Introduction to XML

Presented for UC Santa Cruz Extension
January 13, 2004

iDevelopSoftware, Inc.
Copyright (c) 2001 - 2004, All Rights Reserved

Copyright (c) 2001 - 2004 by iDevelopSoftware, Inc.

All Rights Reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author.

Permission is expressly granted to the University of California Santa Cruz Extension to duplicate these notes for students enrolled in the "Introduction to XML" course.

Under no condition is permission granted to make additional copies of the notes for other use.

This page must accompany all copies of these notes.

Bennett Smith
bennettsmith@idevelopsoftware.com
www.idevelopsoftware.com

MODULE 1

Business Case for Using the eXtensible Markup Language (XML)

Learning Objectives

The objectives for this module are as follows:

- Learn what XML is and is not;
- Learn why one might apply XML technologies;
- Learn how XML technologies help address significant business problems
- Learn which tools and platforms support development with XML

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

What is XML, and what is it not?

Before we can speak intelligently about when to apply XML technologies, it is probably a good idea to define what XML is and is not. This section will attempt to answer these questions.

XML is a Markup Language

The Extensible Markup Language (XML) is a markup language used to represent semi-structured data in an open format.

A markup language is a language used to express or convey metadata (that is, information about a dataset) in the context of the data it marks up.

Markup languages predate computers. An example of a non-computer related markup language is the set of proofreader's marks used to indicate necessary grammatical or stylistic changes to a manuscript.

Here is a quote from the Extensible Markup Language (XML) 1.0 standard that defines XML.

> "Extensible Markup language, abbreviated XML, describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them. XML is an application profile or restricted form of SGML, the Standard Generalized markup Language [ISO 8879]. By construction, XML documents are conforming SGML documents.
>
> XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure."

The full specification for the XML markup language can be read and printed from the World Wide Web Consortium (http://www.w3.org) using this URL:

http://www.w3.org/TR/REC-xml

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

XML is not for Presentation

Hyper-text Markup Language (HTML) is another popular markup language. It is used to describe the content and presentation of a web page. XML and HTML share a common ancestor in the markup language SGML. However, **HTML is about layout and presentation and XML is about structure and data**.

With HTML the data you wish to present is buried within the tags which express the layout and presentation of a page.

For example, here is a stock portfolio as HTML.

Listing 1: HTML Stock Portfolio

```
 1:  <html>
 2:    <head><title>Stock Portfolio</title></head>
 3:    <body>
 4:      <h1>Stock Portfolio for: Bill Gates</h1>
 5:      <h2>Close of Trade Day Summary: 04/17/2001</h2>
 6:      <table border="1">
 7:      <tr>
 8:        <th>Symbol</th>
 9:        <th>Corporate Name</th>
10:        <th>Price at Close</th>
11:      </tr>
12:      <tr><td>INTC</td><td>INTEL CORP</td><td>29.22</td>
13:      </tr>
14:      <tr><td>ORCL</td><td>ORACLE CORP</td><td>17.35</td>
15:      </tr>
16:      <tr><td>CSCO</td><td>CISCO SYSTEMS</td><td>17.9</td>
17:      </tr>
18:      <tr><td>SUNW</td><td>SUN MICROSYS</td><td>17.45</td>
19:      </tr>
20:      <tr><td>IBM</td><td>INTL BUS MACHINE</td><td>103</td>
21:      </tr>
22:      <tr><td>RHAT</td><td>RED HAT INC</td><td>5.9</td>
23:      </tr>
24:      </table>
25:    </body>
26:  </html>
```

To highlight the primary limitation of this markup language for business systems I have marked the data in ***bold italics***. As you can see the data is buried deep inside the markup necessary to describe the page layout.

One could imagine collecting this data over the course of a year and then wanting to graph the price trend for each security. It would be difficult to parse each saved HTML file to gather the data for the day, even if the page layout did not change at all during the course of the year.

XML is All About Data

XML provides a way to markup the data independent of the presentation. XML is represented as text using the **ISO/IEC 10646 character set** (basically Unicode). By selecting this character set in favor of something like ANSI or ASCII, it is possible to exchange data using XML globally.

Previously we saw an example of HTML markup. Below is the same content, but this time it is described using XML markup tags.

Listing 2: XML Stock Portfolio

```
 1:  <?xml version="1.0" encoding="UTF-8"?>
 2:  <portfolio>
 3:    <investor>Bill Gates</investor>
 4:    <trade_day>20010417</trade_day>
 5:    <securities>
 6:      <stock>
 7:        <symbol>INTC</symbol>
 8:        <name>INTEL CORP</name>
 9:        <close_price>29.22</close_price>
10:      </stock>
11:      <stock>
12:        <symbol>ORCL</symbol>
13:        <name>ORACLE CORP</name>
14:        <close_price>17.35</close_price>
15:      </stock>
16:      <stock>
17:        <symbol>CSCO</symbol>
18:        <name>CISCO SYSTEMS</name>
19:        <close_price>17.9</close_price>
20:      </stock>
21:      <stock>
22:        <symbol>SUNW</symbol>
23:        <name>SUN MICROSYS</name>
24:        <close_price>17.45</close_price>
25:      </stock>
26:      <stock>
27:        <symbol>IBM</symbol>
28:        <name>INTL BUS MACHINE</name>
29:        <close_price>103</close_price>
```

 Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

```
30:      </stock>
31:      <stock>
32:        <symbol>RHAT</symbol>
33:        <name>RED HAT INC</name>
34:        <close_price>5.9</close_price>
35:      </stock>
36:    </securities>
37:  </portfolio>
```

As you can see, with XML the data is marked up to express it's structure and nothing is expressed with regards to the presentation of the data. In fact, the same XML data can be presented many different ways. We will come to this shortly.

XML is eXtensible!

The XML specification defines what the markup language looks like. The specification does not define the tags used in the markup of a document, nor does it define any requirements for specific tags to appear within a document. That is 100% the responsibility of the XML document designer.

The XML document designer is the person or organization defines how a given XML document should be marked up.

You define the tags, what the data looks like, and what the semantics are for interpreting or using the data. In short, **you extend XML by introducing new XML document types**.

The flexibility and expressiveness of XML can be both a blessing and a curse. On the one hand, it allows you to represent any semi-structured data you wish in a format that is easy to share with others. On the other hand, if everyone in the world comes up with their own data format for similar data entities chaos would ensue.

There are a number of standards that related to XML which help address this problem. Three of these will be discussed over the next few lectures. They are Document Type Definitions (DTDs), XML Schemas, and XML Namespaces.

In addition to the above mentioned standards, many industry standards have been developed for the representation of various data. For example, there are emerging standards for **medical transactions** such as laboratory orders and prescriptions. There are also standards to help solve supply-chain management problems such as **component procurement**.

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

XML is for Data Interchange and Document Exchange

XML is an excellent choice for the exchange of business documents related to commerce transactions. A whole industry has grown up around Business-to-Business, Business-to-Consumer, and Consumer-to-Consumer electronic transactions.

An example of Business-to-Consumer data interchange is the purchase of consumer goods via a web store. Imagine that the user visits the web store to purchase a lawnmower and a baby monitor. These items are selected, a credit card number and shipping information is entered, and the order is placed. The web store may not actually have these items. It may be necessary to purchase them from a distributor.

In order to accomplish this electronically we need an agreed upon mechanism for moving a purchase order from the web store to the distributor. How can this be done?

A system to trade purchase orders could be build around any of the following technologies:

- Proprietary protocol transmitted using sockets (TCP/IP).

  This would be a very expensive way to implement a system like this. Each distributor would have to build a system that understood the proprietary protocol. If a distributor worked with many web stores it might become burdensome to build custom interfaces to each of them.

- Distributed component object model such as CORBA or DCOM.

  Distributed object communications tend to be a good solution for tightly coupled distributed systems. If all of the trading partners were on the same private network, and they all used similar computing environments this may be an appropriate technology. However, CORBA and DCOM require special configuration of corporate firewalls in order to pass method requests between clients and servers. This is a shortcoming.

- Text files

  The old standby! Electronic commerce could be conducted using something as simple as a text file and an e-mail system. There are issues of security as well as data integrity when using the e-mail system to route sensitive information such as a credit card number.

Of course, the ultimate solution (at least in this class) is to use XML for the data interchange.

