XML

What and why XML?
XML basics
XML and DB, some technical underpinning
XML, semantic Web, and Web services
Case study – XML applications

Ref: Deitel’s XML – How to program
http://www.w3schools.com/xml/default.asp
Many other sources of XML materials
Motivation

...
<table>
<thead>
<tr>
<th>Date/Time</th>
<th>Date/Time</th>
<th>Duration</th>
<th>Rate</th>
<th>Call ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001-10-03 11:58:13</td>
<td>2001-10-03 11:58:32</td>
<td>19</td>
<td>0</td>
<td>02275****</td>
</tr>
<tr>
<td>2001-10-03 16:25:31</td>
<td>2001-10-03 16:26:21</td>
<td>50</td>
<td>0</td>
<td>07571**** 09299****</td>
</tr>
<tr>
<td>2001-10-03 17:05:02</td>
<td>2001-10-03 17:05:49</td>
<td>47</td>
<td>10</td>
<td>07781**** 09199****</td>
</tr>
<tr>
<td>2001-10-03 17:43:01</td>
<td>2001-10-03 17:43:27</td>
<td>26</td>
<td>0</td>
<td>02233**** 09357****</td>
</tr>
<tr>
<td>2001-10-03 17:43:50</td>
<td>2001-10-03 17:44:13</td>
<td>23</td>
<td>0</td>
<td>02233**** 03957****</td>
</tr>
<tr>
<td>2001-10-03 17:44:59</td>
<td>2001-10-03 17:45:19</td>
<td>20</td>
<td>0</td>
<td>02233**** 03957****</td>
</tr>
</tbody>
</table>

...
<?xml version="1.0" encoding="Big5"?>
<WORKFLOW workflow-id="Leave request" job-id="101">
  <TYPE_NAME>Wf_instance</TYPE_NAME>
  <VERSION>1.0</VERSION>
  <JOB_NAME>Leave: Fin007</JOB_NAME>
  <INITIATOR>Fin007</INITIATOR>
  <STATE state-id="Start">
    <INTERACTOR)NULL</INTERACTOR>
    <INIT_DATE>2001/8/5 AM 10:39:39</INIT_DATE>
    <DUE_DATE>NULL</DUE_DATE>
    <COMPLETE_DATE>2001/8/5 AM 10:39:39</COMPLETE_DATE>
    <STATUS>complete</STATUS>
    <DATA>NULL</DATA>
  </STATE>
  <STATE state-id="st_Petition">
    <INTERACTOR>
      <ROLE state-id="st_Petition" role-id="role_Petitioner" role-type="user">Fin007</ROLE>
    </INTERACTOR>
    <INIT_DATE>2001/8/5 AM 10:39:40</INIT_DATE>
    <DUE_DATE>2001/8/5 PM 1:39:40</DUE_DATE>
    <COMPLETE_DATE>2001/8/5 AM 10:50:08</COMPLETE_DATE>
    <STATUS>complete</STATUS>
    <DATA dataitem-id="data_Leave" datatype="value">
      <INPUT><![CDATA[<Leave><SN></SN><Name></Name><Type></Type><Date></Date><Manager></Manager><HRD></HRD><Anew></Anew></Leave>]]></INPUT>
      <OUTPUT><![CDATA[<Leave><SN>Fin007</SN><Name>John Lee</Name><Type>sick</Type><Date>2001/8/14</Date><Manager></Manager><HRD></HRD><Anew></Anew></Leave>]]></OUTPUT>
    </DATA>
  </STATE>
</WORKFLOW>
XML (eXtensible Markup Language)

- Gaining importance for common data representation
- “Simpler” than SGML (subset of SGML)
- More “general” than HTML
- Example:
  
  ```xml
  <CUSTOMER>
    <NAME> Joe </NAME>
    <ADDRESS> NY </ADDRESS>
  </CUSTOMER>
  ```

- Great deal of activity in Ecommerce (competition to EDI), messaging middleware, data transformers, data management, publishing, etc.
What is XML

