Web Application Engineering: XML and the Web

Service Oriented Computing Group, CSE, UNSW

Week 4

References used for the Lecture:

- XML in a nutshell, Chapters 9 and 10

Acknowledgement: Some examples in these notes are originated from Dr. David Edmond from QUT, Brisbane
eXtensible Markup Language (XML)

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 issues from the actual data

---

**in HTML ...**

```html
<html>
  <h1>Bibliography</h1>
  <ol>
    <li><i>Foundation of Databases</i>, <b>Abiteboul, Hull</b>, 1995</li>
    <li><i>Database Systems</i>, <b>Elmasri, Navathe</b>, 1994</li>
  </ol>
</html>
```

**in XML ...**

```xml
<bibliography>
  <book>
    <title>Foundation of Databases</title>
    <author>Abiteboul</author>
    <author>Hull</author>
    <year>1995</year>
  </book>
  <book> <!-- continues --></book>
</bibliography>
```
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 ...
**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?

Benefits of using XML in document (data) exchange

- Self-describing, modular and portable 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
  - quick and simple search (XPath)
  - more comprehensive keyword + structure based search possible as well (XQuery)
- Extensible document descriptions
  - XML is flexible (cf. relational tables)!
  - reuse, adaptation of existing documents
Separating the Content from Presentation

XML

```xml
<?xml version="1.0" ?>
<?xml-stylesheet type="text/css" href="staffcard.css" ?>
<staff>
  <name>Helen Paik</name>
  <title>Lecturer, UNSW</title>
  <email>hpaik@cse</email>
  <extension>54095</extension>
  <photo src="me.gif" />
</staff>
```

CSS

```
staff{background-color: #cccccc; ...}
name{display: block; font-size: 20pt; ...
title{display: block; margin-left: 20pt;}
email{display: block; font-family: monospace;
extension{display: block; margin-left: 20pt;}
```
Like any other good inventions, XML is now used for things that are far beyond its creators original imagination.

- A set of 'tags' that are developed for specific types of documents.

  e.g., Chemical Markup Language (CML)

<atom id="caffeine_karne_a_1">
  <float builtin="x3" units="A">-2.8709</float>
  <float builtin="y3" units="A">-1.0499</float>
  <float builtin="z3" units="A">0.1718</float>
  <string builtin="elementType">C</string>
</atom>
Really Simple Syndication (RSS)

```xml
<rss version="0.91">
  <channel>
    <title>CNN.com</title>
    <item>
      <title>July ends with 76 ... killed</title>
      <link>http://www.cnn.com/.../story.html</link>
      <description>Three U.S. soldiers were ...</description>
    </item>
  </channel>
</rss>
```
XML is ...

- **Is a Language**
  - there is a grammar, and it can be parsed by machines
- **Is a Markup Language**
  - XML looks a bit like HTML (tags).
  - But it describes what things are, not what they are supposed to do
- **Is Extensible**
  - you can define more words and add to the language
- **XML is for structuring data.**
- **XML is for describing data.**
- **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.**
  - it is not something you can 'compile'
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.
- RDF: a language for describing resources.
Quick XML Syntax

An XML document is a tree ...

```xml
<office>
    <phone>1235</phone>
    <person>
        <name>Alan</name>
        <age>29</age>
        <phone>2044</phone>
    </person>
    <person>
        <name>Sue</name>
        <age>45</age>
        <phone>2043</phone>
    </person>
</office>
```
XML can be used for more free-form documents (e.g., business reports, magazine articles, essays, short stories, etc.)

Example

```xml
<biography>
  <name><first_name>Alan</first_name> <last_name>Turing</last_name>
  </name> was one of the first people to truly deserve the name <emphasize>computer scientist</emphasize>. Although his contributions to the field are too numerous to list, his best-known are the eponymous <emphasize>Turing Test</emphasize> and <emphasize>Turing Machine</emphasize>.

  <definition>The <term>Turing Test</term> is to this day the standard test for determining whether a computer is truly intelligent. This test has yet to be passed.</definition>
</biography>
```

Mixed content: some elements may contain both character data and child elements (e.g., `<definition>` and `<biography>`)
## 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>

... ... ... ... ... ...

