Introduction to Chapter Title

Objectives

- Define XML
- Review the history of XML
- Discuss the advantages of XML
- Explain XML's growth on the Internet
- Review and explain basic XML terminology.
- Define the structure of an XML document.
- Learn how to create XML documents.
- Look at the feature set of XML.

Just What is XML Anyway?

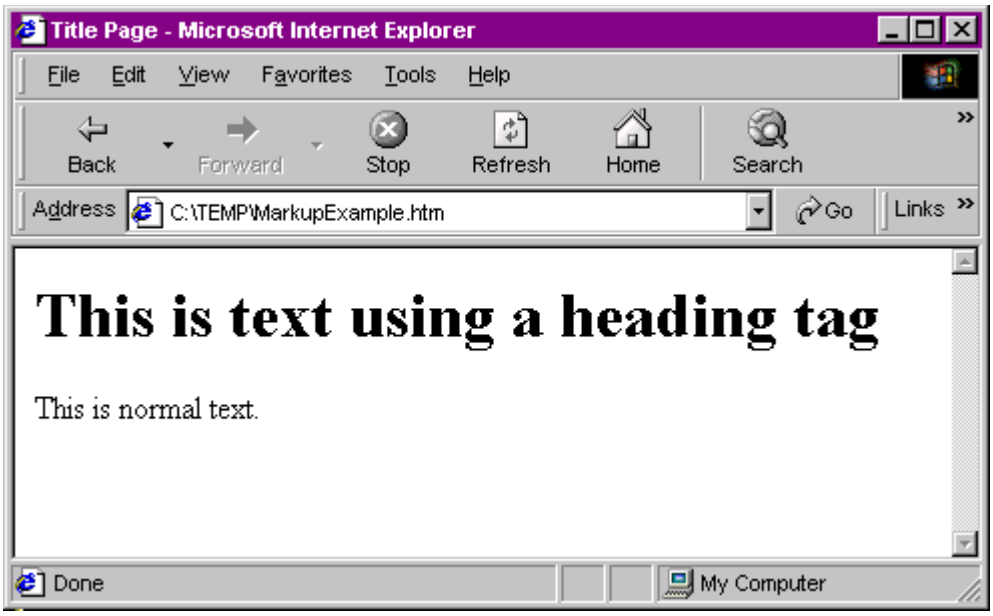
XML is an acronym for *eXtensible Markup Language*. Like its predecessor Hyper Text Markup Language, or HTML, it has its root in a standard known as the *Standard Generalized Markup Language*, or SGML. To understand why we need XML, let's take a look at both SGML and HTML. Understanding their respective strengths and weaknesses will shed some light on why XML exists and why it is gaining such strong momentum.

Markup Languages

Before we talk about the features of a markup language, let's define what a markup language it. In simple terms, a *markup* refers to the use of characters within a piece of information that can be used to process or identify that information in a particular way. A good example of a markup can be seen in an HTML document. The following example uses greater-than and less-than symbols to identify markup elements, or tags, that have specific purposes.

```
<HTML>
   <HEAD>
      <TITLE>Title Page</TITLE>
   </HEAD>
   <BODY>
      <H1>This is text using a heading tag</H1>
      This is normal text.
   </BODY>
</HTML>
```

If the above HTML document was viewed in a browser, the browser would interpret the markup elements and display the content to reflect the authors intentions. For example, the <H1> and </H1> tags are used to display the text in a large font, whereas the text immediately following it is displayed in the browser's standard font.



There are a number of different markup languages and types. We are going to review three of the most common.

SGML

Standard Generalized Markup Language (SGML) was designed as a standard way to store data independent of any software application or platform. SGML is often referred to as a *meta language*. Meta languages are languages that are used for describing *markup* languages. HTML is a derivative of SGML and is therefore called an SGML application. There are a number of languages based on SGML. There are also a number of standard data formats based on SGML.

The real power behind SGML is its ability to declare *Document Type Definitions,* or DTD's, which we will discuss in detail later. For now, it is only important to understand that SGML provides the ability to define the contents of the document, its markup characteristics, and its information model.

