IT2605
Applications of Web Services

L01 XML Fundamentals

Nanyang Polytechnic

Objectives

- XML and its benefit
- XML vs HTML
- XML elements and attributes
- Well-formed XML document
- Using XML

Introducing XML

- XML stands for Extensible Markup Language. A markup language specifies the structure and content of a document.
- XML focus on how to represent data.
- XML consists of user defined tag
- Sample XML

```
<?xml version="1.0"?>
<customers>
    <customer ID="C001>
        <name>NYP</name>
        <phone>65501234</phone>
    </customer>
<customers>
```

Benefits of XML

- XML is an industry standard
  - XML is W3C recommendation . It gains huge acceptance
- XML is self describing
  - The markup tags are more readable unlike csv file
- XML is extensible
  - It allows you to add your own markup tags
- XML is used to exchange data between 2 incompatible system
  - XML can be used as a means for sending data for distributed computing. Using XML and SOAP reduces the complexity of exchanging data between incompatible system over the internet
- XML can be use to create specialized vocabularies
  - Using XML as a base to create other vocabularies such as WSDL, SOAP, RSS etc

Self Describing

```
09111A,2009,S2,IT9999,68,54,90
09112B,2009,S2,IT9999,97,83.4,90
09333C,2009,S2,IT9999,89,89.4,93.3
09444D,2009,S2,IT9999,60,63.1,90
09555E,2009,S2,IT9999,87,75.7,90
09666E,2009,S2,IT9999,63.5,70.3,93.3
09777E,2009,S2,IT9999,75,79.1,86.7
09888E,2009,S2,IT9999,61,66,96.7
09228B,2009,S2,IT9999,60.5,66,93.3
09229E,2009,S2,IT9999,80,88,96.7
09334X,2009,S2,IT9999,80,69.1,93.3
09335E,2009,S2,IT9999,77,76.3,86.7
09456A,2009,S2,IT9999,77,71.1,86.7
09678E,2009,S2,IT9999,77,86.6,90
09688E,2009,S2,IT9999,84,78.6,92
```

Can you understand this comma separated value (csv) file?

Self Describing

Instead of using csv file,
We model it in XML document

√ Readable
√ Self explained
√ Don't need to refer to other document to understand

```xml
<?xml version="1.0" encoding="utf-8" ?>
<score>
  <student admno="09111A">
    <yr>2009</yr>
    <sem>S2</sem>
    <test1>68</test1>
    <test2>54</test2>
    <test3>90</test3>
  </student>
  <student admno="09112B">
    <yr>2009</yr>
    <sem>S2</sem>
    <module>IT9999</module>
    <test1>97</test1>
    <test2>83.4</test2>
    <test3>90</test3>
  </student>
  <student admno="09333C">
    <yr>2009</yr>
    <sem>S2</sem>
    <module>IT9999</module>
    <test1>89</test1>
    <test2>89.4</test2>
    <test3>93.3</test3>
  </student>

</score>
```

Comparing XML and HTML

HTML CODE

```
<H2>Kind of Blue</H2>
<H3>Miles Davis</H3>
<OL>Tracks
  <LI>So What (9:22)</LI>
  <LI>Freddie Freeloader (9:46)</LI>
  <LI>Blue in Green (5:37)</LI>
  <LI>All Blues (11:33)</LI>
  <LI>Flamenco Sketches (9:26)</LI>
</OL>
```

HTML code describes the format of the data, but not the data content

XML CODE

```
<CDTITLE>Kind of Blue</CDTITLE>
<ARTIST>Miles Davis</ARTIST>
<CONTENTS>
  <TRACK>So What (9:22)</TRACK>
  <TRACK>Freddie Freeloader (9:46)</TRACK>
  <TRACK>Blue in Green (5:37)</TRACK>
  <TRACK>All Blues (11:33) </TRACK>
  <TRACK>Flamenco Sketches (9:26) </TRACK>
</CONTENTS>
```

XML code describes the data content, but not the data format

Standard tag

Self defined

XML Parsers

- An XML processor (also called XML parser) evaluates the document to make sure it conforms to all XML specifications for structure and syntax.
- All modern browsers have a built-in XML parser.

