# Contents

| Chapter 1 - Course Introduction | ................................................................. | 7 |
| Course Objectives | ................................................................. | 8 |
| Course Overview | ................................................................. | 10 |
| Using the Workbook | ................................................................. | 11 |
| Suggested References | ................................................................. | 12 |

| Chapter 2 - Getting Started with XML | ................................................................. | 15 |
| Data and Document Structure | ................................................................. | 16 |
| XML | ................................................................. | 18 |
| Well-Formed XML | ................................................................. | 20 |
| Valid vs. Well-Formed XML | ................................................................. | 22 |
| XML Schema | ................................................................. | 24 |
| Presentation Style | ................................................................. | 26 |
| XSL-FO and XSLT | ................................................................. | 28 |
| Using XML | ................................................................. | 30 |
| Labs | ................................................................. | 32 |

| Chapter 3 - Writing Well-Formed XML | ................................................................. | 35 |
| XML Fundamentals | ................................................................. | 36 |
| Tag Attributes | ................................................................. | 38 |
| Naming Rules | ................................................................. | 40 |
| Empty and Non-Empty Elements | ................................................................. | 42 |
| Nesting and Hierarchy of Tags | ................................................................. | 44 |
| Processing Instructions and the XML Declaration | ................................................................. | 46 |
| Other XML Constructs | ................................................................. | 48 |
| Entity and Character References | ................................................................. | 50 |
| Labs | ................................................................. | 52 |
# Introduction to XML

## Chapter 4 - Namespaces

- Why Namespaces? ................................................................. 56  
- Namespace Prefixes and Declaration ........................................ 58  
- Multiple Namespace Declarations ........................................ 60  
- Declaring Namespaces in the Root Element .......................... 62  
- Default Namespaces ............................................................ 64  
- Labs .................................................................................. 66

## Chapter 5 - Validating XML with XML Schemas

- Schema Overview ............................................................... 70  
- Associating XML with a Schema ......................................... 72  
- Simple and Built-in Types .................................................... 74  
- Complex Types ................................................................. 76  
- Element Declarations ......................................................... 78  
- Attribute Declarations ....................................................... 80  
- Choices ............................................................................... 82  
- Named Types and Anonymous Types ................................. 84  
- Labs .................................................................................. 86

## Chapter 6 - Introduction to XSLT

- Stylesheet, Source, and Result .............................................. 90  
- XSLT Processors ............................................................... 92  
- Processor Implementations .................................................. 94  
- XPath Basics ....................................................................... 96  
- `xsl:stylesheet` ............................................................... 98  
- `xsl:template` ................................................................. 100  
- `xsl:value-of` ................................................................. 102  
- `xsl:apply-templates` ....................................................... 104  
- `xsl:output` ................................................................. 106  
- Labs ................................................................................. 108

