COS10024 Web Development

Lecture 11 – Introduction to XML

Dr. Gamunu Dassanayake

TP2/2022

Contents

What is XML?

What is XML ?

EXAMPLES!
TXT vs Database vs XML vs HTML

- XML is a simple *structured general mark-up language.*
- XML enables *structured data* to be marked-up, searched and utilized in *XML Applications* ***… e.g., using the DOM*** ☺
- XML data can be *exchanged*:
  - between computers,
  - between computer applications,
  - between organizations.
- Electronic document *data exchange* is now easily arranged with XML and the Web, ***e.g., using Web Services as the API***. (Application Programming Interface)
- XML was designed to be self-descriptive.
- XML is a W3C Recommendation.

```
<item>
  <title>Empire Burlesque</title>
  <note>Special Edition</note>
  <quantity>1</quantity>
  <price>10.90</price>
</item>
```

What is XML ? [2]

- **Extensible Markup Language (XML) is**
  - ✓ *a human-readable,*
  - ✓ *machine-understandable,*
  - ✓ *general syntax for describing hierarchical data,*
  - ✓ *applicable to a wide range of applications*

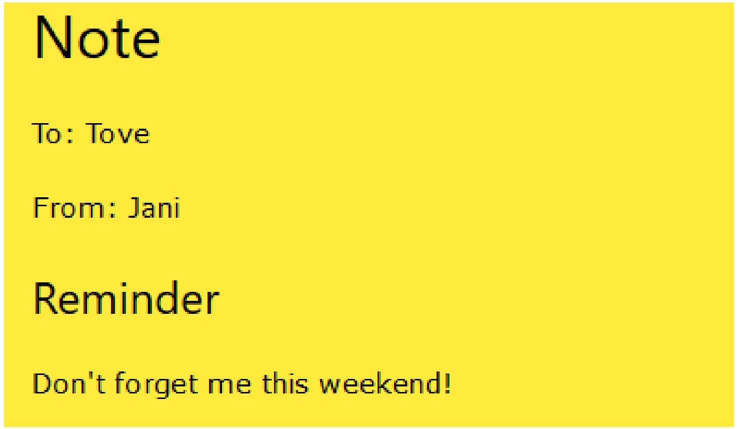
- **XML is an ISO compliant *subset* of Standard Generalized Markup Language (SGML).**
- **XML (and SGML) is a meta-language.**
- **XML is *extensible.***

XML Example [1]

XML Does Not DO Anything.

Maybe it is a little hard to understand, but XML does not DO anything.

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```



Note

To: Tove

From: Jani

Reminder

Don't forget me this weekend!

The XML above is quit self-descriptive

- It has sender information.
- It has receiver information
- It has a heading
- It has a message body

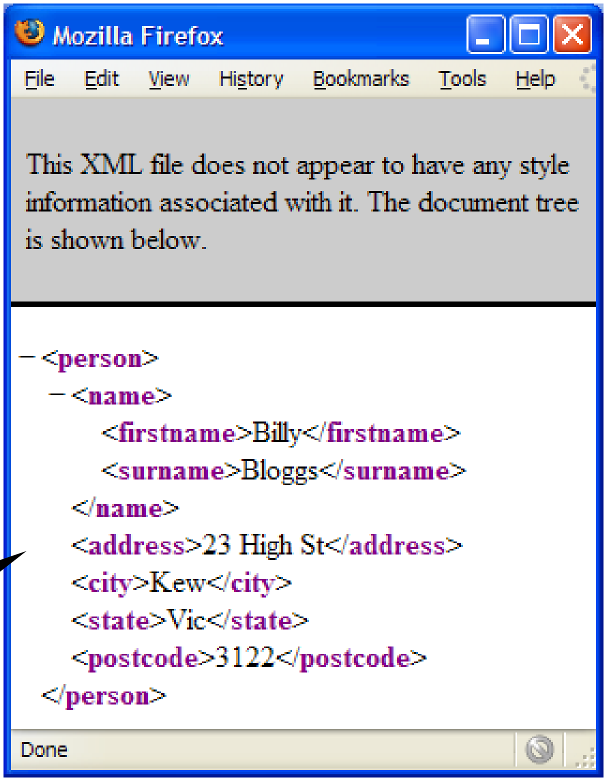
DEMO –slide6.xml

XML

- A quick look:

  Any structured data can be marked up with XML

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE person SYSTEM "person.dtd">
<person>
  <name>
      <firstname>Billy</firstname>
      <surname>Bloggs</surname>
  </name>
  <address>23 High St</address>
  <city>Kew</city>
  <state>Vic</state>
  <postcode>3122</postcode>
</person>
```



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
- <person>
  - <name>
      <firstname>Billy</firstname>
      <surname>Bloggs</surname>
    </name>
    <address>23 High St</address>
    <city>Kew</city>
    <state>Vic</state>
    <postcode>3122</postcode>
  </person>
```

Most current browsers will render ***well-formed*** XML documents. ☺

DEMO –slide7.xml

The Difference Between XML and HTML

XML and HTML were designed with different goals:

- XML was designed to carry data - with focus on what data is.
- HTML was designed to display data - with focus on how data looks.
- XML tags are not predefined like HTML tags are.

XML Does Not Use Predefined Tags

- The XML language has no predefined tags.
- The tags in the example above (like <to> and <from>) are not defined in any XML standard. These tags are "invented" by the author of the XML document.
- HTML works with predefined tags like <p>, <h1>, <table>, etc.
- With XML, the author must define both the tags and the document structure.

XML Technologies

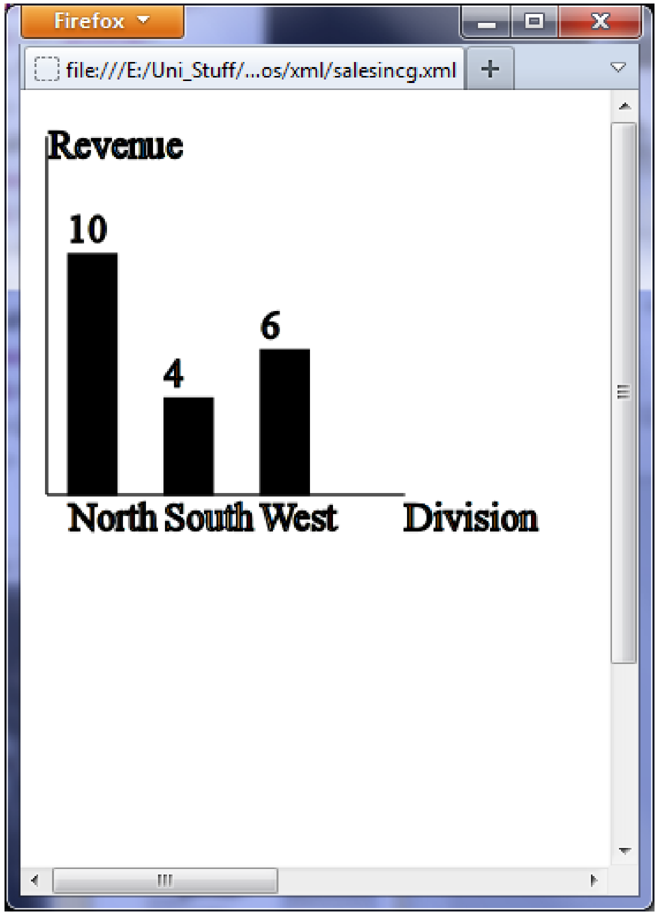
- XML is also a family of technologies
  - **XML** Syntax (Core) defines what “tags” and “attributes” are.
  - **XLink** defines how to add ***hyperlinks*** to an XML file.
  - **XPointer** defines how to ***point to parts*** of an XML file.
  - **XSL** (Extensible Style Sheet Language) can ***transform*** an XML
  - **XML Schema** used to ***define the structure*** on an XML.
  - **XML DOM** is used to access XML objects
- XML is extended and supported by many associated technologies: such as Document Type Definitions (DTDs), XML Namespaces, XML Schema and Resource Description Framework (RDF).
- These technologies, and many more, are in varying stages of the W3C specification process, and adoption.