Parsing XML Documents

- A non-validating parser only parses the document and makes a check for well-formedness.
- A validating parser parses the document and checks its structure against the rules of a DTD or XML schema. If the document fails to meet the DTD or schema rules, the parser will return an error.
- **DTD = Document Type Definition**
- Validating and non-validating parsers are used to transform an XML document into its tree structure, so that applications can access the document's contents.

Well-Formed and Valid XML Documents

- There are two categories of XML documents
  - **Well-formed** : contains no syntax errors and satisfies the specifications for XML code as laid out by the W3C
  - **Valid**: a well-formed document that also satisfies the rules laid out in the DTD or XML schema attached to the document

Well-Formed XML

Well-Formed XML Document

- A well-formed XML document is a document that conforms to the XML syntax rules, like:
  1. it must begin with the XML declaration
  2. it must have one unique root element
  3. start-tags must have matching end-tags
  4. Element name are case sensitive
  5. all elements must be closed
  6. all elements must be properly nested
  7. all attribute values must be quoted
  8. entities must be used for special characters

*refer to slide 21 for XML Syntax rules for element name
*refer to slide 26 for XML Syntax rules for attributes

The Structure of an XML Document

- XML documents consist of three parts
  - The prolog
    - Provides information about the document itself
  - The document body
    - Contains the document's content in a hierarchical tree structure
  - The epilog
    - Optional, contains any final comments or processing instructions

XML Document Example:

Prolog

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- This document contains data on Jazz Warehouse special offers -->
```

Document Elements

```
<SPECIALS>
<TITLE>Monthly Specials at the Jazz Warehouse</TITLE>
   <CD>Kind of Blue
      <ARTIST>Miles Davis</ARTIST>
      <TRACK length="9:22">So What</TRACK>
      <TRACK length="9:46">Freddie Freeloader</TRACK>
      <TRACK length="5:37">Blue in Green</TRACK>
      <TRACK length="11:33">All Blues</TRACK>
      <TRACK length="9:26">Flamenco Sketches</TRACK>
   </CD>
   <CD>Cookin'
      <ARTIST>Miles Davis</ARTIST>
      <TRACK length="5:57">My Funny Valentine</TRACK>
      <TRACK length="9:53">Blues by Five</TRACK>
      <TRACK length="4:22">Airegin</TRACK>
      <TRACK length="13:03">Tune-Up</TRACK>
   </CD>
   <CD>Blue Train
      <ARTIST>John Coltrane</ARTIST>
      <TRACK length="10:39">Blue Train</TRACK>
      <TRACK length="9:06">Moment's Notice</TRACK>
      <TRACK length="7:11">Locomotion</TRACK>
      <TRACK length="7:55">I'm Old Fashioned</TRACK>
      <TRACK length="7:03">Lazy Bird</TRACK>
   </CD>
</SPECIALS>
```

The Prolog

- The prolog consists of 4 parts in the following order:
  - XML declaration
  - Miscellaneous statements or comments optional
  - Processing instructions optional *Instruction to process the xml document*
  - Document type declaration (DTD) optional Confirm to the validation rule
- This order has to be followed or the parser will generate an error message.
- Only the XML declaration is required, the others are optional, but it is good form to include them.

The XML Declaration

- The XML declaration is always the first line of code in an XML document. It tells the processor what follows is written using XML. It can also provide any information about how the parser should interpret the code.

- The complete syntax is:

  <?xml version="*version number*" encoding="*encoding type*" standalone="*yes | no*" ?>

  * encoding and standalone attributes are optional

- A sample declaration might look like this:

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

Unicode Transformation Format

The XML Declaration

<?xml version="1.0" ?>

<?XML VERSION="1.0" ENCODING="UTF-8"
STANDALONE="YES" ?>

<?xml version= 1.0 encoding= UTF-8
standalone= yes ?>

Comments

- Comments or miscellaneous statements go after the declaration. Comments may appear anywhere after the declaration.
- The syntax for comments is:

  <!- - *comment text* - ->

Elements

- Elements are the basic building blocks of XML files.

- XML supports two types of elements:
  - closed
  - empty (also called open)

Elements

- A closed element, has the following syntax:

  <element_name>PCDATA </element_name>

- PCDATA contains characters describes the content of element.
- Example:

  <Artist>Miles Davis</Artist>

  <Price>100</Price>

PCDATA aka **p**arse as **c**haracter **data**

Document Body

Elements

- Element names are:
  - Case sensitive
  - Must begin with a letter or the underscore character (_)
  - May not contain blank spaces
  - Cannot begin with the letters “xml”
  - Must match in both the opening and closing tags
- Elements can be nested, as follows:

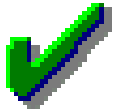
```
<CD>Kind of Blue
  <TRACK>So What (9:22)</TRACK>
  <TRACK>Blue in Green (5:37)</TRACK>
</CD>
```

Elements

- Nested elements are called **child elements**
- Elements must be nested correctly. Child elements must be enclosed within their parent elements.

<CD>Kind of Blue<ARTIST>Miles Davies</CD> </ARTIST>

<CD>Kind of Blue<ARTIST>Miles Davies</ARTIST></CD>



Elements

- All elements must be nested within a single **root element**. There can be only **one** root element.