## Chapter 7 - XPath Nodetypes

- XPath Expressions .............................................................. 112  
- XPath Context ................................................................. 114  
- XPath Location Steps ......................................................... 116  
- Element and Root Nodes ...................................................... 118  
- Text and Attribute Nodes ................................................... 120  
- Comment and Processing Instruction Nodes .......................... 122
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Namespace Nodes</td>
<td>124</td>
</tr>
<tr>
<td>Wildcards</td>
<td>126</td>
</tr>
<tr>
<td>whitespace</td>
<td>128</td>
</tr>
<tr>
<td>Default Template Rules</td>
<td>130</td>
</tr>
<tr>
<td>Labs</td>
<td>132</td>
</tr>
<tr>
<td>Chapter 8 - XPath Axes and Predicates</td>
<td>135</td>
</tr>
<tr>
<td>Location Paths and Location Steps</td>
<td>136</td>
</tr>
<tr>
<td>Peer Axis Types</td>
<td>138</td>
</tr>
<tr>
<td>More Peer Axis Types</td>
<td>140</td>
</tr>
<tr>
<td>Descendant Axis Types</td>
<td>142</td>
</tr>
<tr>
<td>Ancestor Axis Types</td>
<td>144</td>
</tr>
<tr>
<td>Node Tests</td>
<td>146</td>
</tr>
<tr>
<td>Predicates</td>
<td>148</td>
</tr>
<tr>
<td>Functions</td>
<td>150</td>
</tr>
<tr>
<td>Labs</td>
<td>152</td>
</tr>
<tr>
<td>Chapter 9 - XSLT Flow Control</td>
<td>155</td>
</tr>
<tr>
<td>xsl:if</td>
<td>156</td>
</tr>
<tr>
<td>xsl:choose</td>
<td>158</td>
</tr>
<tr>
<td>xsl:for-each</td>
<td>160</td>
</tr>
<tr>
<td>xsl:sort</td>
<td>162</td>
</tr>
<tr>
<td>Named Templates</td>
<td>164</td>
</tr>
<tr>
<td>Mode</td>
<td>166</td>
</tr>
<tr>
<td>Labs</td>
<td>168</td>
</tr>
<tr>
<td>Chapter 10 - XML in Applications</td>
<td>171</td>
</tr>
<tr>
<td>Reasons and Places for Using XML</td>
<td>172</td>
</tr>
<tr>
<td>DOM Parsers</td>
<td>174</td>
</tr>
<tr>
<td>SAX Parsers</td>
<td>176</td>
</tr>
<tr>
<td>Web Services</td>
<td>178</td>
</tr>
<tr>
<td>Appendix A - Using XML Schema with Namespaces</td>
<td>181</td>
</tr>
<tr>
<td>Qualified and Unqualified XML</td>
<td>182</td>
</tr>
<tr>
<td>Associating Qualified XML with a Schema</td>
<td>184</td>
</tr>
<tr>
<td>Associating a Schema with a Namespace</td>
<td>186</td>
</tr>
<tr>
<td>Controlling Element and Attribute Qualification</td>
<td>188</td>
</tr>
<tr>
<td>Merging Schema with the Same Namespace</td>
<td>190</td>
</tr>
</tbody>
</table>

© 2013 ITCourseware, LLC

Rev 6.1.1

Page v
Course Objectives

- Explain what XML is, and how it is used in application and document development.
- Write well-formed documents that conform to XML's basic rules of syntax.
- Use XML namespaces to distinguish between XML tags.
- Validate XML documents with XML Schemas.
- Transform an XML document into an HTML document using XSLT.
- Use XPath to navigate a document tree.
- Explain how programs can use DOM and SAX to parse XML documents.
Course Overview

* **Audience:** Application developers, web developers and administrators, and XML authors.

* **Prerequisites:** *HTML*. Familiarity with web and data processing concepts. Programming experience is helpful, but not necessary.

* **Classroom Environment:**
  - Workstation per student, with a text editor and an XML-compliant browser.
Chapter 1

Using the Workbook

This workbook design is based on a page-pair, consisting of a Topic page and a Support page. When you lay the workbook open flat, the Topic page is on the left and the Support page is on the right. The Topic page contains the points to be discussed in class. The Support page has code examples, diagrams, screen shots and additional information. Hands On sections provide opportunities for practical application of key concepts. Try It and Investigate sections help direct individual discovery.

In addition, there is an index for quick lookup. Printed lab solutions are in the back of the book as well as online if you need a little help.

The Topic page provides the main topics for classroom discussion.

The Support page has additional information, examples, and suggestions.

Code examples are in a fixed font and shaded. The online file name is listed above the shaded area.

Topics are organized into first (♦), second (☼), and third (★) level points.

Callout boxes point out important parts of the example code.

Pages are numbered sequentially throughout the book, making lookup easy.

Screen shots show examples of what you should see in class.
Suggested References


*http://www.xml.com/*
*http://www.w3c.org/*
Chapter 2 - Getting Started with XML

OBJECTIVES

- Write and view a simple XML document.
- Distinguish between well-formed and valid XML.
- Use an XML Schema to validate an XML document.
- Control the presentation style of an XML document.
Data and Document Structure

* In general terms, a document is any set of information or data.
  - Publications (web, print, and so on) for people to read.
  - Information passed by programs to other programs.

