XML + JSON

Saba sadat Faghih Imani

2024

Questions

- Q1) How to define the data that is transferred between web server and client?
- Q2) Which technologies?
- Q3) Is data correctly encoded?
- Q4) How to access the data in web pages?
- Q5) How to present the data?

Outline

- Introduction
- Documentation, Interaction, Validation
- Processing (using JavaScript)
- Conclusion

Introduction

- HTML + CSS + JavaScript → Interactive Web pages
  - Web server is not involved after page is loaded
    - JavaScript reacts to user events
- However, most web applications needs data from server after the page is loaded
  - e.g., new emails data in Gmail
  - A mechanism to communication: AJAX
  - A common (standard) format to exchange data
- In most applications, the data is structured

Introduction (cont'd)

- In general (not only in web) to store or transport data, we need a common format, to specify the stucture of data; e.g.,
  - Documents: PDF, DOCx, PPTx, ...
- How to define the data structure?
  - Binary format (similar to binary files)
    - Difficult to develop & debug, machine depended, …
  - Text format (similar to text files)
    - Human readable, machine independent & easier

Introduction (cont'd)

- Example: Data structure of a class
  - Course name, teacher, # of students, each student information

```
WP
Imani
31
Ali
Hassani
1111
Babak
Hosseini
2222
….
```

```
Student num: 31
Name: WP
Teacher: Imani
Ali Hassani 1111
Babak Hosseini 2222
….
```

```
class Course{
   string name;
   string teacher;
    integer num;
    Array st of Students;
}

c  = new Course();
c.name = WP;
c.teacher = Imani;
c.num = 32
st[1] = new Student();
st[1].name=Ali; st[2].fam=Hassani
….
```

Outline

- Introduction
  - XML
- Documentation, Interaction, Validation
- Processing (using JavaScript)
- Conclusion

XML

- W3C's approach
  - XML: eXtensible Markup Language
  - A meta-markup language to describe data structure
    - In each application, a markup language (set of tags & attributes) are defined

```
<course>
  <title> WP </title>
  <num> 32 </num>
  <teacher> Imani </teacher>
  <students>
      <student><name>Ali</name> <fam>Hassani</fam>
              <id> 1111 </id></student>
       …
  </students>
</course>
```

XML

- Standard Generalized Markup Language (SGML)
  - Expensive, complex to implement
- XML: a subset of SGML
  - Goals: generality while simplicity and usability
  - Simplifies SGML by:
    - leaving out many syntactical options and variants
  - XML = SGML – {complexity, document perspective} + {simplicity, data exchange perspective}

Why to Study XML: Benefits

- Simplify data sharing & transport
  - XML is text based and platform independent
- Extensive tools to process XML
  - To validate, to present, to search, …
- In web application, data separation from HTML
  - E.g., table structure by HTML, table data by XML
- Extensible for different applications
  - A powerful tool to model/describe complex data
  - E.g., MS Office!!!

XML Document Content (Markups)

- Three markup types in XML:
- 1) Elements
  - Tag + Attributes
  - Content
    - Parsed Character Data
    - Unparsed Character Data (CDATA)
- 2) Comments
- 3) Processing instructions

XML Elements

- XML element structure
  - Tag + Attribute + Content

<tagname attribute=“value”>
Content
</tagname>

- *No predefined tag*
- If content is not CDATA, is parsed by parser
  - A value for this element
  - Child elements of this element

XML Elements' Attributes

- Tags (elements) are customized by attribute
  - *No predefined attributes*

```
<os install="factory">Windows</os>
<os install="user">Linux</os>
```

- Attribute vs. Tags
  - Attributes can be replaced by elements
    - Attribute cannot be repeated for an element
    - Attribute cannot have children
  - Attributes mainly used for metadata, e.g., ID, class

Processing Instructions

- Processing instructions pass information (instruction) to the application that processes the XML file
  - They are not a part of user data

```
<?Target  String ?>
```

- Common usage

```
<?xml-stylesheet href="URL" type="text/xsl"?>
```

- XML Declaration is a special PI

```
<?xml version="1.0" encoding="UTF-16"?>
```

  - XML Declaration is always first line in file

Basic XML Document Structure

```
<?xml version="1.0" encoding="UTF-16"?>
<root-tag>
<inner-tags>
  Data
</inner-tags>
<!-- Comment -->
</root-tag>
```

