Introduction to XML

- What is XML
- The structure of an XML document
- Parsing XML in Java
  - SAX
  - DOM
  - JDom

What is XML?

- Extensible Markup Language
- A syntax for documents
- A Meta-Markup Language
- A Structural and Semantic language, not a formatting language

XML Applications

- A specific markup language that uses the XML meta-syntax is called an XML application.
- Different XML applications have their own more constricted syntaxes and vocabularies within the broader XML syntax.
- Further syntax can be layered on top of this; e.g. data typing through schemas.

Some XML Applications

- Clinical Trial Data Model for drug trials
- National Library of Medicine (NLM) XML Data Formats for MEDLINE data over FTP replacing ELHILL Unit Record Format (EURF) on magnetic tape
- Health Level Seven XML Patient Record Architecture
- ASTM XML Document Type Definitions (DTDs) for Health Care
- Molecular Dynamics Markup Language (MoDL)
- BIOpolymer Markup Language (BIOML)
- Gene Expression Markup Language (GEML)
- Chemical Markup Language
- Bioinformatic Sequence Markup Language
- Open Financial Exchange
- Human Resources Markup Language
- XML Court Interface
- and many more...

A simple example

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<?xml-stylesheet type="text/css" href="song.css"?>
<!DOCTYPE SONG SYSTEM "song.dtd">
<SONG xmlns="http://www.cafeconleche.org/namespace/song"
      xmlns:xlink="http://www.w3.org/1999/xlink">
  <TITLE>Hot Cop</TITLE>
  <PHOTO
    xlink:type="simple" xlink:show="onLoad" xlink:href="hotcop.jpg"
    ALT="Victor Willis in Cop Outfit" WIDTH="100" HEIGHT="200"/>
  <COMPOSER>Jacques Morali</COMPOSER>
  <COMPOSER>Henri Belolo</COMPOSER>
  <COMPOSER>Victor Willis</COMPOSER>
  <PRODUCER>Jacques Morali</PRODUCER>
  <!-- The publisher is actually Polygram but I needed
       an example of a general entity reference. -->
  <PUBLISHER xlink:type="simple" xlink:href="http://www.amrecords.com/">
    A &amp;amp; M Records
  </PUBLISHER>
  <LENGTH>6:20</LENGTH>
  <YEAR>1978</YEAR>
  <ARTIST>Village People</ARTIST>
</SONG>
<!-- You can tell what album I was
     listening to when I wrote this example -->
```

Markup and Character Data

- **Markup includes:**
  - **Tags**
  - **Entity References**
  - **Comments**
  - **Processing Instructions**
  - **Document Type Declarations**
  - **XML Declaration**
  - **CDATA Section Delimiters**
- **Character data includes everything else**

Markup and Character Data Example

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<?xml-stylesheet type="text/css" href="song.css"?>
<!DOCTYPE SONG SYSTEM "song.dtd">
<SONG xmlns="http://www.cafeconleche.org/namespace/song"
      xmlns:xlink="http://www.w3.org/1999/xlink">
  <TITLE>Hot Cop</TITLE>
  <PHOTO
    xlink:type="simple" xlink:show="onLoad" xlink:href="hotcop.jpg"
    ALT="Victor Willis in Cop Outfit" WIDTH="100" HEIGHT="200"/>
  <COMPOSER>Jacques Morali</COMPOSER>
  <COMPOSER>Henri Belolo</COMPOSER>
  <COMPOSER>Victor Willis</COMPOSER>
  <PRODUCER>Jacques Morali</PRODUCER>
  <!-- The publisher is actually Polygram but I needed
       an example of a general entity reference. -->
  <PUBLISHER xlink:type="simple" xlink:href="http://www.amrecords.com/">
    A &amp;amp; M Records
  </PUBLISHER>
  <LENGTH>6:20</LENGTH>
  <YEAR>1978</YEAR>
  <ARTIST>Village People</ARTIST>
</SONG>
<!-- You can tell what album I was
     listening to when I wrote this example -->
```

Elements and Tags

- Elements are delimited by a start-tag like `<LENGTH>` and a matching end-tag like `</LENGTH>`:

```
<LENGTH>6:20</LENGTH>
```

- Elements contain content which can be text, child elements, or both:

```
<LENGTH>6:20</LENGTH>
<PRODUCER>
        <NAME>
                <GIVEN>Jacques</GIVEN>
                <FAMILY>Morali</FAMILY>
        </NAME>
</PRODUCER>
<PARAGRAPH> The <ARTIST>Village People</ARTIST> were a
popular <GENRE>Disco</GENRE> band in the 1970's
</PARAGRAPH>
```

- The element is the tags plus the content.
- Empty-element tags both start and end an element:

```
<PHOTO/>
```

- This element has no content. It is equivalent to `<PHOTO></PHOTO>`
- Elements can have attributes:

```
<PHOTO xlink:type="simple" xlink:show="onLoad"
xlink:href="hotcop.jpg" ALT="Victor Willis in Cop
Outfit" WIDTH="100" HEIGHT="200"/>
```

Entities

- An XML document is made up of one or more physical storage units called *entities*
- Entity references:
  - Parsed internal general entity references like &
  - Parsed external general entity references
  - Unparsed external general entity references
  - External parameter entity references
  - Internal parameter entity references
- Reading an XML document is not the same thing as reading an XML file
  - The file contains entity references.
  - The document contains the entities' replacement text.
  - When you use a parser to read a document you'll get the text including characters like <. You will not see the entity references.