* A person can infer a document's structure intuitively.
  - You can distinguish between a grocery list and a parts list by reading the items.
  - You can identify an introductory paragraph by its position.

* To a computer program, a document is a sequence of characters.
  - A user must point out parts of a document to the program.
Consider a simple document, like a memo.

bossmemo.txt
To: Boss
From: Me
Hey, I am really glad I'm in XML class this week.

You can easily identify the different parts of your memo: the sender, the recipient, the message.
XML stands for *Extensible Markup Language*.

XML defines a way of marking up text to describe the structure of data.

- *Tags* identify the parts of the document.
- These tags build a hierarchy of elements that, in its entirety, makes a document.

XML is a way of creating your own markup language.

- You define the tags to explain your data.
- Your tags describe meaning and structure, not appearance.

XML is a standard for creating markup languages.

- That is, XML is a *meta-markup* language.
- Industries and organizations use XML to write rules defining their own markup languages.
- The World-Wide Web Consortium (W3C) created and maintains the definition of XML.
Jon Bosak of Sun Microsystems formed the XML Working Group in conjunction with the W3C. His efforts were focused on the evolution of markup languages from way too complex (SGML) to not powerful enough (HTML). The driving force behind XML is to build an infrastructure of markup languages that would allow industries to have a standard means of data interchange. Current examples are Scalable Vector Graphics, MathML, and XHTML. Each industry defines the rules that make a document valid for its particular use. XHTML, for example, is a redefinition of HTML as a markup language that complies with XML standards. XML is about creating these specialized languages.

1986 – SGML becomes a standard.
1996 – XML defined.
2006 – XML 1.1 recommendation released.

**Hands On:**
Create a file called `bossmemo.xml`. Add the following content to your file:

```
<memo>
  <to>Boss</to>
  <from>Me</from>

  <message>Hey, I am really glad I'm in XML class this week.</message>
</memo>
```

Load `bossmemo.xml` in your browser.
Well-Formed XML

A well-formed document conforms to XML's basic rules of syntax.

- Every open tag must be closed.
- The open tag must exactly match the closing tag: XML is case-sensitive.
- All elements must be embedded within a single root element.
- Child tags must be closed before parent tags.

"Well-formed" doesn't apply any validation tests to the content within the document.

- That is, a well-formed document has correct XML tag syntax, but the elements might be invalid for the specified document type.

Applications must reject your XML if it is not well-formed.
Hands On:

Modify bossmemo.xml, changing the name of the first tag to all uppercase letters:

```xml
<MEMO>
  <to>Boss</to>
  <from>Me</from>

  <message>Hey, I am really glad I'm in XML class this week.</message>
</MEMO>
```

bossmemo.xml is no longer well-formed, because the open and close tags do not match: XML tag names are case-sensitive. Try loading bossmemo.xml in your browser. What happens?

Now try this: fix the first tag, but change the order of the tags `<to>` and `<from>`:

```xml
<MEMO>
  <to>Boss</to>
  <from>Me</from>

  <message>Hey, I am really glad I'm in XML class this week.</message>
</MEMO>
```

bossmemo.xml is no longer well-formed, because the tags are now mismatched. See what happens when you reload it in your browser. The browser (and any other application) is supposed to reject any XML that isn't well-formed, regardless of what the document type is.

Restore bossmemo.xml to its well-formed state:

```xml
<memo>
  <to>Boss</to>
  <from>Me</from>

  <message>Hey, I am really glad I'm in XML class this week.</message>
</memo>
```
Valid vs. Well-Formed XML

* A valid document conforms to the predefined rules of a specific type of document.
  - These rules can be written by the author of the XML document or by someone else.
    - They might be from the same company or the same industry.
  - The rules determine the type of data that each part of a document can contain.

* Not every application requires your XML to be valid in order to complete its task.
  - For example, a browser can just display a document with no special treatment of the element structure.
  - That is, some applications validate and others don't.