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

The web store server could leverage XML to perform this purchase by generating a purchase order and transmitting it over the Internet to the distributor. This purchase order may look something like the following:

Listing 3: e-Commerce Purchase Order

```
 1:  <?xml version="1.0"?>
 2:  <purchaseOrder orderDate="1999-10-20">
 3:    <shipTo country="US">
 4:      <name>Alice Smith</name>
 5:      <street>123 Maple Street</street>
 6:      <city>Mill valley</city>
 7:      <state>CA</state>
 8:      <zip>90952</zip>
 9:    </shipTo>
10:    <billTo country="US">
11:      <name>Robert Smith</name>
12:      <street>8 Oak Avenue</street>
13:      <city>Old Town</city>
14:      <state>PA</state>
15:      <zip>95819</zip>
16:    </billTo>
17:    <comment>Hurry, my lawn is going wild!</comment>
18:    <items>
19:      <item partNum="872-AA">
20:        <productName>Lawnmower</productName>
21:        <quantity>1</quantity>
22:        <USPrice>148.95</USPrice>
23:        <comment>Confirm this is electric</comment>
24:      </item>
25:      <item partNum="926-AA">
26:        <productName>Baby Monitor</productName>
27:        <quantity>1</quantity>
28:        <USPrice>39.98</USPrice>
29:        <shipDate>1999-05-21</shipDate>
30:      </item>
31:    </items>
32:  </purchaseOrder>
```

In fact, the distributor may aggregate these purchase orders and place another order to the actual manufacturer of the lawnmower. There are numerous places in this chain of events where electronic data exchange with XML documents can be useful.

Other standards such as EDI do exist that address the same fundamental business problems. However, the ease with which XML documents can be created, read, and

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

manipulated makes it easier for more organizations to do electronic commerce than if they invested in implementing EDI.

XML Helps Build Dynamic Web Applications

Finally, XML is a good technology to consider when building a data driven web site. Since XML is all about data and HTML is all about presentation it makes sense to combine the two when building data driven web sites.

The general strategy is to store the data either directly in XML documents (If it doesn’t change very often.) or to store it in a data base (RDBMS) system. When a page needs to present some information to a person browsing the site the following steps are performed:

1. The page request is decoded to determine what is being requested by the browser.
2. The necessary data is retrieved from the RDBMS system and formatted into one or more XML documents.
3. The necessary style sheet is selected based on which page is to be generated.
4. (Optional) If the client browser does not support XML a full HTML page can be generated on the server by transforming the XML document using the style sheet.
5. The XML document and the style sheet are sent to the browser.
6. The browser displays the new page for the user. This may involve parsing the XML on the client, performing transformations, or dynamically generating HTML based on the contents of the XML document.

This is a very powerful way to construct dynamic data driven web sites. Later in the course we will look at some examples where this is done.

Goals of XML

The list below is a restatement of the goals of XML as stated in the standard, and subsequently expanded upon by Tim Bray, a member of the XML standard editing team.

1. XML shall be straightforwardly usable over the Internet.

 Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

This was not taken to mean that you could feed XML to the browsers of the day, but that the design would have regard at all times to the needs of distributed applications working on large-scale networks.

2. XML shall support a wide variety of applications.

   This can be seen as something of a counterweight to the first design goal. Whereas XML is built for the Internet, it is not limited to the Internet. Specifically, we wanted to enable the creation of applications including authoring tools, simple filters, formatting engines, and translators.

3. XML shall be compatible with SGML.

   Of our design goals, this proved to be the most troublesome in practice. Our subgoals included:

- Existing SGML tools will be able to read and write XML data.
- XML instances are SGML documents as they are, without changes to the instance.
- For any XML document, a DTD can be generated such that SGML will produce “the same parse” as would an XML processor.
- XML should have essentially the same expressive power as SGML.

That we largely succeeded in accomplishing these sub-goals is due to two factors. First, the determination of many members of the WG and SIG to stick with SGML compatibility, even when there were good design reasons for abandoning it. Second, the active participation and cooperation of the SGML community, led by Charles Goldfarb, in ensuring that XML and SGML remained compatible.

4. It shall be easy to write programs that process XML documents.

   Specifically, we wanted it to be straightforward to create useful XML programs that did not depend on reading the DTD. We were ambitious, and declared that “easy” meant that the holder of a CS bachelor’s degree ought to be able to write basic XML processing machinery in less than a week.

   The motivation here is obvious - data formats succeed or fail depending on whether there are good tools available. If the format is easy to process, good tools will be available, otherwise not.

   The processor-in-a-week goal has proved elusive, so in that respect we failed to meet this design goal at a quantitative level. However, the fact that within months of the appearance of the first drafts of the XML spec, there were a substantial number of freeware XML processors on the market serves as evidence that perhaps we did meet our goal, qualitatively.

5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.

One of the problems with SGML has historically been that in its (laudable) attempt to maximize generality, it adopted a large number of optional features. This meant, in practice that if I had a conforming SGML document and you had another, we might not be able to interchange them. This design goal was met; XML has no options, and every XML processor in the world should be able to read every XML document in the world, assuming that it can decode the characters.

6. XML documents should be human-legible and reasonably clear.

 This goal was motivated simply by the perception that textual formats are more open, more useful, and more pleasant to work with than binary formats. One of the substantial benefits of XML is that no matter how bad a day your tools are having, you can always pull an XML document into Emacs or Notepad or whatever your favorite editor is, and get useful work done.

7. The XML design should be prepared quickly.

 This goal was motivated largely by fear. We perceived that many of the Internet's powers-that-be did not share our desire for widespread use of open non proprietary textual data formats. We believed that if we didn't toss XML's hat into the ring soon, the Web's obvious need for extensibility would be met by some combination of binary gibberish and proprietary kludges.

8. The design of XML shall be formal and concise.

 This is closely related to design goal #4 above; a data format is programmer-friendly if programmers can read and use the defining documents; otherwise not. Too many other standards and specifications have relied too heavily on prose and not enough on formalisms. This is one area where some, in particular Dan Connolly, have urged that the actual XML spec falls short of the goal; that a version could have been created which was substantially more formal and concise than the current document.

9. XML documents shall be easy to create.

 There is more at work here than the simple realization that if the documents are easy to create, then they will be created. The main goal was in fact to design XML in such a way that it would be tractable to design and build XML authoring systems. Our success in meeting this design goal remains to be established in the marketplace.

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

10. Terseness in XML markup is of minimal importance.

The historical reason for this goal is that the complexity and difficulty of SGML was greatly increased by its use of minimizations, i.e., the omission of pieces of markup, in the interest of terseness. In the case of XML, whenever there was a conflict between conciseness and clarity, clarity won.

The Family of XML Standards

The acronym "XML" stands for eXtensible Markup Language. The W3C has published a standards document describing the specifics of this markup language. By itself this standard is not terribly interesting or powerful. It is only when combine with a myriad of other related standards that the true power of XML can be appreciated.

Here is a list of standards related to XML. This is not a complete list. Remember, XML is *extensible*, so new XML vocabularies are introduced on a regular basis. All of the standards listed here should be considered the "core" of XML and you should have at least a general understanding of each.

Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation 6 October 200, Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler. [http://www.w3.org/TR/REC-xml]

Namespaces in XML, W3C Recommendation 14 January 1999, Tim Bray, Dave Hollander, Andrew Layman. [http://www.w3.org/TR/REC-xml-names]

XML Inclusions (XInclude), Candidate Recommendation, Jonathan Marsh, David Orchard. [http://www.w3.org/TR/xinclude]

XML Information Set, W3C Recommendation 24 October 2001, John Cowan, Richard Tobin. [http://www.w3.org/TR/xml-infoset]

XML Base, W3C Recommendation 27 June 2001, Jonathan Marsh. [http://www.w3.org/TR/xmlbase]

Associating Stylesheets with XML, W3C Recommendation 29 June 1999, James Clark. [http://www.w3.org/TR/xml-stylesheet]

XML Path Language (XPath) Version 1.0, W3C Recommendation 16 November 1999, James Clark, Steve DeRose. [http://www.w3.org/TR/xpath]

XML Schema Part 1: Structures, W3C Recommendation 2 May 2001, Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn. [http://www.w3.org/TR/xmlschema-1]

XML Schema Part 2: Datatypes, W3C Recommendation 2 May 2001, Paul V. Biron, Ashok Malhotra. [http://www.w3.org/TR/xmlschema-2]

XSL Transformations (XSLT) Version 1.0, W3C Recommendation 16 November 1999, James Clark. [http://www.w3.org/TR/xslt]

Extensible Stylesheet Language (XSL) Version 1.0, W3C Recommendation 15 October 2001, Sharon Adler, Anders Berglund, Jeff Caruso, Stephen Deach, Tony Graham, Paul Grosso, Eduardo Gutentag, Alex Milowski, Scott Parnell, Jeremy Richman, Steve Zilles. [http://www.w3.org/TR/xsl]

XHTML 1.0 The Extensible HyperText Markup Language (Second Edition) : A Reformulation of HTML 4 in XML 1.0. W3C Recommendation 26 January 2000, revised 1 August 2002. [http://www.w3.org/TR/xhtml1]

Scaled Vector Graphics (SVG) 1.1 Specification, W3C Recommendation 14 January 2003, Jon Ferraiolo, FUJISAWA Jun, Dean Jackson. [http://www.w3.org/TR/SVG11]