The downside to SGML is that it is all encompassing, and has a lot of rules. It has so many aspects to it that it is almost impossible to implement all of them. For this reason, SGML is rarely used by itself. Instead, subsets have been created that target niche applications and needs.

HTML

Hyper Text Markup Language (HTML) is the first internationally accepted derivative of SGML. HTML is really the document language of the World Wide Web. However, HTML is rather limiting. It was originally designed to represent document based data within a browser in very basic form. It has

evolved over time to support application features through the use of JavaScript, Java Applets, and the inclusion of client-side plug-ins, but as a whole it is still very basic. Furthermore, it is not a good language for applications to store and share data. One of the primary reasons for this is its lack of support for DTD's. Remember DTD's are external elements that define the contents and structure of the data. HTML's structure is extremely limited compared to its predecessor SGML.

Then again, that is not what HTML was designed for. It was designed as a document language, not a data language. This is where XML enters the picture.

XML

XML incorporates many of the features of SGML while learning from the limitations of HTML. Like SGML, XML utilizes DTD's, making it flexible and extensible. The goals of XML as defined by its creators were more focused than those of SGML making it much easier to implement. These goals included:

- XML could be used with existing Internet protocols (HTTP, MIME, etc.). This makes it the ideal format for sharing information on the Internet.
- XML support is application independent. Any application can utilize and support XML documents.
- XML is platform independent. Its use of technologies such as Unicode make it portable across machine types.
- XML is license free. It is controlled by an international standards organization. This means that it isn't going to cost you anything to use it.
- XML is compatible with SGML.
- The feature set of XML was kept to a minimum so that applications could support it. Compare this goal with that of SGML.
- XML is a family of technologies. XML has already evolved to include support for such things as style sheets, hyperlinks, and the Document Object Model (DOM).

XML takes the best of SGML (structured data definition capabilities) and the best of HTML (web addressing) . The result is a portable, highly usable, markup language that can be used by any number of applications to store and share structured data. Applications that will benefit or are already benefiting from XML include:

- Office applications (word processors, spreadsheets, etc.)
- Web applications (browsers, e-mail, etc.)
- Server applications (database servers, e-mail servers, etc.)

At its core, XML appears very simple. However, the implications of its use are very complex. It is already changing the way that people store information and build applications. Microsoft, Netscape, Sun, and many others are already using XML today in their applications, database servers, and e-commerce platforms.

XML Processors

An XML Processor is a piece of software, either an application or a library, that can process XML. A good example of an XML processor is XML document validation software. There are a number of such packages available for free and for sale on the Internet. Such applications can be used to validate the contents of an XML document and make sure that it is *well-formed*. A well-formed document is one that adheres to the rules of XML and any associated DTD's.

Other good examples include XML document viewers and XML document code libraries that can be used by software that you create to manipulate XML documents.

XML Evolution

XML is still evolving. There are a number of derivatives and extensions to XML that are being used today. Some of these include:

- **eXtensible Style Sheets (XSL/XSLT)**

  XSL is a technology by which you can embed XML within an HTML page and have the HTML processor (browser) populate the contents of the page using the embedded XML. This is a very powerful technology, although not many browsers currently support it. This is one of the most exciting XML implementations as it directly affects the way that data can be presented on the World Wide Web.

- which

What is XML Used For?

XML has created a quite revolution on the Internet. It is the first truly portable data format that was designed for Internet and multi-language support. The number of applications for XML are limitless. Here are just a few of the areas that XML has gained momentum.

Business to Business E-Commerce

This is one of the areas that XML has moved most rapidly. Getting businesses to talk the same data "language" has always been difficult. Differences in software and hardware platforms have always been big issues for companies that want to communicate electronically.