* Application-to-application usage usually requires valid XML.
  - The receiving application needs to know which data elements to expect.
XML Schema

* An XML Schema defines rules for a specific type of document, including:
  - Names of elements, and how and where they can be used
  - The order of elements
  - Proper nesting and containment of elements
  - Element attributes

* XML Schema is another W3C recommendation.
  - Schemas are expressed in the form of a separate, well-formed, XML file.
  - The syntax of XML Schema is XML, so it is composed of a series of XML tags.

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="memo">
    <xsd:complexType>
      Tags, such as `element` and `attribute`, describe the location of the tags that make up your XML file.
    </xsd:complexType>
    Tags, such as `complexType` and `simpleType`, help to define the type of content the elements and attributes can have.
  </xsd:element>
</xsd:schema>
```
Hands On:

memo.xsd (provided in your chapter directory) is an XML Schema that validates bossmemo.xml:

```xml
memo.xsd
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="memo">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="to" type="xsd:string"/>
        <xsd:element name="from" type="xsd:string"/>
        <xsd:element name="message" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

The extension .xsd is the common convention for naming what are termed "XML Schema Definition" files.

In order to validate against XML Schema, the XML file needs to indicate the location of the schema. Update the `<memo>` tag to include additional information:

```xml
<memo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="memo.xsd">
  <to>Boss</to>
  <from>Me</from>

  <message>Hey, I am really glad I'm in XML class this week!</message>
</memo>
```

Ask your instructor how to validate the bossmemo.xml file against the XML Schema. If there is any sort of failure, verify that your information is typed in the correct case and verify that bossmemo.xml and memo.xsd are located in the same directory.

What happens if you add a new element after the `</message>` tag in your XML document — say, a P.S. element?

```xml
...  
  <message>Hey, I am really glad I'm in XML class this week.</message>
  <ps>You should take this class yourself!</ps>
</memo>
```

Is your document still valid? (Try it and see what happens, then remove this from your memo.)
Presentation Style

XML describes the structure of your document.

- Structure helps applications identify and manipulate parts of your document.

Some applications present or transform your document (or a subset) in some manner.

- A browser could present your document as a web page.
- A spreadsheet could present your document as a data table.

XML, by itself, says nothing about presentation.

You can use various languages to manipulate XML data for presentation.

- Cascading Style Sheets (CSS) are designed for specifying the display characteristics of data (HTML or XML) in a browser.
- Extensible Stylesheet Language (XSL) defines general-purpose formatting characteristics for XML data.
XSL-FO and XSLT

* **XSL Formatting Objects (XSL-FO)** allows you to define rules that directly manipulate the presentation of XML data.
  
  - You can use XSL-FO to define formatting rules that would apply to printers, typesetters, and various file formats.
  
  - XSL-FO can be directly transformed using an XSL-FO processor into various formats, such as postscript and PDF files.

```
XSL-FO  FO Processor  PDF, postscript
```

* **XSL Transformations (XSLT)** allows you to define how to transform an XML document into a different document.
  
  - For example, you could use XSLT to define how to transform your XML document into HTML, for use with HTML browsers.
  
  - The application reads both the XML and XSL documents, formatting the display accordingly.
Hands On:
Now let’s use XSLT to define a transformation of an XML memo into an HTML file. A stylesheet called `colormemo.xsl` is provided in the chapter directory.

colormemo.xsl

```xml
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <html>
            <body bgcolor="lightblue">
                <b>To: </b> <font color="red">
                    <xsl:value-of select="memo/to"/>
                </font>
                <br><br>
                <b>From: </b> <font color="red">
                    <xsl:value-of select="memo/from"/>
                </font>
                <br>
                <hr>
                <font color="black">
                    <xsl:value-of select="memo/message"/>
                </font>
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

Add a processing instruction to the top of your XML file referencing the XSLT stylesheet. The `xml-stylesheet` processing instruction is used by browsers to format the presentation of the XML file.