Example

```
<?xml version="1.1" encoding="UTF-8" ?>
<notebook>
<name>ThinkPad</name>
<model>T500</model>
<spec>
  <hardware>
       <RAM>4GB</RAM>
  </hardware>
  <software>
       <OS>Linux, FC27 </OS>
  </software>
</spec>
</notebook>
```

Example (CDATA)

```
<?xml version="1.1" encoding="UTF-8" ?>
<operator>
  <mathematic>
      + - * / %
  </mathematic>
  <comparison>
      <![CDATA[
            < <= == >= > !=
      ]]>
  </comparison>
</operator>
```

XML Syntax Rules

- Start-tag and End-tag, or self-closing tag
- Tags can't overlap
- XML documents can have only one root element
- XML naming conventions
  - Names can start with letters or the dash (-) character
    - After the first character, numbers, hyphens, and periods are allowed
  - Names can't start with "xml", in uppercase or lowercase
  - There can't be a space after the opening < character
  - XML is case sensitive
  - Value of attributes must be quoted
- White-spaces are preserved
- **&**, **<**, **>** are represented by **`&`** **`<`** **`>`**

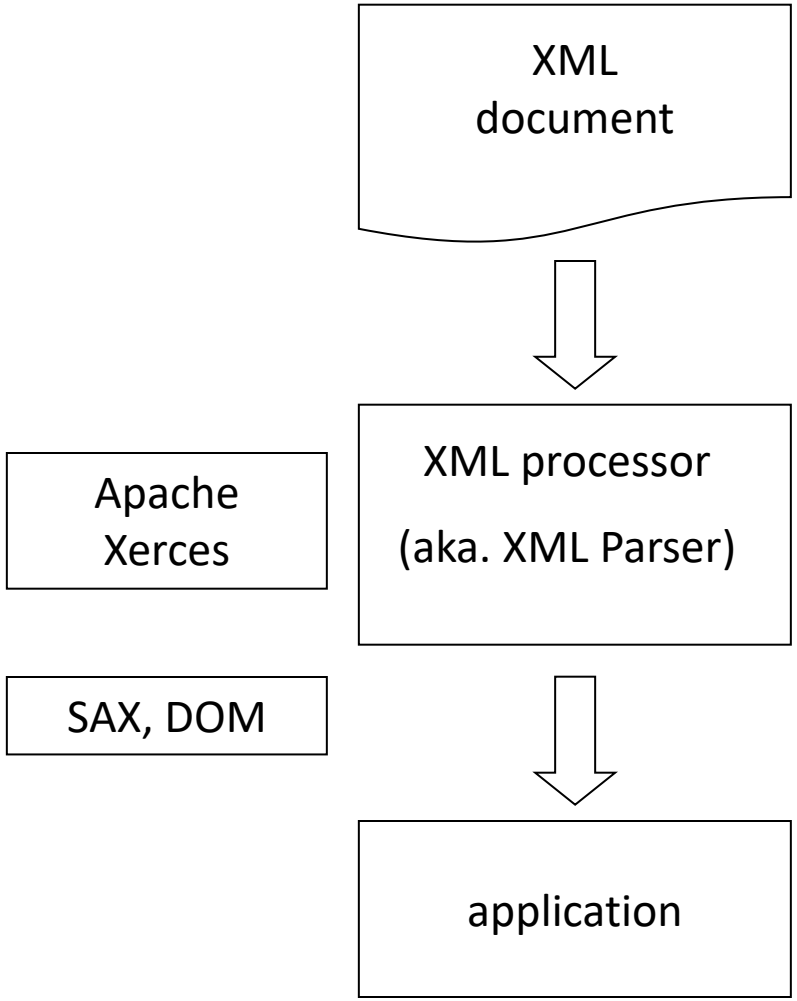
XML vs. HTML

- Tags
  - HTML: Predefined fixed tags
  - XML: No predefined (meta-language)
    - User defined tags & attributes
- Purpose
  - HTML: Information display
  - XML: Data structure & transfer (which is displayed)

XML in General Application

- XML by itself does not do anything
- XML just describes the structure of the data
- Other applications parse XML and use it