Mathematical Markup Language (MathML) Version 2.0 (Second Edition), W3C Recommendation 21 October 2003, David Carlisle, Patrick Ion, Robert miner, Nico Poppelier. [http://www.w3.org/TR/MathML2]

Cascading style Sheets, level 1, W3C Recommendation 17 December 1996, revised 11 January 1999, Hakon Wium Lie, Bert Bos. [http://www.w3.org/TR/CSS1]

Cascading Style Sheets, level 2, W3C Recommendation 12 May 1998, Bert Bos, Hakon Wium Lie, Chris Lilley, Ian Jacobs. [http://www.w3.org/TR/CSS2]

Document Object Model (DOM) Level 1 Specification, W3C Recommendation 1 October 1998. [http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001]

Document Object Model (DOM) Level 2 {all specification}, W3C Recommendation. [http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113]

RDF/XML Syntax Specification (Revised), W3C Proposed Recommendation 15 December 2003, Dave Beckett, Brian McBride. [http://www.w3.org/TR/rdf-syntax-grammar]

Uniform Resource Identifiers (URI): Generic Syntax [http://www.ietf.org/rfc/rfc2396.txt]

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

Tools and Platforms Supporting XML Development

Here are links to some tools and platforms that support XML development. These are a good place to start. Many more exist so don't forget to search google!

Parser Implementations

This is a short list of parsers for XML. To do anything interesting with XML you will need one of these. Your choice will depend on the platform and language used in developing your applications.

- SAXON (Java) [http://saxon.sourceforge.net]
- Libxml (C/C++) [http://xmlsoft.org]
- MSXML [http://msdn.microsoft.com/xml]
- Xerces (Java, C++, Perl) [http://xml.apache.org]
- eXpat (C) [http://expat.sourceforge.net]

XML Editors

You need a good text editor when building XML applications. Any editor will do (Vi, Emacs, Notepad, etc.) The editors listed below have specific support for XML and include special features for document validation and often support XPath and XSLT directly.

- XmlSpy [http://www.altova.com]
- Syntext Serna [http://www.syntext.com]
- XMLmind [http:www.xmlmind.com]
- oXygen [http://www.oxygenxml.com]
- XMLwriter [http://xmlwriter.net]
- Xeena [http://www.alphaworks.ibm.com/tech/xeena]

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

MODULE 2

Basic eXtensible Markup Language Document Structure

Learning Objectives

The objectives for this module are as follows:

- Gain an understanding of elements, attributes, processing instructions, and comments.
- Learn how to construct an XML document that is well-formed.
- Become familiar with some of the tools used to manipulate XML documents.

The XML 1.0 Recommendation can be downloaded from http://www.w3.org/XML. An annotated version of the specification is also available at http://www.xml.com/axml/axml.html.

General XML Document Structure

XML Documents are comprised of three parts or sections. These are:

- Prolog

  The prolog of an XML document will contain processing instructions, comments, and possibly a data type definition (DTD).

- Body

  The body of an XML document will contain a minimum of one element, and possibly more nested child elements. It may also contain comments.

- Epilog

  The epilog follows the closing tag of the root element in the body. It may contain additional processing instructions and/or comments.

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

Formatting Conventions

Case Matters

When working with XML documents it is always important to remember that an XML processor does distinguish case. This means the following are all considered different names in XML:

purchaseOrder

purchaseorder

PURCHASEORDER

PurchaseOrder

Naming Things

XML is a tagged markup language. Each tag used in an XML document has a name. The XML standard defines a name like this:

NameChar ::= Letter | Digit | ‘.’ | ‘-’ | ‘_’ | ‘:’)

Name ::= (Letter | ‘_’ | ‘:’) (NameChar)*

It is important to remember that letters are not limited to ASCII only. Since XML uses the ISO/IEC 10646 character set all Unicode characters can be represented. In addition to the constraints stated above the standard does not permit a tag name to begin with the characters X M L, in either upper or lower case.

The table below shows some valid and invalid names in XML.

TABLE 1. XML Tag Names

Valid Names	Invalid Names
doghouse	-Stagecoach
Dog-House	14Stagecoach
Dog_House25	StageCoach@
Dog.House	XMLStageCoach

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

End-of-line Handling

XML documents are cross-platform. Many platforms encode the end-of-line with a single CR or LF. Yet others encode it as CR-LF together. The XML specification requires that all XML processors strip off the CR if found, and ensure that all lines end with a single LF.

Elements

Elements are the most basic component of an XML document.

Basics

Elements are comprised of a start tag, content, and an end tag.

For example:

```
1:   <message>
2:     Hello, World!
3:   </message>
```

This very simple example of XML contains a start element on line one, the content on line two, and the end stag on line three.

As you can see, a start tag is simply a tag name enclosed in angle brackets. This is very similar to HTML tags such as <head>, <body>, or <table>. The difference is that **you get to decide what the name of the tag is**.

XML processors ignore white-space in the content area of the document. This means we could have also written the example as follows:

```
1:   <message>Hello, World!</message>
```

These two messages are equivalent so far as the XML processor is concerned.

Each element must contain a matching end tag to be valid. If no end tag is found the XML processor will emit an error and processing will terminate.

 Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

Empty-Element Tags

There are times when an element may appear without any content. In this case it is valid to represent the element in either of these ways:

```
1:   <message></message>
```

or

```
1:   <message/>
```

Root Element

Each XML document must contain exactly one root element. The root element is special only in so far as the contents of this element represent the data of the XML document.

There can be no sibling elements to the root element. All other elements must be children of the root element.

The following example illustrates a root element (<messages>) and a number of nested elements.

```
1:   <messages>
2:     <message>
3:       Hello, World!
4:     </message
5:     <message>
6:       Goodbye
7:     </message>
8:   </messages>
```

In this case the root element contains two additional tags for separate messages.

Content

There are some rules governing content as well. Content is the text between opening and closing tags. In order to be able to tell the difference between tags and content, a small number of characters and character patterns are not allowed in the content area. Specifically:

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

- Less-than (<) signs are not allowed because they are reserved for starting tags. The greater-than sign (>) is allowed because it must be preceeded by the illegal less-than sign before it becomes significant.
- The ampersand (&) is not allowed because it is reserved for entity references, which are discussed later in the course.
- White space characters (spaces, tabs, CR+LF, and so on) are ignored. However, spaces between words are left intact.

There are times when you need one of the illegal symbols, or when you wish to preserve the spacing within some block of content. To deal with these situations the XML specification offers two techniques.

First, you can use entity references to "escape" the illegal characters within the content. This technique will be discussed at a later time.

Second, you may include a character data section (CDATA) within the XML content. When the XML processor sees a CDATA section it will ignore all text found within the section. This provides a very convenient means for preserving a block of text exactly as entered.

An example of using the CDATA section is shown below:

```
1:  <message>
2:    <![CDATA[ <h1>Hello, World!</h1> ]]>
3:  </message>
```

Notice that the message contains some embedded HTML.

Child Elements

Child elements are permitted within the content area of any XML element.

In this example we have a root element (<messages>) that contains two child elements (<message>). In the second message there are still more child elements each of which contains the message in a different language.

```
1:  <messages>
2:    <message>
3:      Hello, World!
4:    </message
5:    <message>
6:      <english>Goodbye</english>
7:      <french>au revoir</french>
```


Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

```
 8:      <italian>Arrivederci</italian>
 9:    </message>
10:  </messages>
```

Element Nesting

An important consideration when creating an XML document is the proper nesting of the elements. The nesting of elements defines an abstract tree structure where each element is a node in the tree.

The tree structure would be broken if child elements were improperly nested. The following is an example of what not to do.

```
 1:  <messages>
 2:    <message>
 3:      Hello, World!
 4:    </message>
 5:    <message>
 6:      <bold><english>Goodbye</bold></english>
 7:      <bold><french>au revoir</bold></french>
 8:      <bold><italian>Arrivederci</italian></bold>
 9:    </message>
10:  </messages>
```

Attributes

Attributes are additional data elements that help to more accurately describe an element. The following example encodes the language of a message in an attribute instead of a nested element as done above.

```
 1:  <messages>
 2:    <message language="english">
 3:      Goodbye
 4:    </message>
 5:    <message language="french">
 6:      Au revoir
 7:    </message>
 8:    <message language="italian">
 9:      Arrivederci
10:    </message>
11:  </messages>
```

Used correctly, attributes add value to an entity without blurring the line between content and metadata (data about data). This is sometimes a difficult balance to

achieve. When considering using an attribute, as yourself, “Should this value be visible to a reader as content? Or, should this value remain in the background and simply help out with the processing?”

If elements are the “nouns” of XML, then attributes are its “adjectives”. Attributes are attached to an element.

Attributes are only permitted to be listed in an element tag once.