### Phonebook with Attributes

```xml
<Phonebook>
  <Entry entryNumber="001">
    <Name Title="Miss">
      <Last>Edgar</Last>
      <First>Pam</First>
    </Name>
    <School Campus="KG">Optometry</School>
    <Room Building="B" Level="5">01</Room>
  </Entry>
</Phonebook>
```

- Attribute order is not significant
- Sometimes using attributes can make an XML document concise
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>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Phonebook with many attributes ...

```xml
<Phonebook>
  <Entry entryNumber="001">
    <Name Title="Miss" LName="Edgar" FName="Pam"/>
    <Location Campus="KG" School="Optometry" Building="B" Room="501"/>
  </Entry>
</Phonebook>
```

- Avoid using too many (loses structure, more parsing effort ...)

H. Paik, S.Venugopal (CSE, UNSW)
The character data inside an element must not contain certain characters with special meanings (e.g., < means start of a tag). You must escape the characters using entity references.

XML predefines exactly five entity references:

- &lt; - The less then sign (<)
- &amp; - The ampersand (&)
- &gt; - The greater than sign (>)
- &quot; - The straight double quotation marks ("")
- &apos; - The apostrophe, single quote (’)

```xml
<image source='koala.gif' width='122' height='66'
    alt = 'Powered by O'Reilly Books'
/>
```
Sometimes the character data of an element might contain too many characters that need to be escaped (e.g., chunk of other XML parts or HTML code).

CDATA section lets you enclose the character data as literal.

Example

```xml
<p>You can use default <code>xmlns</code> to attribute to avoid having to add the svg prefix to all your elements:</p>

```xml
<![[CDATA[
  <svg xmlns="http://www.w3.org/2000/svg"
       width="12cm" height="12cm"
       <eclipse rx="110" ry="130"/>
       <rect x = "4cm" y="1cm" width="3cm" height="6cm"/>
  </svg>
]]>
```

Everything between `<!CDATA[ and ]]>` is treated as raw characters, not markups.
Quick XML syntax

- All XML documents must have 'a' root element
- All XML elements must have a closing tag
  - Empty element tags end with />
- XML tags are case sensitive (NAME vs. Name)
- All XML elements must be nested (<p><q></p><q>??)
- Element Naming
  - letters, numbers, and other characters
  - must not start with a number, '. (period)' or '- (hyphen)'
  - must not start with 'xml' (or XML or Xml ..)
  - cannot contain spaces
- Attribute values must always be quoted (single or double)
- Comments in XML: <!-- This is a comment -->
Defining the document structure

Phonebook.xml

```xml
<Phonebook>
  <Entry>
    <LastName Title="Miss">Edgar</LastName>
    <FirstName>Pam</FirstName>
    <School>Optometry</School>
    <Campus>KG</Campus>
    <Room>B501</Room>
    <Extension>5695</Extension>
  </Entry>
  <!-- and so on -->
</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:

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

**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>
```
<?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

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

```xml
<Book>
  <Author>
    <Name>J.K. Rowling</Name>
  </Author>
  <Detail>
    <Series>Seventh</Series>
    <Title>Harry Potter and the Deathly Hallows</Title>
  </Detail>
</Book>
```

Creating Elements:

```xml
<!ELEMENT book (author, detail)>    <!ELEMENT detail (series, title)>
<!ELEMENT author (name)>             <!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>
```

1. `?`: optional element (only once)
2. `+`: mandatory element (1 or more)
3. `*`: optional element (0 or more)

