XML

A Sample XML Code

XML Declaration: It is a processing instruction

Root Element

Attribute

Empty Element

```
<?xml version="1.0"?>
<BOOK>
    <TITLE>King of the Murgos</TITLE>
    <AUTHOR>Eddings, David</AUTHOR>
    <PUBLISHER>Del Ray</PUBLISHER>
    <COVER TYPE="PAPERBACK" />
    <CATEGORY CLASS="FANTASY" />
    <ISBN>0-345-41920-0</ISBN>
    <RATING NUMBER="5" />
    <COMMENTS>Book 2 of the Malloreon. </COMMENTS>
</BOOK>
```

- **An XML document with correct syntax is "Well Formed".**
- **The syntax rules:**
  - must begin with the XML declaration
  - XML documents must have a root element
  - XML elements must have a closing tag
  - XML tags are case sensitive
  - If an element is empty, it still must be closed.
  - XML elements must be properly nested
  - XML attribute values must be quoted

```
<?xml version="1.0"?>
<Employee>
  <ECode>1111</ECode>
  <Ename>
     <Fname>Neeta</Fname>
     <Lname>Singh</Lname>
  </Ename>
  <Desig desigId="4"/>
  <Salary> 21000 </Salary>
</Employee>
```

Examples

```
<?xml version="1.0"?>
<note time="12:03:46">
  <to>Tove</to>
  <from>John</from>
  <heading>Reminder</heading>
  <body>Meeting this weekend!</body>
</note>
```

```
<?xml version="1.0" ?>
<BankAccount acctId="1234">
    <Name>Darshan Singh</Name>
    <Type>Checking</Type>
    <OpenDate>11/04/1974</OpenDate>
    <Balance>25382.20</Balance>
</BankAccount>
```

```
<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>
```

```
<?xml version="1.0" ?>
<Employees>
   <Employee>
      <empid>1001</empid>
      <EmpName>Vipul</EmpName>
      <Desig>Software Analyst</Desig>
   </Employee>
 <Employee>
      <Empid>1002</Empid>
      <EmpName>Vivek</EmpName>
      <Desig>Software Analyst</Desig>
   </Employee>
</Employees>
```

Valid XML

- **Well-formed vs. Valid : "well formed" XML document is not the same as a "valid" XML document.**
- **A "valid" XML document must be well formed. In addition, it must conform to a document type definition.**
- **There are 2 different document type definitions that can be used with XML:**
  - DTD - The original Document Type Definition
  - XML Schema - An XML-based alternative to DTD
- **Validation of XML is always done against DTD / Schema by Parser**
- **Parser is a program that parses XML document & occasionally modifies it.**
- **XML Parsers can be classified as**
  - Non Validating Parsers - Only checks for structure problems in the code
    - sufficient when there is no DTD or schema linked to the XML code; Most browsers
  - Validating Parsers - Checks for validation rules specified in DTD or Schema

XML DTD (Document Type Declaration)

- **DTD defines the structure of an XML document.**
  - It defines the structure with a list of legal elements
  - Internal DTD: is wrapped inside the <!DOCTYPE> definition. Syntax:
    - <!DOCTYPE root-element [element declarations]>
  - External DTD Declaration : DTD is declared in an external file;
    - Syntax: <!DOCTYPE root-element SYSTEM “filename”>

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend</body>
</note>
```

```
<?xml version="1.0"?>
<!-- note.xml →
<!DOCTYPE note SYSTEM "note.dtd">
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Lets meet this weekend</body>
</note>
```

```
<!– note.dtd -- >
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

Using DTD : Working with Elements and Attributes

- **Element Declarations: First declaration inside a DTD**
  - syntax: <!ELEMENT name content> ;where “name” is a standard XML name
- **Empty Elements: have no content &are marked up as either :**
    - <empty_element/>
    - <empty_element></empty_element>
  - Eg : <!ELEMENT empty_element EMPTY>
- **Elements with Parsed Character Data : Elements with only parsed character data are declared with #PCDATA inside parentheses:**
- **<!ELEMENT element-name (#PCDATA)>**
- **Unrestricted Elements: Opposite of an empty element**
    - An unrestricted element can contain any element that is declared elsewhere in the XML document’s DTD.
  - An unrestricted element’s content is declared as follows:
    - <!ELEMENT any_elementANY>

Working with Elements and Attributes

- **Element Sequences:**
  - It is a simplest form of element content model - a list of the possible elements, enclosed in parentheses and separated by commas.
  - Example:
    - <!ELEMENT counting (first, second, third, fourth)>

```
<counting>
  <first>one</first>
  <second>Two</second>
  ….
</ counting>
```

