An Introduction to XML

Nancy McCracken,

Ozgur Balsoy

Northeast Parallel Architectures Center at Syracuse University

111 College Place, Syracuse, NY 13244

http://www.npac.syr.edu/projects/webtech/xml

Outline

- Overview of XML and its relationship to HTML and SGML
- XML extensible tags: DTD
- Some simple examples
- XSL and XLL
- XML tools
- Details on writing XML
- Example showing the use of XML to store a simple database

References

- ***XML Complete*, Steven Holzner [McGraw-Hill, 1998], old but good for DOM processing**
- **"XML, Java, and the future of the Web", Jon Bosak, Sun Microsystems, 1997**
- **"Weaving a Better Web", S. Mace, U. Flohr, R. Dobson, T. Graham, *Byte*, March 1998, pp.58-68**
- **NPAC's XML Resources page, http://www.npac.syr.edu/projects/webtech/xml**
- **Sun XML tutorial: java.sun.com/xml/tutorial_intro.html, concentrating on SAX and DOM API's**
- **XML and XSL tutorials: www.xml101.com, and its predecessor: www.refsnesdata.no**

Overview of HTML

- **HTML = Hypertext Markup Language**
  - the lingua franca of the World Wide Web
  - HTML is a simple language well suited for hypertext, multimedia and the display of small and reasonably simple documents
- **HTML 2.0 spec completed in Nov 95**
- **HTML+ and HTML 3.0 never released**
- **HTML 3.2 (Jan 97) added tables, applets, and other capabilities (approximately 70 tags)**
  - this is what most people are familiar with today
- **HTML 4.0 spec released in Dec 97**

Beyond HTML

- **Limitations of HTML:**
  - Extensibility: HTML does not allow users to specify their own tags or attributes in order to parameterize or otherwise semantically qualify their data.
  - Structure: HTML does not support the specification of deep structures needed to represent database schema or object-oriented hierarchies.
  - Validation: HTML does not support the kind of language specification that allows applications to check data for structural validity when it is imported.

What is XML?

- **XML = eXtensible Markup Language**
- **XML is a subset of Standard Generalized Markup Language, but unlike the latter, XML is specifically designed for the web**
- **Specification of W3C: http://www.w3.org/XML**
- **XML 1.0 in February 98, related specifications since then**
- **How XML fits into the new HTML world:**
  - XML describes the logical structure of the document.
  - CSS (Cascading Style Sheets) or other style language describes the visual presentation of the document.
  - The DOM (Document Object Model) allows scripting languages, such as JavaScript to access document objects.
  - DHTML (Dynamic HTML) allows a dynamic presentation of the document.

Logical vs. Visual Design

- **The logical design of a document (content) should be separate from its visual design (presentation)**
- **Separation of logical and visual design**
  - promotes sound typography
  - encourages better writing
  - is more flexible
- **XML can be used to define the logical design, while the XSL (Extensible Style Language) is used to define the visual design.**

What is SGML?

- SGML = Standard Generalized ML
- A SGML document carries with it a grammar called a Document Type Definition (DTD). The DTD defines the tags and the meaning of those tags
- Presentation is governed by a style sheet written in the Document Style Semantics and Specification Language (DSSSL)
- Note that HTML is a fixed SGML application, a hard-wired set of about 70 tags and 50 attributes, and does not need to have a DTD.

SGML Example

- **A simple SGML document with embedded DTD:**

```
<!DOCTYPE DOCUMENT [
  <!ELEMENT DOCUMENT O O (p*,BIGP*)>
  <!ELEMENT p - O (#PCDATA)>
  <!ELEMENT BIGP - O (#PCDATA)>
 ]>
 <DOCUMENT>
  <p>Welcome to
  <BIGP>XML Style!
 </DOCUMENT>
```

SGML Example (cont’d)

- A corresponding DSSSL style sheet:

```
<!DOCTYPE style-sheet PUBLIC "-//James
    Clark//DTD DSSSL Style Sheet//EN">

(root (make simple-page-sequence))

(element p (make paragraph))
(element BIGP (make paragraph
                    font-size: 24pt
                    space-before: 12pt))
```

