References used for the Lecture:
- XML In A Nutshell, E.R. Harold and W.S. Means, O'Reilly
- XML, Patric Carey, Thomson Course Technology

Acknowledgement: Some examples in these notes are originated from Dr. David Edmond from School of Information Systems, QUT, Brisbane. Revised by Helen Paik, 2005

10th August 2006

A simple, very flexible and extensible text data format
- “extensible” because the markup format is not fixed like HTML
- It lets you design your own customised markup
- XML is a language that describes data
- It separates presentation from the actual data

<xml version="1.0" ?>
  <xml-stylesheet type="text/css" href="staffcard.css" ?>
    <staff>
      <name>Helen Paik</name>
      <title>Wester, UNSW</title>
      <email>hpak@cse</email>
      <extension>54095</extension>
      <photo src="me.gif" />
    </staff>
  </xml-stylesheet>
</xml>
Consider the following snippet of information from a staff list:

<table>
<thead>
<tr>
<th>LName</th>
<th>Title</th>
<th>FName</th>
<th>School</th>
<th>Campus</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edgar</td>
<td>Miss</td>
<td>Pam</td>
<td>Optometry</td>
<td>KG</td>
<td>B501</td>
</tr>
<tr>
<td>Edmond</td>
<td>Dr</td>
<td>David</td>
<td>Information Systems</td>
<td>GP</td>
<td>S842</td>
</tr>
<tr>
<td>Edmonds</td>
<td>Dr</td>
<td>Ian</td>
<td>Physical Sciences</td>
<td>GP</td>
<td>M206</td>
</tr>
</tbody>
</table>

In XML ...

```xml
<Phonebook>
  <Entry>
    <LastName>Edgar</LastName>
    <Title>Miss</Title>
    <FistName>Pam</FirstName>
    <School>Optometry</School>
    <Campus>KG</Campus>
    <Room>B501</Room>
  </Entry>
  <Entry>
    <LastName>Edmond</LastName>
    <Title>Dr</Title>
    <FistName>David</FirstName>
    <School>Information Systems</School>
    <Campus>GP</Campus>
    <Room>S842</Room>
  </Entry>
  <!-- Entry continues ... --!
</Phonebook>
```

Why XML? – Background

**Early Web**
- Used to publish documents to be read by humans
- HTML was designed for the purpose

**Today’s Web**
- Many business activities are performed on the Web
  - Dynamic interactions:
    - Web app ⇔ people / Web app ⇔ Web app
    - Web becomes a platform for data exchange
  - XML provides a simple, cross-platform data format
- Web contains vast amount of data published in HTML format
  - Many programs process or analyse such data
  - HTML changes ... (when data inside does not) → the program that reads the HTML page must change too
  - XML provides a long-term, reliable data format for publishing

Why XML?

**Scenario One: Data Processing**
- An organisation publishes financial data. The data is generated from a relational database. Web pages are generated on demand by invoking an SQL query and formatting its output into HTML. Now, a second organisation wants to obtain some financial analysis of this data but only has access to the HTML page(s). What is a solution in these cases and what are the problems with the solution?

**Scenario Two: Data Exchange**
- The sales department in Company A, which has offices all over the world, wants to get a list of last year’s gold customers from Customer Relations departments. Sales department maintains a central database, but each local Customer Relation branch may use their own software solution. In the past, there has been some confusion in the text files that were exchanged (eg., Fname, then Lname?, Postcode or Zipcode?, currency AUD or USD, YEN?)

Why XML? – Background

**Benefits of using XML in document (data) exchange**
- Self-describing, modular data
- A common, widely accepted data representation language
- Standard supports available for creating/parsing XML docs
- Standard supports for checking validity of data
- Efficient search of business information
  - standard support for querying XML docs
  - keyword + structure based search possible (XQuery)
- Extensible document descriptions
  - XML is flexible!
  - reuse, adaptation of existing documents
XML Applications

A set of ‘tags’ that are developed for specific types of documents.

Math Markup Language (MathML)

\[
\begin{bmatrix}
X & Y \\
Z & W
\end{bmatrix}
\]

- XML is for structuring data.
- XML is for describing data.
- XML looks a bit like HTML (tags).
- XML is text, but isn’t meant to be read.
- XML is verbose by design.
- XML is a family of technologies.
- XML is license-free, platform-independent and well-supported.
- XML is NOT a programming language.