Processing Instructions

XML documents are processed by XML processors. Some XML processors require additional information regarding “how” to process the documents. The XML standard provides processing instructions (PI) as a means of communicating with the XML processor during the parse process.

All PIs begin with an opening angle bracket and a question mark. This is followed by the name of the PI and any attributes that are a part of the instruction. The PI is closed with another question mark and a closing angle bracket.

Presently there is one processing instruction defined that must be present in an XML document for it to be considered valid. It is the processing instruction that informs the processor that the file is an XML document. It looks like this:

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

This processing instruct must appear as the very first line in an XML document file. If this PI is not present the file will not be considered well-formed. Refer to the W3C XML 1.0 Recommendation for a full definition of well-formed.

Comments

A comment is started with an angle bracket , an exclamation point, and two dashes. It is terminated by two dashes and a closing angle bracket. For example:

```
1:  <?xml version="1.0" ?>
2:  <!-- This is a comment line. -->
3:
4:  <!--
```

 Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

```
 5:     This is a comment that spans
 6:     multiple lines in the document.
 7:   -->
 8:
 9:   <messages>
10:   </messages>
```

XML comments are formatted like HTML comments. Since they lack the entity-style delimiters (< and />) you might conclude that comments are stripped out and ignored by XML processors, but that is not the case. Comments may remain visible to the parsing process. This is a deliberate design decision , making it possible for comments to pass through the parsing process.

A document must not begin with a comment. This would violate the rule that the XML processing instruction must appear on the first line of the file.

Be careful not to introduce a nesting error by commenting out the start or end tag for an element. It is okay to comment out an entire element, but it is an error to comment out part of an element. The XML processor will notify you if such an error is encountered.

It is invalid to enter a comment within the tag portion of an element. For example, the following is invalid:

```
1:   <?xml version="1.0" ?>
2:   <messages <-- these are messages --> >
3:   </messages>
```

Finally, since the two dashes are used to signal the end of a comment they may not appear within a comment.

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

MODULE 3

XML Namespaces

Learning Objectives

The objectives for this module are as follows:

- Learn what an XML Namespace is.
- Learn why an XML Namespace is important.
- Learn how an XML Namespace is both defined and declared within an XML document.

Standards Documents

The following standards may be of use in studying the topics in this module:

Namespaces in XML, http://www.w3.org/TR/REC-xml-names

Uniform Resource Identifiers (URI): Generic Syntax, http://www.ietf.org/rfc/rfc2396.txt

What is an XML Namespace?

From the W3C “Namespaces in XML” recommendation:

> “XML namespaces provide a simple method for qualifying element and attribute names used in Extensible Markup Language documents by associating them with namespaces identified by URI references.”

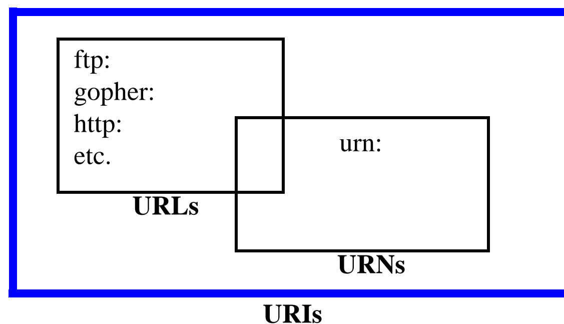
Stated another way, an XML namespace is a grouping of related element and attribute names whose purpose is disambiguation between multiple XML vocabularies within a single XML documents.

Naming “Stuff” on the World-Wide-Web

Before we proceed with a discussion of XML namespaces we need to understand how things are identified and named on the web. Specifically, we must learn about URIs, URLs, and URNs.

A Uniform Resource Identifier (URI) is a compact string of characters for identifying an abstract or physical resource. A URI is the one and only naming/addressing technology used on the world-wide-web. The URI standard is extensible to support a number of different addressing schemes. Two common schemes are URLs and URNs.

FIGURE 1. Relationship between URI, URL, and URN.



A URL (Uniform Resource Locator) is an informal term associated with popular URI naming schemes such as http, ftp, mailto, etc. This term is no longer used in technical specifications.

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

A URN (Uniform Resource Name) is a URI that has an institutional commitment to persistence, availability, etc. URNs follow a particular scheme, “urn:”, specified by RFC2141 and related documents, intended to serve as persistent, location-independent, resource identifiers. *Note that a URN may be a URL since a registered DNS domain name is considered universally unique.* This explains the overlap between URLs and URNs in the above figure.

What is an XML Vocabulary?

From the book “Professional XML” by Wrox Press,

> “An XML ‘vocabulary’ is a description of XML data that is used as the medium for information exchange, often within a specific domain of human activity (business, chemistry, law, music, for example).”

As we work through this module it will become clear that without XML Namespaces the inherent flexibility of XML would result in chaos as everyone defined their own XML vocabularies to describe the same or similar things.

Why use XML Namespaces?

When a developer publishes an XML vocabulary it contains a specific set of elements and attributes. The developer documents the semantics according to which these elements and attributes should be evaluated.

Now, imagine a situation where two developers each develop XML vocabularies for a similar purpose. Perhaps they both create XML vocabularies for the representation of names.

Developer one designs a vocabulary for listing the names of her friends. A sample document for this vocabulary might look like the following:

Listing 1: Names of Friends

```
1:  <?xml version="1.0"?>
2:  <friends>
3:    <friend>
4:      <name>John Doe</name>
5:      <phone>(408) 555-1234</phone>
6:    </friend>
7:    <friend>
8:      <name>Mary Smith</name>
```

```
 9:      <phone>(415) 627-9837</phone>
10:    </friend>
11:  </friends>
```

Developer two designs a vocabulary for listing the names of his pets. A sample document for this vocabulary might look like the following:

Listing 2: Names of Pets

```
1:  <?xml version="1.0"?>
2:  <pets>
3:    <pet type="dog">
4:      <name>Addie</name><!-- Mary's dog -->
5:    </pet>
6:    <pet type="cat">
7:      <name>Gracie</name><!-- John's cat -->
8:    </pet>
9:  </pets>
```

Suppose someone wants to create a document with the names of all her friends and their pets. Instead of building a third XML vocabulary why not just reuse the two existing vocabularies? The combine XML document might look like this:

Listing 3: Friends and Their Pets

```
 1:  <?xml version="1.0"?>
 2:  <animal.lovers>
 3:    <animal.lover>
 4:      <friend>
 5:        <name>John Doe</name>
 6:        <phone>(408) 555-1234</phone>
 7:      </friend>
 8:      <pet type="cat">
 9:        <name>Gracie</name><!-- John's cat -->
10:      </pet>
11:    </animal.lover>
12:    <animal.lover>
13:      <friend>
14:        <name>Mary Smith</name>
15:        <phone>(415) 627-9837</phone>
16:      </friend>
17:      <pet type="dog">
18:        <name>Addie</name><!-- Mary's dog -->
19:      </pet>
20:    </animal.lover>
21:  </animal.lovers>
```

There is a problem with the document displayed above. The <name> element is used in two different contexts within the document. Within the <friend> section it is

 Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

used to refer to a person while within the <pet> section it is used to refer to an animal.

Looking at this document is may be obvious to you or I that one is the name of a person and the other is the name of an animal, but to an XML processor this is not so clear. Remember, the meaning or semantics of a tag are defined by the XML document designer, not by the XML standard.

In order for an XML processor to clearly distinguish between the two <name> elements we must introduce the use of XML Namespaces. With the help of XML Namespaces it is possible to precisely identify a name as either that of a person or a pet.

Reasons to use XML Namespaces

To summarize, the following are all reasons to consider using XML namespaces when designing XML documents.

1. XML Namespaces allow for the mixing of XML vocabularies.
2. XML Namespaces allow for partitioning of large or difficult problems into smaller more soluble problems.
3. XML Namespaces when combine with XML Schemas help promote reuse.

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

Using XML Namespaces

This section of the module discusses how one goes about using XML Namespaces within an XML document.

Declaration

An XML Namespace is declared as an attribute of an element within an XML document. Before examining the details of a namespace declaration let's look at an example:

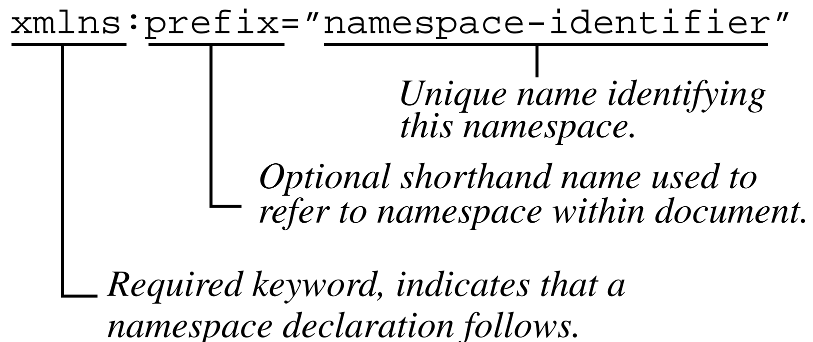
Listing 4: Simple XML Namespace

```
1:  <?xml version="1.0"?>
2:  <messages xmlns:m="http://www.yourcompany.com/ns/messages">
3:    <m:message m:language="english">
4:      Hello, World!
5:    </m:message>
6:  </messages>
```

In the above listing an XML Namespace is declared for the messages element. The namespace will be referenced using the short-hand "m" within the document.