XML is SGML Lite

- **XML is also an SGML application, but since XML is extensible (XML is also a metalanguage), every XML document must be accompanied by its DTD**
- **XML is a compromise between the non-extensible, limited capabilities of HTML and the full power and complexity of SGML**
- **XML offers "80% of the benefits of SGML for 20% of its complexity"**
  - XML designers tried to leave out all the SGML that would be rarely used on the web
  - Note that XML specification is 30 pages and the SGML specification is 500 pages.
- **XML allows you to define your own tags and to describe nested hierarchies of information.**

XML Design Goals

- 1) XML shall be usable over the Internet
- 2) XML shall support a variety of applications
- 3) XML shall be compatible with SGML
- 4) It shall be easy to write programs that process XML documents
- 5) Optional features in XML shall be kept to the absolute minimum, ideally zero
- 6) XML documents should be human-legible and reasonably clear
- 7) Design of XML should be prepared quickly
- 8) Design of XML shall be formal and concise
- 9) XML documents shall be easy to create
- 10) Terseness in XML markup is of minimal importance

Features of XML

- **The documents are stored in plain text and thus can be transferred and processed anywhere.**
- **Inline-reusability - documents can be composed of many pieces**
- **Unifying principles make it easily acceptable**
  - "everything is a tree"
  - UNICODE for different languages
- **XML documents enable several types of uses**
  - traditional data processing - XML documents can be the data interchange medium
  - document-driven programming
  - archiving

Origins of XML

- First draft of XML spec released by W3C in Nov 96 (four other drafts published in 1997)
- The first XML parser (written in Java) released by Microsoft in July 97
- Microsoft released version 1.8 of its XML parser (which supports XML 1.0) in Jan 98
- W3C finalized the XML 1.0 spec in Feb 98
- First XML-aware beta versions of NC and IE5.0 released in June 98
- Sun announced Java Standard Extension for XML (XML API) in March 99
- W3C working drafts for extensions - 99/00

“Hello World!” in XML

- **An XML document with external DTD:**

```
<?xml version="1.0"?>
<!DOCTYPE greeting SYSTEM "hello.dtd">
<greeting>Hello World!</greeting>
```

- **An XML document with embedded DTD:**

```
<?xml version="1.0"?>
<!DOCTYPE greeting [
  <!ELEMENT greeting (#PCDATA)>
]>
<greeting>Hello World!</greeting>
```

XML and Related Acronyms

- ***Document Type Definition*** **(DTD), which defines the tags and their relationships**
- ***Extensible Style Language*** **(XSL) style sheets, which specify the presentation of the document**
- ***Extensible Link Language*** **(XLL), which defines link-handling details**
- ***Resource Description Framework*** **(RDF), document metadata**
- ***Document Object Model*** **(DOM), API for converting the document to a tree object in your program for processing and updating**
- ***Simple API for XML*** **(SAX), "serial access" protocol, fast-to-execute protocol for processing document on the fly**
- ***XML Namespaces*****, for an environment of multiple sets of XML tags**
- ***XHTML*****, a definition of HTML tags for XML documents (which are then just HTML documents)**
- ***XML schema*****, to offer a more flexible alternative to DTD**

Document Type Definition

- **The DTD specifies the logical structure of the document; it is a formal grammar describing document syntax and semantics**
- **The DTD does *not* describe the physical layout of the document; this is left to the style sheets and the scripts**
- **It is no mean task to write a DTD, so most users will adopt predefined DTDs (or can write an XML document without a DTD).**
- **DTDs can be written in separate files to facilitate re-use.**
- **Content-providers, industries and other groups can collaborate to define sets of tags.**

XML must be “well-formed”

- **For the data contained in an XML document to be parsed correctly, its markup must be well-formed, meaning that properly nested and nonabbreviated starting and ending tags are used.**
  - **This well-formedness provides the encapsulation mechanism allowing designated sections of the data to be accessed programmatically.**
  - **It is important to remember that XML is a markup language, not a programming language**
- ***Scenario #1:* the server offers the XML document without its DTD, the parser does a syntax check, and the DTD follows if the XML document is “well-formed”**
- ***Scenario #2:* the server checks the XML document against its DTD (“validity”) before sending the document to the client**