XML Vs. EDI

The first initiative to solve this problem was a standard called *Electronic Data Interchange*, or EDI. While the goals of EDI where lofty, the implementation was less than perfect. For many companies, implementing EDI in their processes was not something that yielded much in the way of results. The same issues of differences in hardware and software still existed Differences in data representation between organizations was another major hurdle. Having a standard for content modeling that could be interrogated by software was not part of the EDI model. This required that the software that would use the EDI output would have to understand its content model inherently. Each company had one or more content models that it would create for different things (e.g. purchase orders, proposals, etc.) . Things got real complicated real fast. A good analogy is that of the telephone. The problem with EDI was that each company that wanted to talk to each other had to reinvent the telephone for each type of discussion they wanted to have. Needless to say, EDI was not the panacea that many envisioned it to be.

XML addresses these issues in the way that it was designed. It is platform independent, it is structured, and it has a mechanism for strict content modeling that can be interrogated by any software using a standard XML processor. The result is that companies can communicate with each other using XML and engage in business to business e-commerce without having to worry about hardware and software platform issues, or content modeling issues.

Many large organizations are now using XML for such things as requests for proposals, purchase orders and transaction records to name a few. Since XML is designed for use with Internet protocols, the Internet has become the medium for transferring this XML data back and forth. The Internet is becoming the ideal place to do business to business e-commerce.

Catalogs

XML makes the perfect storage and transfer mechanism for information such as catalogs. This includes not only catalogs like those used for e-commerce, but also things like parts and inventory. Companies can use XML to keep lists of information (catalogs) that can be shared and transferred between multiple departments, managers, outside vendors, etc.

Data Wherehousing and Archiving

Storing large amounts of information for access by multiple applications is known as Data Wherehousing. XML can be used to store such information in usable chunks (documents) that can be retrieved from anywhere on the network or Internet, allowing users to get the information from anywhere. This is especially useful when sales teams who travel need to get at pertinent information.

Another use of XML is that of data archiving. A good example is the archiving of relational database data. Relational databases are extremely effective at storing and retrieving information (data) quickly. However, lots of infrequently used data can slow them down. Why not store such data in XML documents, leaving only the data that is commonly used within the relational database? This allows the database server to do what it does best while making the older data still available.

Data Migration

Migrating data from one system to another has been the bane of application programmers for years. Developers have often resorted to creating custom file formats that can be used to exchange data between systems. The problem however is that each system must understand the format of the data if they are going to be able to share such information.

XML is the perfect mechanism to be used for data migration. By storing application data in XML, other applications can interrogate and query such information using XML's open standards. Each application needs to know how to read one type of data storage, namely XML. This makes it much easier for application developers as they can use XML to export and import data.

A good example of this is migrating data from one database type to another (e.g. SQL Server to Oracle). By outputting the data to XML, it can be accessed by an application that knows how to process XML and put import the data into the target application or software.

Is XML just for Programmers?

No. XML is making its way into all kinds of Internet technologies, including HTML. Microsoft is leading the way in incorporating XML into its browser. Starting with Microsoft Internet Explorer 4.0, you can create XML *data islands* within your HTML that can be used to dynamically update data within the document instead of having to retrieve it from the server. It also incorporated XML support inside of the Document Object Model (DOM), making it easy to request XML documents from the server without having to constantly refresh your HTML pages.

It is expected that both Netscape Navigator and Internet Explorer are going to increase their support for XML as new versions of their respective browsers become available. Before long, XML will be as common to Web content providers and authors as HTML currently is. Since XML is truly extensible, do not be surprised to see XML versions of HTML documents supported in the future.

XML is becoming the standard for all kinds of data storage. New word processors and spreadsheets are outputting their data in XML, making it much easier to import and export data between platforms and applications.

XML is not just for programmers, even though most users will never directly interact with it. It is a world wide standard that makes data storage and transfer much easier and reliable than ever before.

XML References

There are a number of references on-line to help you understand XML and keep up to date on the tools and technologies that support it.

- **World Wide Web Consortium** (http://www.w3c.org/xml)

The W3C is maintaining the standard for XML. This is a great site to periodically visit to keep up to date on what is going in the world of XML.

- **Microsoft** (http://www.microsoft.com/xml)

Microsoft was one of the first to jump on the XML bandwagon with Internet Explorer 4.0 and has incorporated it into their office applications, developer tools, and e-commerce platforms.