The general format of an XML Namespace declaration is illustrated below.

FIGURE 2. General Form of XML Namespace Declaration



The namespace-identifier portion of an XML Namespace declaration should follow URI naming conventions.

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

Note that even though many XML Namespace declarations are in the form of a URL, the XML processor is not required to resolve the URL. In terms of XML Namespaces, the URL meets the requirement that it be unique by virtue of the fact that the registered domain name portion of a URL is unique.

When you design an XML vocabulary you may select an XML Namespace for it. If you do, then it is your responsibility to ensure that the selected namespace-identifier is unique.

Ways to Ensure Uniqueness

1. Use a URL that contains a fully qualified domain name (FQDN) that has been registered on the Internet. This FQDN is already known to be unique, so by appending a unique path, it is possible to create a unique namespace-identifier.
2. Use a URN. See an earlier section in this module for a discussion of URNs. The URN contains a unique identifier (number) assigned by the IANA (Internet Assigned Number Authority; http://www.iana.org).

Qualified Names

A qualified name is used to identify an element or attribute as a member of a particular namespace. Qualified names are composed using the prefix for a previously declared XML Namespace, a colon, and the name of the element or attribute.

The prefix used in the qualified name must have already been seen, either as part of the xmlns: declaration for this element, or previously in a parent element.

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

Namespace Scoping

Declarations of XML Namespaces are not global to the document. The declaration is scoped to the element where it appears, and to all child elements that are part of the element's content. A namespace declaration may be overridden in a child element by declaring a new namespace with the same prefix.

Listing 5: Overriding a Namespace Declaration

```
1:  <?xml version="1.0"?>
2:  <foo xmlns:a="http://www.xyz.com/nsa">
3:    <!-- a:bar is from nsa -->
4:    <a:bar>
5:      <!-- a:jar is from nsa2 -->
6:      <a:jar xmlns:a="http://www.xyz.com/nsa2">
7:      </a:jar>
8:    </a:bar>
9:  </foo>
```

Default Namespace

An XML document may have a default namespace. If declared, the default namespace applies to all non-qualified element and attributes within the document.

By default an XML document **does not** have a default namespace. A default namespace must be explicitly declared. To declare a default namespace use the same syntax as a qualified declaration, but omit the prefix.

Listing 6: Default Namespace

```
1:  <?xml version="1.0"?>
2:  <foo xmlns:="http://www.xyzm.com/ns">
3:    <a>
4:      Blah, Blah, Blah!
5:    </a>
6:  </foo>
```

Multiple Namespaces in an XML Document

As seen in listing 5 above, it is permissible to declare multiple namespaces within the same document. You may even have multiple namespaces declared within a single element. This is normally seen in the root element of a document.

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

Another look at Friends and Their Pets

Here are listings for the two XML vocabularies introduced earlier in the module. They have been modified to use XML Namespaces. The combine “animal lovers” XML document has also been modified to solve the ambiguity problem with the <name> element.

Listing 7: Names of Friends (w/XML Namespace)

```
 1:  <?xml version="1.0"?>
 2:  <f:friends xmlns:f="http://www.xyz.com/ns/friends">
 3:    <f:friend>
 4:      <f:name>John Doe</f:name>
 5:      <f:phone>(408) 555-1234</f:phone>
 6:    </f:friend>
 7:    <f:friend>
 8:      <f:name>Mary Smith</f:name>
 9:      <f:phone>(415) 627-9837</f:phone>
10:    </f:friend>
11:  </f:friends>
```

Listing 8: Names of Pets (w/XML Namespace)

```
1:  <?xml version="1.0"?>
2:  <p:pets xmlns:p="http://www.abc.com/ns/pets">
3:    <p:pet p:type="dog">
4:      <p:name>Addie</p:name><!-- Marys dog -->
5:    </p:pet>
6:    <p:pet p:type="cat">
7:      <p:name>Gracie</p:name><!-- Johns cat -->
8:    </p:pet>
9:  </p:pets>
```

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

Listing 9: Friends and Their Pets (w/XML Namespaces)

```
 1: <?xml version="1.0"?>
 2: <animal.lovers xmlns:="http://www.spca.org/ns/alvr"
 3:             xmlns:f="http://www.xyz.com/ns/friends"
 4:             xmlns:p="http://www.abc.com/ns/pets">
 5:   <animal.lover>
 6:     <f:friend>
 7:       <f:name>John Doe</f:name>
 8:       <f:phone>(408) 555-1234</f:phone>
 9:     </f:friend>
10:     <p:pet p:type="cat">
11:       <p:name>Gracie</p:name>
12:     </p:pet>
13:   </animal.lover>
14:   <animal.lover>
15:     <f:friend>
16:       <f:name>Mary Smith</f:name>
17:       <f:phone>(415) 627-9837</f:phone>
18:     </f:friend>
19:     <p:pet type="dog">
20:       <p:name>Addie</p:name>
21:     </p:pet>
22:   </animal.lover>
23: </animal.lovers>
```


Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

MODULE 4

Document Validation

Learning Objectives

The objectives for this module are as follows:

- Learn what document validation is and why it is important.
- Learn the difference between “well-formed” and “valid” XML documents.
- Learn what a Document Type Definition (DTD) is.
- Learn what XML Schema is.
- Learn how to create a DTD as part of designing an XML vocabulary.
- Learn how to create an XML Schema as part of designing an XML vocabulary.

Standards Documents

The following standards may be of use in studying the topics in this module:

Extensible Markup Language (XML) 1.0 (Second Edition), http://www.w3.org/TR/REC-xml.

Namespaces in XML, http://www.w3.org/TR/REC-xml-names.

XML Schema Part 0: Primer, http://www.w3.org/TR/xmlschema-0.

XML Schema Part 1: Structures, http://www.w3.org/TR/xmlschema-1.

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

XML Schema Part 2: Datatypes, http://www.w3.org/TR/xmlschema-2.

Document Validation

The idea behind document validation is to allow the XML processor to leverage the power of the XML parser for initial correctness checking of any input XML document.

The goal is to provide a means for a XML vocabulary designer to clearly state the allowable combinations of elements and attributes that produce "valid" instances of the XML vocabulary.

In order to acomplish this it is necessary to have a language capable of specifying the valid combinations. Two such languages exist for XML vocabularies. The first is the Document Type Definition (DTD) and the second is the XML Schema.

Well-Formed

An XML document is said to be well-formed when it adheres to the syntax outlined in the XML 1.0 standard. This means that all opening tags are matched with closing tags, all element and attribute names are properly formed, and that the nesting of elements is correct.

Well-formed does **not** mean the document structure is "correct" for a given XML processor.

Valid

An XML document is said to be valid when it has been checked against the defintion for the appropriate XML vocabulary. Validity is checked by the XML parser by comparing the structure of the document with the rules as expressed in either a Document Type Definition or an XML Schema.

A document must be well-formed in order to be valid. A document may be well-formed without being valid .

A "validating" XML Parser is required in order to perform these checks. Many parsers exist which only check for well-formed XML, and do not attempt to validate the content of the document.

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

What is a Document Type Definition?

This section will discuss the both document type definitions and document type declarations.

Document Type Definition

A document type definition is a grammar that describes a class of documents, and consists of markup declarations. The document type definition is usually refered to as the DTD.

Another way to think about a DTD is that it defines the universe of all possible XML documents for a particular purpose. The DTD lists all permissible element and attribute tag names, and the valid combinations for those tags.

The following listing illustrates a very simple external document type definition. It describes a class of documents that can contain exactly one empty element, named "foo."

Listing 1: Simple External Document Type Definition

```
1:   <?xml encoding="UTF-8"?>
2:   <!ELEMENT foo EMPTY>
```

Document Type Declaration

A document type declaration is part of an XML document that contains or points to markup declarations that provide a grammar for a class of documents. The declaration can point to a DTD stored in an external file, or can contain the markup declaratioins directly in the XML document.

An XML document that contains the DTD within it is called a **standalone** document. It contains an "internal" DTD. A standalone document may be marked as such using the following attribute:

```
1:   <?xml version="1.0" standalone="yes"?>
```

An XML document that contains a pointer to a DTD stored in a separate file has an "external" DTD. When the DTD is stored external to the XML document, it may be indicated as follows:

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

```
1:  <?xml version="1.0" standalone="no"?>
```

The following listsing illustrate how to use an external document type declaration in an XML document.