http://www.w3.org/XML/

XSL – eXtensible Stylesheet Language

- An XML document can be transformed into HTML, SVG, or PDF, etc, using XSL.

```
<?xml version="1.0" encoding="UTF-8"?>
<sales>
        <division id="North">
                <revenue>10</revenue>
                <growth>9</growth>
                <bonus>7</bonus>
        </division>

        <division id="South">

                <revenue>4</revenue>
                <growth>3</growth>
                <bonus>4</bonus>
        </division>

        <division id="West">
                <revenue>6</revenue>

                <growth>-1.5</growth>
                <bonus>2</bonus>
        </division>
</sales>
```



See http://www.w3.org/TR/xslt

XML Document

- **Should contain a simple version declaration that tells the processor what version of XML the document conforms to:**

```
<?xml version="1.0" Encoding=‘UTF-8” ?>
```

  A Unicode-based encoding such as UTF-8 can support many languages and can accommodate pages and forms in any mixture of those languages.

- **Is considered "well-formed" if it strictly follows the syntax requirements of XML.**

- **Can be read by any XML-parser, if it is a well-formed XML document.**

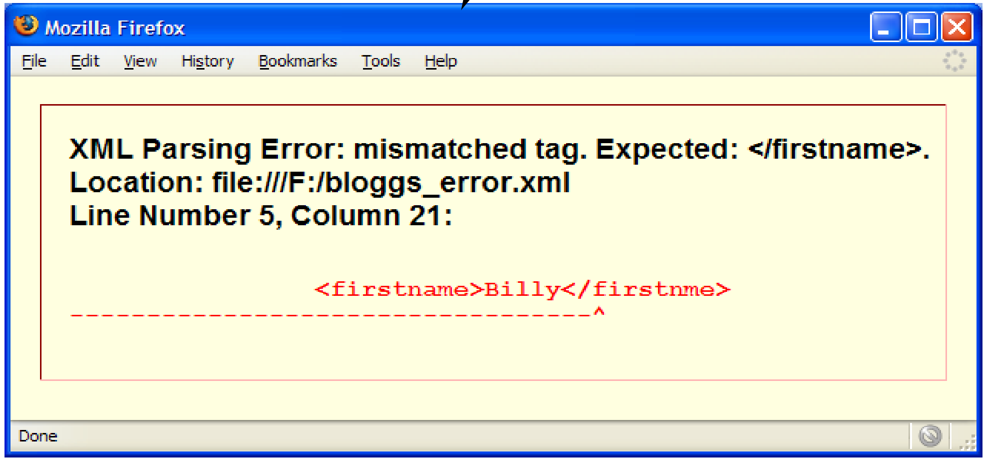
DEMO -Books.xml and nba.xml

Well-Formed XML

- **Not well-formed:**

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE person SYSTEM "person.dtd">
<person>
   <name>
        <firstname>Billy</firstnme>
        <surname>Bloggs</surname>
   </name>
   <address>23 High St</address>
   <city>Kew</city>
   <state>Vic</state>
   <postcode>3122</postcode>
</person>
```

Most browsers will **not** render XML documents that are **not** well-formed. ☹



```
XML Parsing Error: mismatched tag. Expected: </firstname>.
Location: file:///F:/bloggs_error.xml
Line Number 5, Column 21:

          <firstname>Billy</firstnme>
--------------------------------------------^
```

DEMO –slide13.xml

What is "well-formed" ?

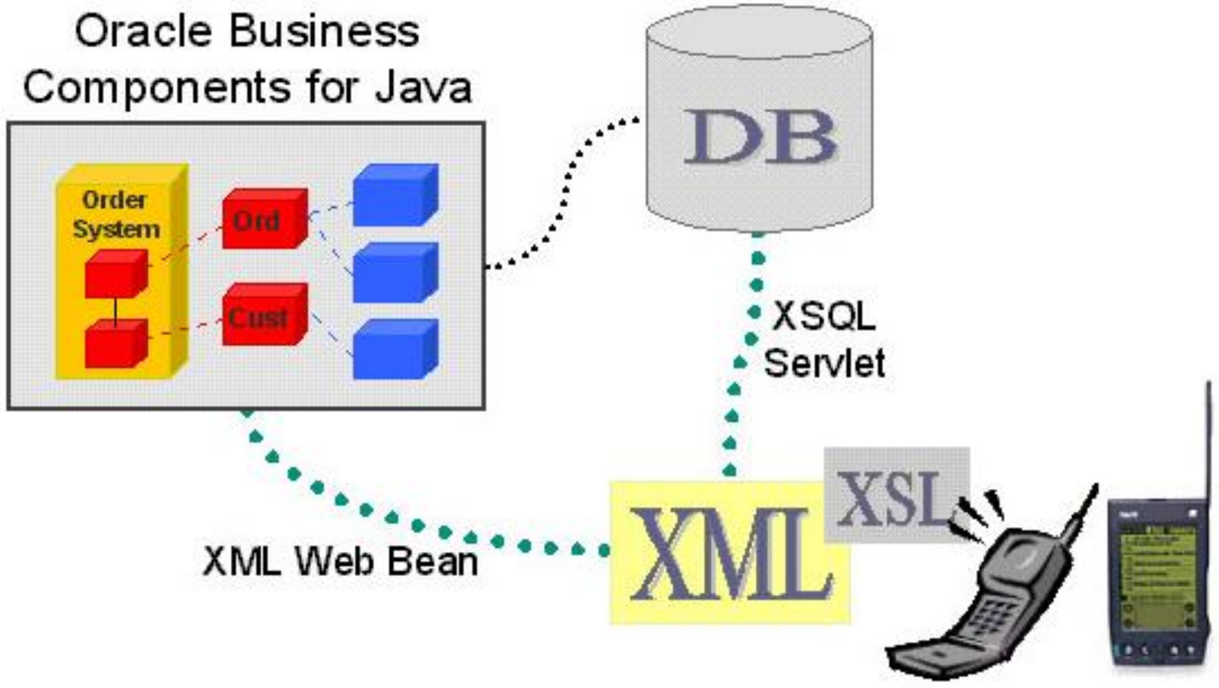
- **Rule 1: All tags must be properly closed :** ✗
  - Incorrect: `<name>Billy Bloggs`
  - Correct:: `<name>Billy Bloggs</name>` ✓
  - Correct: `<employee><name /></employee>` ✓
- **Rule 2: All tags must be properly nested:**
  - Incorrect: `<employee><name> ... </employee ></name>` ✗
  - Correct: `<employee><name> ... </name></employee>` ✓
- **Rule 3: All attribute values must be in double quotes.**
  - Incorrect: `<price currency=AUD>` ✗
  - Correct:: `<price currency="AUD">` ✓

SWIN BUR NE SWINBURNE UNIVERSITY OF TECHNOLOGY

What is "well-formed" ?

- **Rule 4: An element may not have two attributes with the same name.**
  - Incorrect: `<price currency="AUD" currency="USD">` ✗
  - Correct: `<price currency="AUD">` ✓
- **Rule 5: XML is case sensitive.**
  - `<Atag>` `<atag>` , and `<ATAG>` are *three* ***different*** *tags*
  - Incorrect: `<price>100.00</PRICE>` ✗
  - Correct: `<price>100.00</price>` ✓
- **Rule 6: There must be exactly one root element.**

XML Applications



XML Applications

- **XML files are still simple text files *(just like HTML).***
- **When XML is used for a particular project or task, it is called an "XML application", such as:**
  - XHTML: An XML application of HTML.
  - **KML / GML**: XML applications for geography, e.g., Google Maps)
  - **Ajax**: An XML application for transferring data from server to Web applications.
  - Web Services: An XML application for Service Provision
- **XML documents use the file extension .xml. Specific "XML applications" can use them however they want.**

XML Document (continued)

