Working with JSON and XML

Objectives

- In this session, you will learn to:
  - Handle AJAX requests using jQuery
  - Identify JSON
  - Work with JSON
  - Identify XML
  - Identify DTD
  - Validate XML using DTD
  - Implement PCDATA
  - Identify and work with namespaces

Handling AJAX Requests using jQuery

Can I use AJAX with jQuery so that I have to write less code and update parts of a Web page?

Handling AJAX Requests using jQuery (Contd.)

- jQuery library provides you with various methods, known as jQuery AJAX methods, that allow you to make a call to the AJAX code.
- These methods allow you to perform various tasks.
- Few of these tasks are given in the following list:
  - Load data from an external file directly into a selected HTML element of you Web page.
  - Request data in a specific format, such as text, XML, or JSON.
- The following list depicts the jQuery AJAX methods:
  - `load()`
  - `get()`
  - `post()`
  - `ajax()`

Handling AJAX Requests using jQuery (Contd.)

- The `load()` method is used to load or fetch data from a Web server into a selected HTML element.
- The syntax to use the `load()` method is as follows:

```
$(selector).load(URL[,data][,complete])
```

- In the preceding syntax:
  - `URL`: Is used to specify the URL from where you want to load the data. The URL can be used to refer to any resource, such as text file or html file.
  - `data`: Is used to pass an object of a key/value pair along with the request to the server.
  - `complete`: Is used to refer to a callback method that will be executed after the successful execution of load() method.
- The following code snippet depicts how to load content of the file “Test.txt” into a specific <div> element of an HTML Web page:

```
$("#div1").load("Test.txt");
```

Handling AJAX Requests using jQuery (Contd.)

- The following code snippet depicts how to load content of an element with ID “para1” inside the file “Test.txt” into a specific <div> element of an HTML Web page:

```
$("#div1").load("Test.txt #para1");
```

- There might be situation when retrieving data from a URL resource fails.
- In such a scenario, you can use the following code snippet to display the status of the retrieval:

```
$("button").click(function(){
    $("#div1").load(“Test.txt", function(responseTxt,
statusTxt, xhr){
        if(statusTxt == "success")
            alert("External content loaded
successfully!");
        if(statusTxt == "error")
            alert("Error: " + xhr.status + ": " +
xhr.statusText);
    });
});
```

Handling AJAX Requests using jQuery (Contd.)

- The following code snippet load data from an external source and displays the AJAX call status:

LoadAJAX.htm

```
<html>
<head>
<script
src="https://ajax.googleapis.
com/ajax/libs/jquery/1.11.3/jq
uery.min.js"></script>
<script>
$(document).ready(function(){
$("button").click(function(){
$("#div1").load("Test.txt",
function(responseTxt,
statusTxt, xhr){
if(statusTxt == "success")
alert("External Data Retrieved
successfully!");
if(statusTxt == "error")
alert("Error: " +
xhr.status + ": " +
xhr.statusText);});});});
</script>
</head>
<body>
<div id="div1"><h2>The
external content will be
displayed here.</h2></div>
<button>Load External
Data</button>
</body>
</html>
```

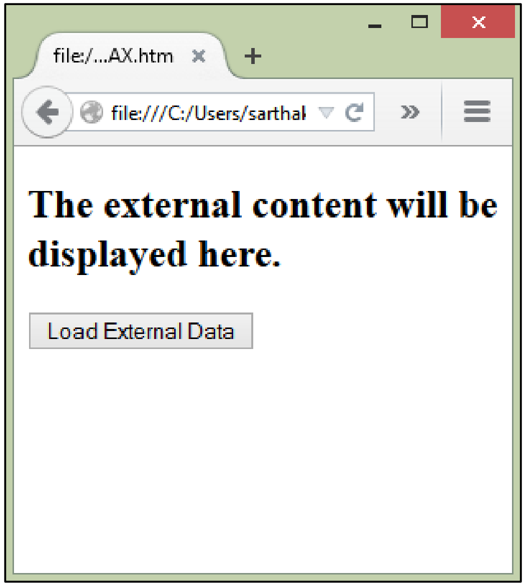
Handling AJAX Requests using jQuery (Contd.)

- The following code depicts the data inside the “Test.txt” file:

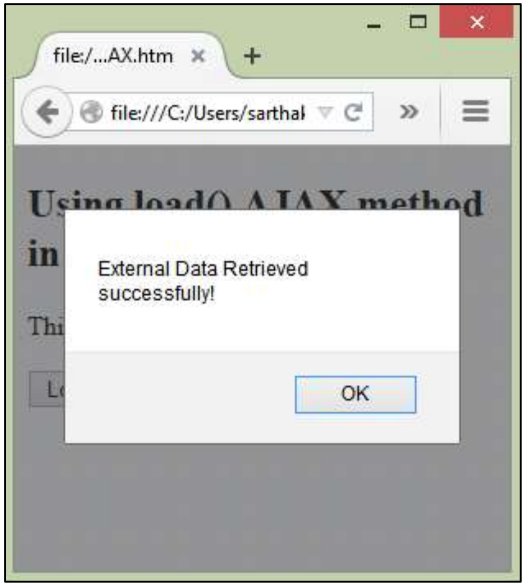
Test.txt

```
<h2>Using load() AJAX method in jQuery</h2>
<p id="p1">This is data from external resource.</p>
```

- The following figures depict the outputs of the code snippet on the preceding slide.



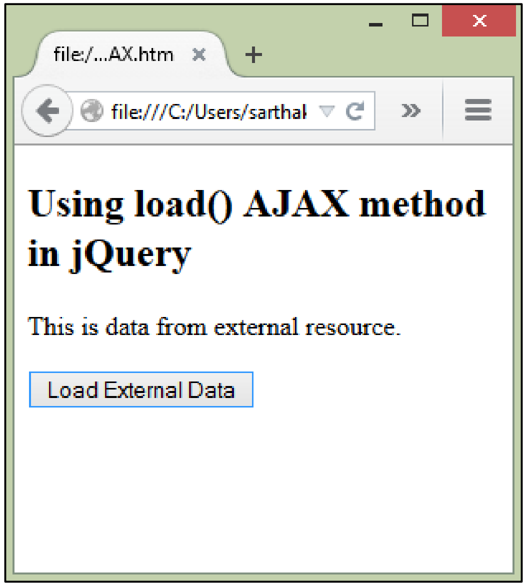
The external content will be displayed here.



Using load() AJAX method in

Thi

External Data Retrieved successfully!



Using load() AJAX method in jQuery

This is data from external resource.

Handling AJAX Requests using jQuery (Contd.)

- The `get()` method is used to load data from a Web server using the HTTP GET request.
- The syntax to use the `get()` method is as follows:

```
$(selector).get(url[,data][, callback],[datatype])
```

- In the preceding syntax:
  - `URL`: Is used to specify the URL from where you want to load the data. The URL can be used to refer to any resource, such as text file or html file.
  - `data`: Is used to pass an object of a key/value pair along with the request to the server.
  - `callback`: Is used to refer to a callback method that will be executed after the successful execution of load() method.
  - `datatype`: Is used to specify the type of data, such as text, JSON, or XML, that is expected in return from the server.

Handling AJAX Requests using jQuery (Contd.)

- The following code snippet depicts how to use `get()` method to retrieve data from a web server:

```
$("button").click(function(){
  $.get(“Test.aspx", function(data, status){
      alert("Data: " + data + "\n Status: " + status);
  });
});
```

Handling AJAX Requests using jQuery (Contd.)

- The `post()` method is:
  - Similar to the `get()` method.
  - Used to load data from a Web server using the HTTP POST request.
  - Used when the requested is large in amount.
  - Used to send the data in an encrypted format.
- The syntax to use the `post()` method is as follows:

```
$(selector).post(url[,data][, callback],[datatype])
```

- In the preceding syntax:
  - `URL`: Is used to specify the URL from where you want to load the data. The URL can be used to refer to any resource, such as text file or html file.
  - `data`: Is used to pass an object of a key/value pair along with the request to the server.
  - `callback`: Is used to refer to a callback method that will be executed after the successful execution of load() method.
  - `datatype`: Is used to specify the type of data, such as text, JSON, or XML, that is expected in return from the server.

Handling AJAX Requests using jQuery (Contd.)

- The following code snippet depicts how to use `post()` method to retrieve data from a web server:

```
$("button").click(function(){
  $.post(“LoginPage.aspx",
  {
     Uname: “User1",
     Pass: “password123"
  },
  function(data, status){
    alert("Data: " + data + "\nStatus: " + status);
  });
});
```

Handling AJAX Requests using jQuery (Contd.)