```
<!ELEMENT book (author, detail*)>
<!ELEMENT detail (series?, title)>
<!ELEMENT author (name+)>
<!ELEMENT series (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT title (#PCDATA)>
```
Element Choices

<!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

<!ELEMENT BR EMPTY>

<BR/> is called an empty element
Mixed content: mixture of elements and text

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

<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)>
<book>
  <author period="classical" category="children">
    <name type="normal">J.K. Rowling</name>
  </author>
  <title>Harry Potter and the Half-Blood Prince</title>
</book>

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>
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)
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>
```
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">
%seldef;
<!ENTITY % fromdef SYSTEM "from.dtd">
%fromdef;
<!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)>  
```

and "from.dtd" is:

```xml
<!ELEMENT from (table+)>  
<!ELEMENT table (#PCDATA)>  
```
Well-formedness and Validity of XML

Well-formedness Rules:

- Open and close all tags
- Empty-element tags end with />
- There is a unique root element
- Elements may not overlap
- Attribute values are quoted
- < and & are only used to start tags and entity references, respectively
- Only the five predefined entity references are used

Validity Rules:

- Well-formed
- Must have a Document Type Definition (DTD)
- Must comply with the constraints specified in the DTD
Limitations of DTD

- Limited support for constraining attribute values
- No limits on character data
- Limited support for namespaces
- No self-documentation, does not have XML syntax
XML Schema Definition

XML Schema - W3C’s recommendation for replacing DTD with features such as:

- Simple and complex data types
- Type derivation and inheritance
- Namespace-aware element and attributes
- Limits on number of appearances by an element
- Combining with regular expressions for finer control over document structure

Most importantly, XML Schemas are well-formed XML documents themselves. But first, what is a namespace?
XML Namespaces

- XML elements can have any names.
- What if a name could mean two different things (i.e., name clash)?

The following two XML documents that describe student information.

From University X:
```
<student>
  <id>12345</id>
  <name>Jeff Smith</name>
  <language>C#</language>
  <rating>9.5</rating>
</student>
```

From University Y:
```
<student>
  <id>534-22-5252</id>
  <name>Bob Citizen</name>
  <language>Spanish</language>
  <rating>3.2</rating>
</student>
```

How could a program distinguish the different elements?
Another example ...
A namespace is a set of names in which all names are unique.

- The name ‘title’ can now be identified as: Book.title, Project.title, Employee.title ...
- These names are called “qualified names”.
- XML namespaces give elements and attributes a unique name across the Internet.
- XML namespaces enable programmers to process the tags and attributes they care about and ignore those that don’t matter to them.
The previous examples can now have qualified names:

namespace:UniversityX/Student

- ID
- rating
- name
- language

namespace:UniversityY/Student

- ID
- rating
- name
- language

Book Namespace

- <Books>
  - <Book>
    - <Name>
    - <ISBN>
    - <Ed>
    - <Author>
      - <Name>
        - <First>
        - <Last>
      - <Email>

Author Namespace
XML Namespace Syntax

- `xmlns:<prefix>='namespace identifier'`

- eg. `<books xmlns:xdc="http://www.xml.com/books">

- not a normal XML attribute (treated differently)

- the URI must be unique, but may not represent a 'useful' resource

- the prefix is by convention or author's choice

Consider the following XML document: painting.xml

```xml
<catalog>
  <rdf:RDF xmlns:rdf="http://www.w3.org/TR/REC-rdf-syntax#">
    <rdf:Description xmlns:dc="http://purl.org/dc/" about="painting.xml">
      <dc:title>Impressionist Paintings</dc:title>
      <dc:creator>Elliotte Rusty Harold</dc:jcreator>
      <dc:description>impressionist paintings</dc:description>
      <dc:date>2000-08-22</dc:date>
    </rdf:Description>
  </rdf:RDF>
  <painting>
    <title>Memory of the Garden at Etten</title>
    <artist>Vincent Van Gogh</artist>
    <date>1888</date>
    <description>Two women look to the left.</description>
  </painting>
</catalog>
```
XML Schema Definition

The elements of an schema are derived from the XML Schema namespace (http://www.w3.org/2001/XMLSchema)

- Every schema document should start with xs:schema or schema element.

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.example.org"
xmlns:tns="http://www.example.org"
elementFormDefault="qualified">
```

