# Lecture Notes 13
# Introduction to XML

**Anand Tripathi**

**CSci 4131**

## Internet Programming

# Reference Materials

- **Book -- Building Web services with Java making sense of XML, SOAP, WSDL, and UDDI**
  - Please see Chapter 2
    - **This book is available online through U of M Library.**
    - **Check Course Webpage**
- **Deitels book (4th edition –available online)**
  - **See Chapter 14**
  - **Or, Chapter 15 of the 5th edition**

# Basic Concepts

- XML – eXtensible Markup Language - is a meta language for defining any desired markup language for an application domain.
  - XML has no tags of its own,
- An XML-based markup language can be used to store and communicate information.
- Using XML, new tags and structures can be defined to describe structure of the information to be stored and communicated.
- Newly created tags must adhere to the rules of XML specification.

# Basic Concepts

- Information structured according to XML rules can be processed by computer programs in a meaningful way.
  - It can be parsed and manipulated according to given set of rules.
- The structures used for information organization can be extended, and structures developed by different organizations can be combined.
- An XML code is both data and its description.
- Information can be structured between disparate systems and organizations.

# Basic Concepts

- *Elements* are the basic building blocks of an XML document.
- An element contains a *tag*, and possibly some *attributes*.
- A element may contain some *child elements*.

- A valid XML document must conform to certain structuring rules. The rules are defined either by a DTD (Document Type Definition) or an XML Schema.

# Example 1

```
<?xml version="1.0" encoding="UTF-8">

<person>
    <name>
        <firstname>    George        </firstname>
        <lastname>     Washington    </lastname>
    </name>
    <height unit = "inches" >    75    </height>
</person>
```

# Document Prolog

**<?xml** version="1.0" encoding="UTF-8" **?>**

- The above is a processing instruction. A document can have one or more processing instructions.
- A processing instruction is contained in brackets:

  <?PITarget … ?>

  where the PITarget is a keyword for the document processing applications.

# Document Prolog

Document prolog has three functions:

1. Indicate that this is an XML document.
2. Include some comments about the documents
   - <!-- This is a comment -->
   - Warning: No nesting of comments. Do not use double hyphen within a comment
3. Include some meta information about the contents of the document.

# Root element, and  Element nesting

- Every document must have one and only one root element.
- In this example <person> … </person> is the root element.
- All elements have an opening tag and a closing tag.
- An element can have other nested elements:
  - Element person contains two elements name and height.
  - Element name contains firstname and lastname.

# Attributes and Values

- An element may contain one or more attributes.
- The typical use of an attribute is to represent some meta data for the data contained in the element.
- It defines properties or purpose of the content.
- In the previous example, element height contains attribute named units.

<height unit = “inches”> 75 </height>

*attribute specifying that 75 is in inches*

- A commonly used attribute is “id” as seen in XHTML, which must have a unique value in the document.

# Special Symbols

- Five special symbols:
  1. &amp to represent &
  2. &lt to represent <
  3. &gt to represent >
  4. &quot for the double quote mark “
  5. &apos for the single quote mark ‘

# CDATA to include verbatim text

- To include some text that should not be parsed by an application, use the following construct:

```
<![CDATA[
   … some text data that will not be parsed
]]>
```

See an example on pages 43 and 44 of the Web Services book.

Warning: Do not nest CDATA sections

```
<example-to-show>
  &lt;?xml version=&quot;1.0&quot;?&gt;
  &lt;rootElement&gt;
    &lt;childElement id=&quot;1&quot;&gt;
      The man said: &quot;Hello, there!&quot;.
    &lt;/childElement&gt;
  &lt;/rootElement&gt;
</example-to-show>
```

Above using CDATA

```
<example-to-show><![CDATA[
    <?xml version="1.0"?>
      <rootElement>
          <childElement id="1">
              The man said: "Hello, there!".
          </childElement>
       </rootElement>
]]></example-to-show>
```

# Schemas: DTD and XML Schema

- An XML document must be structured according some specific rules to be considered as a valid document.
- The structuring rule are given by a schema by a DTD or XML Schema
  - DTD (Document Type Definition) is an older way of specifying such structuring rules.
  - XML Schema is now a more recognized and current method, giving several advantages over DTD.

# Advantages of DTDs

- They used to be widely used and supported by most XML parsers.
- They are compact and easy to comprehend.
- They can be defined inline for quick development.

# Disadvantages of DTD

- DTD is not an XML document , so a different set of processing tools are needed for it.
- More importantly, it does not support integration of different namespaces:
  - It is not possible to integrate different DTDs if they use the same element names for different structures.
- DTD does not provide data typing, thus reducing the validation capabilities.
- Not possible to define how many child elements may nest within a parent element.