The XML Family

- XML: a language used to describe information.
- DOM: a programming interface for accessing and updating documents.
- DTD/XML schema: a language for specifying the structure and content of documents.
- XSLT: a language for transforming documents.
- XPath: a query language for navigating XML documents.
- XPointer: for identifying fragments of a document.
- XLink: generalises the concept of a hypertext link.
- XInclude: for merging documents.
- XQuery: a language for making queries across documents.
- ....

Quick XML syntax

- All XML elements must have a closing tag
- XML tags are case sensitive
- All XML elements must be properly nested
- Element Naming
  - letters, numbers, and other characters
  - must not start with a number or ‘.’
  - must not start with ‘xml’ (or XML or Xml ..)
  - cannot contain spaces
- Attribute values must always be quoted
- Comments in XML <!-- This is a comment -->
- All XML documents must have a root element
Quick XML Syntax

An XML document is a tree ...

```
<office>
  <phone>1235</phone>
  <person>
    <name>Alan</name>
    <phone>2044</phone>
  </person>
  <person>
    <name>Sue</name>
    <phone>2043</phone>
  </person>
</office>
```

Attributes in XML tags

<table>
<thead>
<tr>
<th>LName</th>
<th>Title</th>
<th>FName</th>
<th>School</th>
<th>Campus</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edgar</td>
<td>Miss</td>
<td>Pam</td>
<td>Optometry</td>
<td>KG</td>
<td>B501</td>
</tr>
</tbody>
</table>

- Sometimes using attributes can make an XML document concise
- Avoid using too many (loses structure, more parsing effort ...)

Some more XML examples

How would you write an SQL query in XML?

**Select CarNr, Make, Cost From Cars Order by Cost**

```
<sql>
  <select order="Cost">
    <col>carNr</col>
    <col>Make</col>
    <col>Cost</col>
  </select>
  <from>
    <table>Cars</table>
  </from>
</sql>
```

The XML Specification

- The core of the XML specification is a series of production rules.
- Each production has the form: symbol ::= expression
- Some of these productions are:
  - `document ::= prolog element Misc*`
  - `prolog ::= XMLDecl? Misc* (doctypedecl Misc*)?`
  - `element ::= EmptyElemTag | STag content ETag`
  - `Misc ::= Comment | PI | S`
  - `Comment ::= "<!--" ((Char - '-') | ('-' (Char - '-')))"-->
  - `S ::= (#x20 | #x9 | #xD | #xA)+`

(Refer to the next slide for the EBNF grammar)
Extended Backus-Naur Form (EBNF)

The following forms of expression are permitted by the grammar.

- A? matches A or nothing; optional A.
- A B matches A followed by B.
- A | B matches A or B but not both.
- A - B matches any string that matches A but does not match B.
- A+ matches one or more occurrences of A.
- A* matches zero or more occurrences of A.
- #xN? matches the corresponding ISO/IEC 10646 character.
- ’string’ matches the enclosed string exactly.
- "string" matches the enclosed string exactly.
- (expression) expression is treated as a unit and may be combined as described in this list.

Topics

In these notes, we discuss the structure of XML documents. In particular, we discuss how to:

- Define the overall structure of an XML document (DTDs).
- Describe the allowable content of an element.
- Make use of existing DTDs by reference.
- Define different kinds of attributes.
- Identify individual elements.
- Cross reference elements.
- Make use of entities to construct our XML documents.

- http://www.w3.org/TR/REC-xml

Part II

Document Type Definition: DTD

Defining the document structure

Phonebook.xml

<Phonebook>
  <Entry>
    <LastName Title="Miss">Edgar</LastName>
    <FirstName>Pam</FirstName>
    <School>Optometry</School>
    <Campus>KG</Campus>
    <Room>B501</Room>
    <Extension>5695</Extension>
  </Entry>
</Phonebook>

How would we communicate the nature of this document? If we were to describe the document to someone over a phone line, we might say:

- It’s a kind of internal (staff) phone book.
- It’s made up of a number of individual entries.
- Each entry contains the staff member’s last name, title, first name ...
- A person’s title must be Miss or Mrs or Ms or Mr or Dr or Prof ...
Document Type Definitions

DTDs are used to ensure that XML docs adhere to a "agreed" structure.