```xml
<?xml-stylesheet type="text/xsl" href="colormemo.xsl"?>
...
```

What do you see when you load `bossmemo.xml` in a browser?
Using XML

   - This sets the rules for which elements are permitted and where they are permitted.

2. Create an XML document and mark up data using the elements defined in the XML Schema.

3. Optionally, create a stylesheet that defines presentation rules for your XML.
Use the `maxOccurs="unbounded"` attribute to indicate that multiple `item` children are allowed within a `shoppingcart` element.

You can nest a `complexType` within an `element` to specify child elements.

Validate `shoppingcart.xml` with `shoppingcart.xsd`.

Often, XML designers will prototype their XML document first and then generate the schema using a tool.

```
shoppingcart.xsd
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="shoppingcart">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="item" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="id" type="xsd:string"/>
              <xsd:element name="description" type="xsd:string"/>
              <xsd:element name="quantity" type="xsd:int"/>
              <xsd:element name="price" type="xsd:decimal"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

```
shoppingcart.xml
<shoppingcart xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="shoppingcart.xsd">
  <item>
    <id>GWS27757</id>
    <description>Get Well Soon Bouquet</description>
    <quantity>1</quantity>
    <price>48.95</price>
  </item>
  <item>
    <id>CS24623</id>
    <description>Colors of Summer Bouquet</description>
    <quantity>1</quantity>
    <price>69.95</price>
  </item>
</shoppingcart>
```
**Labs**

1. Add a **subject** element to your memo XML file. Try to validate your memo file using your validation software. Does the software validate your XML?
   (Solution: `bossmemo1.xml`)

2. Change your XML Schema to support a **subject** element between the **from** and **message** elements.
   (Solutions: `bossmemo2.xml, memo2.xsd`).

3. In your XML document, place the **subject** element after the **message** element. What happens when you try to validate your XML file? Is your XML valid? If it isn't valid, fix your XML.
   (Solution: `bossmemo3.xml`).

4. (Optional) Add formatting rules to `colormemo.xsl` so that the subject is displayed in bold text.
   (Solutions: `bossmemo4.xml, colormemo2.xsl`).

5. The following table represents a list of books and their associated data. Use XML tags to describe the data. The data for the table can be found in `books.txt`. Make sure your tags describe all of the data's structure.
   **Note:** Just create an XML document with tags describing the data; don't attempt to format the data as a table.
   (Solution: `books.xml`)

<table>
<thead>
<tr>
<th>ISBN</th>
<th>Title</th>
<th>Author</th>
<th>Year Published</th>
</tr>
</thead>
<tbody>
<tr>
<td>0316769487</td>
<td>The Catcher in the Rye</td>
<td>J. D. Salinger</td>
<td>1951</td>
</tr>
<tr>
<td>0446310786</td>
<td>To Kill a Mockingbird</td>
<td>Harper Lee</td>
<td>1960</td>
</tr>
<tr>
<td>0684801221</td>
<td>The Old Man and the Sea</td>
<td>Ernest Hemingway</td>
<td>1952</td>
</tr>
<tr>
<td>0451526341</td>
<td>Animal Farm</td>
<td>George Orwell</td>
<td>1945</td>
</tr>
</tbody>
</table>

   (Solutions: `books2.xml, books.xsd`).
Chapter 3 - Writing Well-Formed XML

Objectives

- Describe the sections of an XML document.
- Write well-formed XML.
- Use attributes to describe an instance of an element.
- Properly nest elements.
- Use empty tags.
- Comment your XML data.
- Embed raw data in an XML file.
- Use entities to represent special characters.
**XML Fundamentals**

- An XML document is made up of two sections.
  - A *prolog*, if present, describes what data is to follow.
    - This may include processing instructions and/or an `xml` declaration.
  - A *data section* starts at the root element and contains nested elements and their associated data.
- An *element* is made up of an opening tag, a closing tag, and everything in-between.
  - Angle brackets, `< >`, delimit a tag.
  - Every tag must have a name.
    ```xml
    <booklist>
    </booklist>
    ```
  - Every open tag must be closed.
- An XML document is well-formed when it conforms to all of the syntax rules for XML.
  - An XML document that is not well-formed cannot be processed by an XML application.
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="memo.xsl"?>
<memo>
  <to>You</to>
  <from>Boss</from>
  <subject>XML Training</subject>
  <message>Study hard, you've got to use this soon!</message>
</memo>
Tag Attributes