- All the names defined in a schema belong to its targetNamespace.
- Element declarations can refer to names in other source namespaces as well.
  - A form of inheritance.
Types and Declarations

- Simple Types - Basic data types such as strings, integers, boolean, etc.
- Complex Types - Composed of simple types. Consists of an arrangement of elements and attributes.
- Element declarations - Associates an element name to an instance of a simple or complex type
- Attribute declaration - Associates an attribute name with an instance of a simple type.
Simple Types

Simple Type Declaration

- Cannot contain elements or attributes
- Can be pre-defined or user-defined.

Examples:

```xml
<element name='sname' type="string" />
<element name='age' type="integer" />
<element name='course' type='string'/>
```

Which defines:

```xml
<sname>John Doe</sname>
<age>24</age>
<course>COMP9321</course>
```
Attributes

Attributes are simple types as well (however, simple types themselves CANNOT contain attributes)

<attribute name="currency" type="string"/>

Attributes can have default or fixed values and can be optional or required

<attribute name="currency" type="string" default="EUR"/>
<attribute name="currency" type="string" fixed="AUD"/>
<attribute name="currency" type="string" use="required"/>
Type Restrictions

Restrictions are used to specify a range of acceptable values for XML elements or attributes.

Examples:

```xml
<simpleType name="nameString">
  <restriction base="string">
    <pattern value="([a-zA-Z])+/"/>
  </restriction>
</simpleType>

<element name='sname' type="tns:nameString" />

<simpleType name="ageNum">
  <restriction base="integer">
    <minInclusive value="1"/>
    <maxInclusive value="80"/>
  </restriction>
</simpleType>

<element name='age' type="tns:ageNum" />

<simpleType name="courseString">
  <restriction base="string">
    <enumeration value="COMP9321"/>
    <enumeration value="COMP9322"/>
    <enumeration value="COMP9323"/>
  </restriction>
</simpleType>

<element name='course' type="tns:courseString"/>
```
Complex Types

Complex Types can be empty or composed of only elements, or only text, or a mix of both elements and text. The number of elements are controlled by the indicators as below:

Order indicators:
- All - all elements specified in the type can occur in any order but must occur only once
- Choice - either one or the other element must be present
- Sequence - all elements must occur in the order specified. You can have more than one element.

Occurrence indicators:
- maxOccurs
- minOccurs
<complexType name="student">
  <sequence>
    <element name='sname' type="tns:nameString" />
    <element name='age' type="tns:ageNum" />
    <element name='course' type="tns:courseString"/>
  </sequence>
</complexType>

which defines:

@student>
  <sname>John Doe</sname>
  <age>24</age>
  <course>COMP9321</course>
</student>
Inside an XML file, there are ...

- **Markup:**
  - Tags, Entity References, Comments, Processing Instructions, DTD declarations, XML declaration, and CDATA Section Delimiters
- and **Character Data** which includes everything else
  - **Parsed Character Data (PCDATA):** character data left after entity references are replaced with their text
  - e.g. Given `<PUBLISHER>A & M Records</PUBLISHER>`, the parsed character data is `A & M Records`
What do you mean by ’parsing (or processing) XML docs’?

- Parsing makes an interface available to your application that needs to make use of the document.
- Through the interface, you can modify, retrieve the document contents.