```
<?xml version="1.0"?>
<course>
   <subject>
        <code>COS10005</code>
        <code>COS60002</code>
        <title>Web Development</title>
        <credit>12.5</credit>
   </subject>
   <subject>
        <code>COS20022</code>
        <title>Web Programming</title>
        <credit>12.5</credit>
   </subject>
</course>
```

XSML!
e**X**tensible **S**ubject **M**arkup **L**anguage!
It does not exist, yet.

DEMO –slide18.xml

Document Type Definition (DTD)

Document Type Definition

- **Sometimes XML is too flexible.**

- **When XML documents are used to exchange data, the format (e.g., structure, elements and attributes) must be fixed.**

- **Document Type Definition (DTD) is used to specify the allowed format for the data (e.g., structure, elements and attributes).**

DTD – Example

```
<!ELEMENT course     (subject+)>
<!ELEMENT subject    (code,title,credit)>
<!ELEMENT code       (#PCDATA)>
<!ELEMENT title      (#PCDATA)>
                        TA)>
```

Content of a `<course>` element is one or many `<subject>` elements.

Content of the `<code>` element is parsed character data.

Content of a `<subject>` element is one or many `<code>` elements, a `<title>` element and a `<credit>` element.

XML validators follow those rules to validate XML documents.

SWINBURNE UNIVERSITY OF TECHNOLOGY

DTD – Element Declarations

- **For each element:**

<!ELEMENT element_name element_content>

- **Possible values for element_content:**
  - (#PCDATA): parsed character data
- <!ELEMENT title (#PCDATA)>
  - (child): one child element type
- <!ELEMENT course (subject+)>
  - (child1, ..., childn): a sequence of child element types
- <!ELEMENT subject (code+,title,credit)>
  - (child1|...|childn): one of the elements

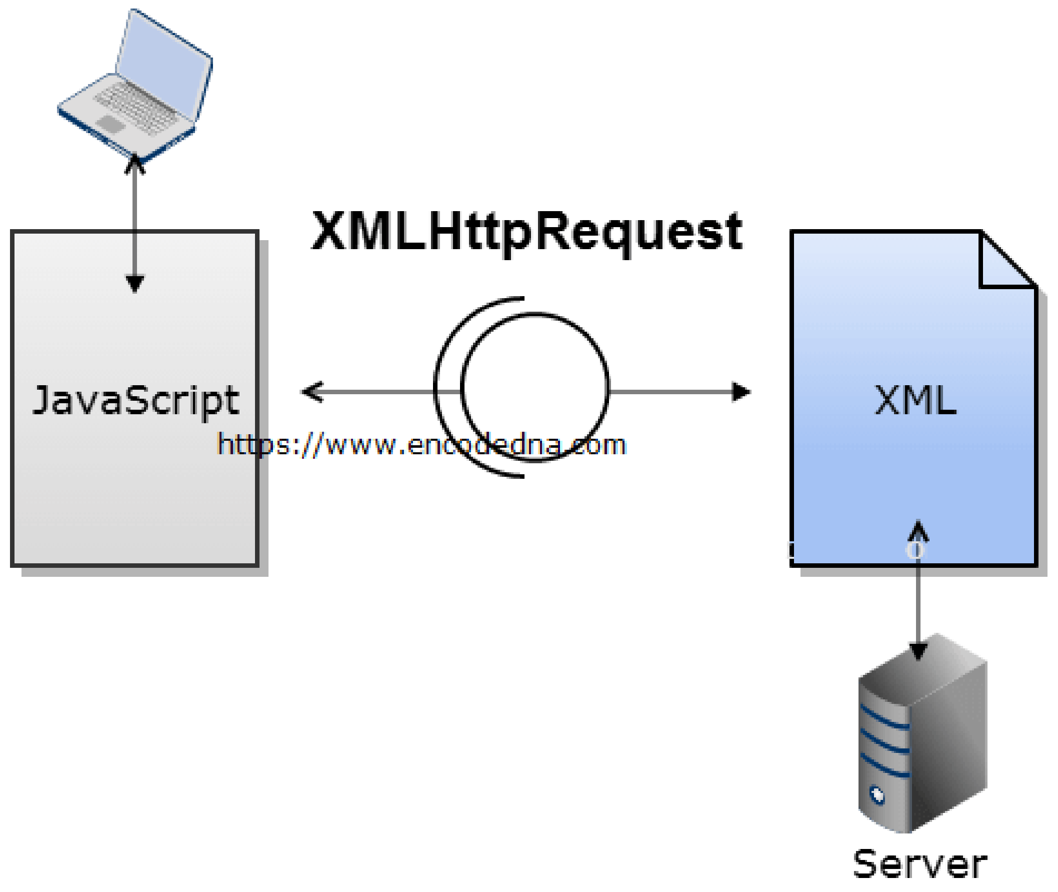
DTD – Element Declarations

<!ELEMENT element_name element_content>

- **For each child element child, possible counts can be specified:**
  - child: exactly one such element
  - child+: one or many such elements
  - child*: zero or many such elements
  - child?: zero or one such element

<!ELEMENT subject (code+,title,credit)>

Using JavaScript to Read Local XML Data



XML File

