XML and JSON

XML: Introduction

- e**X**tensible **M**arkup **L**anguage (XML) defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.
- XML defined in the XML 1.0 Specification produced by the W3C, and several other related specifications
- Extensible
  - Can be used for both documents and messages
  - Unlike HTML, new “tags” can be defined

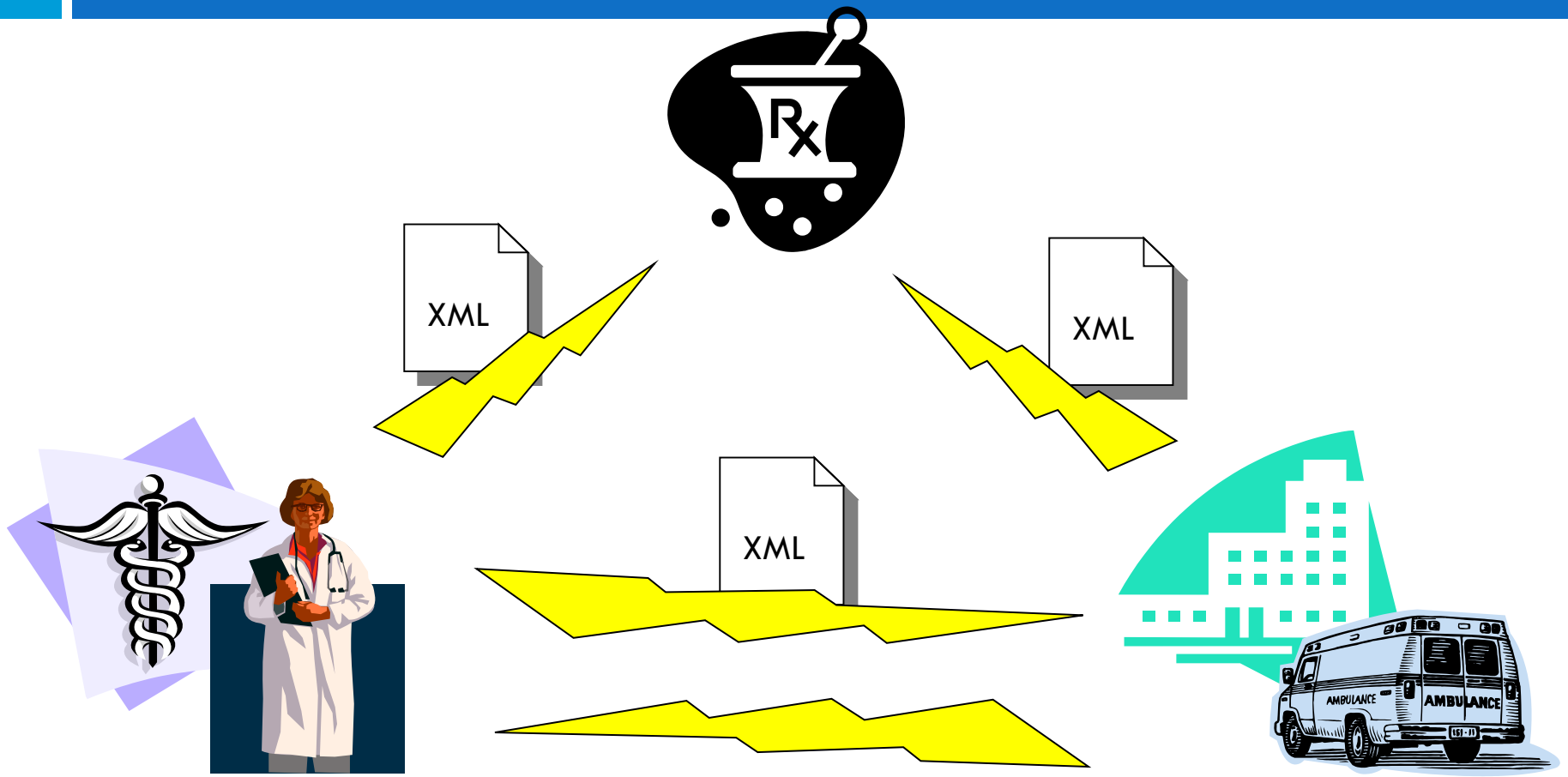
Why do we need XML?

- to present complex data in human-readable form
  - "self-describing data"
- to interchange data between different platforms

The Business Connection

- Protocol independence
  - Eases intra-business communication
  - Allows information interchange with partners
- Platform independence
  - Bridges legacy systems to new applications
- Open standard
  - Everyone “speaks” the same language

The “Big” Picture: An Example



Anatomy of an XML file


```xml
<?xml version="1.0" encoding="UTF-8"?> <!-- XML prolog -->
       <note> <!-- root element -->
              <to>Tove</to>
              <from>Jani</from> <!-- element ("tag") -->
              <subject>Reminder</subject> <!-- content of
element -->
              <message language="english"> <!-- attribute
and its value -->
                     Don't forget me this weekend!
              </message>
       </note>
```

XML

- begins with an <?xml ... ?> header tag ("prolog")
- has a single root element (in this case, note)
- tag, attribute, and comment syntax is just like XHTML

Uses of XML

- XML data comes from many sources on the web:
  - **web servers** store data as XML files
  - **databases** sometimes return query results as XML
  - **web** services use XML to communicate
- XML is the de facto universal format for exchange of data
- XML languages are used for music, math, vector graphics
- popular use: RSS for news feeds & podcasts

Pros and cons of XML

pro:

- easy to read (for humans and computers)
- standard format makes automation easy
- international, platform-independent, open/free standard
- can represent almost any general kind of data (record, list, tree)

Pros and cons of XML

con:

- bulky syntax/structure makes files large; can decrease performance
  - example: quadratic formula in MathML
- can be hard to "shoehorn" data into a good XML format

What tags are legal in XML?