```
<?xm1 version="1.0" ?>
<CD> Kind of Blue
        <TRACK> So What (9:22)</TRACK>
        <TRACK> Blue in Green (5:37)</TRACK>
        <TRACK> All Blues (11:33)</TRACK>
</CD>
<CD> Cookin'
        <TRACK> My Funny Valentine(5:57)</TRACK>
        <TRACK> Blues by Five (9:53)</TRACK>
        <TRACK> Tune-Up (13:03)</TRACK>
</CD>
```

```
<?xm1 version="1.0" ?>
<ARTIST>
   <CD> Kind of Blue
        <TRACK> So What (9:22)</TRACK>
        <TRACK> Blue in Green (5:37)</TRACK>
        <TRACK> All Blues (11:33)</TRACK>
   </CD>
   <CD> Cookin'
        <TRACK> My Funny Valentine(5:57)</TRACK>
        <TRACK> Blues by Five (9:53)</TRACK>
        <TRACK> Tune-Up (13:03)</TRACK>
   </CD>
</ARTIST>
```

Document Body

Elements

- An **open** or **empty** element is an element that contains no content. They can be used to mark sections of the document for the XML parser.
- It has the following syntax:
  - <element_name></element_name>
  - <element_name/> **Self-Closing Tag**

Attributes

- An **attribute** is a feature or characteristic of an element. Attributes are text strings and must be placed in single or double quotes.
- Syntax
  - *<element_name attribute="value"> … </element_name>*
  - *<element_name attribute="value"/>*
- Example
  - <TRACK length="9:22">So What</TRACK>

Attributes

- Attribute name constraints:
  - Must begin with a letter or underscore (_)
  - No spaces allowed
  - Should not begin with the text string “xml”
  - Can only appear once in the SAME starting tag
- Example
  - <student ID=034321A>Jimmy Tan</student>
  - <student adm no=“031234C”>Rose Lee</student>
  - <student id=“0123456D” name=“Ah Lien” id =“S8733221A” />

Character References

- Some reserved characters cannot be included in PCDATA because they are used in XML syntax such as **<** and **&** characters
- These special characters can be inserted into your XML document by using a **character reference**.
- SYNTAX
  - &#reference OR
  - &#characterunicode;

Start with &# and end with a semicolon

Character References

- Example
  - *<Title> Made <IT> Difference</Title>* ✗
  - *<Title> Made* ***<****IT****>*** *Difference</Title>* ✓
  - *<Title> Made* ***&#60****;IT****&#62****; Difference</Title>* ✓

Commonly used Character References

SYMBOL	CHARACTER REFERENCE	CHARACTER NAME	DESCRIPTION
<	<	<	Less than symbol
>	>	>	Greater than symbol
&	&	&	Ampersand
“	&#22	"	
‘	&#27;	'	Apostrophe
£	£	£	Pound sign
€	€	€	Euro sign
¥	$#165;	¥	Yen sign

CDATA Sections

- The document becomes unreadable if there is lot of character references. In this case CDATA become useful.
- A CDATA section is a large block of text the XML processor will interpret only as text.
- Syntax:
  - <element1>
    <! [CDATA [*Text Block ….*]]>
    </element1>

CDATA Sections

- Constraints:
  - May contain most markup characters (e.g. <, > and &) and these will be interpreted by the XML parser as text and not markup commands
  - May be placed anywhere that text occurs in the document, (e.g. between opening and closing element tags)
  - Cannot be nested within one another
  - Cannot be empty
  - Cannot have string sequence “]]>”

CDATA Sections

- In this example, a CDATA section stores several HTML tags within an element named HTMLCODE:
- Example