* Opening tags can contain *attributes*, which further describe a specific instance of an element.

```xml
<tree type="Deciduous">
  
  Every attribute has a name, an equal sign, and a quoted value.

  - Single or double quotes can be used.
    ```xml
    <tree type="Evergreen" height='60'>
    ```
  
  - Use whitespace to separate one attribute from another.

  - XML is not well-formed if an attribute is missing a value or quotes around that value.

* Order of attributes is insignificant.

```
x<tree height="60" type="Evergreen">
```

* The presence or lack of attributes does not affect whether an XML document is well-formed.

  - Rather, this is a question of the document's validity.
The question of when to use attributes and when to use child elements to hold information about an element is sometimes difficult to decide. Some say that attributes are for metadata and child elements are for data, but it can be hard to distinguish between the two. As a general rule, multi-valued data or complex structures should be in child elements. This is because of the basic rules associated with attributes. First, an element cannot have more than one attribute with the same name. Second, an attribute's value must be made up of quoted character data, and this data is interpreted as a pure string even if you tried to incorporate markup syntax within it. Thus, attribute values should be kept elementary in nature.

In this example, the differentiation between attribute and child element is fairly obvious:

```xml
addressbook.xml
<addressbook>
  <card>
    <firstname>Sally</firstname>
    <lastname>Smith</lastname>
    <phone type="home">303-555-1234</phone>
    <phone type="cell">720-555-3256</phone>
    <address type="home">
      <street>1601 Main Street</street>
      <city>Denver</city>
      <state>Colorado</state>
      <zip>80202</zip>
    </address>
  </card>
</addressbook>
```

But in this example, either way works:

```xml
tree1.xml
<tree>
  <type>Deciduous</type>
  <height>60</height>
</tree>
```

or

```xml
tree2.xml
<tree type="Deciduous" height="60"></tree>
```

Ultimately, the decision rests with the designers and developers, and should best support the needs of the application.
**Naming Rules**

* Tag and attribute names must conform to XML's basic syntax rules.

  - *Name-start characters*, those characters permitted at the beginning of a name, are:
    - Letters, including A-Z, a-z (and characters from many other languages).
    - An underscore, _.

  - The following *name characters* are legal beyond the beginning of the name:
    - Name-start characters
    - Digits, 0 through 9
    - A dash, -
    - A period, .
    - A colon, : (though you should not use this yourself, since it is reserved for special uses)

  - Names beginning with xml, in any combination of case, are reserved and should not be used.
    - xml, Xml, XML are not legal starts to names.

  - Tag and attribute names are case-sensitive.
    - PRICE is not the same as price.
    - <price currency="USD"> is not the same as <price Currency="USD">.
Empty and Non-Empty Elements

- Most elements are non-empty elements, which contain data.
- A non-empty element must have separate opening and closing tags.

  `<saying>`
  - The quick brown fox jumped over the lazy dog.
  `</saying>`

  - A closing tag has the same name, but with a slash in front of it.
  - Attributes can appear only in the opening tag.

- Elements that are empty elements do not contain any data or sub-elements.
  - Empty elements are allowed to have attributes.
  - You can write such an element as an opening and closing tag with nothing between them.

    `<snow></snow>`

    `<snow accumulation="5"></snow>`

  - Alternatively, you can write a single tag that ends with `/>`.

    `<snow/>`

    `<snow accumulation="5"/>`

- For an XML document to be well-formed, every opened element must be closed, even if it is closed within a single tag.
weather.xml