- **Element Choices:**
  - A choice of elements in an element content model is indicated by a vertical line (|) between the alternatives, as shown below:
    - <!ELEMENT choose (this_one | that_one)>
  - Example:

```
<choose>
  <this_one> choose this one</this_one>
</choose>
and then
<choose>
   <that_one>chose that one</that_one>
</choose>
```

Working with Elements and Attributes

- **Combined Sequences and Choices:**
  - Content sequence & choices can be combined by grouping the element content into model groups. For example:

```
!ELEMENT lots_of_choice (may_be | could_be), (this_one, that_one)>
```

    - The “lots_of_choice” element can consist of either a may_be element or a could_be element; followed by this_one element & then that_one element.

- **Element Occurrence Indicators: specify how many times elements can appear**
  - The **? character** indicates that the element or group of elements may be omitted or may occur just once.
  - The *** character** indicates that an element or group of elements may be omitted or may appear zero or more number of times.
  - The **+ character** indicates that an element or group of elements must appear at least once and may appear one or more number of times.

Working with Elements and Attributes

- **Character Content**
  - # PCDATA (Parsed Character data) in the content model
  - Text is allowed in the element
  - Eg declarations:
    - <!ELEMENT para (title, text)>
    - <!ELEMENT title (#PCDATA)>
    - <!ELEMENT text (#PCDATA)>
  - XML document could look like :

```
<para>
    <title>My Life</title>
    <text>My life is full of joy</text>
</para>
```

- **Mixed Content Elements:**
  - Elements that can contain text, elements, or both are called “mixed content models”:
    - <!ELEMENT pick (#PCDATA | aaa | bbb | ccc | ddd)*>

Working with Elements and Attributes

- **Attribute Declaration: attributes are declared with an ATTLIST declaration**
  - You can declare one element at a time.
  - Elements can have lots of attributes.
  - Attributes are all declared at once in an attribute declaration list.
  - An attribute declaration list has the following form:

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

  - The attribute-type can be one of the following:

Type	Description
CDATA	The value is character data
(en1|en2|..)	The value must be one from an enumerated list
ID	The value is a unique id
IDREF	The value is the id of another element
NMTOKEN	The value is a valid XML name
NMTOKENS	The value is a list of valid XML names
ENTITY	The value is an entity
ENTITIES	The value is a list of entities

Working with Elements and Attributes .:Egs

- **The attribute-value can be one of the following:**

Value	Explanation
value	The default value of the attribute
#REQUIRED	The attribute is required
#IMPLIED	The attribute is optional
#FIXED value	The attribute value is fixed

```
DTD: <!ATTLIST person number CDATA #REQUIRED>
Valid XML: <person number="5677" />
Invalid XML: <person />
```

```
DTD: <!ATTLIST contact fax CDATA #IMPLIED>
Valid XML: <contact fax="555-667788" />
Valid XML: <contact />
```

```
DTD: <!ATTLIST sender company CDATA #FIXED “Abc">
Valid XML: <sender company=“Abc" />
Invalid XML: <sender company=“Xyz" />
```

Working with Elements and Attributes

- **Default Attribute Value**
  - DTD:
  - Valid XML:
    <square width="100" /> If no width specified, has a default value of 0

```
<!ELEMENT square EMPTY>
<!ATTLIST square width CDATA "0">
```

- **Enumerated Attribute Types:**
  - They have values that are simply lists of possible values.
  - Each value has to be a valid name token (NMTOKEN).

```
Enumerated attrs syntax:
<!ATTLIST element-name attribute-name (en1|en2|..) default-value>
DTD: <!ATTLIST payment type (check|cash) "cash">
XML example: <payment type="check" />
Or <payment type="cash" />
Invalid XML : <payment type="EFT" />
```

  - Eg:

```
<!ATTLIST paint color (RED | YELLOW | GREEN ) "RED">
```

Value given in quotes is a default value for this attribute.