- A similar approach is used for formats (event user-defined format); so, what is the advantages of XML?!!!
  - *XML is standard*
  - *Available XML tools & technologies*



XML Technology Components

- Data structure (tree) representation
  - XML document (a text file)
- Validation & Conformance
  - Document Type Definition (DTD) or XML Schema
- Element access & addressing
  - XPath, DOM
- Display and transformation
  - XSLT or CSS
- Programming, Database, Query, …

Outline

- Introduction
  - JSON
- Documentation, Interaction, Validation
- Processing (using JavaScript)
- Conclusion

JSON

- JavaScripters' approach
  - JSON: JavaScript Object Notation
    - Data is represented as a JS object

```
{"course":
  "title": "WP",
  "num": 32,
  "teacher": "Imani",
  "students":
  [
       { "name": "Ali", "fam": "Hassani", "id": 1111 }
       …
  ]
}
```

Why to Study JSON: Benefits

- Simplify data sharing & transport
  - JSON is text based and platform independent
- JSON is simple, efficient, and popular
- Extensive libraries to process JSON
  - To validate, to present, …
- In web application, data separation from HTML
  - E.g., table structure by HTML, table data by JSON
- JSON ←→ JS Objects
  - Any differences?!

JSON Syntax

- Data is in name/value pairs
  - Field name in double quotes, followed by a colon, followed by a value
    - In JSON, *keys* must be strings
- Data is separated by commas
  - Data types: string, number, object, array, boolean, and null
- Curly braces hold objects
- Square brackets hold arrays

JSON Syntax (cont’d)

- Object field access by .

```
myObj = { "name":“Ali", "age":30, "car":null };
x = myObj.name;
```

- Object field access similar to array

```
myObj = { "name":“Ali", "age":30, "car":null };
x = myObj["name"];
```

- Looping on Object

```
myObj = { "name":“Ali", "age":30, "car":null };
for (x in myObj) {
     y = x;
```

Outline

- Introduction
- Documentation, Interaction, Validation
- Processing (using JavaScript)
- Conclusion

Documentation, Interaction, Validation

- Assume that application A exchange data with application B
- How does A’s developer document the data format?
- How does the receiver know the structure of the data?
- How can the receiver validate the data?