- XML stands for E\textsc{x}\textsc{xtensible} \textsc{m}arkup \textsc{l}anguage
- XML is a system for defining, validating, and sharing document formats
- XML is a \textbf{markup language} much like HTML.
- XML was designed to \textbf{describe data}.
- XML uses a DTD (\textbf{Document Type Definition}) to describe the data.
- XML with a DTD is designed to be \textbf{self-descriptive}.
- XML uses \textbf{tags} (for example, \texttt{<age>35</age>}) to distinguish document structures, and
- \textbf{attributes} (for example, in \texttt{<A HREF="http://www.xml.com/"/>}, \texttt{HREF} is the attribute name, and \texttt{http://www.xml.com/} is the attribute value) to encode extra document information
- XML tags are not predefined in XML. You must \textbf{define your own tags}. 
XML

- XML, a universal data format
  - Widely agreed upon standard
  - Markup language for documents or semi-structured data
- Inter-operability and integration; for data exchange between applications
- XML tags, a form of metadata that are user-definable, can be useful semantics for search and other purposes
- Example: the same XML data can be
  - presented to the consumer on the web
  - processed by the consumer’s bank, and
  - transferred to a third party for an electronic funds transfer
XML vs. HTML

- XML and HTML were designed with different goals:
  - XML was designed to describe data and to focus on what data is
  - HTML was designed to display data and to focus on how data looks.
- HTML is about displaying information, XML is about describing information.
- XML is not a replacement for HTML; it is a complement to HTML.
Characteristics and Significance
(from Deitel’s)

- XML is both human-readable and computable
- The separation of content and its presentation is the essential characteristics of XML
- Because an XML document describes data, it can conceivably be processed by any application
- Ideal for data exchange
- Being integrated into many different applications, interoperability is achieved
- Likely to become the universal language for representing data
- An open standard with a wide selection of tools for its implementation
- Business is implementing XML to accomplish the goal of EAI for interoperability both within and between organizations
The resource definition framework (RDF), developed by the W3C, adds to XML’s metadata capabilities.

Many vocabularies are being developed and are in use in major sectors such as the automotive and airline industries.

XML can be used to create new languages such as WML.

Because XML data is easy for computers to read and convert between formats, it can be used as middleware to integrate legacy systems with other applications and networks.

Possible uses of XML are endless.

With XML, information can be described precisely and therefore applications can work with the information more intelligently.
XML Family

XML (Extensible Markup Language)

XML Variants
- WML (wireless)
- VML (voice)
- MathML

XML Other Features
- XML Link
- XML Signature
- XML Path

XML DTD for Validation

XSL for Presentation

XML Query Language

XML Schema for Data Representation
XML Basics

- XML Syntax
- XML Elements
- XML Attributes
- DTD
- Valid & well-formed XML
- Displaying XML with CSS
- Displaying XML with XSL
- XML in Data Islands
XML: What is it composed of

- Elements and sub-elements
  - Attributes
- Entity References
  - Used to represent special characters
- Comments
  - Comments begin with <!-- and end with -->.
- Processing Instructions
  - Escape hatch to provide information to an application
- CDATA sections
  - Instructs the parser to ignore most markup characters
- Document Type Declarations (DTD)
  - Required in SGML, optional in XML
XML Syntax

- All XML elements must have a closing tag
- XML tags are case sensitive
- All XML elements must be properly nested
- All XML documents must have a root tag
- Attribute values must always be quoted
- With XML, White Space is Preserved

```xml
<?xml version="1.0"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```
XML Elements

XML Elements are Extensible

- XML documents can be extended to carry more information.

```xml
<note>
  <to>Tove</to>
  <from>Jani</from>
  <body>Don't forget me this weekend!</body>
</note>
```
XML Elements

- XML Elements have Relationships
  - Elements are related as parents and children

<note>
  <date>1999-08-01</date>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>

Root (Parent)

Child

<to> and <from> Are sister elements
XML Elements

- Elements have Content
  - Elements can have different content types.
  - mixed content, simple content, or empty content. (An element can also have attributes)

- XML elements must follow these naming rules:
  - Names can contain letters, numbers, and other characters
  - Names must not start with a number or other punctuation characters
  - Names must not start with the letters xml (or XML or Xml ..)
  - Names cannot contain spaces
XML Attributes

- XML elements can have attributes in the start tag, just like HTML.
- Attributes are used to provide additional information about elements.
- Data can be stored in child elements or in attributes.
XML Attributes

- Use Attributes or Elements?
- some of the problems using attributes:
  - attributes cannot contain multiple values (child elements can)
  - attributes are not easily expandable (for future changes)
  - attributes cannot describe structures (child elements can)
  - attributes are more difficult to manipulate by program code
  - attribute values are not easy to test against a DTD
DTD (Document type def.)

- DTD’s
  - Allows a document to communicate meta-information to parser about its content
- DTD’s contain:
  - Element Type Declarations
    - Identify names of elements and nature of their content.
    - `<!ELEMENT customer (name, address)>`
    - `<!ELEMENT address (type, street+, city, state, zip)>`
  - Attribute List Declarations
    - Identify which elements may have attributes, what attributes they may have, what values the attributes may hold, and what value is the default
  - Entity Declarations
    - Allow you to associate a name with some other fragment of content
A DTD (document type definition) defines the document structure with a list of legal elements; it is a grammar or set of rules that define what tags can appear in the document and how they must nest within each other.

