, secondary information to an element.

Simple XML Documents

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
    <book category="children">
      <title>Harry Potter</title>
      <author>J K. Rowling</author>
      <year>2005</year>
      <price>29.99</price>
   </book>
   <book category="web">
      <title>Learning XML</title>
      <author>Erik T. Ray</author>
      <year>2003</year>
      <price>39.95</price>
    </book>
</bookstore>
```

In the example above:

• <title>, <author>, <year>, and <price> have **text content** because they contain text (like 29.99 or Harry Potter).

• <bookstore> and <book> have **element contents**, because they contain another element.

• <book> has an attribute (category = “children”).

XML Example Revisited

- A flat text file is not nearly so clear.

Alice Lee

alee@aol.com

212-346-1234

1985-03-22

- Markup for the data aids understanding of its purpose.

```
<?xml version="1.0"/>
<address>
   <name>Alice Lee</name>
   <email>alee@aol.com</email>
   <phone>212-346-1234</phone>
   <birthday>1985-03-22</birthday>
</address>
```

Expanded Example

```
<?xml version = “1.0” ?>
<address>
   <name>
      <first>Alice</first>
      <last>Lee</last>
    </name>
    <email>alee@aol.com</email>
    <phone>123-45-6789</phone>
    <birthday>
      <year>1983</year>
      <month>07</month>
      <day>15</day>
   </birthday>
</address>
```

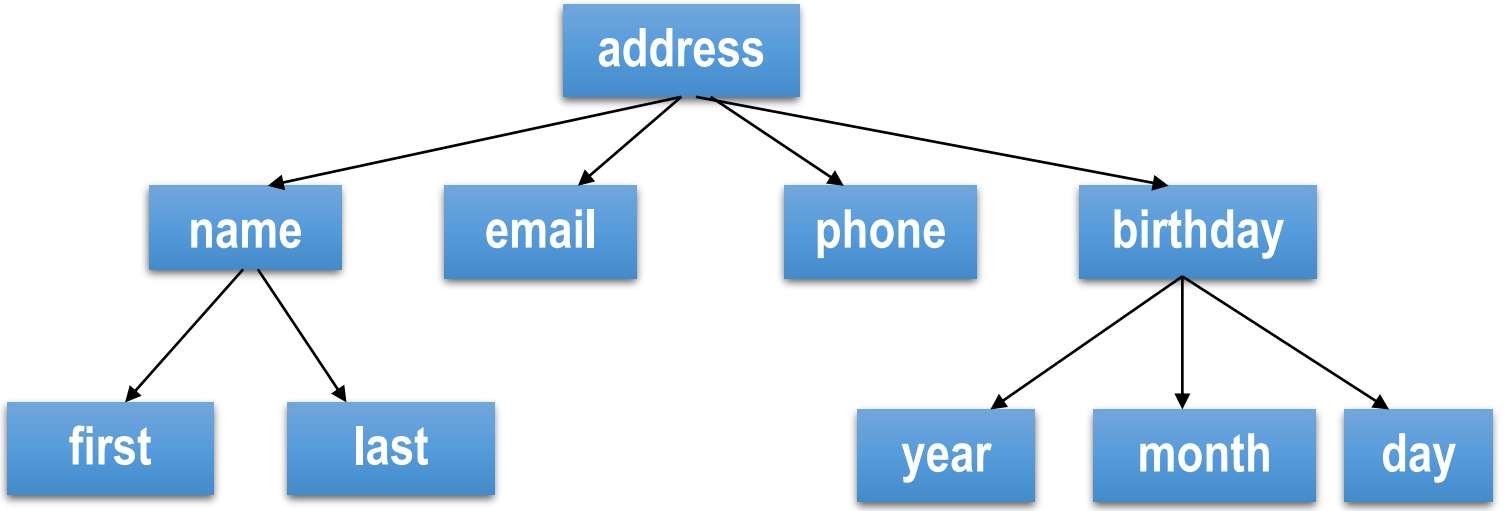
XML Tree Structure

- XML documents are formed as element trees.
- An XML tree starts at a root element and branches from the root to child elements.
- All elements can have sub elements (child elements):
- Example:
-

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

Exercise:

- Create an xml file based on the provided tree.
- Your elements should be as same as the tree elements
- You can use any value as you want.



What is a well-formed XML document (1) ?

Well-formed documents follow basic syntax rules e.g.

- there is an XML declaration in the first line
- there is a single document root
- all tags use proper delimiters
- all elements have start and end tags
    - But can be minimized if empty:
 instead of
</br>
- all elements are properly nested

  <author> <firstname>Mark</firstname>

  <lastname>Twain</lastname> </author>
- appropriate use of special characters
- all attribute values are quoted:

  <subject scheme="LCSH">Music</subject>

What is a well-formed XML document (2) ?

- Bad example

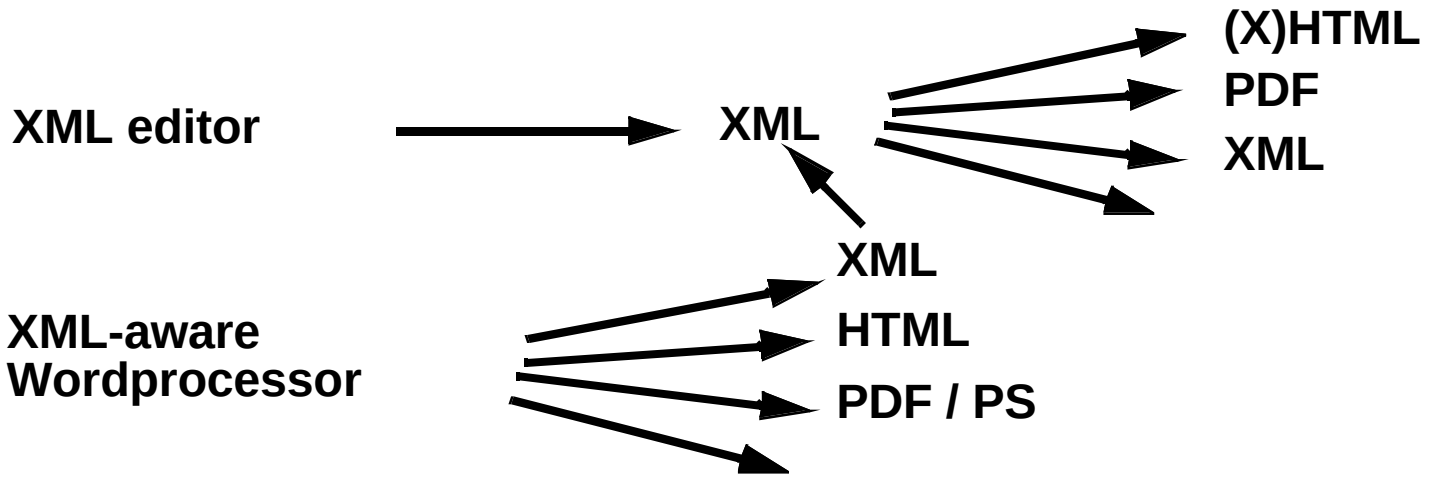
```
<addressBook>
  <address>Rue de Lausanne, Genève <person></address>
  <name>
    <family>Schneider</family> <firstName>Nina</firstName>
  </name>
  <email>nina@nina.name</email>
 </person>
 <name><family> Muller </family> <name>
</addressBook>
```

- Good example