- **XML.COM** (http://www.xml.com)

XML.COM is a great resource for articles relating to the use of XML, application development and XML, XML processors for download, and XML standards.

- **XML.ORG** (http://www.xml.org)

XML.ORG is the self-proclaimed XML "portal". Lot's of articles and reference material on XML standards and implementations as well as links to downloadable XML processors and software. A good site to visit when you are interested in what is going on in the XML community.

Anatomy of an XML Document

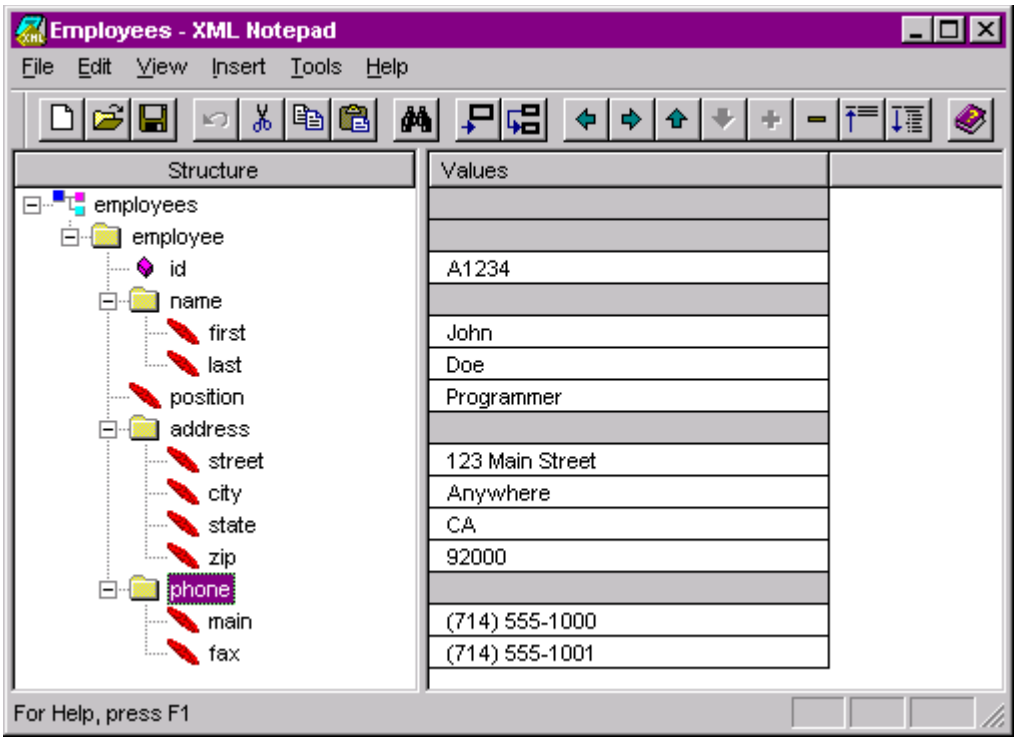
Before we go any farther, let's look at an example XML document. The following document is a sample that stores employee information.

```
<?xml version="1.0"?>
<employees>
   <employee id="A1234">
      <name>
            <first>John</first>
            <last>Doe</last>
      </name>
      <position>Programmer</position>
      <address>
            <street>123 Main Street</street>
            <city>Anywhere</city>
            <state>CA</state>
            <zip>92000</zip>
      </address>
      <phone>
            <main>(714) 555-1000</main>
            <fax>(714) 555-1001</fax>
      </phone>
   </employee>
</employees>
```

File: Chapter 2\Employees.xml

As you can see, XML is very easy to read and understand. If you have spent time creating documents in HTML, you can see how XML is both similar and very different. It is similar in the fact that it is a markup language and uses tags. The tags however are not HTML tags. Unlike HTML, you can create your own tags as we have done here.