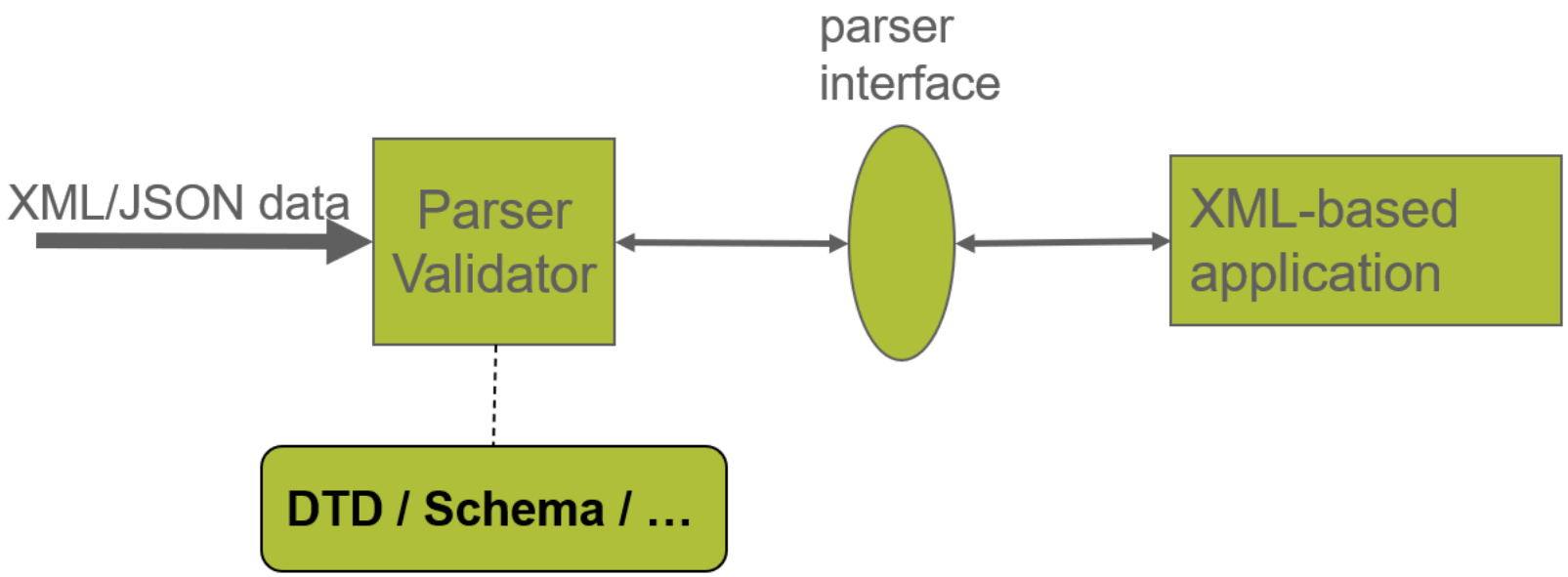
Valid Data

- Syntax
  - Syntax rules
    - E.g., all XML tags must be closed
    - E.g., all keys must be double quoted in JSON
  - Syntax error → parser fails to parse the file
- Symantec (structure)
  - Application specific rules
    - E.g. student must have ID
  - Error → the application fails

How to Validate structure?

- 1) Application specific programs need to check structure of data
  - Different applications → different programs
  - Change in data structure → code modification
- 2) General validator + reference document
  - Reference document
    - Tag/key names, attributes, tree structure, tag relations, …
  - Different reference documents
    - XML: DTD, XML Schema, RELAX NG
    - JSON: JSON Schema

Reference Based Validation



The Reference Documents Usage

- The Reference document is the answer of
  - Documentation
    - It describes the structure of data which is human readable
  - Interaction
    - The description is machine readable
  - Validation
    - There are validators to validate the data based on it

Outline

- Introduction
- Documentation, Interaction, Validation
  - XML
- Processing (using JavaScript)
- Conclusion

DTD

- DTD is a set of structural rules called declarations, specify
  - A set of elements and attributes that can be in XML
    - Where these elements and attributes may appear
  - **<!** *keyword* **…>**
    - **ELEMENT**: to define tags
      - For leaf nodes: Character pattern
      - For internal nodes: List of children
    - **ATTLIST**: to define tag attributes
      - Includes: name of the element, the attribute's name, its type, and a default option

XML Schema

- XML Schema describes the structure of an XML file
  - Also referred to as XML Schema Definition (XSD)
- XML Schemas benefits (DTD disadvantages)
  - Created using basic XML syntax (DTD has its own syntax)
  - Validate text element content based on built-in and user-defined data types (DTD does not fully support data type)
- Similar to OOP
  - Schema is a class & XML files are instances
  - Schema specifies
    - Elements and attributes, where and how often
    - Data type of every element and attribute

Schema (cont'd)

- XML schema is itself an XML-based language
  - Has its own predefined tags
- Two categories of data types
  - *Simple*: Cannot have nested *elements* or *attribute* (i.e., itself is a leaf or attribute)
    - Primitive: **`string`**, **`Boolean`**, **`integer`**, **`float`**, …
    - Derived: **`byte`**, **`long`**, **`unsignedInt`**, …
    - User defined: restriction of base types
  - *Complex*: Can have attribute or/and nested elements

XML Schema (cont'd)

- Simple element declaration

```
<xs:element name="a name" type="a type" />
```

- Complex element declaration

```
<xs:element name="a name">
  <xs:complexType>
    <xs:sequence> or <xs:all> or <xs:choice>
       <xs:element name
            minOccurs="…" maxOccurs="…"/>
    </xs:sequence> or </xs:all> or </xs:choice>
  </xs:complexType>
</xs:element>
```

XML Schema (cont'd)

- Notes on **`minOccurs`** & **`maxOccurs`**
  - Using the **`all`** indicator
    - **`minOccurs`** & **`maxOccurs`** indicator can only be 0 or 1
  - The default value for **`minOccurs`** and **`maxOccurs`** is 1
    - To set **`maxOccurs`** > 1, we should set **`minOccurs`** too
  - To allow an element to appear an unlimited number of times, use the **`maxOccurs="unbounded"`** statement

XML Schema Example: note.xsd

```
<?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="date" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

XML Schema Example: note.xml

```
<?xml version="1.0"?>
<note xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance" xsi:schemaLocation="note.xsd">