```
<addressBook>
 <person>
  <name> <family>Wallace</family> <given>Bob</given> </name>
  <email>bwallace@megacorp.com</email>
  <address>Rue de Lausanne, Genève</address>
 </person>
</addressBook>
```

XML in the documentation world

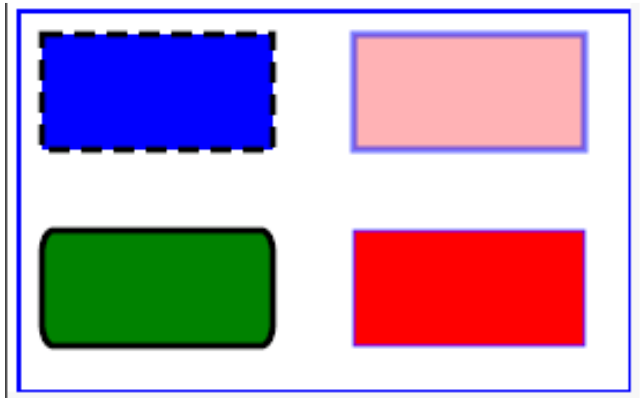
- XML is popular in the documentation world
- Specialized vocabularies to write huge documents (e.g. DocBook or DITA)
- Domain-specific vocabularies to enforce semantics, e.g. legal markup, news syndication



SVG

- SVG = Scalable Vector Graphics (as powerful as Flash)
- Partically supported in Firefox, plugin needed for IE

```
<?xml version="1.0" standalone="no"?>
<svg width="270" height="170" xmlns="http://www.w3.org/2000/svg">
 <rect x="5" y="5" width="265" height="165"
       style="fill:none;stroke:blue;stroke-width:2" />
 <rect x="15" y="15" width="100" height="50" fill="blue"
       stroke="black" stroke-width="3" stroke-dasharray="9 5"/>
 <rect x="15" y="100" width="100" height="50"
       fill="green" stroke="black" stroke-width="3" rx="5" ry="10"/>
<rect x="150" y="15" width="100" height="50" fill="red"
      stroke="blue" stroke-opacity="0.5" fill-opacity="0.3" stroke-width="3"/>
<rect x="150" y="100" width="100" height="50"
      style="fill:red;stroke:blue;stroke-width:1"/>
</svg>
```

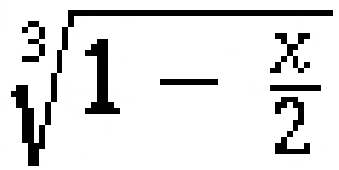


MathML

- Mathematical formulas
- Firefox, plugin needed for IE

Example:

```
<mroot>
  <mrow>
    <mn>1</mn>
    <mo>-</mo>
    <mfrac>
      <mi>x</mi>
      <mn>2</mn>
    </mfrac>
  </mrow>
  <mn>3</mn>
</mroot>
```



JSON

JavaScript Object Notation

What is JSON?

- “JSON” stands for “JavaScript Object Notation”
  - Lightweight data-interchange format
  - Despite the name, JSON is a (mostly) language-independent way of specifying objects as name-value pairs
  - Structured representation of data object
  - Can be parsed with most modern languages
- JSON Schema can be used to validated a JSON file

JSON Syntax Rules

- JSON is almost identical to python dictionary except for
  - In JSON, true and false are not capitalized
  - In JSON, null is used instead of None
- Uses key/value pairs: {“name”: “John”}
- Uses double quotes around KEY and VALUE
- Must use the specified types
- File type is “.json”
- A value can be: A string, a number, true, false, null, an object, or an array
- Strings are enclosed in double quotes, and can contain the usual assortment of escaped characters

JSON Example

```
{
    "name": "John Smith",
    "age": 35,
    "address": {
            "street": "5 main St.",
            "city": "Austin"
     },
     "children": ["Mary", "Abel"]
}
```

- This is a single JSON object for a person. It's wrapped in curly braces and uses the key/value pairs.
- The first pair, which is the name, uses a string as the value, John Smith, must be wrapped in double quotes.
- The second one is the number and note that the number is not wrapped in double quotes.
- The next is address which is an embedded object with a street and a city.
- The last one is an array is an array of strings.

JSON Schema

- A JSON Schema allows you to specify what type of data can go into your JSON files.
- It allows you to restrict the type of data entered.

JSON Schema

```
{
  "type": "object",
  "properties": {
     "name": {
         "type": "string"
      },
     "age": {
        "type": "integer"
     },
   },
}
```

```
"address": {
    "type": "object",
    "properties": {
       "street": {
          "type": "string"
       },
      "city": {
          "type": "string"
       },
    },
}
```

```
"children": {
    "type": "array",
    "items": {
       "type": "string"
     }
 }
```

Validating JSON file

- The following website can be used to validate a JSON file against a schema

  https://www.jsonschemavalidator.net/

- Paste both the schema and the corresponding JSON file

Using JSON with Python

- In Python, JSON exists as a string.
- p = '{"name": "Bob", "languages": ["Python", "Java"]}‘
- It's also common to store a JSON object in a file.
- Opening a file using with is as simple as: with open(filename) as file: # it automatically closes the file when done.
- To work with JSON (string, or file containing JSON object), you can use Python's json module.

```
import json
```

Loading JSON data from a file

- Example:

```
def load_json(filename):
    with open(filename) as file:
        jsn = json.load(file)
        #file.close()
    return jsn
person = load_json('person.json')
```

- This command parse the above person.json using json.load() method from the json module. The result is a Python dictionary.

Writing JSON object to a file

- Example:

```
person = { "name": "John Smith", "age": 35, "address": {"street":
"5 main St.", "city": "Austin"}, "children": ["Mary", "Abel"]}


with open('person_to_json.json', 'w') as fp:
    json.dump(person, fp, indent=4)
```

- Using json.dump(), we can convert Python Objects to JSON file.

Accessing JSON Properties in Python

- Example:

Assume that you already loaded your person.json as follows.

```
person = load_json('person.json')
```

To access the property "name"

```
Print(person["name"])
-> John Smith
```

Accessing JSON Properties in Python

1. To access the property “age"

```
person["age"]
```

```
-> 35
```

2. To access the property “street"

```
print(person["address"]["street"])
```

```
-> 5 main St.
```

3. To access the property “city"

```
print(person["address"]["city"])
```

```
-> Austin
```

4. To access the property “children"

```
print(person["children"][0])
```

```
-> Mary
```

Python – JSON Objects

array: Ordered list of 0 or more values. Wrap in square brackets
object: Unordered collection of key/value pairs
number: No difference between integers and floats
Integer: typical integer rules
string: Use double quote
boolean: true or false
null: Empty value

Python	JSON Equivalent
`dict`	object
`list`, `tuple`	array
`str`	string
`int`, `float`, `int`	number
`True`	true
`False`	false
`None`	null

Credit

- https://www.youtube.com/watch?v=wI1CWzNtE-M
- https://www.programiz.com/python-programming/json