```
<HTMLCODE>
    <![CDATA[
      <h1>The Jazz Warehouse</h1>
      <h2>Your Online Store for Jazz Music</h2>
    ] ]>
</HTMLCODE>
```

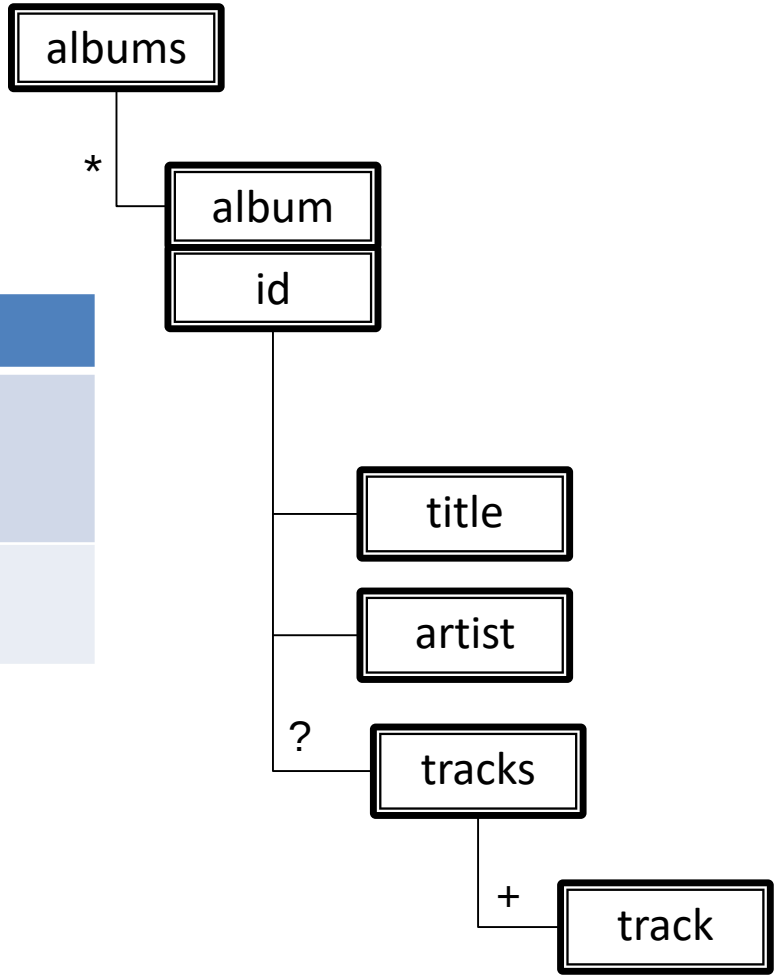
XML Document Tree

- The elements in an XML document form a document tree. The tree starts at the root and branches to the lowest level of the tree.
- All elements can have child elements
- Parent, child, and sibling are used to describe the relationships between elements.
- Parent elements have children.
- Children on the same level are called siblings