```xml
<!ELEMENT WORKFLOW (TYPE_NAME, VERSION, DEF_TIME, STATE, INTERACTOR, DATA, ACTIVITY, RULE, TIME)>
<!ELEMENT TYPE_NAME (#PCDATA)>
<!ELEMENT VERSION (#PCDATA)>
<!ELEMENT DEF_TIME (#PCDATA)>
<!ELEMENT STATE (STATE_NAME, STATE_NAME+)>
<!ELEMENT STATE_NAME (#PCDATA)>
<!ATTLIST STATE_NAME state-id ID #REQUIRED time-id ID #REQUIRED>
<!ELEMENT INTERACTOR (ROLE*)>
<!ELEMENT ROLE (#PCDATA)>
<!ATTLIST ROLE role-id ID #REQUIRED>
<!ELEMENT DATA (DATA_ITEM)*>
<!ELEMENT DATA_ITEM (DATA_NAME, DATA_TYPE, DATA_LOCATION)>
<!ATTLIST DATA_ITEM dataitem-id ID #REQUIRED>
...
Well-formed XML document

- XML with correct syntax is well-formed
- All the begin-tags and end-tags match up
- Empty tags use the special XML syntax (e.g. `<empty/>`)
- All the attribute values are nicely quoted (e.g. `<a href="http://www.textuality.com/xml.html">`)
- All the *entities* are declared (entities are re-usable chunks of data, much like macros, part of XML's inheritance from SGML).
- A document that is well-formed is easy for a computer program to read
Validity

- A valid document must have a DTD, and
- It conforms to the rules in the document type declaration.
- Validity is useful because an XML-savvy editor can use the type declaration to help (and in fact require) users to create documents that are valid; such documents are much easier to use and (especially) re-use.

- Errors in XML documents will stop the XML program.
- With HTML it was possible to create documents with lots of errors.
Well-formed versus Valid

- **A Well-Formed Document:**
  - adheres to the syntactic rules defined by the XML standard
    - E.g. Tags are delimited by `<` and `>`

- **A Valid Document:**
  - A well-formed document that also adheres to the rules of a specified Document Type Definition (DTD)

- **DTD:**
  - specifies a set of rules for the structure of the document
A DTD for a Customer

```xml
<!ELEMENT customer (name, address?, phone?)>
<!ATTLIST customer id CDATA #REQUIRED>
<!ELEMENT name (first, middle?, last)>
<!ELEMENT address (street+, city, state, zip)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT first (#PCDATA)>
<!ELEMENT middle (#PCDATA)>
<!ELEMENT last (#PCDATA)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT zip (#PCDATA)>

... We can save this into a file called customer.dtd
Valid XML document

```xml
<?xml version="1.0" standalone="no"?>
<!DOCTYPE customer SYSTEM "customer.dtd">
<customer id="12345">
  <name>
    <first>Joe</first>
    <last>Bubba</last>
  </name>
  <address>
    <street>Building2</street>
    <street>445 Swamp Street</street>
    <city>Joytown</city>
    <state>NJ</state>
    <zip>07555</zip>
  </address>
  <phone>933-555-1111</phone>
</customer>
```
Well-formed but *not* Valid XML document