XML Example

- Another example which could be used for URL exchanges between network capable applications:

```
 <LINK>
<TITLE>XML Recommendation</TITLE>
<URL>
  http://www.w3.org/TR/REC-xml
</URL>
<DESCRIPTION>
  The official XML spec from W3C
</DESCRIPTION>
</LINK>
```

XML Example (cont'd)

- **A document may have many such links:**

```
 <DOCUMENT>
<LINKS>
  <LINK>…</LINK>
  <LINK>…</LINK>
  …
</LINKS>
</DOCUMENT>
```

XML Example (cont'd)

- **Now write a DTD for this document:**

```
<!ELEMENT DOCUMENT (LINKS)>
<!ELEMENT LINKS (LINK)*>
<!ELEMENT LINK
  (TITLE,URL,DESCRIPTION)>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT URL (#PCDATA)>
<!ELEMENT DESCRIPTION (#PCDATA)>
```

- **PCDATA stands for "parsed character data"**

XML Example (cont’d)

- Store the DTD in a file (links.dtd) and write an XML document based on this DTD:

```
 <?XML version="1.0"?>
<!DOCTYPE DOCUMENT SYSTEM "links.dtd">
<DOCUMENT>
  <LINKS>
    <LINK>…</LINK>
    <LINK>…</LINK>
    …
  </LINKS>
</DOCUMENT>
```

- Note that you need an XML compiler to generate regular HTML in Netscape browsers - Internet Explorer 5.0 has a compiler built in.

XML Prolog and instructions

- **Every XML file starts with the prolog, giving information about the document. The minimal prolog identifies it as an xml document**

  <? version="1.0"?>

- **The prolog may also include the encoding and whether it is a standalone document:**

  <?xml version="1.0"

  encoding="ISO-8859-1" standalone="yes"?>

- **If it is not standalone, it may specifiy external "entities" which may be named in the document or an external DTD**
- **An XML file may also contain processing instructions for the application processing the document:**

  <?target instructions?>

  where target is the name of the application.

XML tag structure

- **In XML terminology, a pair of start and end tags is an element.**
- **XML documents allow only one root element.**
- **XML documents must have a strict hierarchical structure.**
  - All start tags must have an end tag.
  - Any element must be properly nested within another.
    <LI> XML requires <B><I>proper nesting</I></B>.</LI>
- **Empty tags are allowed as elements in XML documents.**
  - An empty tag is a start and end tag together and is identified by a trailing / after the tag name.
    <LI>XML requires empty tags, such as <BR/>.</LI>
  - A start tag and end tag with nothing in-between can also be considered an empty tag.
    <IMG SRC="face.gif"></IMG>

XML tag details

- **All attribute values must be within single or double quotes. <FONT COLOR="#FF00CC"> quoted attribute </FONT>**

Syntactic details

- **XML tags are case-sensitive. (<H1> is not the same as <h1>.**
- **White space in the data between tags is relevant. But within the markup itself and within quoted attribute values, white space is normalized (removed).**
- **XML allows you to specify different character set encodings. <?xml version=‘1.0’ encoding=‘UTF-8’ ?>**
- **Predefined entities:**

– &lt;	replaced by	<
– &amp;		&
– &gt;		>
– &apos;		‘
– &quot;		“

Document Type Definition

- **An optional, but powerful feature of XML that provides a formal set of rules to define a document structure**
- **Defines the elements that may be used, and dictates where they may be applied in relation to each other; therefore specifies the document hierarchy and granularity**
- **Comprises a set of declarations that define a document structure tree**
- **Declarations stored either at the top of each document that must conform to the rules, or alternatively, and more usually, in separate data files, referred by a special instruction at the top of each document.**

Document Type Definition

- **Each DTD element must either be a container element, or be empty (a place holder). Container elements may contain text, child elements, or a mixture of both.**
- **DTD also specifies the names of attributes, and dictates which elements they may appear in. For each attribute it specifies whether it is optional or required.**
- **DTD tree describing a book as containing a number of Chapter elements, with each chapter containing either a number of Paragraph elements, or a single Sections element**
- **A particular document tree has a node for each actual chapter and paragraph present, and may omit some of the optional elements**