- All elements can have text content and attributes

Element Hierarchy: The Document Tree

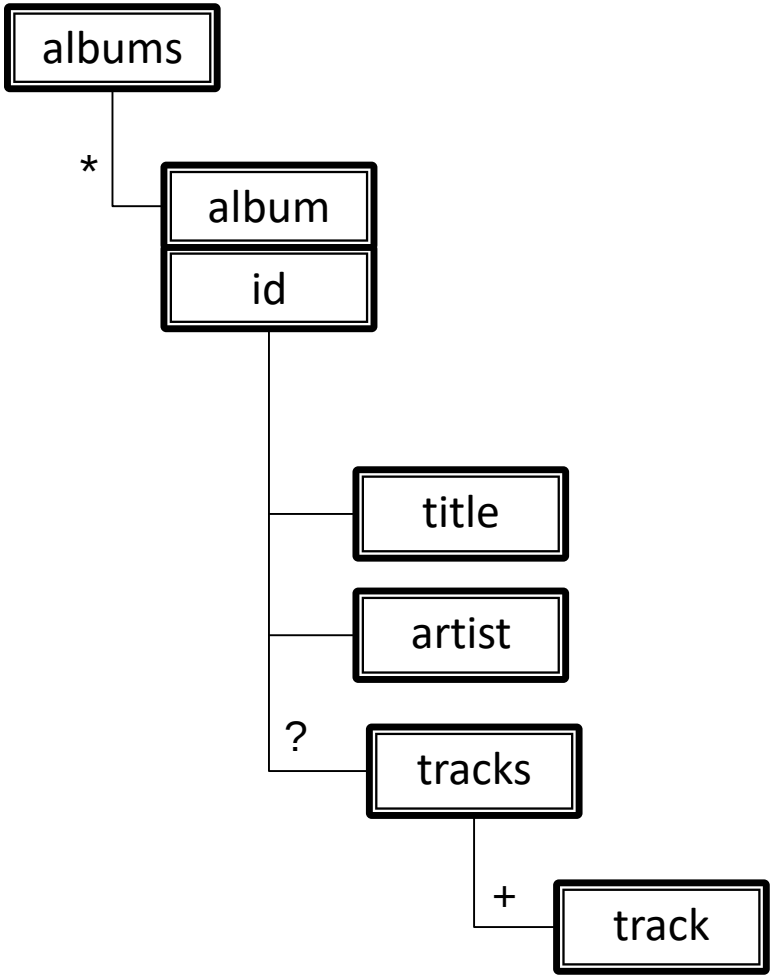
ID is the attribute to identify each album

ID	Album Title	Artist	Tracks
1	Blue Haze	Miles Davis	Four, Tune Up, Miles Ahead
2	The Great Summit	Louis Armstrong	Cottontail, Solitude



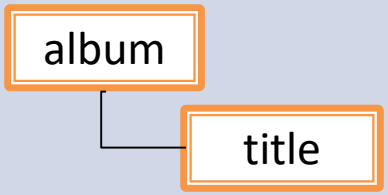
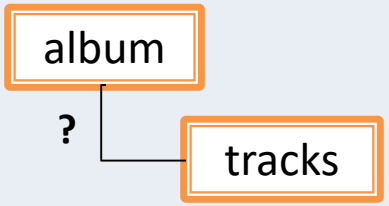
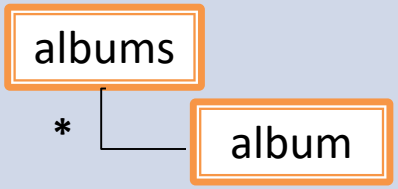
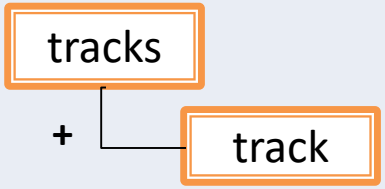
Element Hierarchy: The Document Tree