# XML Schema

- They are themselves XML documents, based on XML structuring rules.
  - This makes it easier to process them using the XML framework.

- Most importantly, they support integration of different namespaces in a document, thus avoiding any problems with the same element names used differently by different applications.

# Examples from Web Services Book

Refer to Chapter 2 on “XML Primer” in the book “Building Web Services with Java”

It develops an XML based framework for handling purchase orders for a company dealing with skateboards.

# Example from Web Services Book

```
<?xml version="1.0" encoding="UTF-8"?>
<! -- This is being used for illustration in CSci 5131 -->
<po id="43871" submitted="2004-01-05" customerId="73852">
   <billTo>
        <company>The Skateboard Warehouse</company>
        <street>One Warehouse Park</street>
         <street>Building 17</street>
        <city>Boston</city>
        <state>MA</state>
        <postalCode>01775</postalCode>
    </billTo>
```

```
    <shipTo>
      <company>The Skateboard    Warehouse</company>
      <street>One Warehouse Park</street>
      <street>Building 17</street>
       <city>Boston</city>
       <state>MA</state>
       <postalCode>01775</postalCode>
    </shipTo>
    <order>
          <item sku="318-BP" quantity="5">
              <description>Skateboard backpack; five pockets </description>
          </item>
          <item sku="947-TI" quantity="12">
              <description>Street-style titanium skateboard.</description>
           </item>
          <item sku="008-PR" quantity="1000">        </item>
    </order>
</po>
```

# Conceptual Structure

- This structure of a purchase order could be represented by a Java class as follows:

```
class PO {
    int id;
    Date submitted;
    int customerId;
    Address billTo;
    Address shipTo;
    Item   order[];
}
```

# Refining the XML Structure

- The information contained in the <billTo> and <shipTo> elements are duplicates.
- Such duplicate information structures can be represented more concisely using the ID and IDREF mechanisms of XML.

A unique ID attribute is associated with the billTo element.

The content of the shipTo element is defined as a reference to the content of the billTo element using that id as reference.

```
<po id="43871" submitted="2004-01-05" customerId="73852">
   <billTo id="addr-1">
        <company>The Skateboard Warehouse</company>
        <street>One Warehouse Park</street>
        <street>Building 17</street>
        <city>Boston</city> <state>MA</state>
        <postalCode>01775</postalCode>
   </billTo>
   <shipTo href="addr-1"/>
    ...
  </po>
```

# XML Namespaces

- Many applications need XML documents from different sources to be combined.
- This requires that the element names and ids should not cause any conflicts when they are not unique.
- To void such problems, each application defines its own namespaces, and documents are required to identify the namespace to which a particular element name belongs.

# Example of Name Conflict

- An application dealing with people's postal addresses may define an address structure as:

```
<address>
    <name> John Smith </name>
    <street> 200 Union Street </street>
    <city>   Minneapolis> </city>
</address>
```

- An application dealing with Internet addresses may define an address structure as:

```
<address>
    <internet-address> 128.101.38.120 </internet-address>
    <domain-name>  kepler.cs.umn.edu </domain-name>
</address>
```

# Example of Name Conflict

- Consider an XML based messaging application that wants to include a purchase order as an attachment in a message.
- A message is defined with the following kind of structure:

```
<message from="..." to="..." sent="...">
    <text> This is the text of the message. </text>
    <!-- A message can have attachments -->
    <attachment>
        <description>Brief description of the attachment. </description>
        <item>
            <!-- XML of attachment  such as PurchaseOrder goes here -->
        </item>
    </attachment>
</message>
```

```
<message from="bj@bjskates.com" to="orders@skatestown.com" sent="2004-01-
    05">
    <text> Hi, here is what I need this time. Thx, BJ. </text>
    <attachment>
      <description> Your new purchase order </description>
    <item>
        <po id="43871" submitted="2004-01-05" customerId="73852">
         <billTo id="addr-1">
              <company>The Skateboard Warehouse</company>
              <street>One Warehouse Park</street>
             <city>Boston</city> <state>MA</state>     <
              postalCode>01775</postalCode>
          </billTo>
          <shipTo href="addr-1"/>
          <order>
          <item sku="318-BP" quantity="5">
              <description> Skateboard backpack; five pockets </description>
          </item>
        rest of the purchase order part follows here
```

# Problems

1. In the previous document it is not clear which elements are defined by the messaging application and which elements are defined by the purchase order application.
2. There are several element tags which are common in the two applications, but they have different structure or meaning:
   - description and item tags are defined by both the messaging system as well as the purchase order