```xml
<?xml version="1.0" standalone="no"?>
<!DOCTYPE customer SYSTEM "customer.dtd">
<foo1 id="12345">
  <foo2>
    <foo3>Sam</foo3>
    <foo4>Sheppard</foo4>
  </foo2>
  <foo5>
    <foo6>Office</foo6>
    <foo7>445 Street</foo7>
    <foo8>Hilltown</foo8>
    <foo9>MI</foo9>
    <foo10>48190</foo10>
  </foo5>
  <foo11>734-555-2222</foo11>
</foo1>
```
With CSS (Cascading Style Sheets) you can add display information to an XML document.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="cd_catalog.css"?>
<CATALOG>
  <CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
  </CD>
  ...
  ...
  ...
</CATALOG>
```
Displaying XML with XSL

- With XSL you can add display information to your XML document
- One way to use XSL is to transform XML into HTML

```xml
<?xml version="1.0"?>
<?xml:stylesheet type="text/xsl" href="simple.xsl" ?>
<breakfast_menu>
  <food>
    <name>Belgian Waffles</name>
    <price>$5.95</price>
    <description>two of our famous Belgian Waffles</description>
    <calories>650</calories>
  </food>
</breakfast_menu>
```
<?xml version="1.0"?>
<HTML xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <BODY STYLE="font-family:Arial, helvetica, sans-serif; font-size:12pt; background-color:#EEEEEE">
    <xsl:for-each select="breakfast_menu/food">
      <DIV STYLE="background-color:teal; color:white; padding:4px">
        <SPAN STYLE="font-weight:bold; color:white"><xsl:value-of select="name"/></SPAN>
        - <xsl:value-of select="price"/>
      </DIV>
      <DIV STYLE="margin-left:20px; margin-bottom:1em; font-size:10pt">
        <xsl:value-of select="description"/>
        <SPAN STYLE="font-style:italic">(<xsl:value-of select="calories"/> calories per serving)</SPAN>
      </DIV>
    </xsl:for-each>
  </BODY>
</HTML>
XML in Data Islands

With Internet Explorer 5.0, XML can be embedded within HTML pages in Data Islands.

- The unofficial `<xml>` tag is used to embed XML data within HTML.

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

<html><body>
    <xml id="cdcat" src="cd_catalog.xml"></xml>
    <table border="1" datasrc="#cdcat">
        <tr>
            <td><span datafld="ARTIST"></span></td>
            <td><span datafld="TITLE"></span></td>
        </tr>
    </table>
</body></html>
XML and DB: Data Model & Query Lang.

- Semi-structured data model
  - Edge-labeled graph
  - SSD-expression (semi-structured data)

- XML data model
  - Similar to semi-structured data model

- Query language for data in those models
  - The core language
  - Lorel
  - UnQL
  - XML-QL
  - XQuery
<?DOCTYPEDb [
<!ELEMENTdb(r1*, r2*)>
<!ELEMENTr1(a, b, c)>  
<!ELEMENTr2(c, d)>  
<!ELEMENTa(#PCDATA)>  
<!ELEMENTb(#PCDATA)>  
<!ELEMENTc(#PCDATA)>  
<!ELEMENTd(#PCDATA)>  
]>  
<db><r1><a>a1</a> <b>b1</b> <c>c1</c> </r1>
<r1><a>a2</a> <b>b2</b> <c>c2</c> </r1>
<r2><c>c2</c> <d>d2</d> </r2>
<r2><c>c3</c> <d>d3</d> </r2>
<r2><c>c4</c> <d>d4</d> </r2>
</db>
<?xml version="1.0" standalone="yes"?>
<customer>
  <name>
    <first>Pat</first>
    <last>Hemsath</last>
  </name>
  <address>
    <street>Dead-end Building</street>
    <street>Horror Street</street>
    <city>Morristown</city><state>NJ</state>
    <zip>07960</zip>
  </address>
  <phone>973-555-3114</phone>
</customer>
Figure 3.3 Trees for (a) XML data and (b) ssd-expression.
XQuery

FOR $b IN document("bib.xml")/bib/book
WHERE $b/publisher/text() = "Addison-Wesley"
    AND $b/@year = "1994"
RETURN $b/title

<title>TCP/IP Illustrated</title>

http://131.107.228.20/xquerydemo/demo.aspx
Storage & Indexing

- Stored as text file
- Stored in relational database form
- Stored in OODB form

- Simple indexing for tree-shaped semi-structured data
- Region algebra
Figure 8.8 Some tree semistructured data; leaf values are omitted.
A simple index for tree-shaped SSD