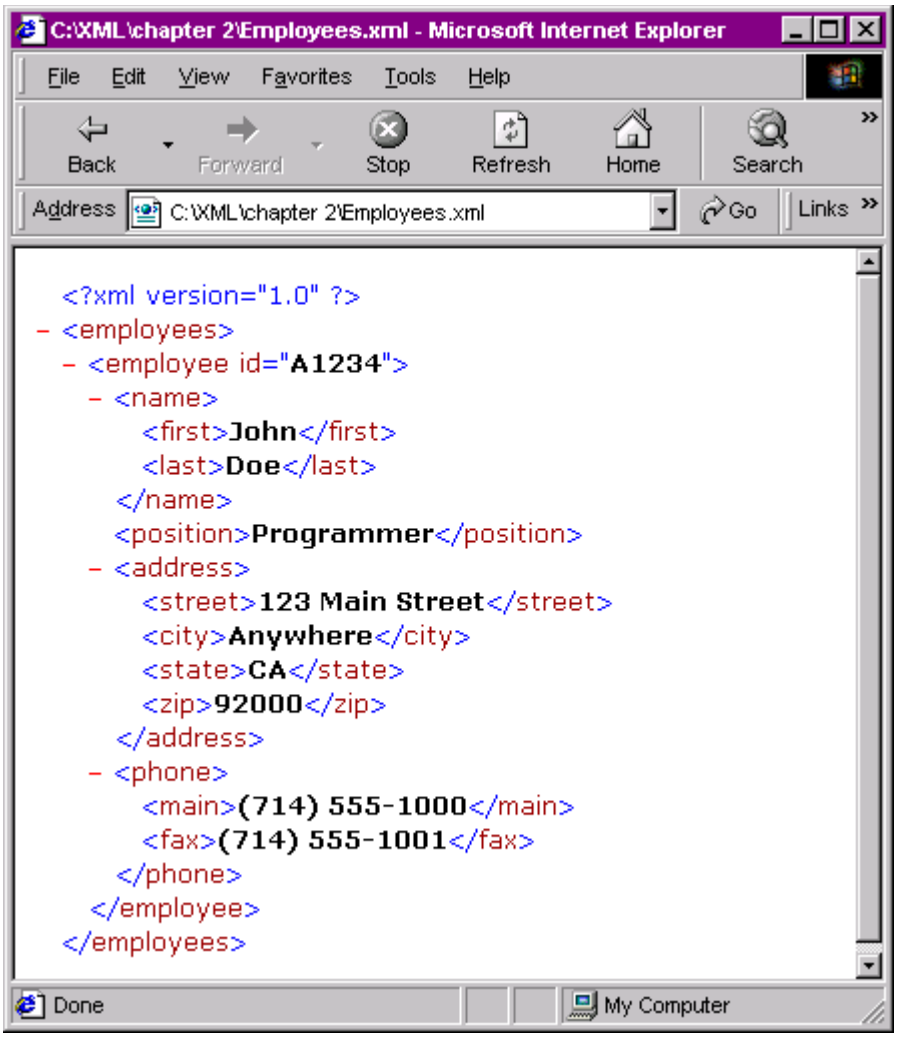
We can view this document in an XML viewer such as Microsoft's XML Notepad and see the logical structure of our XML document.



File: Chapter 2\Employees.htm in XML Notepad

The Microsoft XML Notepad is available for free download from the Microsoft web site at http://msdn.microsoft.com/xml/notepad/download.asp. It is a simple tool for creating and manipulating XML documents.

Another way to view an XML document is by using a browser that supports XML such as Internet Explorer 5.0 (IE5). When we open the same XML document using IE5, we get a view more consistent with how the document is physically formed.



File: Chapter 2\Employees.htm in IE5

IE5 is also available for free download from the Microsoft web site at http://www.microsoft.com/ie.

XML Terminology

As with all technologies, XML has its own terminology. Before we get into creating XML documents, it is important that you understand the terms that are going to be used throughout the rest of this course.