Listing 2: Referencing External DTD

```
1:  <?xml version="1.0" standalone="no"?>
2:  <!DOCTYPE foo SYSTEM "Listing-1.dtd">
3:  <foo/>
```

The next listing is equivalent to the previous, but makes use of an internal document type declaration.

Listing 3: Referencing Internal DTD

```
1:  <?xml version="1.0" standalone="yes"?>
2:  <!DOCTYPE foo [
3:    <!ELEMENT foo EMPTY>
4:  ]>
5:  <foo/>
```

Issues with Document Type Definitions

Document Type Definitions are an excellent way to express constraints on the structure of an XML document.

The following are some issues with DTDs:

1. Syntax not easy to understand, and not easy to parse.
2. Namespaces not easily supported.
3. Closed document model. Not easy to extend a DTD.
4. All content is expressed as strings. No data types for content validation.

As more developers begin to use XML documents as a means of exchanging data between applications these issues become more significant.

In the next section we will look at XML Schema, which is another way of expressing the design of an XML vocabulary. XML Schema addresses many of these shortcomings of Document Type Definitions.

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

What is an XML Schema?

An XML Schema is similar to a DTD in that it specifies the structure of “valid” XML document instances.

A schema differs from a DTD in three key ways:

1. The syntax of an XML Schema is similar to a regular XML file. You can use an XML parser to read and validate an XML Schema just like any other XML document!
2. An XML Schema is extensible. You can reuse existing XML Schema in new schema. XML Schemas support the use of XML Namespaces since they are just regular old XML files!
3. An XML Schema allows for the specification of data types within the content of elements. A rich set of simple and complex data types are available for expressing the content of an XML document.

In general, if you are building a new XML vocabulary it is wise to consider expressing the vocabulary using an XML Schema instead of using the older Document Type Definition format.

Note: There are actual a number of different versions of the XML Schema specification. These notes refer to the current W3C version. Many tools and some of the books on XML refer to earlier versions of XML Schema under names like XML-Data, Document Content Description, Schema for Object-Oriented XML, or Document Description Markup Language.

Here is a very simple XML Schema that represents our single element XML vocabulary.

Listing 4: Simple XML Schema Definition

```
1:   <?xml version="1.0" encoding="UTF-8"?>
2:   <xsd:schema
3:        xmlns:foo="http://www.abc.com/ns/foo"
4:        xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
5:        targetNamespace="http://www.abc.com/ns/foo">
6:     <xsd:element name="foo">
7:        <xsd:complexType/>
8:     </xsd:element>
9:   </xsd:schema>
```

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

This XML Schema defines a new namespace called “foo” that contains all documents consisting of a single element named “foo”. This element is empty.

An example of a XML document instance using this schema is:

Listing 5: Simple Document Instance Referencing XML Schema

```
1:  <?xml version="1.0" encoding="UTF-8"?>
2:  <foo
3:    xmlns="http://www.abc.com/ns/foo"
4:    xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
5:    xsi:schemaLocation="http://www.abc.com/ns/foo
6:                         Listing-4.xsd">
7:  </foo>
```

The default namespace for this document is “http://www.abc.com/ns/foo”. The schemaLocation attribute contains a pair of values. The first value is the name of an XML Namespace and the second value is a hint to the XML processor as to where it can find this XML Schema. In this case, the schema file is stored in “Listing-4.xsd.”

How is a Document Type Definition Constructed?

A DTD contains markup declarations. Each markup declaration describes one of the following:

- element type declaration
- attribute list declaration
- entity declaration
- notation declaration

In order to construct a document type definition we must understand how to express each of these declarations. Minimally, element type and attribute list declarations must be understood. These are discussed below.

For a discussion of entity and notation declarations refer to the XML 1.0 standards document, or the text for this course.

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

Element Type Declaration

When validating a document against a DTD every element found in the document must be listed within the DTD. Elements not listed in the DTD are not permitted to appear in the document instance.

Element type declarations are used to identify the elements that can appear in the document instances. The general form of an element type declaration is:

```
1:   <!ELEMENT ElementName Rule>
```

One occurance of this statement must appear for each element allowed in the document. The “ElementName” is the tag name selected for the element and the “Rule” lists the allowable content for the element.

Remember, every XML element has three parts; the start tag, content, and end tag. This element type declaration simply allows you to express the tag name and valid content for the element.

The “Rule” is used to express the contents of the element. In an XML document the content of an element can be one or more of the following things:

- Empty
- Child Element(s)
- Characters (i.e. string data)

Expressing the rule is done using content declarations.

For empty content the declaration is **EMPTY**.

For character (string data) content the declaration is **#PCDATA**.

For child element content the declaration is the name of the child element. More on this syntax in a bit.

Additionallyl, it is possible to have an element with unconstrained content. This is expressed using the declaration **ANY**. Note that this defeats the purpose of validation alltogether. Once you have specified the content as ANY, there is no way to determine if the content is valid.

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

Listing 6: Examples of Element Declarations

```
1:  <!ELEMENT title (#PCDATA)><!-- character data -->
2:  <!ELEMENT html ANY>       <!-- anything oaky in content -->
3:  <!ELEMENT foo EMPTY>      <!-- nothing in content -->
4:  <!ELEMENT abc (foo)>      <!-- abc content = foo element -->
```

Including multiple child elements within an element is done with a comma-separated list of child elements. The comman-separated list of children are expected to always appear in the specified order. If you wish to list a number of child elements that may appear in any order, use the "or" symbol '|' instead of the comma-separated list. The "or" symbol indicates that any one of the listed child elements may appear.

Parentheses may be used to group child elements. The number of occurances is specified using special symbols. The table below shows the symbols:

TABLE 1. Element Cardinality

Symbol	Cardinality	Description
	1	No symbol indicates element **must** appear once.
?	0,1	Element appears zero or one times.
+	1-n	Element appears one or more times.
*	0-n	Element appears zero or more times.

The cardinality symbol, if present, appears to the left of the element name within the rule, or just outside a closing parenthesis if the cardinality applies to a group of child elements.

Listing 7: Element Cardinality Examples

```
 1:  <!-- One or more message within the messages element -->
 2:  <!ELEMENT messages message+>
 3:
 4:  <!-- Addresses contains zero or more children.  Each -->
 5:  <!-- child can be either a mailaddr or an emailaddr. -->
 6:  <!ELEMENT Addresses ( mailaddr | emailaddr )*>
 7:
 8:  <!-- The mailaddr contains a US mailing address. -->
 9:  <!-- The second line of the address information is -->
10:  <!-- optional. -->
11:  <!ELEMENT mailaddr ( line1, (line2)?, city, state, zip )>
```


Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

Attribute List Declarations

Attribute list declarations are used to identify the names of the attributes that may appear within the start tag of an element. The general format of the attribute list declaration is:

```
1:  <!ATTLIST ElementName AttributeDefinition+>
```

The "ElementName" must identify one of the element type declarations from the DTD and the "AttributeDefinition" contains specific information about permissible content for this attribute. The format for an "AttributeDefinition" is:

```
1:  AttributeName AttributeType DefaultDeclaration
```

Multiple "AttributeDefinition"s may appear within a single ATTLIST declaration.

The "AttributeName" must follow the naming conventions for an XML identifier. The "AttributeType" is one of the types listed in the following table.

TABLE 2. AttributeType Values

Type	Description
CDATA	General character data.
enumerated	An enumerated list of specific values. (ex. Red, Green, Blue)
ENTITY	Reusable string.
ENTITIES	List of reusable strings.
ID	Unique identifier for this element.
IDREF	Reference to another element in the document, based on the unique identifier of the other element.
IDREFS	List of references to the unique identifiers of other elements within the document.
NMTOKEN	Similar to CDATA, but with more restrictive set of allowable characters. Cannot include a space.
NMTOKENS	List of NMTOKEN strings, separated by spaces.

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

The “DefaultDeclaration” portion of an attribute definition indicates what the default behavior of the XML parser should be for the named attribute. Refer to the following table for a list of allowable values:.

TABLE 3. DefaultDeclaration Values

Type	Description
#REQUIRED	Indicates that the attribute is required to appear within the element.
#IMPLIED	Indicates that the attribute may appear within the element. If the attribute is not present, the XML processor will provide an implied default value. You do not know what this value is from looking at the DTD, unless someone was good enough to mention it in a comment.
#FIXED	Indicates that this attribute has a fixed value. The value may not be changed by explicitly setting it within the document.
character data	Any other character data here represents the default value if no value is provided within the document. The XML parser will add this attribute, using the character data as the value.

Example DTD

This is an example DTD for the Alice30.xml file.