  <to>Ali</to>
  <from>Reza</from>
  <date> 1391-01-01 </date>
</note>
```

Any # of children in any order

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="bar">
 <xs:complexType>
   <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="child1" type="xs:string" />
      <xs:element name="child2" type="xs:string" />
      <xs:element name="child3" type="xs:string" />
   </xs:choice>
 </xs:complexType>
</xs:element>
</xs:schema>
```

Attributes for Complex Elements

- Schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="bar">
<xs:complexType>
   <xs:all>
      <xs:element name="child1" type="xs:string"/>
      <xs:element name="child2" type="xs:string"/>
   </xs:all>
   <xs:attribute name="lang" type="xs:string"/>
</xs:complexType>
</xs:element>
</xs:schema>
```

- XML

```
<bar lang="a">
     <child1>1</child1> <child2>2</child2>
</bar>
```

Attributes for Simple Elements

- Schema

```
<xs:element name="bar">
<xs:complexType>
   <xs:simpleContent>
      <xs:extension base="xs:string">
         <xs:attribute name="lang"
                type="xs:string" />
         </xs:extension>
      </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

- XML

```
<bar lang="123123">abc</bar>
```

XML Validation Tools

- Online validators
  - validator.w3.org
  - www.xmlvalidation.com
- XML tools & commands
  - **`xmllint`** commands in Linux
    - **`xmllint`** xmlfile **`--valid --dtdvalid`** DTD
    - **`xmllint`** xmlfile **`--schema`** schema
- XML libraries
  - LibXML2 for C
  - Java & C# XML libraries

Outline

- Introduction
- Documentation, Interaction, Validation
  - JSON
- Processing (using JavaScript)
- Conclusion

JSON Schema

- JSON schema is JSON also

{

$schema: http://json-schema.org/schema#
$id: URL
title: “String”
description: “String”
type: **string**, **integer**, **number**, **object**, **array**, **boolean**, **null**

type specific descriptions/settings

}

Type Specific Descriptions/Settings

- String
  - Length: **`minLength`** , **`maxLength`**
  - Content: **`pattern`**, **`format`**
- Integer, Number
  - Range: **`minimum`**, **`maximum`**
- Object
  - Members: **`properties`**
- Array
  - Members: **`items`**
  - Length: **`minItems`**, **`maxItem`**

JSON Schema Example

```
{"$schema": "http://json-schema.org/schema#",
 "title": "Product",
 "type": "object",
 "properties": {
     "id": {
         "type": "number",
         "description": "Product identifier"
     },
     "name": { "type": "string" },
     "price": { "type": "number", "minimum": 0 },
     "tags": {
          "type": "array",
          "items": { "type": "string" }
      },
      "stock": {
            "type": "object",
             "properties": {
                 "warehouse": { "type": "number" },
                  "retail": { "type": "number" }
             }
       }
   }
}
```

```
{
    "id": 1,
    "name": "Foo",
    "price": 123,
    "tags": [
        "Bar",
        "Eek"
     ],
     "stock": {
         "warehouse": 300,
         "retail": 20
      }
}
```

JSON Schema Validator

- Validators available
  - As Online tools
  - As programming languages libraries
  - Standalone tools

- Just Google it

Outline

- Introduction
- Documentation, Interaction, Validation
- Processing (using JavaScript)
  - XML
- Conclusion

XML Processor: Parsers

- There are two basic types of XML parsers:
- Tree (DOM)-based parser:
  - Whole document is analyzed to create a DOM tree
  - Advantages: Multiple & Random access to elements, easier to validate the structure of XML
- Event-based parser (SAX):
  - XML document is interpreted as a series of events
  - When a specific event occurs, a function is called to handle it
  - Advantages: less memory usage and no wait to complete faster

XML Parsing in Browser

- Web browsers have built-in XML parser
  - XML parser output: XML DOM
- XML DOM is accessible through JavaScript
  - **`DOMParser`** can parse an input XML string
- How to get XML file in Java Script?
  - Using AJAX
  - (Input) string