Parsed Character Data

- Character data left after entity references are replaced with their text
- Given the element

```
<PUBLISHER>A &amp; M Records</PUBLISHER>
```

- The parsed character data is

```
A & M Records
```

CDATA sections

- **Used to include large blocks of text with lots of normally illegal literal characters like < and &, typically XML or HTML.**

```
<p>You can use a default <code>xmlns</code>
attribute to avoid having to add the svg prefix
to all your elements:</p>
<![CDATA[
    <svg xmlns="http://www.w3.org/2000/svg"
width="12cm" height="10cm"> <ellipse    rx="110"
ry="130" /> <rect x="4cm" y="1cm"
width="3cm" height="6cm" /> </svg>
]]>
```

- **CDATA is for human authors, not for programs!**

Comments

- <!-- Before posting this page, I need to double check the number of pelicans in Lousiana in 1970 -->
- Comments are for humans, not programs.

Processing Instructions

- Divided into a target and data for the target
- The target must be an XML name
- The data can have an effectively arbitrary format

```
<?robots index="yes" follow="no"?>
<?xml-stylesheet href="pelicans.css" type="text/css"?>
<?php
     mysql_connect("database.unc.edu", "clerk",
"password");
     $result = mysql("CYNW", "SELECT LastName,    FirstName
     FROM Employees ORDER BY LastName,    FirstName");
     $i = 0;
     while ($i < mysql_numrows ($result)) {
             $fields = mysql_fetch_row($result);
             echo "<person>$fields[1] $fields[0]
             </person>\r\n";
             $i++;
     }
     mysql_close(); ?>
```

- These are for programs

The XML Declaration

- `<?xml version="1.0" encoding="UTF-8" standalone="no"?>`
- Looks like a processing instruction but isn't.
- `version` attribute
  - required
  - always has the value 1.0
- `encoding` attribute
  - UTF-8
  - ISO-8859-1
  - SJIS
  - etc.
- `standalone` attribute
  - yes
  - no

Document Type Declaration

- `<!DOCTYPE SONG SYSTEM "song.dtd">`

Document Type Definition (DTD)

```
<!ELEMENT SONG (TITLE, PHOTO?, COMPOSER+, PRODUCER*,
 PUBLISHER*, LENGTH?, YEAR?, ARTIST+)>

<!ELEMENT TITLE     (#PCDATA)>
<!ELEMENT COMPOSER  (#PCDATA)>
<!ELEMENT PRODUCER  (#PCDATA)>
<!ELEMENT PUBLISHER (#PCDATA)>
<!ELEMENT LENGTH    (#PCDATA)>
<!-- This should be a four digit year like "1999",
     not a two-digit year like "99" -->
<!ELEMENT YEAR   (#PCDATA)>
<!ELEMENT ARTIST (#PCDATA)>
<!ELEMENT PHOTO EMPTY>
<!ATTLIST PHOTO xlink:type (simple) #FIXED "simple"
                xlink:show (onLoad) #FIXED "onLoad"
                xlink:href CDATA #REQUIRED
                ALT CDATA #REQUIRED
                WIDTH NMTOKEN #REQUIRED
                HEIGHT NMTOKEN #REQUIRED
>
<!ATTLIST PUBLISHER xlink:type (simple) #FIXED "simple"
                    xlink:href CDATA #REQUIRED

>
<!ATTLIST SONG xmlns CDATA       #FIXED "http://www.cafeconleche.org/namespace/song"
               xmlns:xlink CDATA #FIXED "http://www.w3.org/1999/xlink"
>
```

XML processing with Java

- SAX
- DOM
- JDom

The SAX2 Process

- Use the factory method
  `XMLReaderFactory.createXMLReader()`
  to retrieve a parser-specific implementation of the XMLReader interface
- Your code registers a `ContentHandler` with the parser
- An `InputSource` feeds the document into the parser
- As the document is read, the parser calls back to the methods of the `ContentHandler` to tell it what it's seeing in the document.

The ContentHandler interface

```
package org.xml.sax;

public interface ContentHandler {
    public void setDocumentLocator(Locator locator);

    public void startDocument() throws SAXException;

    public void endDocument() throws SAXException;

    public void startPrefixMapping(String prefix, String uri) throws SAXException;

    public void endPrefixMapping(String prefix) throws SAXException;

    public void startElement(String namespaceURI, String localName,
     String qualifiedName, Attributes atts) throws SAXException;

    public void endElement(String namespaceURI, String localName,
     String qualifiedName) throws SAXException;

    public void characters(char[] text, int start, int length)  throws SAXException;

    public void ignorableWhitespace(char[] text, int start, int length) throws SAXException;

    public void processingInstruction(String target, String data) throws SAXException;

    public void skippedEntity(String name) throws SAXException;
}
```

Document Object Model

- Defines how XML and HTML documents are represented as objects in programs
- W3C Standard
- Defined in IDL; thus language independent
- HTML as well as XML
- Writing as well as reading
- Covers everything except internal and external DTD subsets
- DOM focuses more on the document; SAX focuses more on the parser.

The DOM Process

- Create a parser (via JAXP `DocumentBuilderFactory`)
- The parser parses the document and returns a DOM `org.w3c.dom.Document` object.
- The entire document is stored in memory.
- DOM methods and interfaces are used to extract data from this object

JDom

- A Pure Java API for reading and writing XML Documents
- A Java-oriented API for reading and writing XML Documents
- A tree-oriented API for reading and writing XML Documents
- A parser independent API for reading and writing XML Documents

Six packages

- `org.jdom`
  - the classes that represent an XML document and its parts
- `org.jdom.input`
  - classes for reading a document into memory
- `org.jdom.output`
  - classes for writing a document onto a stream or other target (e.g. SAX or DOM app)
- `org.jdom.adapters`
  - classes for hooking up to DOM implementations
- `org.jdom.filter`
  - classes to mask parts of tree while navigating
- `org.jdom.transform`
  - XSLT support via TrAX
- `org.jdom.xpath`
  - XPath courtesy of Jaxen