XML prolog ::= XMLDecl? Misc* (doctypedcl Misc*)?

DTD can be declared:

**within XML:** Internal DTD

```xml
<?xml version="1.0"?>
<!DOCTYPE Login [ 
<!ELEMENT Login (Username,Password)>
<!ELEMENT Username (#PCDATA)>
<!ELEMENT password (#PCDATA)>
]>
<Login>
<Username>hpaik</Username>
<Password>IwillNeverTell</Password>
</Login>
```

**outside XML:** External DTD

```xml
<?xml version="1.0"?>
<!DOCTYPE Login SYSTEM "login.dtd">
<Login>
<Username>hpaik</Username>
<Password>IwillNeverTell</Password>
</Login>
```

Based on COMP9321, s106 (CSE, UNSW)

Phonebook.xml with Internal DTD

```xml
<?xml version="1.0"?>
<!DOCTYPE Phonebook [ 
<!ELEMENT Phonebook (Entry)+ >
<!ELEMENT Entry (LastName, FirstName, School, Campus, Room, Extension)>
<!ELEMENT LastName (#PCDATA)>
<!ELEMENT FirstName (#PCDATA)>
<!ELEMENT School (#PCDATA)>
<!ELEMENT Campus (#PCDATA)>
<!ELEMENT Room (#PCDATA)>
<!ELEMENT Extension (#PCDATA)>
<!ATTLIST LastName Title (Miss | Ms | Mrs | Mr | Dr | Prof) #REQUIRED>
]>
<Phonebook>
<Entry>
<LastName Title="Miss">Edgar</LastName>
<FirstName>Pam</FirstName>
<School>Optometry</School>
<Campus>GP</Campus>
<Room>B501</Room>
<Extension>5695</Extension>
</Entry> <!-- more entries not shown ... -->
</Phonebook>
```

Phonebook.xml with External DTD. Phonebook.dtd

Phonebook.xml

```xml
<?xml version="1.0"?>
<!DOCTYPE Phonebook SYSTEM "Phonebook.dtd">
<Phonebook>
<Entry>
<LastName Title="Miss">Edgar</LastName>
<FirstName>Pam</FirstName> <!-- rest of the entries -->
</Phonebook>
```

Phonebook.dtd

```xml
<!ELEMENT Phonebook (Entry)+ >
<!ELEMENT Entry (LastName, FirstName, School,Campus, Room, Extension)>
<!ELEMENT LastName (#PCDATA)>
<!ELEMENT FirstName (#PCDATA)>
<!ELEMENT School (#PCDATA)>
<!ELEMENT Campus (#PCDATA)>
<!ELEMENT Room (#PCDATA)>
<!ELEMENT Extension (#PCDATA)>
<!ATTLIST LastName Title (Miss | Ms | Mrs | Mr | Dr | Prof) #REQUIRED>
```

Defining XML Content: Elements

A Book

```xml
<book>
<author>
<name>J.K. Rowling</name>
</author>
<detail>
<series>Sixth</series>
<title>Harry Potter and the Half-Blood Prince</title>
</detail>
</book>
```

Creating Elements:

```xml
<!ELEMENT book (author, detail)>
<!ELEMENT author (name)>
<!ELEMENT detail (series, title)>
<!ELEMENT series (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT title (#PCDATA)>
```
Defining XML Content: Modifiers

A Book

```xml
<book>
  <author> <!-- more than one authors? -->
    <name>E. Harold</name> <name>S. Means</name>
  </author>
  <detail> <!-- not every book is in a series -->
    <title>XML in a Nutshell</title>
  </detail>
</book>
```

- ? : optional element (only once)
- + : mandatory element (1 or more)
- * : optional element (0 or more)

```xml
<!ELEMENT book (author, detail*)>
<!ELEMENT author (name+)>  
<!ELEMENT name (#PCDATA)>  
<!ELEMENT detail ((series?, title) | (publisher, release*))>
<!ELEMENT series (#PCDATA)>  
<!ELEMENT title (#PCDATA)>  
<!ELEMENT publisher (#PCDATA)>  
<!ELEMENT release (#PCDATA)>  
```