- The `ajax()` jQuery method:
  - Can be used to call the AJAX requests.
  - Helps in partial-page updates.
- The syntax to use the `ajax()` method is as follows:

```
$(selector).ajax(options)
```

- In the preceding syntax:
  - `options`: Is an optional parameter that helps in configuring the AJAX calls by using key/value pairs.

Handling AJAX Requests using jQuery (Contd.)

- The following table describes few of the important keys that can be specified as an option.

Option	*Description*
async	*A boolean value that indicates whether to execute the request asynchronously.*
complete	*A callback function that executes whenever the request finishes.*
datatype	*A string defining the type of data, such as XML, HTML, JSON, or script, that is expected back from the server.*
success	*A callback function that is executed if the request succeeds.*
type	*A string defining the HTTP method to be used for the request (GET or POST).*
URL	*A required option that refers to the string containing the URL to which the request is sent.*

Handling AJAX Requests using jQuery (Contd.)

- The following code snippet depicts how to use `ajax()` method to retrieve data from a web server:

AJAXMethod.htm

```
<html>
<head>
<title>The jQuery
Example</title>
<script
src="https://ajax.googleapis.com
/ajax/libs/jquery/1.11.3/jquery.
min.js"></script>
<script type="text/javascript"
language="javascript">
$(document).ready(function() {
$("#btn").click(function(event){
 $.ajax( {
        url:'Test.txt',
        type:'GET',
        success:function(data) {
$('#ContentArea').html(data);}
});});});
</script>
```

```
</head>
<body>
<p>Click here to load
Test.txt file:</p>


<div id="ContentArea"
style="background-color:
cyan;">
   Content Area
</div>
<input type="button"
id="btn" value="Load Data"/>
</body>
</html>
```

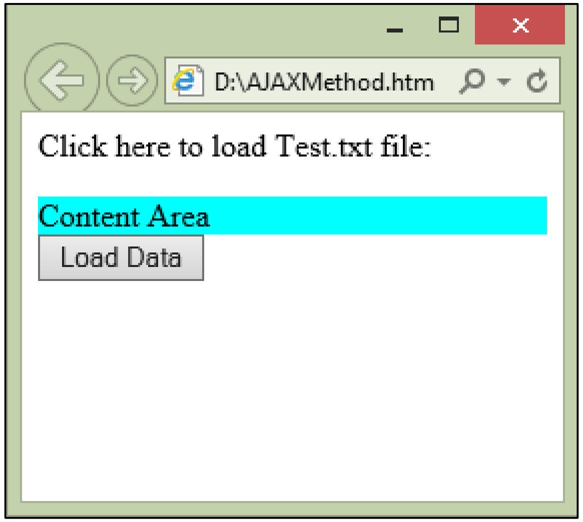
Handling AJAX Requests using jQuery (Contd.)

- The following code depicts the data inside the “Test.txt” file:

Test.txt

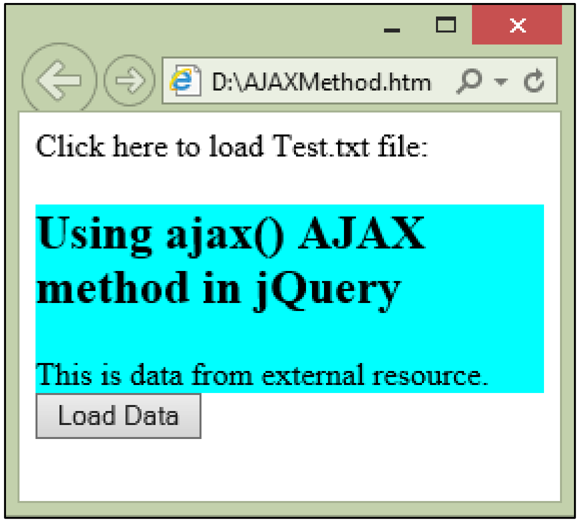
```
<h2>Using ajax() AJAX method in jQuery</h2>
<p id="p1">This is data from external resource.</p>
```

- The following figures depict the outputs of the code snippet on the preceding slide.



Click here to load Test.txt file:

Content Area



Click here to load Test.txt file:

Using ajax() AJAX method in jQuery

This is data from external resource.

Identifying JSON

Is there any other easier and faster way to transfer data other than XML?

Identifying JSON (Contd.)

- JavaScript Object Notation (JSON):
  - Is an open standard light-weight format that is used to store and exchange data.
  - Is an easier and faster alternative to XML.
  - Is language independent format that uses human readable text to transmit data objects.
  - Consists of objects of name/value pairs.
  - Files have the extension .json.
- Syntactically, JSON is similar to the code for creating JavaScript objects.
- Due to this similarity, standard JavaScript methods can be used to convert JSON data into JavaScript objects.
- The following code snippet depicts an example of JSON:

```
{"fName":"Ronald", "lName":"Smith", "Contact":"121 12345"}
```

JSON Syntax

- The following list depicts the similarities between JSON syntax and code for JavaScript object:
  - JSON uses name/value pairs to store data.
  - Commas are used to separate multiple data values.
  - Objects are enclosed within curly braces.
  - Square brackets are used to store arrays.
- The following code depicts how to create name/value pairs:

```
"Name":"Value"
```

- JSON keys must be enclosed within double quotes.

JSON Values

- The following code snippet depicts storing different types of values using JSON name/value pairs:

```
"fName": "Jane"   \\Storing string value
"lName": "Doe"    \\Storing string value
"isAlive": true   \\Storing boolean value
"age": 23         \\Storing integer value
"children": []    \\Storing an array
"spouse": null    \\Storing null
```

- A JSON object can include the following types of values:
  - A numeric value
  - A string
  - A boolean value
  - An array
  - An object
  - A null value

JSON Objects

- JSON objects are enclosed within curly braces.
- Similar to JavaScript objects, JSON objects can be used to store multiple name/value pairs.
- The following code snippet depicts storing data in a JSON object:

```
{
"fName": "Jane",
"lName": "Doe",
"isAlive": true,
"age": 23,
"children": [],
"spouse": null
}
```

JSON Arrays

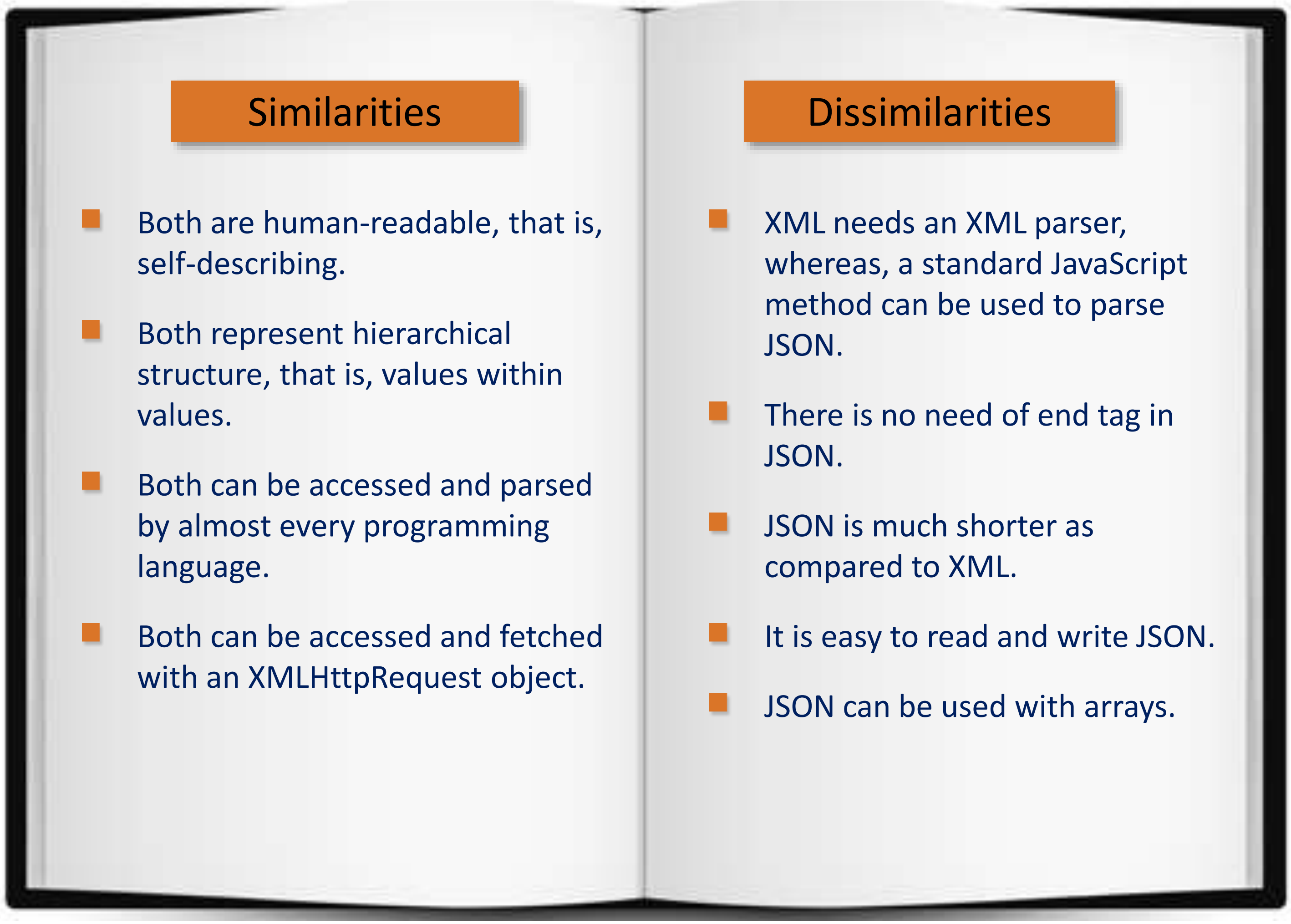
- JSON arrays can be created by using square brackets, as shown in the following code snippet:

```
{
"fName": "Jane",
"lName": "Doe",
"isAlive": true,
"age": 23,
"ContactNumber":[
{"type":"Mobile", "Number":"+9198765" }
{"type":"Office", "Number":"+9124456" }
]
"children": [],
"spouse": null
}
```