What if the interface provided by the parser is parser-specific? → your application will have to be ’parser-specific’.
Obviously ... we want “STANDARD”!
SAX and DOM as the Standard Interfaces

- **SAX** - the **Simple** API for **XML**
- **DOM** - the **Document Object Model**

Why two standards? → trade-off between control and performance

- **DOM** gives you a tree structure
  - you have a complete control over the structure
  - ie., traverse the tree, modify structure, etc.
  - the tree gets stored in memory all at once

- **SAX** lays out the document in time, as a sequence of ‘events’
  - events are associated with each tag (open/close), each tag body, etc.
  - you will write event handlers (ie., you can ignore certain events)
  - It requires much less memory
  - It becomes difficult to use if processing an element depends on earlier/later elements
Document Object Model (DOM)

- DOM is an API for HTML and XML documents, its specification is developed by W3C (http://www.w3.org/DOM/DOMTR).
- It defines the logical structure of documents and the way a document is accessed and manipulated.

```
<TABLE>
  <TBODY>
    <TR>
      <TD>Assignment One</TD>
      <TD>Submission Instructions</TD>
    </TR>
    <TR>
      <TD>Assignment Two</TD>
      <TD>Submission Instructions</TD>
    </TR>
  </TBODY>
</TABLE>
```

http://java.sun.com/j2se/1.5.0/docs/api/org/w3c/dom/package-summary.html
Dealing with Nodes in DOM

In DOM, XML Documents are treated as a tree of nodes

**Types of Nodes**: Twelve different kinds of node are defined by the W3C DOM standard.

- elements
- attributes
- text
- CDATA
- entity reference
- entity
- processing instruction
- comment
- Document
- Document type
- Document fragment
- Notation
<?xml version="1.0"?>
<bibliography>
  <book>
    <author>North, Ken</author>
    <title>Database magic with Ken North</title>
    <address>Upper Saddle River, NJ</address>
    <publisher>Prentice Hall</publisher>
    <year>1999</year>
    <isbn>0136471994</isbn>
  </book>
  <book>
    <author>Elmasri, Ramez, and Shamkant B. Navathe</author>
    <title>Fundamentals of database systems</title>
    <address>Reading, Mass.</address>
    <publisher>Addison-Wesley Pub Co</publisher>
    <year>2000</year>
    <edition>3rd</edition>
    <isbn>0201542633</isbn>
  </book>
  <book>
    <author>Feiler, Jesse</author>
    <title>Database-driven Web sites</title>
    <address>San Francisco</address>
    <publisher>Morgan Kaufmann</publisher>
    <year>1999</year>
    <isbn>0122513363</isbn>
  </book>
<!-- more book -->
</bibliography>
% Text of an element node is stored in a text node.
The root node has no parent but every other node has exactly one parent node.

A node can have any number of children.

Two nodes that have the same parent are called siblings.

A node with no children is called a leaf node.

Among siblings, the node that appears first in sequence is called the first child and the node that appears last is the last child.

Important: Text of an element node is stored in a separate text node.
Using a DOM Parser (eg., Apache Xerces)

```java
import org.w3c.dom.*;
import org.apache.xerces.parsers.DOMParser;

public class DOMCountNames {
    public static void main(String[] args) {
        try {
            DOMParser parser = new DOMParser();
            parser.parse(args[0]);
            Document doc = parser.getDocument();
            // do something ..
        } catch (Exception e) {
            e.printStackTrace(System.err);
        }
    }
}
```

**The DOMParser Class:**
- DOMParser class is derived from the XMLParser class
- parse() method parses the input source given by a system identifier
- Document getDocument() method returns the document itself
Document Interface Methods

Once you have the `Document` object, you can:

- `Attr createAttribute(String name)`: Creates an attribute
- `Element createElement(String tagName)`: Creates an element
- `Text createTextNode(String data)`: Creates a Text Node
- `Element getDocumentElement()`: Gets the root element of the document
- `Element getElementById(String elementId)`: Get the element by ID
- `NodeList getElementsByTagName(String tagname)`: Returns a NodeList of all the elements with a given tag name

`NodeList` Interface Methods:

- `int getLength()`: Gets the number of nodes in this list
- `Node item(int index)`: Gets the item at the specified index value in the collection
Examples of Node Properties (XML), p.9.25

```
document.getElementsByTagName("person")[1]
document.getElementsByTagName("person")[1].parentNode
document.getElementsByTagName("person")[1].childNodes
document.getElementsByTagName("person")[1].firstChild
document.getElementsByTagName("person")[1].lastChild
document.getElementsByTagName("person")[1].previousSibling
document.getElementsByTagName("person")[1].lastChild.firstChild
```
import org.w3c.dom.*;
import org.apache.xerces.parsers.DOMParser;
public class DOMCountNames {
    public static void main(String[] args) {
        try{
            DOMParser parser = new DOMParser();
            parser.parse(args[0]);
            Document doc = parser.getDocument();
            NodeList nodelist = doc.getElementsByTagName("book");
            System.out.println(args[0] + " has " + nodelist.getLength()
                    + " <book> elements.");
        }
        catch(Exception e){
            e.printStackTrace(System.err);
        }
    }
}
Dealing with Nodes in DOM

There is a large range of methods that can be applied to the nodes:

<table>
<thead>
<tr>
<th>Node Interface Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>getNodeName()</td>
</tr>
<tr>
<td>getNodeType()</td>
</tr>
<tr>
<td>getFirstChild()</td>
</tr>
<tr>
<td>getNextSibling()</td>
</tr>
<tr>
<td>insertBefore(...)</td>
</tr>
<tr>
<td>appendChild(...)</td>
</tr>
<tr>
<td>normalize()</td>
</tr>
<tr>
<td>hasAttributes()</td>
</tr>
<tr>
<td>getNodeValue()</td>
</tr>
<tr>
<td>getParentNode()</td>
</tr>
<tr>
<td>getLastChild()</td>
</tr>
<tr>
<td>getAttributes()</td>
</tr>
<tr>
<td>replaceChild(...)</td>
</tr>
<tr>
<td>hasChildNodes()</td>
</tr>
<tr>
<td>isSupported(...)</td>
</tr>
<tr>
<td>getLocalName()</td>
</tr>
<tr>
<td>setNodeValue(...)</td>
</tr>
<tr>
<td>getChildNodes()</td>
</tr>
<tr>
<td>getPreviousSibling()</td>
</tr>
<tr>
<td>removeChild(...)</td>
</tr>
<tr>
<td>cloneNode(...)</td>
</tr>
<tr>
<td>getNamespaceURI()</td>
</tr>
</tbody>
</table>
Consider the following program:

```java
Document doc = parser.getDocument();
Element docRoot = doc.getDocumentElement();
String docRootName = docRoot.getTagName();
System.out.println("Doc root: " + docRootName);
int i = 0;
for (Node node = docRoot.getFirstChild();
    node != null;
    node = node.getNextSibling()) {
    if (node.getNodeType()==Node.ELEMENT_NODE) {
        System.out.println(i+: " + node.getNodeType()
            + node.getNodeName());
    } else {
        System.out.println(i+: " + node.getNodeType());
    }
    i++;
}
```

The method `getNodeType()` returns a number in the range 1 to 12. Thus we can tell which kind of node we are dealing with. What will the output of this program be?
The Element interface

This interface outlines operations that are specific to elements:

- `getTagName()`: This method returns the name of the tag associated with the element.

- `getAttribute(name)`: This method returns a string containing the value of an attribute:
  - `name` is the name of the attribute.

Amend the code on the previous example to print out the name and value of the type attribute attached to the food element.

```java
Element tmp = (Element)node;
if (tmp.getTagName().equals("food"))
{
    String typeVal = tmp.getAttribute("type");
    System.out.println("type: "+typeVal);
}
```
More with DOM ...

Heaps of hands on tutorials on the web ...

- DOM and Javascript: e.g.,
  http://www.sitepoint.com/print/xml-javascript-mozilla
  https://developer.mozilla.org/en/The_DOM_and_JavaScript

- DOM, XML, Javascript and Ajax: e.g.,
  http://www.w3schools.com/Ajax/ajax_intro.asp