Defining XML Content: Choices, Empty

Element Choices

```xml
<!ELEMENT newbooks (book+)>
<!ELEMENT book (author+, detail*)>
<!ELEMENT author (name | penname)>  
<!ELEMENT name (#PCDATA)>  
<!ELEMENT penname (#PCDATA)>  
<!ELEMENT detail ((series?, title) | (publisher, release*))>
<!ELEMENT series (#PCDATA)>  
<!ELEMENT title (#PCDATA)>  
<!ELEMENT publisher (#PCDATA)>  
<!ELEMENT release (#PCDATA)>  
```

Empty Element Content

```xml
<!ELEMENT BR EMPTY>
```

<br/> is called an empty element

Defining XML Content: Mixed content, Any

Mixed content: mixture of elements and text

```xml
<!ELEMENT message (#PCDATA | bold | italic)>  
```

```xml
<message> You <italic> really </italic> <bold> must </bold> try this delicious <bold> new </bold> recipe for <italic> pudding </italic> </message>
```

ANY : Any predefined element could be included

```xml
<!ELEMENT book (author+, description, detail*)>  
<!ELEMENT author (name+)>  
<!ELEMENT name (#PCDATA)>  
<!ELEMENT description ANY>  
<!ELEMENT detail (series?, title)>  
<!ELEMENT series (#PCDATA)>  
<!ELEMENT title (#PCDATA)>  
```

SQL Query example

A SQL query

```xml
<sql>
  <select>
    <col>CarNr</col>
    <col>Make</col>
    <col>Cost</col>
  </select>
  <from>
    <table>Cars</table>
  </from>
</sql>
```

Here is DTD

```xml
<?xml version="1.0"?>
<!DOCTYPE sql [  
  <!ELEMENT sql (select, from)>  
  <!ELEMENT select (col+)>
  <!ELEMENT col (#PCDATA)>  
  <!ELEMENT from (table+)>  
  <!ELEMENT table (#PCDATA)> ]>
```
Defining XML Content: Creating Attributes

Creating Attributes:

```
<!ELEMENT book (author, title)>
<!ELEMENT author (name+)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ATTLIST name type (normal | penname) "normal" #REQUIRED>
<!ATTLIST author period CDATA #REQUIRED
category CDATA #IMPLIED>
```

Based on COMP9321, s106 (CSE, UNSW) XML and DTD 10th August 2006 29 / 38

Example: Hot Gossip Report

Consider the following table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ann</td>
<td>22 Strand Bvd, Copenhagen</td>
</tr>
<tr>
<td>Bill</td>
<td>3391 1615</td>
</tr>
<tr>
<td>Sue</td>
<td>8223 2555</td>
</tr>
<tr>
<td>Doug</td>
<td>3 Via Appia, Rome</td>
</tr>
</tbody>
</table>

Suppose that the document begins:

```
<HotGossip>
  <Friend>Ann</Friend>
  <Contact>
    <Address>22 Strand Bvd, Copenhagen</Address>
  </Contact>
</HotGossip>
```

Let us design the DTD for it ...

Example: Match-making Report

Consider the following table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Sex</th>
<th>Age</th>
<th>Hobbies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bill</td>
<td>m</td>
<td>29</td>
<td>sport, travel</td>
</tr>
<tr>
<td>Sue</td>
<td>f</td>
<td>31</td>
<td>travel</td>
</tr>
<tr>
<td>Ian</td>
<td>m</td>
<td></td>
<td>music, art, photography</td>
</tr>
<tr>
<td>Ann</td>
<td>f</td>
<td>32</td>
<td>Ian</td>
</tr>
</tbody>
</table>

Suppose that the document begins:

```
<MatchMaking>
  <Name Sex="m" Age="29">Bill</Name>
  <Hobby>sport</Hobby><Hobby>travel</Hobby>
</MatchMaking>
```

Let us design the DTD for it ...

Default values for attributes:

The default postcode in an address is to be 4001, state must be QLD.

```
<!ATTLIST Address Postcode CDATA "4001" 
             State CDATA #FIXED "QLD">
```

The above definition has the following effects on the source doc.