Tags

If you are already familiar with HTML, then you are already familiar with the concept of tags. If not, a tag is an identifier to an element, which we will discuss next. Just like HTML, tags are identified using less-than (<) and greater-than (>) symbols. These symbols are considered *markup* and the text between them is known as the tag. Take a look at the following HTML example:

```
<title>This is the title of my HTML page</title>
```

The tag we are using here is the "title" tag. Notice that it has a start tag (<title>), and a different version of the tag at the end (</title>). The slash in the second tag is called a *terminator tag*. The difference between HTML tags and XML tags is that in XML you can name the tags what you want (provided you follow some basic rules). In our example XML document, we have defined lots of tags. In the following example, there are three different tags that are defined: <name>, <first> and <last>.

```
<name>
   <first>John</first>
   <last>Doe</last>
</name>
```

It is very important to understand that in XML you have control of the tag names. There are only a few rules that you have to follow when it comes to naming your tags.

1. The tag name must contain one letter (A-Z or a-z).

2. The tag name can contain digits, but they cannot be the first character.

3. Providing that you adhere to rule 1, you can begin your tag name with an underscore (_) or a colon (:).

4. Spaces and/or tags are not allowed in tag names.

5. The only punctuation signs that you can use in your tag name are the hyphen (-) and a full stop (.). You cannot use underscores to separate long names (e.g. <this_is_my_long_tag_name>). Even though an underscore is allowed as a first character, it is not allowed anywhere else. However, you can use a hyphen or full stop to get the same effect (e.g. <this.is.my.long.tag.name> or <this-is-my-long-tag-name>).

6. Use tag names that make sense. This isn't really a rule because it can not be enforced by an XML processor. However, I recommend that it be a self-imposed rule. One of the features of XML is its readability. The following XML would be well-formed, valid, and perfectly usable, but it is far from readable.

```
<ab>
   <cd>David</cd>
   <_>Doe</_>
</ab>
```

Other than these enforcing these simple rules, XML let's you name your tags as you please.

Elements

In XML documents, data is stored in elements. Elements are identified using a start tag and an end tag. The name of the tag is defined by the author. For those who are familiar with HTML, you immediately can see the difference between HTML and XML in the way that the tags are named. In HTML, you have a predefined set of tags to work with. In XML, you can create your own tag names so that it makes logical sense for what you are doing.

In our example document, we have several elements. For example, the following element called <position> contains a value of "programmer". Note that the end tag uses a backslash ("/") to denote the end of the element.

```
<position>Programmer</position>
```

Elements can be empty, meaning that they do not contain any data. For example, if we did not have a value for the <position> element it would look like this:

```
<position></position>
```

In XML, it is not necessary to use two tags to represent an empty element. Instead, you can use an *end tag*, which allows you to create a start and an end tag within one element tag. An end tag has a backslash incorporated into it as shown:

```
<position/>
```

is equivalent to:

```
<position></position>
```

Attributes

Element tags can contain *attributes*, which give further information about the elements they delimit. An example of an attribute can be seen in the <employee> tag of our example document.

```
<employee id="A1234">
   . . . (contents omitted for brevity)
```