Listing 8: Alice30.dtd

```
 1:  <?xml version="1.0" encoding="UTF-8"?>
 2:  <!-- Top level element should be ebook -->
 3:  <!ELEMENT ebook (the_small_print, ebook_info, chapters)>
 4:
 5:  <!-- The small print contains the license
 6:       for the ebook, as CDATA. -->
 7:  <!ELEMENT the_small_print EMPTY>
 8:
 9:  <!-- General information about the ebook -->
10:  <!ELEMENT ebook_info (title, author, edition)>
11:  <!ELEMENT title (#PCDATA)>
12:  <!ELEMENT author (#PCDATA)>
13:  <!ELEMENT edition (#PCDATA)>
14:
15:  <!-- Chapters within the ebook -->
16:  <!ELEMENT chapters (chapter+)>
17:
18:  <!-- Chapter is comprised of a number of paragraphs. -->
19:  <!ELEMENT chapter (para+)>
```

 Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

```
20:  <!ATTLIST chapter
21:    numberCDATA#REQUIRED
22:    titleCDATA#REQUIRED
23:  >
24:
25:  <!-- Paragraph is the basic building block of the ebook. -->
26:  <!ELEMENT para (#PCDATA | illustration | quote)*>
27:
28:  <!-- Illustration contains only CDATA -->
29:  <!ELEMENT illustration EMPTY>
30:
31:  <!-- Quote contains only CDATA -->
32:  <!ELEMENT quote EMPTY>
33:
```

How is an XML Schema Constructed?

Now that we have an understanding of how a DTD is constructed, and an appreciation for DTDs shortcomings, let's look into how an XML Schema is constructed.

You can create XML Schema files by hand, but it is not advisable due to the complexity of the language. A good tool like XML Spy will help greatly in the preparation of an XML Schema file.

XML Schema Document Structure

A new XML schema document will have a well-defined structure, primarily because it must conform to the DTD for an XML Schema. The example below illustrates a basic XML Schema file. Notice that the "xsd" namespace is used to let the XML processor know we want to use schema element and attribute names.

Listing 9: XML Schema Document Structure

```
 1:  <?xml version="1.0" encoding="UTF-8"?>
 2:  <xsd:schema
 3:      xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
 4:    <xsd:element name="ENTER_NAME_OF_ROOT_ELEMENT_HERE">
 5:      <xsd:annotation>
 6:        <xsd:documentation>
 7:          Comment describing your root element
 8:        </xsd:documentation>
 9:      </xsd:annotation>
10:    </xsd:element>
11:    <xsd:simpleType name="ENTER_NAME_OF_SIMPLE_TYPE_HERE">
```

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

```
12:       <xsd:annotation>
13:         <xsd:documentation>
14:           Comment describing your simple data type
15:         </xsd:documentation>
16:       </xsd:annotation>
17:       <xsd:restriction/>
18:     </xsd:simpleType>
19:   </xsd:schema>
```

The root element of an XML Schema document is named “schema”.

You may document a schema using annotations. Annotations contain documentation.

In general terms, an XML Schema file will be used to create the following:

- Elements
- Attributes
- Simple Types
- Complex Types

Elements may contain attributes, simple types or complex types.

Schema Data Types

Each element within an XML document must have a data type associated with it. XML Schema provides a model for declaring the data type for each element.

There are a large number of simple data types available in XML Schema. For a complete discussion of possible data types refer to the XML Schema Part 0: Primer specification document. The simple types are listed in the table below for reference.

TABLE 4. XML Schema Simple Types

string	CDATA	token	byte
unsignedByte	binary	integer	positiveInteger
nonNegativeInteger	nonPositiveInteger	int	unsignedInt
long	unsignedLong	short	unsignedShort
decimal	float	double	boolean
time	timeInstant	timePeriod	timeDuration

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

TABLE 4. XML Schema Simple Types

string	CDATA	token	byte
date	month	year	century
recurringDay	recurringDate	recurringDuration	Name
QName	NCName	uriReference	language
ID	IDREF	IDREFS	ENTITY
ENTITIES	NOTATION	NMTOKEN	NMTOKENS

An XML Schema Simple Type is made up of three components. These are the value space, lexical space, and facet.

The value space represents all possible values that can be represented in a variable of the specified type.

The lexical space represents all possible literal representations for the values of the specified type. For example, the “positiveInteger” value 100 can be represented as either “100” or “1.0E2”.

The facets of a data type represent characteristics of the type. The XML Schema specification goes into great detail on different types of facets for data types. Facets can be categorized as fundamental or constraining. Fundamental facets define the data type. Constraining facets place some form of constraint on a data type.

Ordered Facets

The following table lists ordered facets for a simple data type:

TABLE 5. Ordered Facets

maxExclusive	minExclusive	maxInclusive	minInclusive
precision	scale		

Unordered Facets

The following table lists unordered facets for a simple data type:

TABLE 6. Unordered Facets

period	length	maxLength	minLength
pattern	enumeration	encoding	

Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

Creating a Simple Data Type

The XML Schema element “simpleType” is used to define a new simple data type. The general form for a simple data type declaration is:

Listing 10: Simple Data Type

```
 1:  <xsd:simpleType name="string50">
 2:    <xsd:annotation>
 3:      <xsd:documentation>
 4:        This is where you write a description of
 5:        what this data type is to be used for.
 6:      </xsd:documentation>
 7:    </xsd:annotation>
 8:    <xsd:restriction base="xsd:string">
 9:      <xsd:maxLength value="50"/>
10:      <xsd:minLength value="0"/>
11:    </xsd:restriction>
12:  </xsd:simpleType>
```

The above sample illustrates how to create a new simple data type that restricts one of the general data types available in XML Schema.

Here we are defining a new data type that is capable of storing a string up to 50 characters in length. This is accomplished by “restricting” the permissable values for the string using the unordered facets “maxLength” and “minLength.”

Creating a Complex Data Type

The XML Schema element “complexType” is used to define a new complex data type. Complex data types are much more expressive than simple data types. A complex data type may contain a number of child elements as well as attributes.

An example of a complex type would be a US Phone Number, represented with an area code, prefix, and number.

Listing 11: Complex Data Type

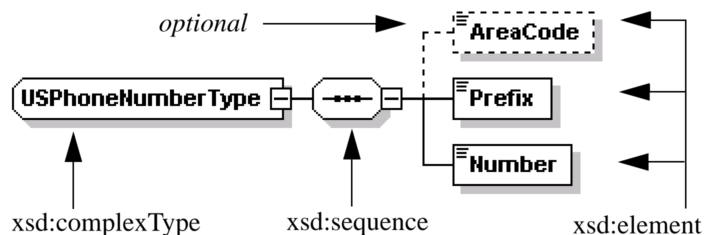
```
 1:  <xsd:complexType name="USPhoneNumberType">
 2:    <xsd:sequence>
 3:      <xsd:element name="AreaCode" minOccurs="0">
 4:        <xsd:simpleType>
 5:          <xsd:restriction base="xsd:positiveInteger">
 6:            <xsd:minInclusive value="0"/>
 7:            <xsd:maxInclusive value="999"/>
 8:          </xsd:restriction>
 9:        </xsd:simpleType>
```


Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

```
10:         </xsd:element>
11:         <xsd:element name="Prefix">
12:           <xsd:simpleType>
13:             <xsd:restriction base="xsd:positiveInteger">
14:               <xsd:minInclusive value="0"/>
15:               <xsd:maxInclusive value="999"/>
16:             </xsd:restriction>
17:           </xsd:simpleType>
18:         </xsd:element>
19:         <xsd:element name="Number">
20:           <xsd:simpleType>
21:             <xsd:restriction base="xsd:positiveInteger">
22:               <xsd:minInclusive value="0"/>
23:               <xsd:maxInclusive value="9999"/>
24:             </xsd:restriction>
25:           </xsd:simpleType>
26:         </xsd:element>
27:       </xsd:sequence>
28:         <xsd:attribute name="status" use="default"
29:                                         value="Provisioned">
30:           <xsd:simpleType>
31:             <xsd:restriction base="xsd:NMTOKEN">
32:               <xsd:enumeration value="Provisioned"/>
33:               <xsd:enumeration value="Not Provisioned"/>
34:               <xsd:enumeration value="Pending"/>
35:             </xsd:restriction>
36:           </xsd:simpleType>
37:         </xsd:attribute>
38:     </xsd:complexType>
```

To better understand this example, refer to the following illustration, generated by XML Spy.

FIGURE 1. Graphical display of <xsd:complexType> in XML Spy



The USPhoneNumberType is comprised of a sequence of elements. Each of these elements has a name (AreaCode, Prefix, Number), a simple type, and some restrictions. The dotted line around the area code indicates that it is optional.