- any tags you want!
- examples:
  - an email message might use tags called **to**, **from**, **subject**
  - a library might use tags called book, title, author
- when designing an XML file, you choose the tags and attributes that best represent the data
- rule of thumb: data = tag, metadata = attribute

Tags and attributes


```xml
<?xml version="1.0" encoding="UTF-8"?> <!-- XML prolog -->
       <note> <!-- root element -->
              <to>Tove</to>
              <from>Jani</from> <!-- element ("tag") -->
              <subject>Reminder</subject> <!-- content of
element -->
              <message language="english"> <!-- attribute
and its value -->
                     Don't forget me this weekend!
              </message>
       </note>
```

XML

Validation of XML Documents

- XML documents *must* be well-formed
- XML documents *may* be valid
  - Validation verifies that the structure and content of the document follows rules specified by grammar
- Types of grammars
  - Document Type Definition (DTD)
  - XML Schema (XSD)

XML DTD

- An XML document with correct syntax is called "Well Formed".
- An XML document validated against a DTD is both "Well Formed" and "Valid".

Valid XML Documents

- **Valid XML Documents**
- A "Valid" XML document is a "Well Formed" XML document, which also conforms to the rules of a DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note SYSTEM "Note.dtd">
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

XML DTD - Example


```
<!DOCTYPE note
[
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

XML Schema

- An XML Schema describes the structure of an XML document.
- The XML Schema language is referred to as XML Schema Definition (XSD).

XML Schema

- The purpose of an XML Schema is to define the legal building blocks of an XML document:
  - the elements and attributes that can appear in a document
  - the number of (and order of) child elements
  - data types for elements and attributes
  - default and fixed values for elements and attributes

XML Schema

- XML Schema is an XML-based (and more powerful) alternative to DTD.
- One of the greatest strength of XML Schemas is the support for data types.
  - It is easier to describe allowable document content
  - It is easier to validate the correctness of data
  - It is easier to define data facets (restrictions on data)
  - It is easier to define data patterns (data formats)

XSD Example


```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

XML Namespaces


```
<root>

<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table xmlns:f="https://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>

</root>
```

XML and Ajax

- web browsers can display XML files, but often you instead want to fetch one and analyze its data
- the XML data is fetched, processed, and displayed using Ajax
  - (XML is the "X" in "Ajax")
- It would be very clunky to examine a complex XML structure as just a giant string!
- luckily, the browser can break apart (parse) XML data into a set of objects
  - there is an XML DOM, very similar to the (X)HTML DOM

XML DOM tree structure


```xml
<?xml version="1.0" encoding="UTF-8"?>
      <categories>
            <category>children</category>
            <category>computers</category>
            ...
      </categories>
```
XML

- the XML tags have a tree structure
- DOM nodes have parents, children, and siblings

Recall: Javascript XML (XHTML) DOM

The DOM properties and methods we already know can be used on XML nodes:

- properties:
  - **firstChild**, lastChild, **childNodes**, nextSibling,
  - previousSibling, **parentNode**
  - **nodeName**, nodeType, **nodeValue**, attributes
- methods:
  - appendChild, insertBefore, removeChild, replaceChild
  - **getElementsByTagName**, getAttribute, hasAttributes, hasChildNodes

- caution: cannot use HTML-specific properties like innerHTML in the XML DOM!

Fetch XML Data Using AJAX

- https://www.w3schools.com/xml/ajax_xmlfile.asp

Fetch XML Data with JQuery and Ajax


```
function loadData() {

        $.ajax({

                url: "cars.xml",
                dataType: "xml",
                success: function(data) {
                    alert("file is loaded");
                    $(data).find('car').each(function(){
                        var title = $(this).find('Title').text();
                        var manufacturer = $(this).find('Manufacturer').text();
                        var info = '<li>Title: ' + title+',   Manufacturer: ' +
                        manufacturer + '</li>';
                        $("ul").append(info);
                    });

                },
                error: function() { alert("error loading file");  }
    });
}
```

JSON: Introduction

- JavaScript Object Notation (JSON) is an open standard format
  - originally specified by Douglas Crockford
- JSON used primarily to transmit data between a server and web application
  - human-readable, light-weight text-data
  - objects consisting of attribute–value pairs
  - an alternative to XML
    - smaller than XML
    - faster and easier to parse
- JSON originally derived from the JavaScript scripting language
  - however, JSON is a language-independent data format
  - libraries to parse and generate JSON data is readily available in a large variety of programming languages