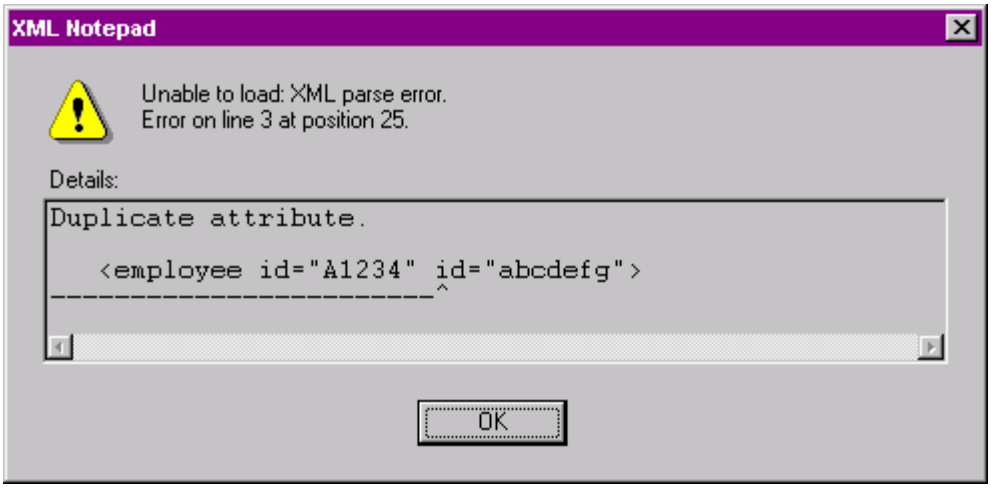
```
</employee>
```

In the above example, the <employee> tag has an attribute called "id". The "id" attribute contains a value of "A1234". Attributes can only be specified in the start tag of an element.

Duplicate attributes are allowed in XML. For example, the following statement is allowed, although it may not make much sense:

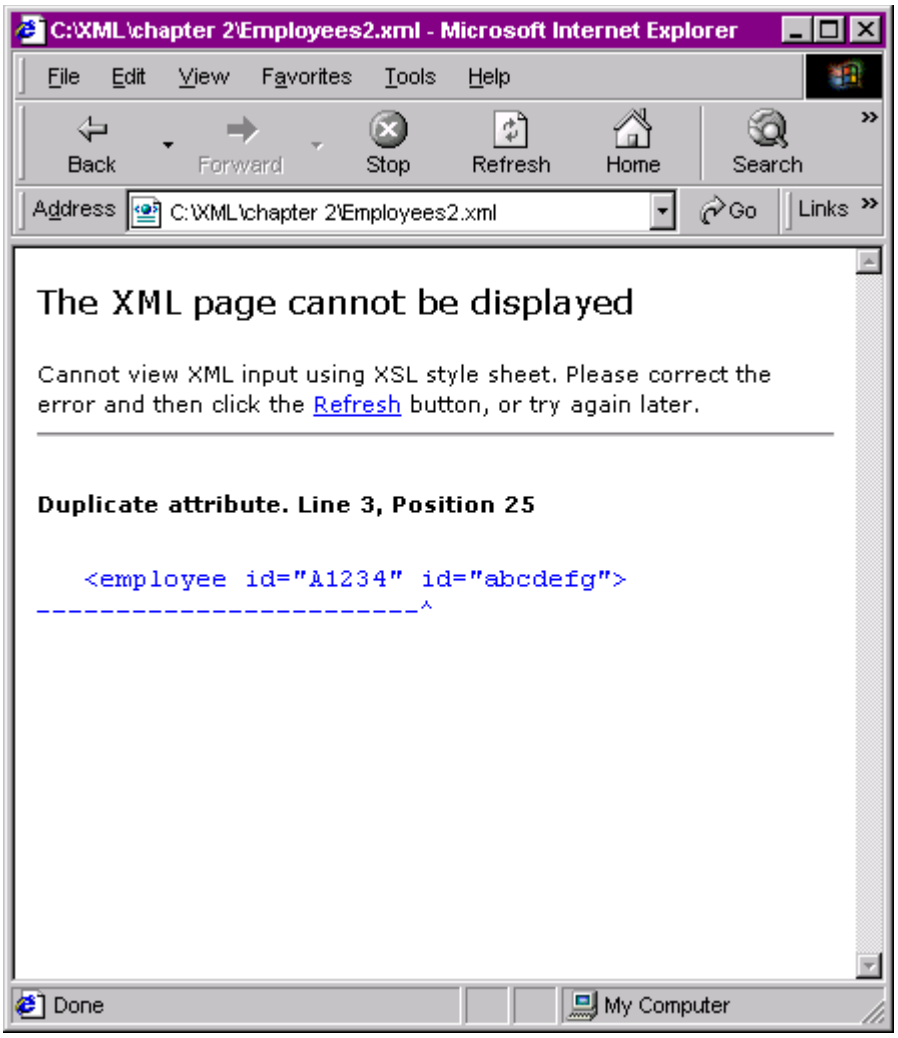
```
<employee id="A1234" id="abcdefg">
```

Unlike SGML and HTML, which consider this an error, the XML specification states that this should be accepted and handled by merging the two attribute values, and that the first declaration of an attribute is the one that can be referenced. Based on the specification, processing of an XML document should continue in this event. However, just like with HTML, viewers may not always implement the standard. Using the version of Microsoft XML Notepad that is available at the time of this writing, an error occurs when it encounters the above declaration.