Working with Elements and Attributes

- **IDREF:**
  - This attribute is a pointer to an ID (an ID reference).
  - Its value must match the value of an ID type attribute that is declared somewhere in the same document.
  - Usage:

```
<!ATTLIST emp deptno IDREF>
```

```
<emp deptno="D10">
```

- **IDREFS:**
  - The value of this attribute consists of one or more IDREF type value, separated by spaces.
  - IDREFS type declaration :

```
<!ATTLIST seminar departments  IDREFS>
```

  - Usage:

```
<seminar departments=" D10 D20 D30">
```

Working with Entities

- **Entities are used to define shortcuts to special characters.**
  - Entities can be declared internal or external.
  - Internal Entity: Syntax : <!ENTITY entity-name "entity-value">
  - External Entity : Syntax : <!ENTITY entity-name SYSTEM "URI/URL">

```
<!ENTITY writer SYSTEM "http://www.mytutorials.com/entities.dtd">
<!ENTITY copyright SYSTEM "http://www.mytutorials.com/entities.dtd">
XML example: <author>&writer;&copyright;</author>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note [
<!ENTITY nbsp " ">
<!ENTITY writer "Writer: mywriter">
<!ENTITY copyright "Copyright: MyTutorials.">
]>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
<footer>&writer; &copyright;</footer>
</note>
```

DTD Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE LIBRARY [
<!ELEMENT LIBRARY (BOOK)*>
<!ELEMENT BOOK (title,author,publisher,cover,category,isbn,rating)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT cover EMPTY>
<!ATTLIST cover type CDATA #REQUIRED>
<!ELEMENT category EMPTY>
<!ATTLIST category class (Fiction|Fantasy|Scifi|Mystery|Horror) "Fiction">
<!ELEMENT isbn (#PCDATA)>
<!ELEMENT rating EMPTY>
<!ATTLIST rating number (1|2|3|4|5) "3"> ]>
<LIBRARY>
  <BOOK>
    <title>King of Murgos</title>
    <author>Eddings, David</author>
    <publisher>Del Ray</publisher>
    <cover type="Paperback"/>
    <category class="Fantasy"/>
    <isbn>0-345-41920-0</isbn>
    <rating number="4"/>
</BOOK>
</LIBRARY>
```

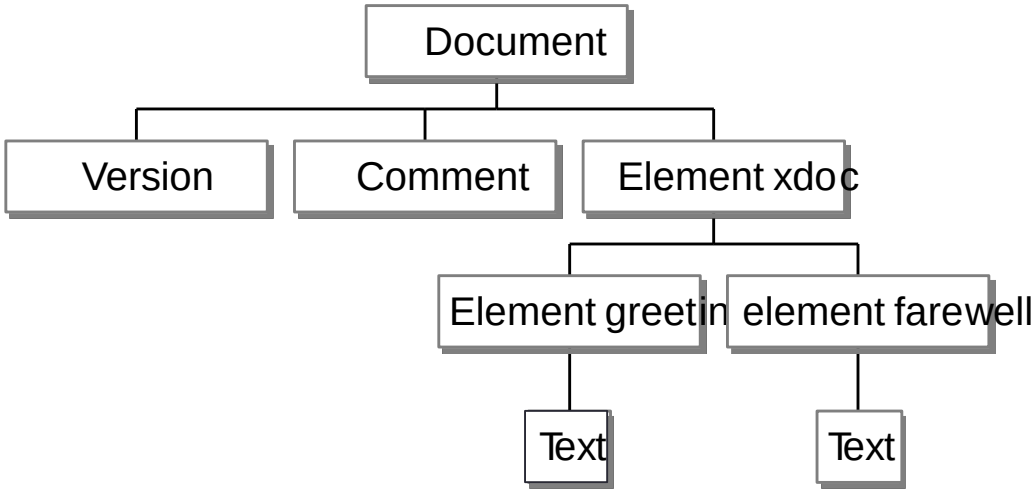
XML Parser

- **Simple API for XML (SAX)**
  - Also called as an event based parser
  - Reads every element from the XML document, so whenever it encounters an XML element or an error it generates and event.
  - SAX is a memory efficient, fast & often used in high performance applications
  - SAX works in serial access mode to parse XML document. .

- **Document Object Model (DOM)**
  - Builds entire XML document structure in memory
  - Standard way to access and manipulate XML documents using programming languages
  - DOM presents the XML document as a tree structure – with the elements, attributes and text defined as nodes. You can access the information in the XML documents in a hierarchical manner.
  - Can be memory and CPU intensive so it is useful when the document is small.

XMLDOM

- **XML DOM defines a standard for accessing & manipulating XML documents**
  - Ie, its a standard for how to get, change, add, or delete XML elements
  - The DOM presents an XML document as a tree-structure
  - In DOM, everything in an XML document is a node.
    - The entire document is a document node
    - Every XML element is an element node
    - The text in the XML elements are text nodes
    - Every attribute is an attribute node
    - Comments are comment nodes