XML DOM

- XML DOM is similar to HTML DOM
  - A tree of nodes (with different types: element, text, attr, …)
  - Nodes are accessed by **`getElementsByTagName`**
  - Nodes are objects (have method & fields)
  - DOM can be modified, e.g., create/remove nodes
- However
  - There is not predefined attributes link **`id`**/**`class`**
    - **`getElementById`** or similar methods are not applicable
  - Since XML is not for presentation
    - Nods have not *event handler* functions
    - Nodes have not *style* field

XML DOM (cont'd)

- Each node have
  - parentNode, children, childNodes, …
- Access to value of a node
  - In the DOM, everything is a node (with different types)
  - Element nodes do not have a content value
  - The content of an element is stored in a child node
    - To get content of a leaf element, the value of the first child node (text node) should got

Example: Message Parser

```
<body>
Enter Your XML:<br />
<textarea name="inputtext1" cols="70"
  rows="10"><root><msg><from></from><to></to><body></body></msg></root></t
  extarea>


<input type="button" onclick="parseXML()" value="Parse" />
  <br />


<div name="outputdiv1" style="border-style:solid; border-
  width:1px; width:70%;"></div>
</body>
```

Example: Message Parser

```
function parseXML(){
  output = "";
  input = document.getElementsByName("inputtext")[0].value;
  parser = new DOMParser();
  xmlDoc = parser.parseFromString(input,"text/xml");
  messages = xmlDoc.getElementsByTagName("root")[0].children;
  for(i=0; i < messages.length; i++){
        msg = messages[i];
        fromNode = msg.getElementsByTagName("from")[0];
        fromText = fromNode.childNodes[0].nodeValue;
        toNode = msg.getElementsByTagName("to")[0];
        toText = toNode.childNodes[0].nodeValue;
        bodyNode = msg.getElementsByTagName("body")[0];
        bodyText = bodyNode.childNodes[0].nodeValue;
        output = output + fromText +" sent following message to " + toText + "<br /> '" +
  bodyText +"'<hr />"
  }
  document.getElementsByName("outputdiv")[0].innerHTML = output;
}
```

Outline

- Introduction
- Documentation, Interaction, Validation
- Processing (using JavaScript)
  - JSON
- Conclusion

JSON in JavaScript

- The JSON object, has two very useful methods to deal with JSON-formatted content
  - **`JSON.parse()`** takes a JSON string and transforms it into a JavaScript object
  - **`JSON.stringify()`** takes a JavaScript object and transforms it into a JSON string

```
myObj = { a: '1', b: 2, c: '3' };
myObjStr = JSON.stringify(myObj);
console.log(myObjStr);
console.log(JSON.parse(myObjStr));
```

Example: Message Parser

```
<body>
Enter Your JSON:<br />

<textarea name="inputtext2" cols="70" rows="10">
{"type": "object", "properties": { "messages": { "type": "array",
"items": { "type": "object", "properties": { "from": { "type": "string" }, "to": { "type": "string" }, "body":
  { "type": "string" }}}}}}
</textarea>

<input type="button" onclick="parseJSON()" value="Parse" /> <br />

<div name="outputdiv2" style="border-style:solid; border-width:1px;
  width:70%;"></div>

</body>
```

Example: Message Parser

```
function parseJSON(){
  output = "";
  input = document.getElementsByName("inputtext2")[0].value;
  jsonData = JSON.parse(input);

  for(i = 0; i < jsonData.messages.length; i++){
       msg = jsonData.messages[i];
       output += msg.from + " sent the following message to " +
  msg.to + "<br /> ''" + msg.body +"''<hr />";
  }

  document.getElementsByName("outputdiv2")[0].innerHTML = output;
}
```

Outline

- Introduction
- Documentation, Interaction, Validation
- Processing (using JavaScript)
- Conclusion

Answers

- 1) Which technology?
  - Text based!
  - XML: A markup met-language with user defined tags
  - JSON: Object notation
- 2) Is data correctly encoded?
  - XML Validation: DTD and Schema
  - JSON Schema
- 3) How to access the data in web pages?
  - Parse XML to DOM, we know how to work with DOM
  - Parse JSON into JS objects

What are the Next?!

- XSL
  - XSLT (XSL Transform) - transforms XML into other format
  - XPath - a language for navigating XML documents
- XML's applications
  - SVG (Scalable Vector Graphics)
    - Defines graphics in XML format
- Other related technologies in data exchange
  - *Protocol Buffers* (Google), *Thrift* (Apache & FB)