- Since JSON uses the same syntax as that of JavaScript objects, JSON arrays can be accessed in the same way as in JavaScript.

JSON vs. XML

- The following figure depicts similarities and dissimilarities between JSON and XML:



Similarities

- Both are human-readable, that is, self-describing.
- Both represent hierarchical structure, that is, values within values.
- Both can be accessed and parsed by almost every programming language.
- Both can be accessed and fetched with an XMLHttpRequest object.

Dissimilarities

- XML needs an XML parser, whereas, a standard JavaScript method can be used to parse JSON.
- There is no need of end tag in JSON.
- JSON is much shorter as compared to XML.
- It is easy to read and write JSON.
- JSON can be used with arrays.

JSON vs XML (Contd.)

- The following code snippet depicts a JSON example that defines a student object containing an array of records of two students:

```
{"students":[
    {"fName":"Jenny", "lName":"Watson"},
    {"fName":"Dean", "lName":"Smith"}
]}
```

- The following code snippet depicts an XML example that defines a student object containing records of two students:

```
<students>
  <student>
    <fName>Jenny</fName>
    <lName>Watson</lName>
  </student>
  <student>
    <fName>Dean</fName>
    <lName>Smith</lName>
  </student>
</students>
```

Reading Data From JSON

- A most common usage of JSON objects is to read/fetch data from a Web server in JSON format, and display it on an HTML Web page.
- To read data from a JSON object, you can use the `JSON.parse()` method provided by JavaScript.
- The syntax for `JSON.parse()` method is as follows:

```
var obj = JSON.parse(text);
```

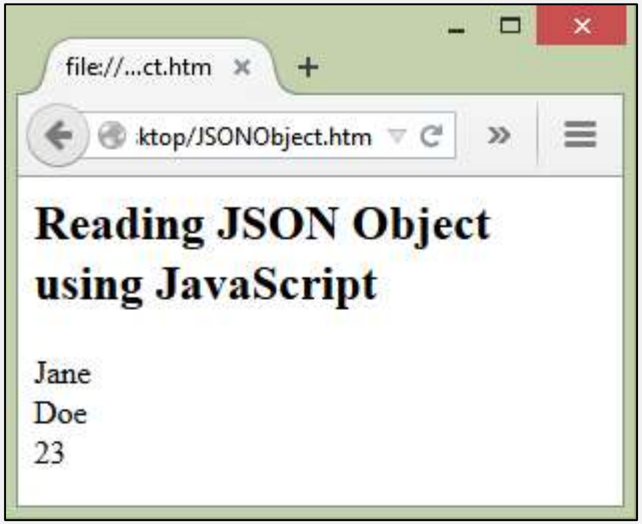
- The following code snippet depicts how to use the `JSON.parse()` method:

```
var jsonData =
'{"fName": "Jane", "lName": "Doe", "isAlive": true,
"age": 23}';
var contact = JSON.parse(jsonData);
document.write(contact.lName+", "+contact.fName);
```

Reading Data From JSON (Contd.)

- The following code snippet depicts how to read data using `JSON.parse()` method:

```
<html>
<body>
<h2>Reading JSON Object using JavaScript</h2>
<p id="pData"></p>
<script>
var jsonData =
'{"fName": "Jane", "lName": "Doe", "isAlive": true, "age":
23}';
var contact = JSON.parse(jsonData);
document.getElementById("pData").
innerHTML =
contact.fName + "<br>" +
contact.lName + "<br>" +
contact.age;
</script>
</body>
</html>
```



Reading JSON Object using JavaScript

```
Jane
Doe
23
```

Creating JSON Text From JavaScript

- JavaScript provides you the `JSON.stringify()` method that allows you to convert JavaScript value to a JSON string.
- The syntax for `JSON.stringify()` method is as follows:

```
var obj = JSON.stringify(value);
```

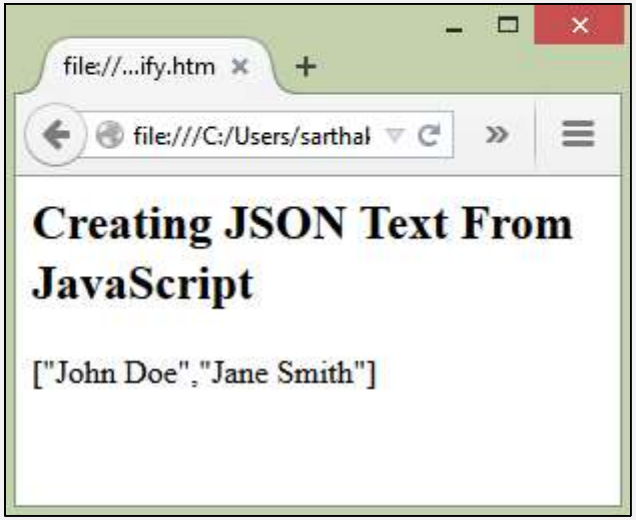
- The following code snippet depicts how to convert JavaScript value into JSON text/string using the `JSON.stringify()` method:

```
var obj = new Object();
obj.fname = "John";
obj.lname = "Doe";
jsonText = JSON.stringify(obj);
document.write(jsonText);

//Output will be {"fname":"John","lname":"Doe"}
```

Reading Data From JSON (Contd.)

- The following code snippet depicts how to convert JavaScript value into JSON text/string using the `JSON.stringify()` method:

```
<html>
<body>
<h2>Reading JSON Object using JavaScript</h2>
<p id="pData"></p>
<script>
var Students = new Array();
Students[0] = "John Doe";
Students[1] = "Jane Smith";
var jsonText = JSON.stringify(Students);
document.getElementById("pData").
innerHTML = jsonText; </script>
</body>
</html>
```