DTD definitions

- **A DTD allows you to create new tags by writing grammar rules which the tags must obey. The rules specify which tags and attributes are valid and their context.**
- **A DTD element declaration looks like:**
  <!ELEMENT person(name, email*)>
  – ELEMENT is the type
  – person is the element declaration
  – (name,email*) is the element content model
  – name and email are the children of person and define the hierarchy of the document.
  – Note that this is called a grammar rule because it could have been written in BNF: person ::= (name, email*)

Document Type Definition

- **Each declaration must follow markup format `<!...>`, and can only use the one of the following keywords:**
  - ELEMENT (tag definition)
  - ATTLIST (attribute definitions)
  - ENTITY (entity definition)
  - NOTATION (data type notation definition)
- **Declarations are grouped within a DTD**

```
<!DOCTYPE MYDTD [
<!-- The MYDTD appears here -->
<!……>
]>
```

Document Type Definition

- **Declarations stored externally and shared by different documents linked as:**

```
<!DOCTYPE MYDTD SYSTEM “EXTRNL.DTD” [
<!-- Some of MYDTD appears here -->
<!……>
]>
```

Element Declarations

- **Keyword ELEMENT Introduces a new element** **`<!ELEMENT title ………>`**
- **Element name must begin with a letter, and may additionally contain digits and some punctuations, i.e. '.', '-', '_', and ':'**
- **If an element can hold no child elements, and also no text, then it is known as empty element and denoted by EMPTY**
- **An element declared to have a content of ANY may contain all of the other elements declared in the DTD**

-
  ```
  <!ELEMENT p ANY>
  <!ELEMENT image EMPTY>
  ```
- **Empty element usage: `<image></image>` or `<image/>`**

Element Declarations

- **A model group is used to define an element that has mixed content or element content.**
- **A model group is bounded by brackets, and contains at least one token.**
- **When a model group contains more than one content token, the child elements are controlled using two logical connector operators; sequence connector ‘,’, and choice connector ‘|’**
- **<!ELEMENT element1 (a, b, c)> indicates a is followed by element b, which in turn is followed by c.**
- **<!ELEMENT element2 (a | b | c)> indicates either one can be selected.**
- **Combinations are possible: (a,b,(c|d)), or ((a,b,c) | d)**

Element Declarations

- **Quantity indicators can also be used.**
  - '?' indicates an element is optional or cannot repeat
  - '+' indicates an element is required and may repeat
  - '*' indicates an element is optional, and also repeatable

- **Document text is indicated by the keyword PCDATA (Parsable Character Data)**

```
<!ELEMENT emph (#PCDATA|sub|super)*>
<!ELEMENT sub (#PCDATA)>
<!ELEMENT super (#PCDATA)>
<emph>H<sub>2</sub>0 is water.</emph>
```

Attributes

- **The rules for attribute declarations follow a similar structure to elements.**
  <!ATTLIST person gender (male|female)#IMPLIED >
  - ATTLIST is the type
  - person is the element
  - gender is the attribute declaration
  - (male|female)#IMPLIED is the attribute definition
- **The keywords following an attribute definition can be**
  - #IMPLIED attribute is optional
  - “unknown” string in quotes is the default attribute
  - #REQUIRED attribute is required
  - #FIXED if attribute is present, it is assigned a fixed value

Attribute Types

- **Enumerated types**
  (male|female|unknown)
- **CDATA type is character data - may include markup**
  <!ATTLIST form method CDATA #FIXED ‘POST’>
- **Tokenized types include the following tokens with special meanings:**
  – ID requires that the attribute have a unique value
  – IDREF is rquired to match an ID attribute within the same document

Entities

- **The DTD of an XML document can contain entity declarations. These are like constants in other languages.**
  - An entity declaration specifies replacement text for the entity including some macro-preprocessing capability.

    ```
    <!ENTITY % pub    “&#xc9;ditions Gallimard”>
    <!ENTITY     rights “All rights reserved”>
    <!ENTITY     book   “La Pest: Albert Camus, &#sA9; 1947
                                        %pub; . &rights;”>
    ```
  - This entity would have replacement text for book:
    La Peste: Albert Camus, c 1947 Editions Gallimard.
    &rights;
    where c would be copyright symbol, and E has accent mark.