```xml
<?xml version="1.0" encoding="utf-8"?>
<albums>
  <album id ="1">
    <title>Blue Haze</title>
    <artist>Miles Davis</artist>
    <tracks>
      <track>Four</track>
      <track>Tune Up</track>
      <track>Miles Ahead</track>
    </tracks>
  </album>
  <album id ="2">
    <title>The Great Summit</title>
    <artist>Louis Armstrong</artist>
    <tracks>
      <track>Cottontail</track>
      <track>Solitude</track>
    </tracks>
  </album>
</albums>
```



Refer to the chart in next slide for cardinality

Charting the Element Hierarchy

Symbol	Description	Chart	Interpretation
[none]	The parent contains a single occurrence of the child element.		An album can only have one title 1..1
?	The child element occurs once or not at all.		An album may or may not have a collection of tracks 0..1
*	The child element occurs any number of times		The albums element can contain zero or more album elements 0..m
+	The child element occurs at least once.		The tracks element must contain at least one music track 1..m

Using XML

Accessing XML data

Common Language to access or transform XML in .NET

- XSLT - EXtensible Stylesheet Language
- XPATH - XML Path Language
- LINQ - Language-Integrated Query
- DOM - Document Object Model

Here we introduce DOM.

XML Document Object Model

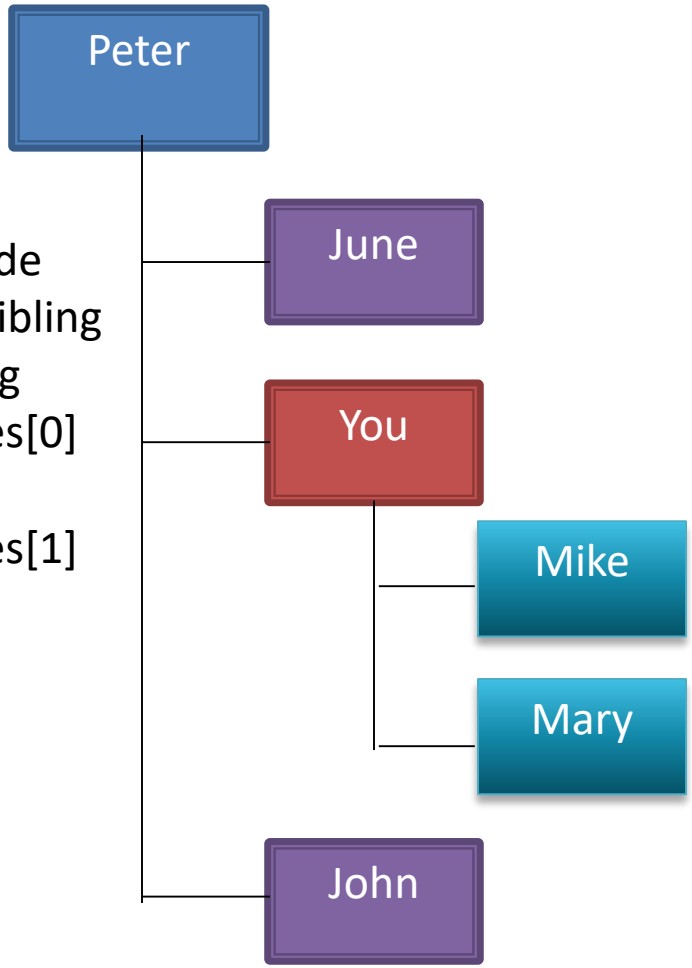
- The Document Object Model (DOM) is an application programming interface (*API*) used to access and manipulate XML.
- DOM provides the quickest way to read XML data as soon as you understand the relationship between the elements in the document.

XML DOM – Tree Traversal (Illustration)

You	doc.getElementByTagName("You")
Peter	doc.getElementByTagName("You").parentNode
June	doc.getElementByTagName("You").previousSibling
John	doc.getElementByTagName("You").nextSibling
Mike	doc.getElementByTagName("You").childNodes[0]
	doc.getElementByTagName("You").firstChild
Mary	doc.getElementByTagName("You").childNodes[1]
	doc.getElementByTagName("You").lastChild

*doc is the variable of an xml document stored in the memory

*childNodes gives an array of nodes



XML DOM – Properties and Methods

- DOM Properties

Note : x is a node object

  - x.Name - the name of x
  - x.Value - the value of x
  - x.InnerText – the value of x and all its child nodes
  - x.Attributes - the attributes nodes of x
  - x.Attributes[“id”].Value – the value of attribute id
- DOM Methods
  - x.getElementsByTagName(*name*) - get all elements with a specified tag name
  - x.appendChild(*node*) - insert a child node to x
  - x.removeChild(*node*) - remove a child node from x

DOM Class in .NET

Class	Description
XmlDocument	Represents the entire document (the root-node of the DOM tree
XmlNode	Represent a single XML document node
XmlElement	Represent an element
XmlAttribute	Represent an attribute
XmlText	Represents text between a starting tag and closing tag
XmlComment	Represents a comment node
XmlNodeList	Represents a collection of nodes

Traversing DOM (Example)