- part.name: (from h1 to h2 then look for name
- part.supplier.name: (similarly
- _*.supplier.name: (all nodes in the tree index (in RAM)
- part._*.subpart.name: (from h1 to h2 then explore the whole subtree dominated by h2

- How to design a data structure (index) that allows the system to answer regular path expressions without traversing the whole graph?
- A simple approach is to make a tree index to summarize path information.
- Each node in the index tree is a hash table, of which each entry contains a list of pointers to the corr. nodes in the data tree.
- In general the index will have one node for every sequence of labels leading to a nonleaf node in the data tree.
Figure 8.9 A simple index for tree-shaped semistructured data.
Region Algebra

- $s_1 \text{ intersect } s_2 \overset{\text{def}}{=} \{ r \mid r \in s_1, r \in s_2 \}$.
- $s_1 \text{ included } s_2 \overset{\text{def}}{=} \{ r \mid r \in s_1, \exists r' \in s_2, r \subseteq r' \}$.
- $s_1 \text{ including } s_2 \overset{\text{def}}{=} \{ r \mid r \in s_1, \exists r' \in s_2, r \supseteq r' \}$.
- $s_1 \text{ parent } s_2 \overset{\text{def}}{=} \{ r \mid r \in s_1, \exists r' \in s_2, r \text{ is a parent of } r' \}$.
- $s_1 \text{ child } s_2 \overset{\text{def}}{=} \{ r \mid r \in s_1, \exists r' \in s_2, r \text{ is a child of } r' \}$.
<table>
<thead>
<tr>
<th>part.name</th>
<th>name child (part child root)</th>
</tr>
</thead>
<tbody>
<tr>
<td>part.supplier.name</td>
<td>name child (supplier child (part child root))</td>
</tr>
<tr>
<td>_*.supplier.name</td>
<td>name child supplier</td>
</tr>
<tr>
<td>part._*.subpart.name</td>
<td>name child (subpart included (part child root))</td>
</tr>
</tbody>
</table>
select X
from _*.subpart: {name: X, _*.supplier.address: "Philadelphia"}

translates into the following region algebra expression:

name child (subpart includes (supplier parent (address intersect "Philadelphia"))))
API, SAX and DOM

- For the manipulation of XML
- SAX (Simple API for XML) is more syntax driven. A SAX parser reads the flow of some XML data, parses it, and detects events when interpreting the tags.
- The Document Object Model (DOM) provides an abstract API for constructing, accessing, and manipulating XML and HTML documents. It provides an object-oriented view of an XML document.
<?xml version="1.0" ?>
<employees>
  <employee>
    <ID>123</ID>
    <name>Jake</name>
    <salary>23000</salary>
  </employee>
  ...
</employees>
Note: The bidirectional arrow indicates a parent-child relationship
XML Parser

- To read and update - create and manipulate - an XML document

JavaScript
var xmlDoc = new ActiveXObject("Microsoft.XMLDOM")

VBScript
set xmlDoc = CreateObject("Microsoft.XMLDOM")

VBScript in an Active Server Page (ASP)
set xmlDoc = Server.CreateObject("Microsoft.XMLDOM")
Importance of XML

- 1st breakthrough – computing machines with proprietary OS and application systems
- 2nd breakthrough – C and UNIX OS being portable to different kinds of hardware platforms but UNIX population yielded to MS Windows, leading to the historic OS battle (OS locks in applications)
- 3rd breakthrough – Web browser and Internet technology including Java made Web-based applications independent of OS and hardware platforms (SunMicro’s Java intended to break the OS lock but MS joined but later tossed it out)
- 4th and the latest – XML (endorsed by all big parties including MS BizTalk, .net) promises to transcend all platform boundaries
XML: How is it used

- Web
  - New browsers starting to support XML

- Electronic commerce
  - Possible replacement for EDI (Electronic Data Interchange)

- Data management
  - XMI: XML Metadata Interchange, used to exchange UML models between vendor tools
  - CWM: Common Warehouse Metadata (Oracle + others)

- Publishing
  - Often used in place of SGML because of its “lightness”

- Application to application message format
Concluding Remarks

- XML’s role on EAI
- Data exchange & interoperability
- Standard – sector/industry
- Intelligent use of data through XML tags (metadata)

- The technology is still evolving.