JSON: Example

```
<!DOCTYPE html>
<html>
<body>
  <h2>JSON Object Creation in JavaScript</h2>
  <p>
    Name: <span id="jname"></span><br />
    Age: <span id="jage"></span><br />
    Address: <span id="jstreet"></span><br />
    Phone: <span id="jphone"></span><br />
  </p>
 <script>
    var JSONObject= {
        "name":"John Johnson",
        "street":"Oslo West 555",
       "age":33,
       "phone":"555 1234567"};
    document.getElementById("jname").innerHTML=JSONObject.name;
    document.getElementById("jage").innerHTML=JSONObject.age;
    document.getElementById("jstreet").innerHTML=JSONObject.street;
    document.getElementById("jphone").innerHTML=JSONObject.phone;
</script>
</body>
</html>
```

JSON: Example

```
{
   "firstName": "John",
   "lastName": "Smith",
   "age": 25,
   "address": {
      "streetAddress": "21 2nd Street",
      "city": "New York",
      "state": "NY",
      "postalCode": 10021
   },
   "phoneNumbers": [
      {
         "type": "home",
         "number": "212 555-1234"
      },
      {
         "type": "fax",
         "number": "646 555-4567"
      }
   ]
}
```

JSON: Syntax

- JSON syntax is a subset of the JavaScript object notation syntax:
  - Data is in name/value pairs
  - Data is separated by commas
  - Curly braces hold objects
  - Square brackets hold arrays

- JSON text-data contains:
  - name/value pairs that consist of:
    - field name (in double quotes)
    - followed by a colon
    - followed by a value that can be:
      - number (integer or floating point)
      - string (in double quotes)
      - Boolean (true or false)
      - array (in square brackets)
      - object (in curly brackets)
      - null
    - example: "firstName" : "John“

JSON: Objects and Arrays

- JSON objects are written inside curly brackets
  - Objects can contain multiple name/values pairs
    - Example: { "firstName":"John" , "lastName":"Doe" }
- JSON arrays are written inside square brackets
  - An array can contain multiple objects
    - Example:

```
{
    "employees": [
        { "firstName":"John" , "lastName":"Doe" },
        { "firstName":"Anna" , "lastName":"Smith" },
        { "firstName":"Peter" , "lastName":"Jones" }
    ]
}
```

JSON Vs XML

- Like XML
  - JSON is plain text
  - JSON is "self-describing" (human readable)
  - JSON is hierarchical (values within values)
  - JSON can be parsed by JavaScript
  - JSON data can be transported using AJAX
- **Unlike XML**
  - No end tag
  - Shorter
  - Quicker to read and write
  - Can be parsed using built-in JavaScript eval()
    - JSON format is syntactically identical to the JavaScript objects creation code
      - Hence, no special parser required in JavaScript; JavaScript can use the built-in eval() function and execute JSON data to produce native JavaScript objects
  - Uses arrays
  - No reserved words

JSON: Need

- For AJAX applications:
  - Using XML
    - Fetch an XML document
    - Use the XML DOM to loop through the document
    - Extract values and store in variables
  - Using JSON
    - Fetch a JSON string
    - eval() the JSON string
- JSON is faster and easier than XML

JSON: Using on Web Browser with JavaScript

- On the web browser, after fetching JSON data from a web server (as file or HttpRequest):
  - convert the JSON data to a JavaScript object
    - JavaScript function eval() can be used to convert a JSON data into a JavaScript object
      - eval() function can compile and execute any JavaScript
      - represents a potential security problem.
    - safer and faster to use a JSON parser natively supported by browser
      - JSON parser will recognize only JSON text and will not compile scripts
      - JSON.parse() function added to ECMAScript 5th Edition Standard and is now supported by the major browsers
        - Example: var p = JSON.parse(contact);
    - The jQuery library has **getJSON()** function

JSON: Using on Web Browser with JavaScript

```
<!DOCTYPE html>
<html>
<body>
    <h2>Create Object from JSON String</h2>
    <p>
       First Name: <span id="fname"></span><br>
       Last Name: <span id="lname"></span><br>
    </p>
    <script>
       var txt = '{"employees":[' +
                 '{"firstName":"John","lastName":"Doe" },' +
                 '{"firstName":"Anna","lastName":"Smith" },' +
                 '{"firstName":"Peter","lastName":"Jones" }]}';

        var obj = JSON.parse(txt);
        document.getElementById("fname").innerHTML=obj.employees[1].firstName;
        document.getElementById("lname").innerHTML=obj.employees[1].lastName;
    </script>
</body>
</html>
```

Fetch JSON Data with JQuery and Ajax


```
$(document).ready(function(){
    $("button").click(function(){
        $.getJSON("demo_ajax_json.js", function(result){
            $.each(result, function(key, value){
                $("div").append(value + " ");
            });
        });
    });
});
```

http://www.w3schools.com/jquery/demo_ajax_json.js

https://learn.jquery.com/using-jquery-core/iterating/

http://www.pureexample.com/jquery/get-json.html

AJAX Example

https://www.youtube.com/watch?v=fEYx8dQr_cQ