```
Using System.Xml;
::::::
XmlDocument doc = new XmlDocument();

// Load xml doc to the memory
doc.Load("d:\\cd.xml");

// return root element items
XmlNode rootnode = doc.DocumentElement;

// return node list of root which are list of items
XmlNodeList  items = rootnode.ChildNodes;
XmlNode title = items[0].FirstChild;
XmlNode tracks = items[0].LastChild;
```

```
<?xml version="1.0" encoding="utf-8"?>
<items>
 <item id="1" category="sentimental">
   <title>Kind of blue</title>
   <artist>Mile Davis</artist>
   <tracks>
    <track length='5.19'>So what</track>
    <track length= "4:25">Blue in green</track>
   </tracks>
 </item>
</items>
```

DOM document method

Method	Description
createElement	Create an element node
createAttribute	Create an attribute node
createTextNode	Create a text node
createComment	Create a comment
createProcessingInstruction	Create a processing instruction node
createCDATASection	Crete a CDATA section node
getDocumentElement	Return the document's root element
appendChild	Append a child node
getChildNodes	Return the child nodes
createXmlDocument	Parses an XML document
write	Output the XML document

Add a Node – append child

```
XmlDocument doc = new XmlDocument();
doc.Load("d:\\cd.xml");

// add a new node price
    XmlNode nodePrice = doc.CreateElement("price");
    //add value for it
    nodePrice.InnerText = "$30";
    // navigate to  <item> node which is 1st childnode of root element
    XmlNodeList items = doc.DocumentElement.ChildNodes;
```

Root -> items

```
    // Add new element to <item> node
    items[0].AppendChild(nodePrice);
```

First element of root -> item

```
    //save back
    doc.Save("d:\\cd.xml");
```

```
<?xml version="1.0" encoding="utf-8"?>
<items>
  <item id="1" category="sentimental">
    <title>Kind of blue</title>
    <artist>Mile Davis</artist>
    <tracks>
      <track length="5.19">So what</track>
      <track length="4:25">Blue in green</track>
    </tracks>
    <price>$30</price>
  </item>
</items>
```

DOM Node method

Method	Description
appendchild	Appends a child node
getAttributes	Returns the node ‘s attributes
getChildNodes	Returns the node’s child node
getNodeName	Returns a node’s name
getNodeType	Returns the node’s type such as element, attribute, text
getNodeValue	Returns the node’s value
getParentNode	Returns the node’s parent
haschileNodes	Returns true if the node has child nodes
removeChild	Remove a child node from the node
setNodevalue	Sets the node’s value
insertBefore	Appends a child node in front of a child node

DOM Element method

Method	Description
appendchild	Appends a child node
getAttributes	Returns the value of node ‘s attributes
getAttributeNode	Returns attribute node
getElementByTagName	Returns a NodeList of matching element nodes, and their children
haschildNodes	Returns true if the node has child nodes
removeChild	Remove a child node from the node
setAttribute	Set an attribute’s value
removeAttribute	Removes an element’s attribute

Summary

- XML and its used
- Rule of Well-formed XML
  - Syntax correct: XML Declaration, Comment, element, attribute, literals….
- Construct XML Document tree
- Accessing XML using DOM

PRACTICAL TIME!