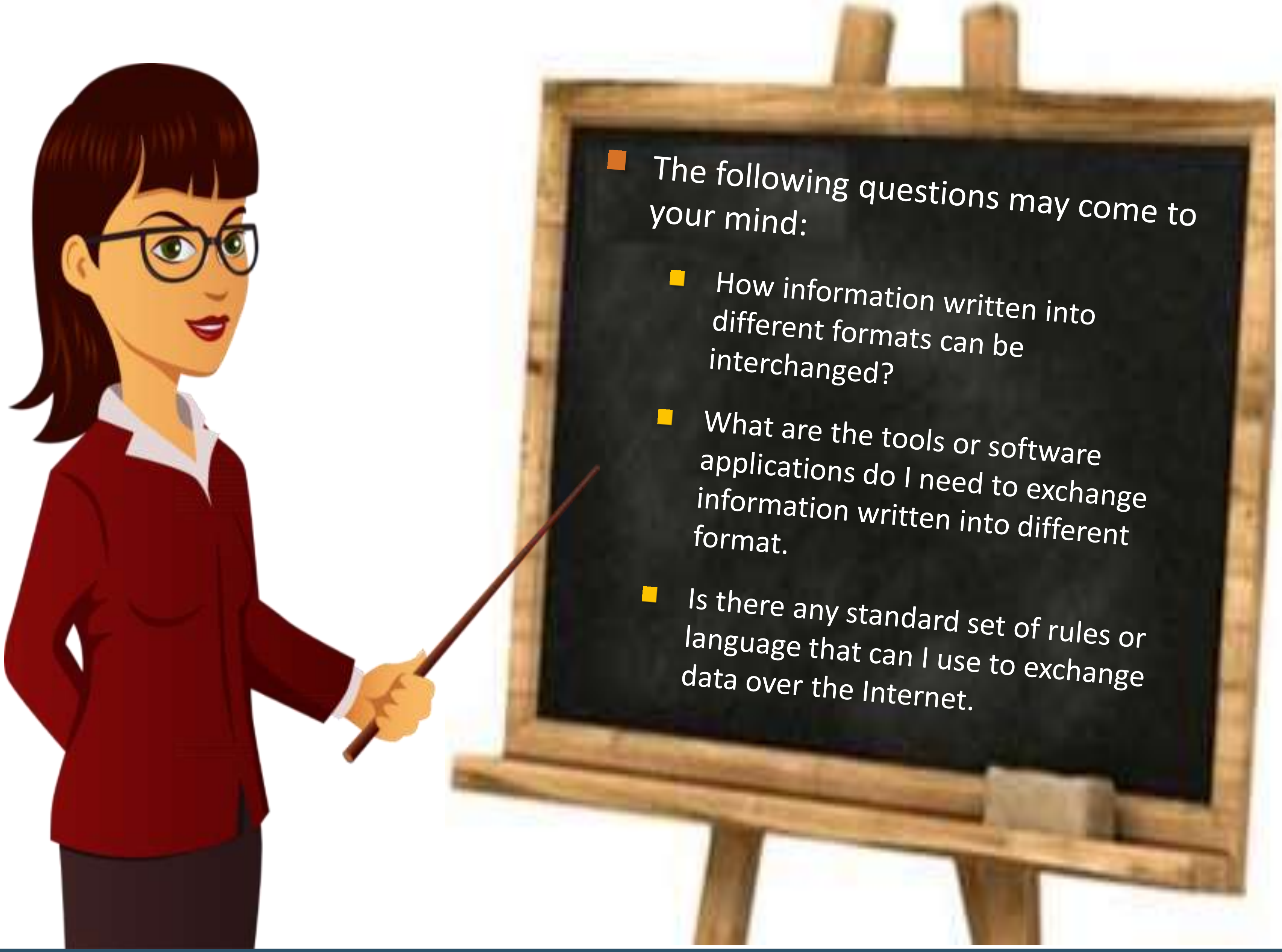
Creating JSON Text From JavaScript

["John Doe","Jane Smith"]

Introduction to XML

Consider a scenario where you need to exchange data in different formats.

Introduction to XML (Contd.)



- The following questions may come to your mind:
  - How information written into different formats can be interchanged?
  - What are the tools or software applications do I need to exchange information written into different format.
  - Is there any standard set of rules or language that can I use to exchange data over the Internet.

Introduction to XML (Contd.)

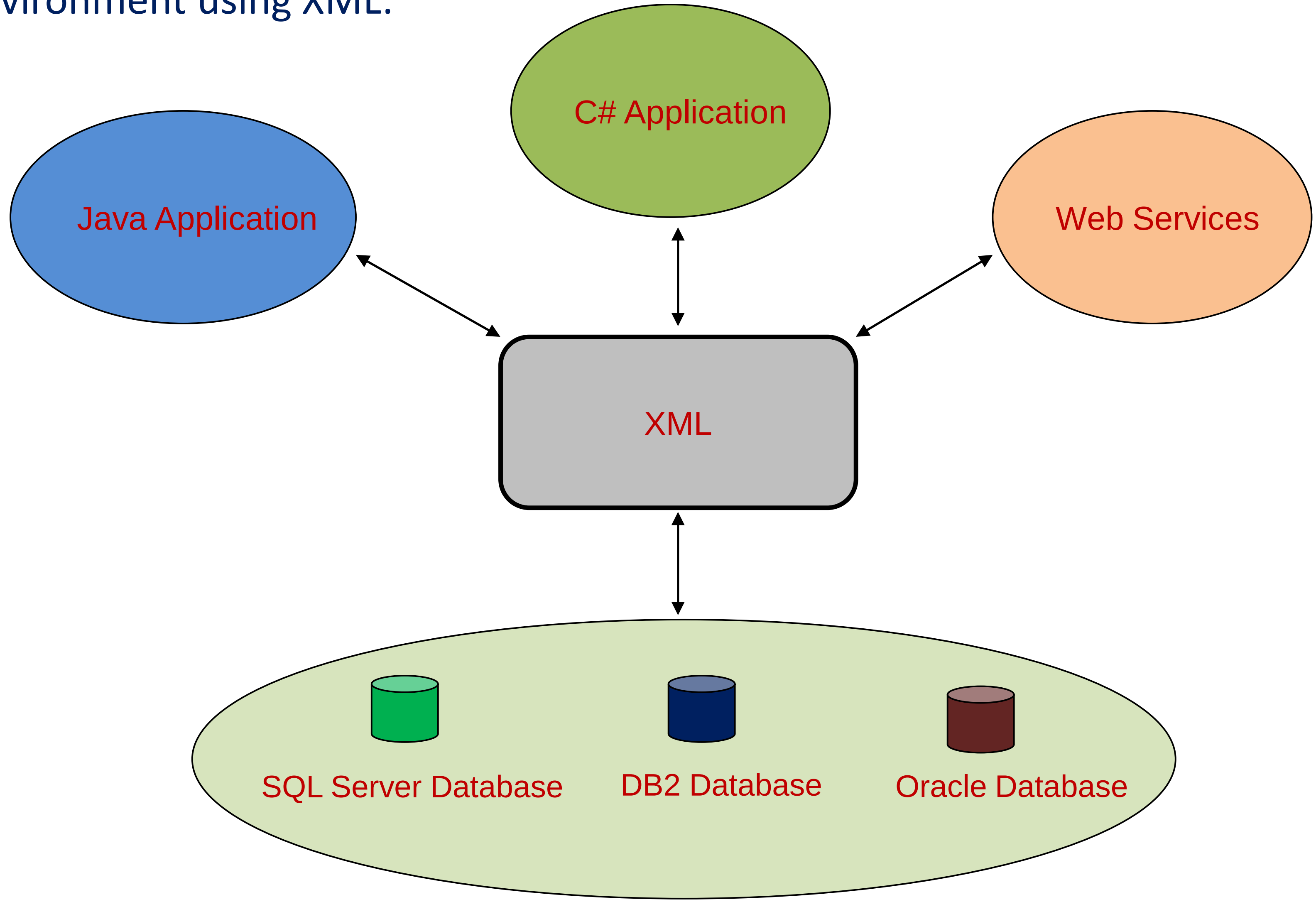
- XML:
  - Stands for Extensive Markup Language.
  - As its name suggests, it has following basic characteristics:
    - Extensible: It is an extensible language that allows programmers to define their own tags.
    - Markup: It is based on markup tags similar to HTML tags.
    - Language: It is similar language to HTML. In addition, offers more flexibility and power to programmers to describe data.

Introduction to XML (Contd.)

- XML:
  - Is a self-describing language.
  - Is recommended by W3C for information exchange over the Internet.
  - Tags are not pre-defined, to use it, you need to define your own tags.
  - Is a software and hardware independent language.
  - Enables programmers to store data in a well-formed structure.
  - Transfers data between various heterogeneous systems over the network.

Introduction to XML (Contd.)

- The following figure depicts how data is exchanged to heterogeneous environment using XML:



Advantages of XML

- Some common advantages of XML are:
  - **Extensible**: It is an extensible language that allows programmers to define their own tags based on the specific requirements.
  - **Data Interchange**: It allows programmers to store data in textual format that can be used as a standard to interchange data.
  - **Smart Searches**: It allows programmers to specify whether they want to search information based on text or tags, and returns the information that matches the search criteria.
  - **Fast Updates**: Use of XML allows offers fast update of information, as only text needs to be updated.
  - **CSS and XSL Support**: XML supports CSS (Cascading Style Sheet) and (Extensible Style Sheet) languages that can be used to apply required formatting of an XML document.
  - **Data Transformation**: As per requirements, data can be stored in the form of text, object, or data in a database. The stored data can be extracted by the client application in the required format.