- `<Address />`  →  `<Address Postcode="4001" State="QLD"/>
- `<Address Postcode="4010" State="QLD"/>
  →  (no error)
- `<Address Postcode="4001" State="NSW"/>
  →  (error)
Attribute Type ID and IDREF

We can introduce attributes whose specific purpose is to identify certain elements and cross-reference between the elements.

```
<DOCTYPE DBC [
<!ELEMENT DBC (Depts, Employees)>  
<!ELEMENT Depts (Dept)*>  
<!ELEMENT Dept (Boss, Owes)>  
<!ATTLIST Dept DeptName ID #REQUIRED>  
<!ELEMENT Boss (#PCDATA)>  
<!ELEMENT Owes (#PCDATA)>  
<!ELEMENT Employees (Employee)*>  
<!ATTLIST Employee Name ID #REQUIRED DeptName IDREF #REQUIRED> 
]>  

<DBC>
  <Depts>
    <Dept DeptName="Bread">
      <Boss>Bruce</Boss><Owes>1000</Owes>
    </Dept>
    <Dept DeptName="Cake">
      <Boss>Cathy</Boss><Owes>1500</Owes>
    </Dept>
    <Dept DeptName="Finance">
      <Boss>Phil</Boss><Owes>60000</Owes>
    </Dept>
  </Depts>
  <Employees>
    <Employee Name="Bruce" DeptName="Bread"/>
    <Employee Name="Bob" DeptName="Bread"/>
    <Employee Name="Betty" DeptName="Bread"/>
    <Employee Name="Bill" DeptName="Bread"/>
    <Employee Name="Cathy" DeptName="Cake"/>
    <Employee Name="Carol" DeptName="Cake"/>
    <Employee Name="Chris" DeptName="Cake"/>
    <Employee Name="Phil" DeptName="Finance"/>
    <Employee Name="Frank" DeptName="Finance"/>
    <Employee Name="Fred" DeptName="Finance"/>
  </Employees>
</DBC>
```

XML Entities

**General entities**

An entity is a reference to other content. Here are a couple of examples of XML general entities:

```
<xsl:value-of select="price[@normal = 200]"/>
<BigGig>U2 &amp; Friends</BigGig>
&amp;lt; (&lt;) &amp;gt; (&gt;) &quot; (") &apos; (')
```

Note: sometimes, you would want to store a block of text that may contain <, &amp; etc. It will be cumbersome to replace all of those with entities. You can use CDATA. XML parser treats CDATA block as a simple text (ie., does not interpret the section).

```
<htmlcode>
<![CDATA [  
  <h1>The Jazz Warehouse</h1>  
  Latest specials from the warehouse. New & Used CDs. ]]>  
</htmlcode>
```

**XML Custom Entities**

The resulting document looks like this:

```
<book>
  <title>Mark Jackson: my words</title>
  <author>Mark Jackson</author>
  <author>Sue Hacker</author>
  <copyright>Mark Jackson</copyright>
</book>
```
XML Custom Entities

Sometimes it might be desirable to construct a document from several (other) XML documents:

```xml
<!DOCTYPE sql [
<!ELEMENT sql (select, from)>  
<!ELEMENT select (col+)>
<!ATTLIST select order CDATA #REQUIRED>
<!ELEMENT col (#PCDATA)>  
<!ELEMENT from (table+)>
<!ELEMENT table (#PCDATA)>  
<!ENTITY select SYSTEM "select.xml">  
<!ENTITY from SYSTEM "from.xml">]>
<sql>&select;&from;</sql>
```

Where "select.xml" contains:

```xml
<select order="cost">
    <col>CarNr</col>
    <col>Make</col>
    <col>Cost</col>
</select>
```

And "from.xml" contains:

```xml
<from>
    <table>Cars</table>
</from>
```

Parameter Entities

It might be a good idea to fragment the DTD in the same way that the document content is partitioned:

```xml
<!DOCTYPE sql [
<!ELEMENT sql (select, from)>  
<!ENTITY % seldef SYSTEM "select.dtd">  
%!ENTITY % seldef SYSTEM "select.dtd">  
%!ENTITY % fromdef SYSTEM "from.dtd">  
%!ENTITY fromdef SYSTEM "from.dtd">  
<!ENTITY select SYSTEM "select.xml">  
<!ENTITY from SYSTEM "from.xml">]>
<sql>&select;&from;</sql>
```

Where select.dtd is defined as:

```xml
<!ELEMENT select (col+)>
<!ATTLIST select order CDATA #REQUIRED>
<!ELEMENT col (#PCDATA)>  
<!ELEMENT table (#PCDATA)>  
```

and "from.dtd" is:

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