The diagram **does not show** the XML Schema **simple data types** used for each of the elements. For that you must look at the source, or use one of the other XML Spy windows. The USPhoneNumberType also defines an **attribute** named “status”. XML Spy does not represent attributes graphically either.

Creating an Element

An XML vocabulary describes the elements and attributes that may appear in a “valid” document instance. In XML Schema the xsd:element element is used to represent an element of the vocabulary. (*Now that’s a lot of elements!*)

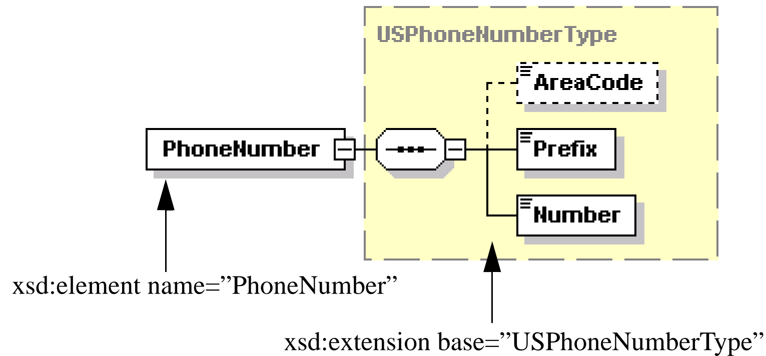
The listing below shows the format of an element definition. In this case the element is a phone number.

Listing 12: Phone Number Element

```
 1:    <xsd:element name="PhoneNumber">
 2:      <xsd:complexType>
 3:        <xsd:complexContent>
 4:          <xsd:extension base="USPhoneNumberType">
 5:            <xsd:attribute name="type" use="default"
 6:                                            value="home">
 7:              <xsd:simpleType>
 8:                <xsd:restriction base="xsd:string">
 9:                  <xsd:enumeration value="home"/>
10:                  <xsd:enumeration value="work"/>
11:                  <xsd:enumeration value="fax"/>
12:                  <xsd:enumeration value="business"/>
13:                  <xsd:enumeration value="mobile"/>
14:                </xsd:restriction>
15:              </xsd:simpleType>
16:            </xsd:attribute>
17:          </xsd:extension>
18:        </xsd:complexContent>
19:      </xsd:complexType>
20:    </xsd:element>
```


Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

FIGURE 2. **Graphical display of <xsd:element> in XML Spy:**



Creating an Attribute

Attributes are created using the <xsd:attribute> element of the XML Schema vocabulary. Attributes may be defined as part of a type or as part of an element.

Attributes may be grouped together to aide in reuse across types.

Refer to earlier listings for complexType and element to see examples of using the attribute tag.

Taking Shortcuts with XML Spy

When creating an XML Schema it is tempting to use XML Spy to "generate" the schema for you based on an existing document. While this does work, the schema generated by XML Spy will not describe a general class of documents. It will specifically describe the source document.

One area where this can be a particular problem is in the use of attributes. For example, when converting the alice30.xml file from the first homework assignment the resulting schema contained the following definition.

Listing 13: Partial alice30.xsd generated by XML Spy

```
1:     <xsd:complexType name="chapterType">
2:       <xsd:sequence>
3:         <xsd:element name="para" type="paraType"
4:                      minOccurs="1" maxOccurs="unbounded"/>
5:       </xsd:sequence>
6:       <xsd:attribute name="number" use="required">
```

```
 7:         <xsd:simpleType>
 8:           <xsd:restriction base="xsd:NMTOKEN">
 9:             <xsd:enumeration value="I"/>
10:             <xsd:enumeration value="II"/>
11:             <xsd:enumeration value="III"/>
12:             <xsd:enumeration value="IV"/>
13:             <xsd:enumeration value="IX"/>
14:             <xsd:enumeration value="V"/>
15:             <xsd:enumeration value="VI"/>
16:             <xsd:enumeration value="VII"/>
17:             <xsd:enumeration value="VIII"/>
18:             <xsd:enumeration value="X"/>
19:             <xsd:enumeration value="XI"/>
20:             <xsd:enumeration value="XII"/>
21:           </xsd:restriction>
22:         </xsd:simpleType>
23:       </xsd:attribute>
24:       <xsd:attribute name="title" type="xsd:string"
25:                                     use="required"/>
26:     </xsd:complexType>
```

Notice that the attribute called “number” is only permitted to hold a value between 1 and 12. This is rather restrictive for a schema that is intended to allow for representing electronic books. Furthermore, the values must be given as Roman numbers. What happend here is XML Spy looked at the document, pulled all of the values used for this attribute, and created a type accordingly. This is probably not what you really want, so this will need to be hand edited to correct the problem.

The listing below shows how this complexType can be changed to support any number of chapters in an ebook. This change requires that the chapter numbers in alice30.xml be converted to standard numbers from Roman numbers. The minimum inclusive value for the chapter number is one (1). No maximum inclusive value is specified. This results in there being no upper bound on the values possible for chapter number.

Listing 14: Improved chapterType Definition

```
 1:     <xsd:complexType name="chapterType">
 2:       <xsd:sequence>
 3:         <xsd:element name="para" type="paraType"
 4:                               maxOccurs="unbounded"/>
 5:       </xsd:sequence>
 6:       <xsd:attribute name="number" use="required">
 7:         <xsd:simpleType>
 8:           <xsd:restriction base="xsd:positiveInteger">
 9:             <xsd:minInclusive value="1"/>
10:           </xsd:restriction>
```


Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

```
11:         </xsd:simpleType>
12:       </xsd:attribute>
13:       <xsd:attribute name="title"
14:                     type="xsd:string" use="required"/>
15:     </xsd:complexType>
```

Sequences and Choices

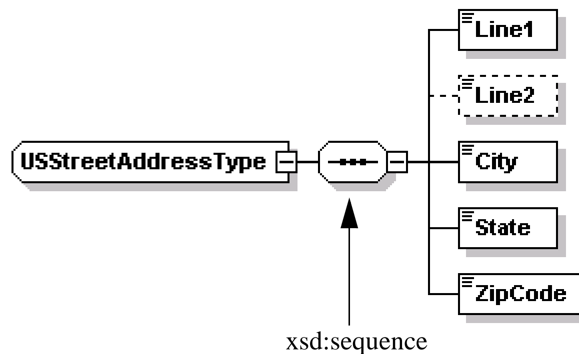
Much of the flexibility of a complexType is derived from two XML Schema constructs. These are the xsd:sequence and xsd:choice.

A sequence represents a **group of child elements** that appear in a particular order. The order is specified in the definition of the complex type. **Some elements** listed in the sequence may be **optional**. The following listing and figure illustrate a sequence of elements for a US street address.

Listing 15: Example of xsd:sequence (US Street Address)

```
 1:   <xsd:complexType name="USStreetAddressType">
 2:     <xsd:sequence>
 3:       <xsd:element name="Line1" type="xsd:string"/>
 4:       <xsd:element name="Line2" type="xsd:string"
 5:                                     minOccurs="0"/>
 6:       <xsd:element name="City" type="xsd:string"/>
 7:       <xsd:element name="State" type="xsd:string"/>
 8:       <xsd:element name="ZipCode" type="xsd:string"/>
 9:     </xsd:sequence>
10:   </xsd:complexType>
```

FIGURE 3. Graphical display of <xsd:sequence> in XML Spy



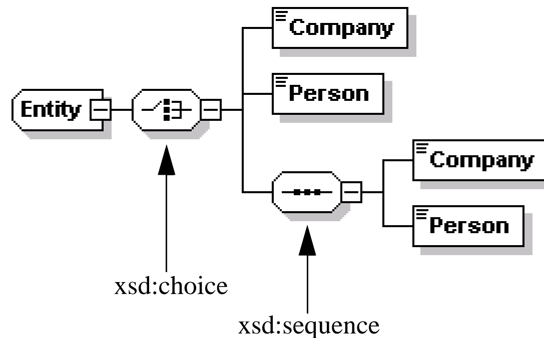
Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.

A choice is used to define a group of elements where one of them will appear within the document instance at this point. The following listing and figure illustrate the use of a choice in a complexType. The type represents an entity in a contact list as either a person, a company, or a person within a company.

Listing 16: Example of xsd:choice (Entity)

```
 1:   <xsd:complexType name="Entity">
 2:     <xsd:choice>
 3:       <xsd:element name="Company" type="xsd:string"/>
 4:       <xsd:element name="Person" type="xsd:string"/>
 5:       <xsd:sequence>
 6:         <xsd:element name="Company" type="xsd:string"/>
 7:         <xsd:element name="Person" type="xsd:string"/>
 8:       </xsd:sequence>
 9:     </xsd:choice>
10:   </xsd:complexType>
```

FIGURE 4. Graphical display of<xsd:choice> in XML Spy:



 Copyright (c) 2001 - 2004, iDevelopSoftware, Inc.