Advantages of XML (Contd.)

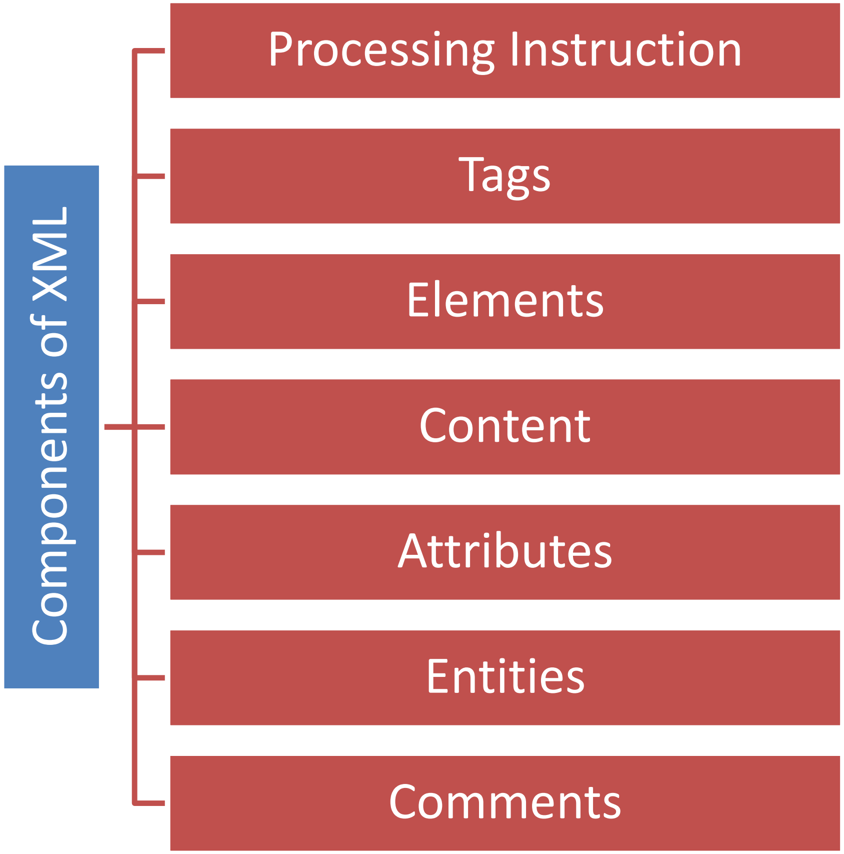
- **Separate Content/Presentation**: XML defines the meaning of data. The representation of data can be controlled with the help of CSS and XSL languages.
- **New Languages**: XML can be also used to develop new languages. Some languages that are derived from XML are:
  - XHTML
  - WSDL
  - SMIL
  - RSS

Creating Well-Formed XML

How can I create my own XML document?

Creating Well-formed XML (Contd.)

- XML is used to represent data in a well-formed structure.
- It provides you various components that you can use to define the structure of data.



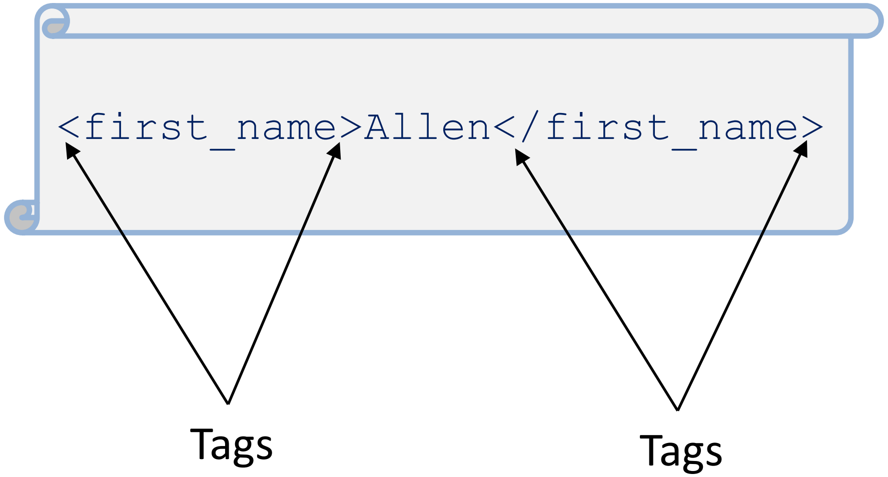
Creating Well-Formed XML (Contd.)

- Processing Instruction (PI):
  - Specifies how an XML document is processed.
  - Is included in an XML document by adding the following line of code at the starting of the file:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Creating Well-formed XML (Contd.)

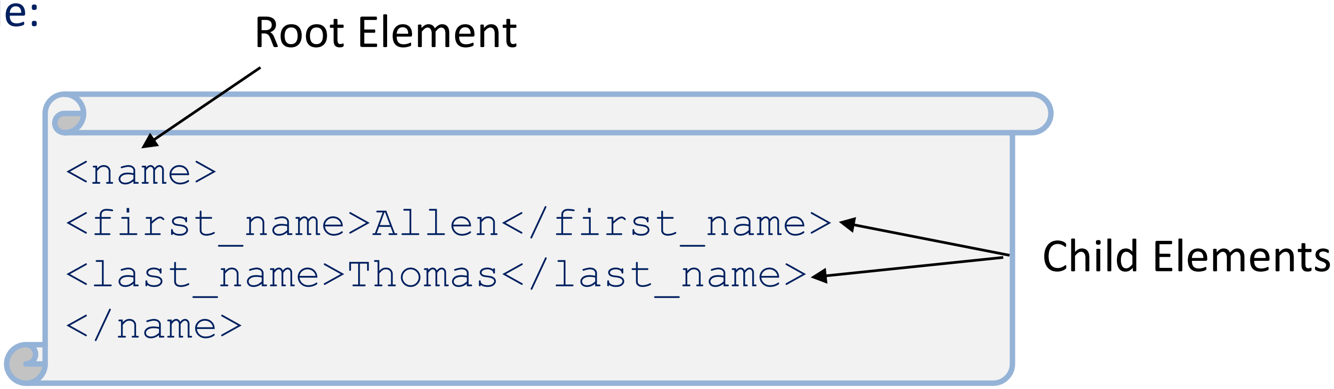
- Tags:
  - Specify the name of information presented in an XML document.
  - Must have opening (<>) and closing (</>) brackets that encloses the name of the tag.
  - For example:



```
<first_name>Allen</first_name>
```

Creating Well-formed XML (Contd.)

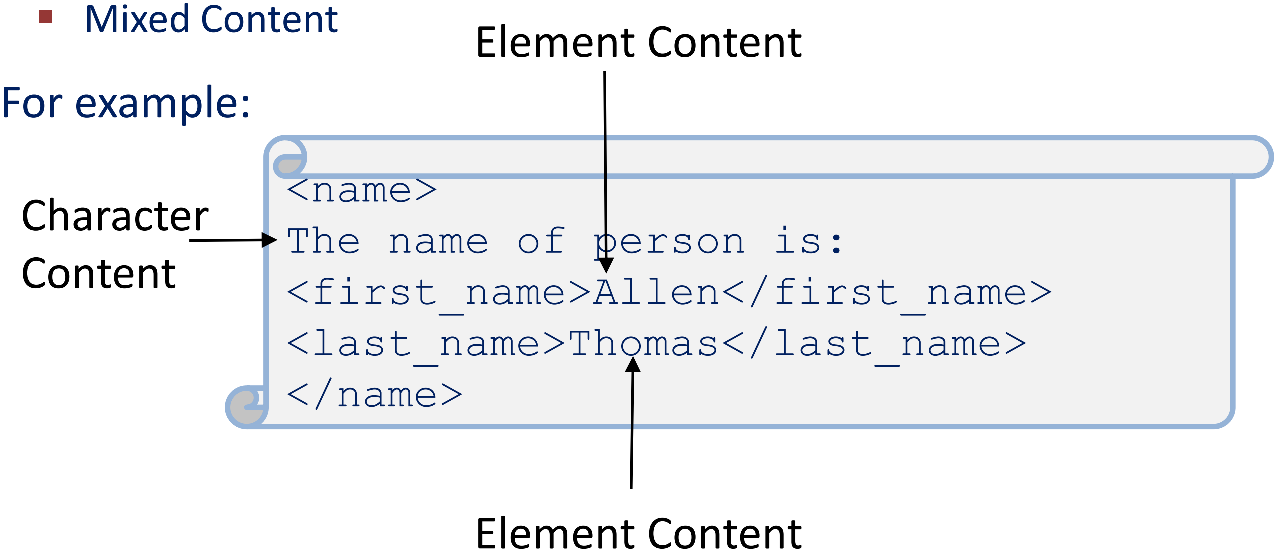
- Elements:
  - Are the basic building blocks of XML.
  - Are represented with the help of tags.
  - Are used to describe data in an XML document.
  - Can contains one or more elements.
  - That contain child elements are known as root element.
  - For example:



```
<name>
<first_name>Allen</first_name>
<last_name>Thomas</last_name>
</name>
```

Creating Well-Formed XML (Contd.)