```xml
<?xml version="1.0"?>
<weather date="12/17/2002">
  <location city="Yellowstone" state="WY">
    <high>26</high>
    <low>19</low>
    <precipitation>
      <amount>0.02</amount>
      <snow/>
    </precipitation>
  </location>
  <location city="Seattle" state="WA">
    <high>43</high>
    <low>39</low>
    <precipitation>
      <amount>0.30</amount>
      <rain/>
    </precipitation>
  </location>
</weather>
```
**Nesting and Hierarchy of Tags**

- Elements can contain other elements.
  - Tags must not overlap.
    - `<quick><brown>fox</brown></quick>` is allowed.
    - `<quick><brown>fox</quick></brown>` is not allowed.

- The first non-prolog element is the *root* element.
  - There is only one root element, containing all other elements and data, excluding the prolog.

- The elements form a hierarchy.
  - *Parent elements* contain *child elements*.
  - A child of a child element is called a *descendant element*.
  - A parent of a parent element is called an *ancestor element*.
    - The root element has no parent.

- For an XML document to be well-formed, it must:
  - Have one and only one root element.
  - Always close nested elements in the order opposite that in which they are opened.
Processing Instructions and the XML Declaration

A Processing Instruction (PI) tag starts with <? and ends with ?>.

```xml
<?output type="HTML"?>
```

- These pass information to an application (applications can elect to ignore PIs that aren't relevant to them).
- PIs are not part of the XML document.

A PI can occur anywhere in an XML document.

- However, they typically appear in the prolog.

The XML declaration is recommended in almost all XML files and, if present, must be the first thing in the file.

```xml
<?xml version="1.0"?>
```

- The **version** attribute, specifying the version of XML, is necessary to process the document.
- The **encoding** attribute is used for internationalization, to tell the parser with which encoding the document was written.
Encoding is part of the larger subject of Internationalization. The encoding attribute of the XML declaration specifies the text encoding with which the document was created.

The character sets that can be used should be defined by the application that is processing the XML document. The default is Unicode (UTF-8 or UTF-16), but there are often many other options for legacy encoding, such as ISO-8859-1 (Latin-1) or ISO-8859-5 (Cyrillic). Even platform-dependent character sets, such as MacRoman, may be understood.

The names for standard encodings are defined by the Internet Assigned Numbers Authority (IANA) and are available at [http://www.iana.org/assignments/character-sets](http://www.iana.org/assignments/character-sets).
Other XML Constructs

- Comment tags start with `<!--` and end with `-->`.
  - You can place these anywhere, except inside a tag.
  - Comments cannot be nested, nor can they contain `-->`.

- A **CDATA** tag encloses raw character data.
  - The tag starts with `<![CDATA[` and ends with `]]>`.
    `<![CDATA[
    raw character data
  ]]>`
  - Anything is allowed within, except for `]]>`.
  - Nothing within the tag will be interpreted as XML.
    `<![CDATA[ A tag like <_XML> is illegal]]>`
You may wonder when it is appropriate to use a **CDATA** section. It can be used around any element text to ensure the text is well-formed. However, this comes at a cost with regard to complexity and processing. The intent of **CDATA** is to prevent certain characters from being parsed at all. Any syntax that would break the rules of a well-formed document can be safely put inside a **CDATA** section.

An example of illegal syntax is the character `<` appearing within the data of an element. However, if put inside **CDATA**, the document is kept well-formed.

```xml
urgentmemo.xml
<?xml version="1.0"?>
<!-- This is a comment -->
<memo type="urgent">
   <!-- This is another comment -->
   <to>Boss</to>
   <from>Me</from>
   <message><![CDATA[Is accounts payable < 5000?]]></message>
</memo>
```
Entity and Character References

- An entity reference represents a specific character.
  - These allow your content to include characters that would otherwise be considered part of the markup.

- An entity reference starts with & and ends with ;.
  - These will not be interpreted within the CDATA tag.

- XML predefines a few entities:
  - &lt; is <
  - &gt; is >
  - &quot; is "
  - &apos; is '
  - &amp; is &

- Any individual character can be represented in a similar fashion.
  - A character reference starts with &#, includes a value, and ends with ;.
    - For value, you must use either a decimal or hexadecimal number (a hex value requires an additional x to be placed after the #).
    - &#38; is the ampersand: &, which is character code 38.
    - &#x00E7; is the lowercase c with cedilla: ç, which is character code 00E7.
To include a character like `<` in your data, use `&lt;`. This allows the computer to literally use the character instead of interpreting it as XML syntax.

urgentmemo2.xml