XML Example - the DTD

- Create a DTD file for an address book named "ab.dtd"
- <!ELEMENT addressBook (person)+>
- <!ELEMENT person (name, email*, link?) >
- <!ATTLIST person id ID #REQUIRED >
- <!ATTLIST person gender (male|female)#IMPLIED >
- <!ELEMENT name (#PCDATA|(family,given))>
- <!ELEMENT family   (#PCDATA)>
- <!ELEMENT given   (#PCDATA)>
- <!ELEMENT email   (#PCDATA)>
- <!ELEMENT link   EMPTY >
  <!ATTLIST link manager IDREF #IMPLIED
                      subordinates IDREFS #IMPLIED >

XML Example - the XML document

-
  ```
  <?xml version="1.0"?>
  <!DOCTYPE addressBook SYSTEM ”ab.dtd">
  <addressBook>
     <person id=“B.WALLACE” gender=“male”>
        <name>
           <family>Wallace</family> <given>Bob</given>
        </name>
        <email>bwallace@megacorp.com</email>
        <link> manager=“C.TUTTLE”/>
     </person>
     <person id=“C.TUTTLE” gender=“femail”>
        <name>
           <family>Tuttle</family> <given>Claire </given
        </name>
        <email>ctuttle@megacorp.com</email>
        <link subordinates=“B.WALLACE”/>
     </person>
  </addressBook>
  ```

Additional XML/DTD Topics

- **Encoding, internalization and languages**
- **Entities: Internal and External, the constants and macro-processing of XML**
- **Processing instructions - allow documents to contain instructions for applications**
- **W3C is considering draft proposals for schemas that would allow you an easier syntax to write DTD grammar rules.**

Extensible Style Language

- **XSL is to XML as Cascading Style Sheets (CSS) are to HTML**
- **Like a CSS, an XSL style sheet describes the presentation of the XML document**
- **Advanced layout features of XSL include: rotated text, multiple columns, and independent regions**
- **Development of XSL lags behind XML - currently a Working Draft of the W3C.**
- **Content of XML documents is intended to be easily read by both people and software, but raw XML data is not suitable for viewing by people who are not interested in structure.**
- **To publish information held in XML format it is necessary to replace the tags with appropriate text styles.**

Document Formatting (XSL)

- **A style rule is used to assign a style to a particular XML element.**
- **It is possible to embed a style rule within an attribute: <p xsl::font-size="9pt" xsl::color="blue">A blue, 9pt paragraph.</p>**
- **The problem with this approach is that the rule must be repeated each time the element is used. The style sheets are developed to solve this problem so that rules are grouped together, and can be shared by multiple files.**

XSL Processing Overview

- **In general, XSL first specifies how to process the source tree to get a result tree.**
- **The pattern of a template rule is matched with the source tree and replaces it with the template.**
- **The result tree is then processed with formatting to achieve a document suitable for display, printing, speech or other media.**
- **XSL and XML may have their own namespaces for rules.**

Document Formatting (XSL)

- **General Structure**

```
<?xml-stylesheet href=“article.stl”
type=“text/xsl”?>
<!DOCTYPE article ………>
<article>
…
</article>
```

- **Each style sheet has a root element called xsl. It may contain any combination of Rule and Stylerule elements.**
- **<xsl>**
  <rule>…</rule><rule>…</rule><stylerule>…</stylerule>
  </xsl>

Applying Styles to XML Elements

- **The simplest XSL templates consist of HTML tags and styles with references to the tags of XML document pieces that should be formatted. For example, putting XML elements tagged TITLE and ARTIST as elements inside a table definition:**

```
<td><xsl:value-of select="TITLE"/></td>
<td><xsl:value-of select="ARTIST"/></td>
```

- **This could be repeated over many elements of the XML document:**

```
<xsl:for-each select="CATALOG/CD”>
<tr>
<td><xsl:value-of select="TITLE"/></td>
<td><xsl:value-of select="ARTIST"/></td>
</tr>
```

XSL Control Structures

- Styles can be conditionally applied with if:

```
<xsl:if match=".[ARTIST='Bob Dylan']">
  ... some output …
 </xsl:if>
```

- Or with conditional choose:

```
<xsl:choose>
<xsl:when match=".[ARTIST='Bob Dylan']">
... some code ...
 </xsl:when>
 <xsl:otherwise>
     ... some code ....
</xsl:otherwise>
 </xsl:choose>
```