```
<?xml version="1.0" encoding="UTF-8"?>
<Teams>
      <Team>
            <TeamName>Lakers</TeamName>
            <Location>Los Angeles</Location>
            <StarPlayer>Kobe Bryant</StarPlayer>
      <Stadium>Staples Center</Stadium>     </Team>
      <Team>
            …
      </Team>
      …
</Teams>
```

DEMO –slide25.xml

STEPS-Using JavaScript to Read Local XML Data

- Step 1: Create A JavaScript Function
- Step 2: Create an XML Object
- Step 3: Setup the Request
- Step 4: Send the Request
- Step 5: Retrieve XML Data
- Step 6: Display XML Data

Step 1: Create A JavaScript Function

```
function parseXML() {
        …
}

//link functions to elements' events
function init() {
        $("#btnExecute").click(parseXML);
}

//the initialise function
$(document).ready(init);
```

Step 2: Create an XML Object

```
function parseXML() {
var xmlhttp;
if (window.XMLHttpRequest) {
  // code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp = new XMLHttpRequest();
} else  {
  // code for IE6, IE5
  xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
}
```

Step 3: Setup the Request

…

xmlhttp.open(*method*, *url*, *async*);

…

where:

method: the type of the request, GET or POST

url: the location of the target file

async: the request should be handled asynchronously or not, true or false

Example:

```
…
xmlhttp.open(“GET", “nba.xml", false);
…
```

Step 4: Send the Request

```
…
xmlhttp.send();
…
```

This statement will send the request to retrieve the XML data specified before using function `open()`, i.e., `nba.xml`.

SWIN BUR * NE * SWINBURNE UNIVERSITY OF TECHNOLOGY

Step 5: Retrieve XML Data

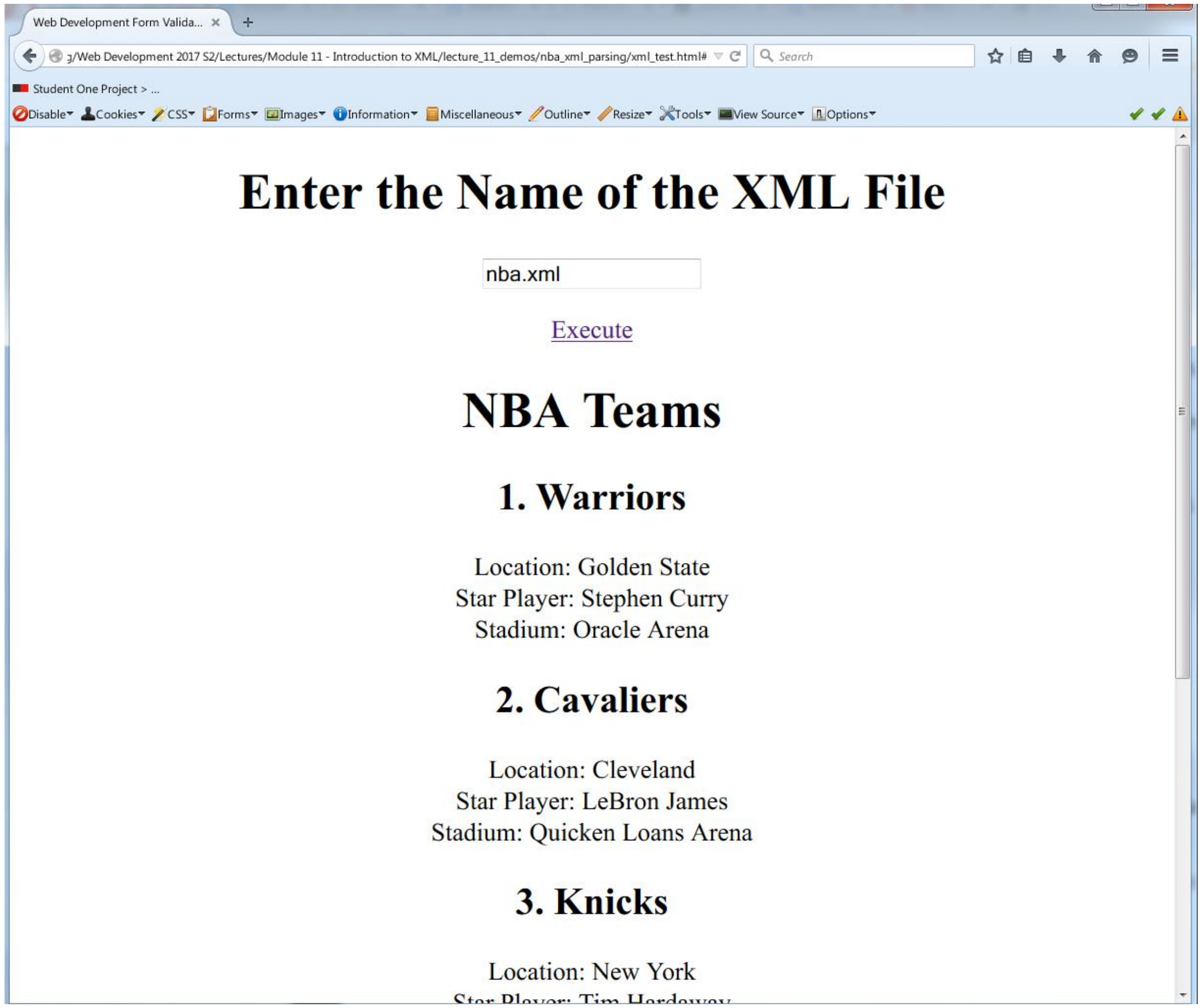
```
…
var xmlDoc = xmlhttp.responseXML;
…
//This statement will retrieve the XML data received
and save it into a variable named xmlDoc.
…
var Teams = xmlDoc.getElementsByTagName("Team");
var TeamNames = xmlDoc.getElementsByTagName("TeamName");
var StarPlayers = xmlDoc.getElementsByTagName("StarPlayer");
var Locations = xmlDoc.getElementsByTagName("Location");
var Stadiums = xmlDoc.getElementsByTagName("Stadium");
…
```

Those statements will retrieve the XML elements using their tag names, i.e., Team, TeamName, StarPlayer, Location and Stadium.

Step 6: Display XML Data

```
for(var i=0; i<Teams.length; ++i) {
    document.write("<h2>");
    document.write(i+"."+TeamNames[i].childNodes[0].nodeValue);
    document.write("</h2>");

    document.write("Location: ");
    document.write(Locations[i].childNodes[0].nodeValue);
    document.write("<br />");

    document.write("Star Player: ");
    document.write(StarPlayers[i].childNodes[0].nodeValue);
    document.write("<br />");

    document.write("Stadium: ");
    document.write(Stadiums[i].childNodes[0].nodeValue);
    document.write("<br />");
}