- Content:
  - Refers the information represented by the elements.
  - Can be categories into following types:
    - Character Content
    - Element Content
    - Mixed Content
  - For example:



Element Content

Character Content

```
<name>
The name of person is:
<first_name>Allen</first_name>
<last_name>Thomas</last_name>
</name>
```

Element Content

Creating Well-formed XML (Contd.)

- Attributes:
  - Allow programmers to provide additional information about the elements.
  - Are created in the form of name-value pair.
  - For example:

```
<emp emp_id="001">
<first_name>Allen</first_name>
<last_name>Thomas</last_name>
</emp>
```

Creating Well-formed XML (Contd.)

- Entity:
  - Allows the insertion of special characters in XML documents.
  - For example, the &lt; entity inserts a literal < character into a document.
  - Supports the following in-build entities:
    - &lt;
    - &gt;
    - &quote;
    - &amp;

Creating Well-formed XML (Contd.)

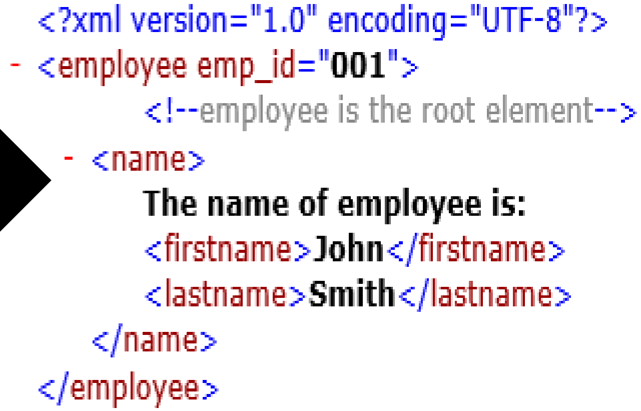
- Comments:
  - Are used to explain the purpose of XML markup.
  - Are not evaluated by the XML parser.
  - Are enclosed within <!..- -> symbols.
  - For example:

```
  <!--emp element has two child elements-->
<emp emp_id="001">
<first_name>Allen</first_name>
<last_name>Thomas</last_name>
</emp>
```

Creating Well-formed XML (Contd.)

- The following code depicts the output well-formed XML document:

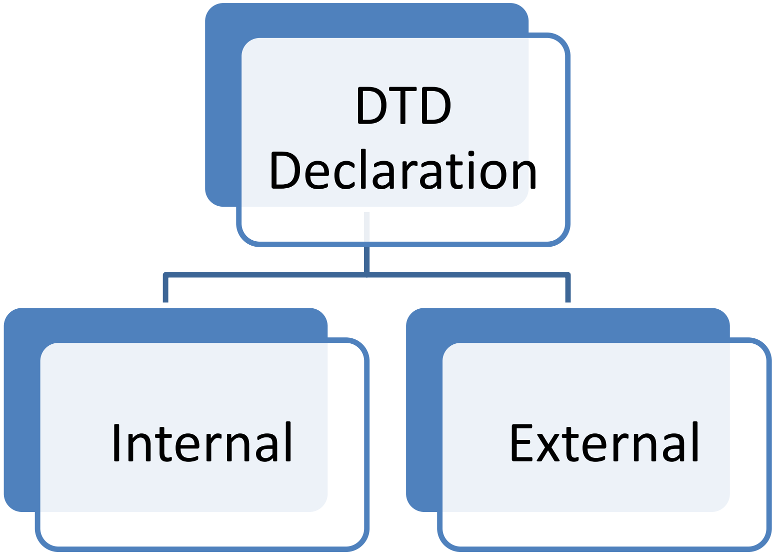
```
<?xml version="1.0" encoding="utf-8"?>
<employee emp_id="001">
<!--employee is the root element-->
<name>
The name of employee is:
    <firstname>John</firstname>
    <lastname>Smith</lastname>
</name>
</employee>
```



```
  <?xml version="1.0" encoding="UTF-8"?>
- <employee emp_id="001">
      <!--employee is the root element-->
  - <name>
      The name of employee is:
      <firstname>John</firstname>
      <lastname>Smith</lastname>
    </name>
  </employee>
```

Specifying Rules Using DTD

- DTD
  - Stands for Document Type Definition.
  - Is used to define the structure and the valid elements and attributes of an XML document.
  - Verifies the data in an XML document.
  - Can be used to verify data of a local file or data received over the network.
  - Can be declared as of the following two types:



Specifying Rules Using DTD (Contd.)

- Internal DTD Declaration:
  - Is declared inside the XML file.
  - Must be enclosed inside the <!DOCTYPE> definition, as:

```
<?xml version="1.0"?>
<!DOCTYPE name [
<!ELEMENT name (first_name, last_name)>
<!ELEMENT first_name (#PCDATA)>
<!ELEMENT last_name (#PCDATA)>
]>
```

  - In the preceding code snippet:
    - !DOCTYPE name: Specifies that the name is the root element of this document.
    - <!ELEMENT name(first_name, last_name): Specifies that the name element must has two elements, first_name and last_name.
    - <!ELEMENT first_name: Specifies the first_name element to be of type PCDATA.
    - <!ELEMENT last_name: Specifies the last_name element to be of type PCDATA.

Specifying Rules Using DTD (Contd.)

- External DTD Declaration:
  - Is declared outside the XML file.
  - References of the DTD file must be included in the `<!DOCTYPE>` definition.
  - For example:

name.dtd

```
<!ELEMENT name
(first_name, last_name)>
<!ELEMENT first_name
(#PCDATA)>
<!ELEMENT last_name
(#PCDATA)>
```

name.xml

```
<?xml version="1.0"?>
<!DOCTYPE name SYSTEM
"name.dtd">
<name>
<first_name>John</first_name
>
<first_name>Thomas</first_na
me>
</name>
```

Creating Valid XML Using DTD

- An XML that has correct syntax is known as a well-formed document.
- An XML document that is validated against a DTD is a well-formed as well as a valid document.
- A valid XML document refers to a well-formed (has correct syntax) and follows all rules specified in a DTD.
- A DTD is used to define the structure of an XML document in the form of a list of valid elements, as shown in the following code snippet:

name.dtd

```
<!DOCTYPE name
[
<!ELEMENT name (first_name, last_name)>
<!ELEMENT first_name(#PCDATA)>
<!ELEMENT last_name(#PCDATA)>
]>
```

Creating Valid XML Using DTD (Contd.)

- The DTD definition given in the previous slide is interpreted as follows:
  - `!DOCTYPE name` defines that the root element of the document is note
  - `!ELEMENT name` defines that the note element must contain two elements: first_name and last_name
  - `!ELEMENT first_name` defines the `first_name` element is to be of type `PCDATA`
  - `!ELEMENT last_name` defines the `last_name` element to be of type `PCDATA`
- When the specified DTD document is referenced in an XML document, the document must follow the rules specified in DTD.

Element Declaration

- You can declare elements by using the `ELEMENT` declaration. In a DTD, you can declare the following type of elements:
  - Empty Elements
  - Elements with Parsed Character Data
  - Elements with any Contents
  - Elements with Children
  - Declaring Only One Occurrence of an Element
  - Declaring Minimum One Occurrence of an Element
  - Declaring Zero or More Occurrences of an Element
  - Declaring Zero or One Occurrences of an Element
  - Declaring either/or Content
  - Declaring Mixed Content
- The general syntax of declaring a DTD is shown in the following syntax:

```
<!ELEMENT element-name category>
```

Element Declaration (Contd.)

- Empty Elements:
  - Are declared with the `Empty` keyword.
  - Syntax:

```
<!ELEMENT element-name category>
```

  - Example:

```
<!ELEMENT p EMPTY>
```

- The syntax of declaring an element is shown in the following code snippet:

Element Declaration (Contd.)