```xml
<?xml version="1.0"?>
<memo type="urgent">
  <to>Boss</to>
  <from>Me</from>
  <message>Is accounts payable &lt; 5000 &lt;= 63;</message>
</memo>
```
Labs

1. Write a well-formed XML document to describe the following invoice:
   - The invoice should have a company name, date, shipping method, customer information, payment information, and several items.
   - Use an attribute to specify the shipping method as FedEx.
   - The customer information should include a customer id, firstname, lastname, address, and zero or more area code and phone number combinations. For every phone number, include an attribute to describe it, using one of the following: home, work, fax, pager, or cell.
   - The payment should be a check with an attribute for a check number.
   - Finally, each item has a quantity, part number, and unit price. Also add an attribute to the item for a description of the item.

   (Solution: invoice.xml)

2. Write a well-formed XML document that defines a schedule for airlines. A schedule has several flights, where each flight has the following:
   - Airline Name
   - Flight Number
   - Equipment (that is, model of aircraft: 737, DC10, etc.)
   - Departure Time and Location
   - Arrival Time and Location
   - Distance

   (Solution: schedule1.xml)

3. To schedule1.xml, add an empty element for the availability of the flight. It should have three attributes that specify whether the flight has seats available in coach, business, or first class.
   (Solution: schedule2.xml)

4. Modify a flight so that it flies out of "Trinidad & Tobago International Airport" for an airline named "Ω Airlines."
   Hint: The hexadecimal code for Ω is 03A9.
   (Solution: schedule3.xml)
Chapter 10 - XML in Applications

Objectives

* Recognize opportunities to utilize XML in application design.

* Describe two major types of XML parsers.

* Explain the difference between validating and non-validating parsers.

* Describe how web services use XML.
ReasOns And PlaceS for Using XML

* Applications need a common format for easy data exchange.
  - Databases provide a common format, as well as persistence.
    - These are expensive in terms of cost and time.
  - Flat files and pipes are another approach to data exchange.
    - They do not address how data is formatted.

* XML is an ideal approach for data formatting.
  - Pipes, sockets, databases, and flat files can still be used for the exchange.
**DOM Parsers**

- *Document Object Model* (DOM) is a standard way to represent information in memory.
- DOM parsers build a representation in memory.
  - This allows a programmer to easily manipulate the objects.
  - Different approaches to iterating through the data are possible.
- From a set of memory-based objects, DOM can generate an XML file.
- There are DOM parsers for a number of languages.
  - C, C++, C#, Java, and Perl are the more commonly-used languages.
  - These are typically validating parsers that will check the XML against a DTD or schema.
SAX Parsers

SIMPLE API FOR XML (SAX) is a fast parser.

- It does not build an internal memory representation.

- It works by calling functions or methods.
  - The programmer codes these functions or methods in a SAX ContentHandler.
  - Select elements are easy to search for or extract.

- Most SAX parsers can be selectively validating.
  - Programmers do not typically worry about validating with SAX.

- There are a number of SAX parsers.
  - Java's API was one of the originals and, as such, is considered the norm.
  - There is also SAX for C, C++, and Perl.
XML Data \[\rightarrow\] SAX Parser \[\rightarrow\] Application \[\rightarrow\] SAX ContentHandler
Web Services

Web services are server applications that are accessed through standard protocols over the Internet.

- Many distributed services architectures are either difficult (CORBA) or proprietary (Java EE).
- XML is both simple and universal.

The simplest web services use HTTP and HTML, with the clients running in web browsers.

- Client-side processing is done in JavaScript and is limited.
- Locating HTML-based web services is performed with search engines, which are tedious and imprecise.

XML-based web services allow more intelligent client applications and more precise web service location.

- Web Service Description Language (WSDL) describes the interface to the web service.
- Simple Object Access Protocol (SOAP) is a protocol the client can use to send a request to the server.