//This for loop will display all the retrieved XML data.
```

Result


Web Development Form Valida...
y/Web Development 2017 S2/Lectures/Module 11 - Introduction to XML/lecture_11_demos/nba_xml_parsing/xml_test.html#
Search
Student One Project > ...
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize Tools View Source Options

Enter the Name of the XML File

nba.xml

Execute

NBA Teams

1. Warriors

Location: Golden State
Star Player: Stephen Curry
Stadium: Oracle Arena

2. Cavaliers

Location: Cleveland
Star Player: LeBron James
Stadium: Quicken Loans Arena

3. Knicks

Location: New York
Star Player: Tim Hardaway

DEMO –open local xml update /openxml.html

teams.xml
nba.xml

Using aJax to Read remote Data

Step 1: Create A JavaScript Function

```
function parseXML() {
      …
}
//link functions to elements' events
function init() {
      $("#btnExecute").click(parseXML);
}
//the initialise function
$(document).ready(init);
```

Same as reading local XML file.

Step 2: Create an XML Object

```
function parseXML() {
var xmlhttp;
if (window.XMLHttpRequest) {
  // code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp = new XMLHttpRequest();
} else  {
  // code for IE6, IE5
  xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
}
…
```

Same as reading local XML file.

Step 3: Create An Event Handling Function

```
xmlhttp.onreadystatechange = function() {
        if((xmlhttp.readyState == 4) && (xmlhttp.status ==
200)) {  //when the xml data is ready
        //obtain received text
        var xmlDoc=xmlhttp.responseText;
        ////update a specific part of the page
        document.getElementById("pResult").innerHTML +=
xmlDoc;

        document.getElementById("pResult").innerHTML +=
"<br />";
        }
}
```

This function has no name. It is only used to handle the `onreadystatechange` event of the request.

Step 4: Setup the Request

```
…
xmlhttp.open(method, url, async);
```

where:

`method:` the type of the request, `GET` or `POST`

`url:` the location of the target file

`async:` the request should be handled asynchronously or not, `true` or `false`

Example:

```
xmlhttp.open("GET", “xml.php", true);
```

Step 5: Send the Request

```
        …
        xmlhttp.send();
} //end of function parseXML()
```

This statement will send the request to target php page specified before using function `open()`, i.e., `xml.php`.

References

- W3C
  http://www.w3.org/xml/ and http://www.w3.org/TR/xml/

- XML in 10 Points
  http://www.w3.org/XML/1999/XML-in-10-points

- W3Schools
  (XML Tutorial, Online Tutorial and Reference)
  http://www.w3schools.com/xml/

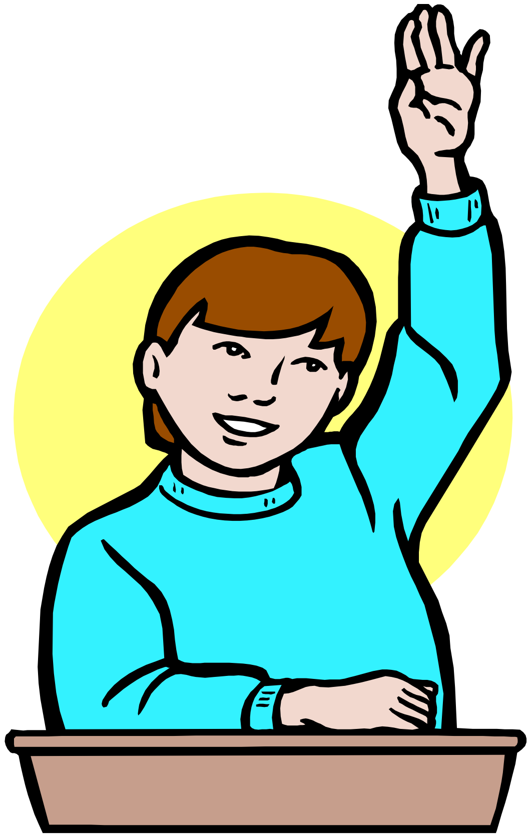
- xml.com
  http://www.xml.com/

Reminder

- Week – 11 Lab Submission
- Assignment 2 (**Demonstration Week 12**)

Question?

- A good question deserve a good grade…



Thanks Lot!!!