- Elements with Parsed Character Data:
  - Are declared with #PCDATA inside parentheses.
  - Syntax:

```
<!ELEMENT element-name (#PCDATA)>
```

  - Example:

```
<!ELEMENT from (#PCDATA)>
```

Element Declaration (Contd.)

- Elements with any Contents:
  - Are declared with the `ANY` keyword inside parentheses.
  - Can contain any kind of allowed XML data.
  - Syntax:

```
<!ELEMENT element-name ANY>
```

  - Example:

```
<!ELEMENT note ANY>
```

Element Declaration (Contd.)

- Elements with Children:
  - Are declared with name of one or more children inside the parentheses.
  - Syntax:

```
<!ELEMENT element-name(child1, child2,...)>
```

  - Example:

```
<!ELEMENT name(first_name, middle_name, last_name)>
```

Element Declaration (Contd.)

- Only One Occurrence of an Element
  - Is used to declare an element such that the child element must occur once and only once inside the root element.

```
<!ELEMENT element-name (child-name)>
```

  - Example:

```
<!ELEMENT name(full_name)>
```

Element Declaration (Contd.)

- Minimum One Occurrence of an Element:
  - Is declared with the plus (+) sign.
  - Syntax:

```
<!ELEMENT element-name (child-name+)>
```

  - Example:

```
<!ELEMENT contacts (mobile_number+)>
```

Element Declaration (Contd.)

- Zero or More Occurrences of an Element:
  - Is declared with the asterisk `(*)` symbol.
  - Syntax:

```
<!ELEMENT element-name (child-name*)>
```

  - Example:

```
<!ELEMENT note (email_id*)>
```

Element Declaration (Contd.)

- Zero or One Occurrences of an Element:
  - Is declared with the question mark `(?)` symbol.
  - Syntax:

```
<!ELEMENT element-name (child-name?)>
```

  - Example:

```
<!ELEMENT note (email_id?)>
```

Element Declaration (Contd.)

- Either/or Content:
  - Is declared with the pipe `(|)` symbol.
  - Syntax:

```
<!ELEMENT note (child1|child2)>
```

  - Example:

```
<!ELEMENT note (email_id|mobie_number)>
```

Element Declaration (Contd.)

- Mixed Content:
  - Is declared with the pipe `(|)` and asterisk `(*)` symbols.
  - Syntax:

```
<!ELEMENT contacts(#PCDATA|child1|child2|child3)*>
```

  - Example:

```
<!ELEMENT contacts(email_id|mobie_number|address)*>
```

Use of PCDATA

- `PCDATA` refers to the parsed character data.
- It is generally rep[resents the character data in the form of text that is found between the start tag and the end tag of an XML element.
- The text of `PCDATA` is parsed by a parser.
- The XML parser examines the `PCDATD` text to find entities and markup.
- Tags found inside the `PCDATA` text are treated as markup.
- Entities found inside the `PCDATA` are explained to their specific meaning.
- The `PCDATD` should not include `&`, `<`, or `>` characters. You can include these characters by using the `&amp;`, `&lt;`, and `&gt;` entities.

Attribute Declaration in DTD

- You can declare attributes with an `ATTLIST` declaration.
- To declare an attribute in DTD, you can use the following code snippet:

```
<!ATTLIST element-name attribute-name attribute-type
attribute-value>
```

- Example:

```
<!ATTLIST employee id CDATA "011">
```

- XML Example:

```
<employee radius="011" />
```

Attribute Types

- While declaring an attribute in DTD, you can use one of the following values for `attribute-type`.

Type	Description
CDATA	*Specifies that the value is character type.*
(en1|en2|..)	*Specifies that the value must be from an enumerated list.*
ID	*Specifies that the value is unique ID.*
IDREF	*Specifies that the value is the ID of another element.*
NMTOKEN	*Specifies that the value is a valid XML name.*
NMTOKENS	*Specifies that the value is a list of valid XML names.*
ENTITY	*Specifies that the value is an entity.*
ENTITIES	*Specifies that the value is an list entities.*
NOTATION	*Specifies that the value is the name of notation*
xml:	*Specifies that the value is the pre-defined XML value.*

Fixed, Default and Optional Types

- In DTD, you can declare attribute values with one of the following types:

Default

REQUIRED

IMPLIED

FIXED

Fixed, Default, and Optional Types (Contd.)

- Attribute with Default Value:
  - In DTD, you can declare an attribute with a default value.
  - If no value is specified for that attribute, the default value is automatically assigned.
  - For example:

```
DTD Declaration:
<!ELEMENT circle EMPTY>
<!ATTLIST circle radius CDATA "0">

XML Markup:
<circle radius="10" />
```

  - In the preceding example, if the value of `radius` attribute is not provided, the default value, 0, is assigned.

Fixed, Default and Optional Types (Contd.)

- Attribute with `#REQUIRED` Keyword:
  - Is used to force the users to provide the value of the attribute.
  - For example:

```
DTD Declaration:
<!ATTLIST employee contact CDATA #REQUIRED>

Valid XML Markup:
<employee contact="897897676" />

Invalid XML Markup:
<employee />
```

Fixed, Default and Optional Types (Contd.)

- Attribute with `#IMPLIED` Keyword:
  - Is used to when the attribute is optional and does not have an default value.
  - For example:

```
DTD Declaration:
<!ATTLIST employee contact CDATA #IMPLIED>

Valid XML Markup:
<employee contact="897897676" />

Valid XML Markup:
<employee />
```

  - In the preceding code snippet, the `contact` attribute is optional. The user is not forced to provide the value of the attribute.

Fixed, Default and Optional Types (Contd.)

- Attribute with #FIXED Keyword:
  - Is used to provide a fixed value to the attribute. The value cannot be changed in the XML document.
  - For example:

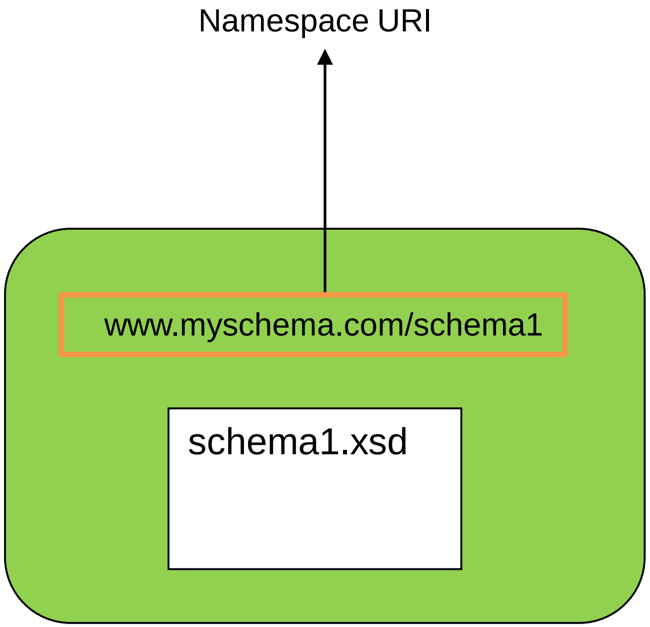
```
DTD Declaration:
<!ATTLIST number base CDATA #FIXED "decimal" >

Valid XML Markup:
<number base="decimal" />

Invalid XML Markup:
<number base="binary" />
```

Namespaces

What is a namespace in XML?

Namespaces (Contd.)

- A namespace can be considered as a virtual space that is identified by a Uniform Resource Identifier (URI).
- It is represented in the form of a string that uniquely identifies the elements and attributes from different schemas, as shown in the following figure.



Namespaces (Contd.)