Extensible Link Language

- **XLL supports simple links (like HTML) plus:**
  - Location-independent naming
  - Bidirectional links
  - External links
  - N-ary hyperlinks
  - Aggregate links and link types
  - Transclusion
- **XLL components: Xlink and XPointer**

Uses of XML

- **XML can be used as Electronic Data Interchange (EDI):**
  – Applications requiring client to mediate between two heterogeneous databases
  – Applications that attempt to redistribute processing load from server to client
  – Applications requiring client to present multiple views of same web document
  – Applications that perform or require detailed, domain-specific search results

DOM

- **The Document Object Model is a specification of the representation of XML and other documents from the W3C consortium.**
- **It is a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents.**
- **Document Object Model Level 2, Candidate Recommendation, March 7, 2000.**
- **DOM consists of a set of core interfaces and also specialized interfaces dedicated to XML, HTML, an abstract view, generic stylesheets, Cascading Style Sheets, Events, traversing the document structure, and a Range object.**
- **There are language bindings for several languages, including JavaScript and Java, which has the package org.w3c.dom**

SAX

- The Simple API for XML (SAX), aka Serial API for XML, is a specification arrived at by a group related to W3C, the XML Dev mailing list.
- While the DOM is the specification of an entire document, the SAX API specifies how to process the document in a serial order by giving processing methods (aka the callback functions) for each node or element of the document.

Java packages for XML

- **javax.xml.parsers contains**
  - SAXParserFactory gives you a SAX parser
  - DocumentBuilderFactory gives you a DocumentBuilder,
  - The Document Builder can process XML to produce a Document object or create an empty Document object..
- **The factory API's allow you to plug-in various vendor parsers at run-time without changing the code. For example, you can load**
  - com.ibm.xml.parser (IBM's parser)
  - com.ms.xml.parser (Microsoft's parser)
  - com.sun.xml.parser or com.sun.xml.ValidatingParser
    - parsers can validate to various degrees depending on whether they process all external entities

Starting the DOM process in Java

- **Java X API:**
  import javax.xml.parsers.DocumentBuilderFactory;
  import javax.xml.parsers.FactoryConfigurationError;
  import javax.xml.parsers.ParserConfigurationException;
  import javax.xml.parsers.DocumentBuilder;
- **Parsing errors come from the SAX implementation:**
  import org.xml.sax.SAXException;
  import org.xml.sax.SAXParseException;
- **The W3C definition for a DOM and DOM exceptions:**
  import org.w3c.dom.Document;
  import org.w3c.dom.DOMException;

Parsing the XML

- Get the parser

  ```
  DocumentBuilderFactory factory =
                   DocumentBuilderFactory.newInstance();
  DocumentBuilder builder =
                       factory.newDocumentBuilder();
  ```

- The default is to get a non-validating parser. Add configurations

  ```
        factory.setValidating(true);
        factory.setNamespaceAware(true);
  ```

- Run the parser:

  ```
  Document document = builder.parse( **XML file** );
  ```

Document

- **The Document interface is actually a child of Node and inherits most of its classes for processing the document from there.**
- **The Document interface itself has methods to create various parts of the document:**
  - createElement, createAttribute, createCDatasection, createTextnode, . . .
- **and other methods such as**
  - getElementsByTagName, which returns a NodeList of all the tags in order as you traverse the document.

The Node interface

- **Has methods to give information about the node:**
  - getNodeValue, getNodeType, getNodeName, getAttributes
- **It primarily has methods to traverse the tree:**
  - getParentNode
  - hasChildNodes (returns Boolean)
  - getChildNodes (returns a NodeList)
  - getFirstChild, getLastChild, getNextSibling, getPreviousSibling
- **And also methods to update the tree:**
  - insertBefore, appendChild, removeChild, replaceChild

Recursively Process Tree

- **Write a method that accepts a Node (and other args as needed):**

```
   void doTree(Node node, . . .)
  { if (! node.hasChildNodes())
      { process all leaf nodes according to node.getNodeType
       }
     else
       { NodeList l = node.getChildNodes();
          for (int i = 0; i<l.getLength(); i++)
            {  doTree ( item(i), . . . ); }
       }
    }
```