# Declaration of Namespaces

- In this example, use of two namespace are declared to be used in this document.
- These are named “msg” and “po” in the document.
- Each element name is prefixed with the namespace to which it belongs.

```
<msg:message
        from="bj@bjskates.com"
        to="orders@skatestown.com"
        sent="2004-01-05"
        xmlns:msg="http://www.xcommercemsg.com/ns/message"
        xmlns:po="http://www.skatestown.com/ns/po“
 >
    <msg:text>
         Hi, here is what I need this time. Thx, BJ.
   </msg:text>
```

```
<msg:attachment>
    <msg:description>The PO</msg:description>
     <msg:item>
        <po:po id="43871" submitted="2004-01-05" customerId="73852">
            <po:billTo id="addr-1">
                <po:company>The Skateboard Warehouse</po:company>
                <po:street>One Warehouse Park</po:street>
                <po:street>Building 17</po:street>
                 <po:city>Boston</po:city> <po:state>MA</po:state>
                 <po:postalCode>01775</po:postalCode>
            </po:billTo>
            <po:shipTo href="addr-1"/>
            <po:order>
               <po:item sku="318-BP" quantity="5">
                      <po:description> Skateboard backpack; five pockets
                      </po:description>
               </po:item> </po:order> </po:po>
</msg:item> </msg:attachment> </msg:message>
```

# Default Namespace

```
<message from="bj@bjskates.com" to="orders@skatestown.com"
    sent="2004-01-05"
    xmlns ="http://www.xcommercemsg.com/ns/message"
    xmlns:po="http://www.skatestown.com/ns/po“ >
   <text> Hi, here is what I need this time. Thx, BJ. </text>
   <attachment>
     <description>The PO</description>
       <item>
         <po:po id="43871" submitted="2004-01-05“ customerId="73852">
           ...
         </po:po>
       </item>
   </attachment>
</message>
```

# Nested Namespace Defaulting

```
<message from="bj@bjskates.com" to="orders@skatestown.com" sent="2004-01-05"
    xmlns="http://www.xcommercemsg.com/ns/message“  >
   <text> Hi, here is what I need this time. Thx, BJ. </text>
   <attachment>
   <description>The PO</description>
     <item>
      <po:po id="43871" submitted="2004-01-05" customerId="73852"
           xmlns:po="http://www.skatestown.com/ns/po">
           <billTo id="addr-1"> ...
           </billTo>
          <shipTo href="addr-1"/>
           <order> ... </order>
     </po:po>
    </item>
   </attachment>
</message>
```

# Prefixing an Attribute with Namespace

- This example shows that one can add an attribute defined in another namespace to an element.

- In this example and attribute name “priority” defined in a different namespace is added to a specific item in a purchase order.

```
<message from="bj@bjskates.com" to="orders@skatestown.com" sent="2004-01-05"
     xmlns="http://www.xcommercemsg.com/ns/message">
   <text> Hi, here is what I need this time. Thx, BJ. </text>
   <attachment>
    <description>The PO</description>
    <item>
       <po:po id="43871" submitted="2004-01-05" customerId="73852"
        xmlns:po="http://www.skatestown.com/ns/po"
        xmlns:p="http://www.skatestown.com/ns/priority“  >
        ...
        <po:order>
          <po:item sku="318-BP" quantity="5" p:priority="high">
            <po:description>
              Skateboard backpack; five pockets
            </po:description>
          </po:item>
          <po:item sku="947-TI" quantity="12">
            <po:description>
              Street-style titanium skateboard.
            </po:description>
           </po:item>
            <po:item sku="008-PR" quantity="1000" p:priority="low“  />
       </po:order> </po:po>
     </item>
  </attachment>
</message>
```

# Namespaces

- All examples use URIs to identify namespace.
- Is there any resource corresponding to these URIs?
  - The answer is NO!
  - These URI simply to represent a unique symbolic name.
- We still need some mechanisms to indicate what can be the valid elements and attributes in a namespace?
  - We also need to show the valid structural relationships between these elements, such as nesting of the elements.
  - This is achieved by defining an XML Schema and giving its location to the an application making use of that namespace.

# Associating Schemas with Namespace

```
<?xml version="1.0" encoding="UTF-8"?>
<po:po xmlns:po="http://www.skatestown.com/ns/po"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation = "http://www.skatestown.com/ns/po
            http://www.skatestown.com/schema/po.xsd"
  id="43871"
  submitted="2004-01-05"
  customerId="73852">
  ...
</po:po>
```