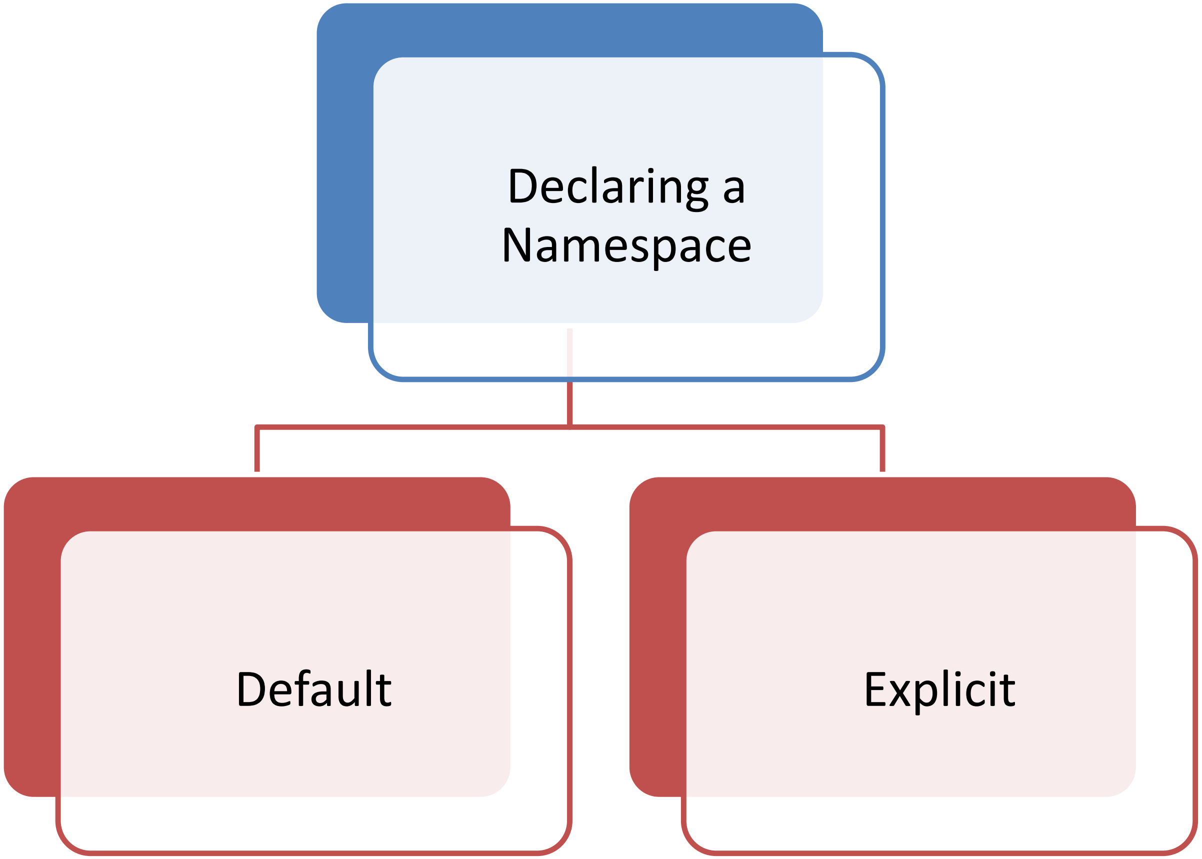
- Namespace URI:
  - Is not a Web URI.
  - does not locate a resource on the Internet.
  - Does not actually point to a resource on the Internet.
  - Is a unique identifier that resolves conflicts between elements having same name.
- Declaring a Namespace:
  - You can declare an namespace with the help of `xmlns` keyword, as shown in the following code snippet:

```
xmlns:prefix="URI"
```

  - In the preceding code snippet, URI is the namespace name, and `prefix` is the alias of the namespace.

Namespaces (Contd.)

- In XML, you can declare namespaces by using the following two methods.



Declaring a Namespace

Default

Explicit

Namespaces (Contd.)

- Default Declaration:
  - Allows programmers to declare a default namespace for a document.
  - Does not require a prefix name.
  - For Example:

```
<schema xmlns="http://myxmlschema.com/XMLSchema">
                  ...
                  ...
</schema>
```

Namespaces (Contd.)

- Explicit Declaration:
  - Requires the use of a prefix with the `xmlns` keyword.
  - Is used to define an XML schema that uses elements and attributes defined in one or more namespaces.
  - For Example:

```
<xsd:schema
xmlns:xsd="http://myxmlschema.com/XMLSchema">
                              ...
                              ...
</schema>
```

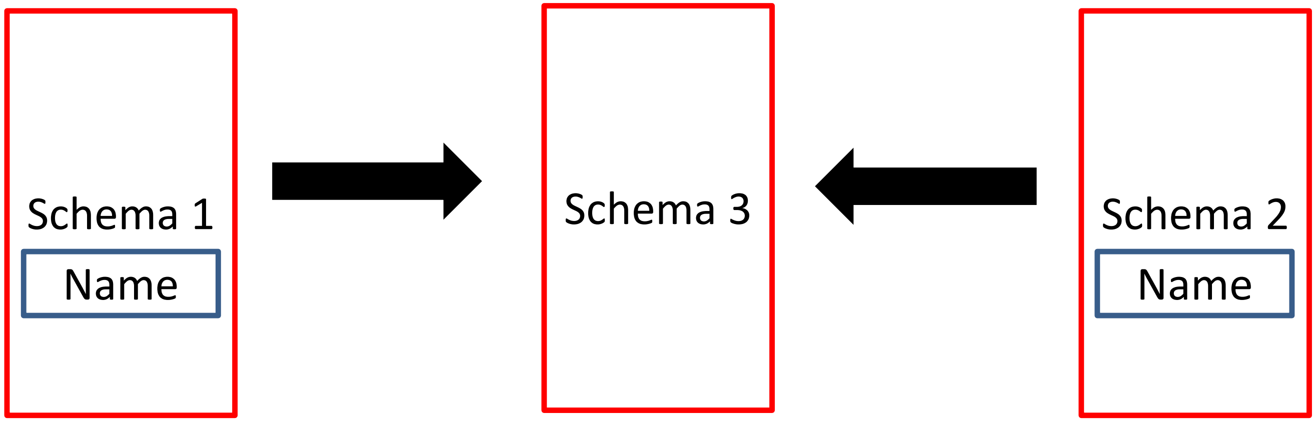
  - In the preceding code snippet, `xsd` is the prefix for the namespace URI.

Why Namespaces?

- In XML, namespaces are used to avoid conflict between the elements having same name.
- To define the structure of an elements XML schema (XSD) files are used.
- An XML document can have the references of multiple schema files.
- In this situation, if an element is defined in two or more schemas with the same name, the conflict of definition may occur.

Why Namespaces (Contd.)?

- For example:
  - **Schema 1** contains the **Name** element that defines the name of a product.
  - **Schema 2** contains the **Name** element that defines the name of a customer.
  - **Schema 1** and **Schema 2** are used in **Schema 3**.
  - Now the conflict between the definition of **Name** element may occur because it is defined in **Schema 1** and **Schema 2** both.



- To avoid such situations, namespaces are used.

What are the Advantages of Using XML Schema Over DTD

- XML schemas follow a universal standard that made data communication over the Internet safe.
- For example, a date like: "01-07-2015" can be interpreted as 7 January in some countries, whereas in other countries as 1 July. However, an XML element accepts date in the fixed format "YYYY-MM-DD" to ensure its correct interpretation.
- XML schemas support data types that allow programmers to:
  - Specify the acceptable content in the document
  - Ensure the validity of data
  - Work with databases
  - Apply restrictions on data
  - Specify data formats
  - Convert data from one data type to another

What are the Advantages of Using XML Schema Over (Contd.)

- You can define number and order of child elements using XML schemas, but not with DTDs.
- XML schemas support namespaces, whereas DTDs do not support namespaces.
- XML schemas can be also extended, therefore, you can:
  - Reuse an XML schema in other schemas.
  - Drive your own data types from the standard types.
  - Add the reference of multiple schemas in a single XML document.

Summary

- In this session, you have learned that:
  - jQuery library provides you with various methods, known as jQuery AJAX methods, that allow you to make a call to the AJAX code.
  - The `load()` method is used to load or fetch data from a Web server into a selected HTML element.
  - The `get()` method is used to load data from a Web server using the HTTP GET request.
  - The `post()` method is used to load data from a Web server using the HTTP POST request.
  - The `ajax()` jQuery method can be used to call the AJAX requests and helps in partial-page updates.
  - JSON is an open standard light-weight format that is used to store and exchange data.
  - Syntactically, JSON is similar to the code for creating JavaScript objects.
  - To read data from a JSON object, you can use the `JSON.parse()` method provided by JavaScript.

Summary (Contd.)

  - To read data from a JSON object, you can use the `JSON.parse()` method provided by JavaScript.
  - JavaScript provides you the `JSON.stringify()` method that allows you to convert JavaScript value to a JSON string.
- XML stands for Extensive Markup Language.
- XML is a self-describing language.
- XML is recommended by W3C for information exchange over the Internet.
- XML transfers data between various heterogeneous systems over the network.
- DTD can be declared as:
  - Internal
  - External
- PCDATA refers to the parsed character data.

Summary (Contd.)

- A namespace can be considered as a virtual space that is identified by a Uniform Resource Identifier (URI).
- XML schemas follow a universal standard that made data communication over the Internet safe.
- XML schemas support namespaces, whereas DTDs do not support namespaces.