```
<?xml version="1.0"?>
<xdoc>
   <greeting>Hello XML</greeting>
  <farewell>Goodbye HTML</farewell>
</xdoc>
```

The Node Object

- **Represents a single node in the document tree.**
  - A node can be an element node, an attribute node, a text node
- **Node Object Properties:**

Property	Description
attributes	A NamedNodeMap containing the attributes of this node (if it is an Element)
childNodes	Returns a NodeList of child nodes for a node
firstChild	Returns the first child of a node
lastChild	Returns the last child of a node
nextSibling	Returns the node immediately following a node
nodeName	Returns the name of a node, depending on its type
nodeType	Returns the type of a node
nodeValue	Sets/returns the value of a node, depending on its type
parentNode	Returns the parent node of a node
previousSibling	Returns the node immediately before a node
textContent	Sets/returns the textual content of a node and its descendants

The Node Object

Node Type	nodeName returns	nodeValue returns
Document	#document	null
Element	element name	null
Attr	attribute name	attribute value
Comment	#comment	comment text
Text	#text	content of node
Entity	entity name	null

- **Some Node Object Methods:**

Method	Description
appendChild()	Appends a new child node to the end of the list of children of a node
hasAttributes()	Returns true if the specified node has any attributes, else false
hasChildNodes()	Returns true if the specified node has any child nodes, else false
insertBefore()	Inserts a new child node before an existing child node
removeChild()	Removes a specified child node from the current node
replaceChild()	Replaces a child node with a new node

NodeList Object

- **The NodeList object represents an ordered list of nodes.**
  - The nodes in the node list can be accessed through their index number (starting from 0).
  - The node list keeps itself up-to-date. If an element is deleted or added, in the node list or the XML document, the list is automatically updated.
  - Note: In a node list, the nodes are returned in the order in which they are specified in the XML document.
- **NodeList Object Property:**
  - length : Returns the number of nodes in a node list
- **NodeList Object Method**
  - item() : Returns the node at the specified index in a node list

Implementing DOM

- **For implementing DOM, you need an application that supports DOM.**
- **Some of the DOM engines are as follows:**
  - Microsoft DOM Engine : available in the latest MSXML.dll & as an ActiveX object
  - IBM DOM Engines

- **DOM using Javascript:**
- **Create an instance of the parser object and DOM engine:**
  - var xmlDoc;
  - xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
- **Load an XML file with the following syntax:**
  - xmlDoc.load("product.xml");
- **Load an XML string with the following syntax**
  - xmlDoc.loadxml(string variable of xml file)

XML file will be parsed as it is loaded. If any errors are found, loading will be aborted

Product.xml

```
<products >
<product >
    <prodid>1000</prodid>
    <pname>Lays</pname>
    <category>chips</category>
    <price>30.00</price>
    <qty>45</qty>
 </product>
 <product >
    <prodid>1001</prodid>
    <pname>Pepsi</pname>
    <category>cold drink</category>
    <price>50.00</price>
    <qty>100</qty>
 </product>
</products>
```

Example

output

```
<script>
var xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.load("product.xml");
var nm=document.getElementById("name");
nm.innerHTML=xmlDoc.getElementsByTagName("pname")[0].childNodes[0].nod
eValue;
var pr=document.getElementById("price");
pr.innerHTML=xmlDoc.getElementsByTagName("price")[0].childNodes[0].nodeV
alue;
var pr=document.getElementById("qty");
qty.innerHTML=xmlDoc.getElementsByTagName("qty")[0].childNodes[0].nodeVa
lue;
</script>
```

To display list

```
<script>
var lst=document.getElementById("showlst");
var pnames=xmlDoc.getElementsByTagName("pname");
var str="<ol>";
alert(pnames.length);
for(var i=0;i<pnames.length;i++){
alert(str+"   "+pnames[i].childNodes[0].nodeValue);
str=str+"<li>"+pnames[i].childNodes[0].nodeValue+"</li>";

}
str=str+"</ol>";
alert(str);
lst.innerHTML=str;
</script>
```

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<note time="12:03:46">
   <to>Tove</to>
   <from>John</from>
   <heading>Reminder</heading>
   <body>Meeting this weekend.</body>
</note>
```

to: Tove
from: John
heading: Reminder
body: Meeting this weekend.

```
<script language="javascript" for="window" event="onload">
  var xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
  xmlDoc.load("note.xml");
  var itemlist=xmlDoc.getElementsByTagName("note");
    for(var i=0;i<itemlist.length;i++) {
         var nodes=itemlist(i).childNodes;
          for(var j=0;j<nodes.length;j++) {
             document.write("<font color='red' size='15'><b>“ +
                                  nodes.item(j).nodeName  +
                                 ":</b></font><font color='blue' size='15'>“ +
                                  nodes.item(j).text+"<br>")
           }
       }
</script>
```