File: Chapter 2\Employees2.xml in XML Notepad

The same is true when the document is loaded in IE5.



File: Chapter 2\Employees2.xml in IE5

In reality, it does not make sense to have multiple declarations of the same attribute. The point here is that the standard says that it should be allowed, even though certain XML processors may not adhere to the rules. This is not a new problem to SGML applications. This is very similar to the way that different browsers will interpret and display HTML documents slightly differently.

Entities

An *entity* has many applications in XML. The official definition of an entity is a *storage object*. The first XML entity that you have encountered so far is the XML document. The document as a whole is an entity. This entity is divided into elements.

There are other types of entities in XML as well. One of the most powerful capabilities of XML is its ability to include *external entities*. In other words, you can reference (and thus include) another file inside of your XML document. You have probably seen this already if you have created web pages with HTML. The <IMG> tag references an external graphics file. The graphics file is not embedded inside of the HTML document, but it references it and the browser requests it from the server.

```
<IMG SRC="./images/somepicture.jpg">
```

We can do the same thing in XML.

```
<picture.1 source="./images/somepicture.jpg"/>
```

In this example, the graphics file is considered an *unparsed entity* as it is not parsed by the XML processor.

You can also include other XML documents. The result is that the XML document is included in the document that referenced it and they are presented by the XML processor as one. This can be a very powerful feature, but beware its misuse. A good example is requesting a document that references another document that references another document, etc. You can easily see how things can get out of hand!

Tip:	Through the use of external references in XML, you can request a small document and get back a lot more than you intended if it has lots of external entity references.

Later, we are going to discuss *internal entities*, but for the remainder if this discussion, it is more important that you understand the external entity and that it is defined as a storage object.

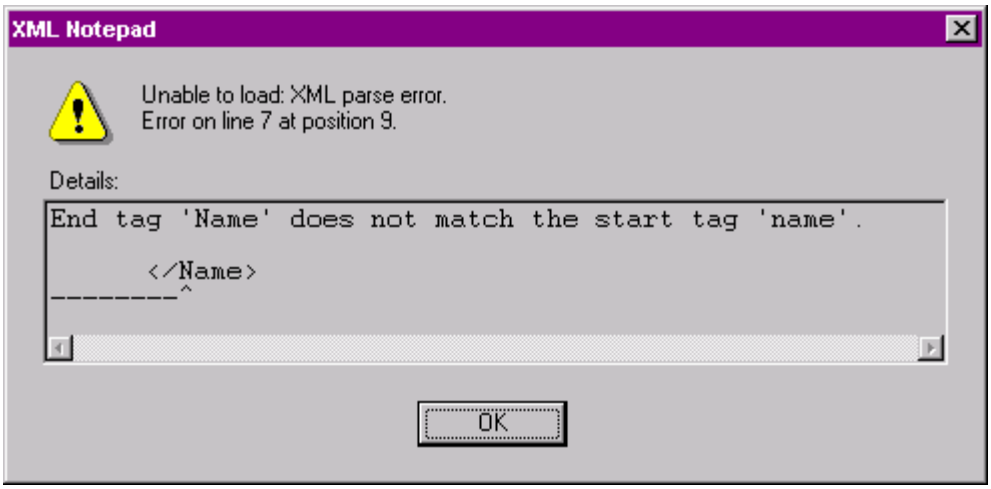
Case Sensitivity

XML is case sensitive, and this is a very important point for creating well-formed documents and for portability. What this means is that XML processors will distinguish upper case letters (A-Z) from lower case letters (a-z). For example, the following will result in an error because the start tag <name> and the end tag </Name> are not recognized as the same.

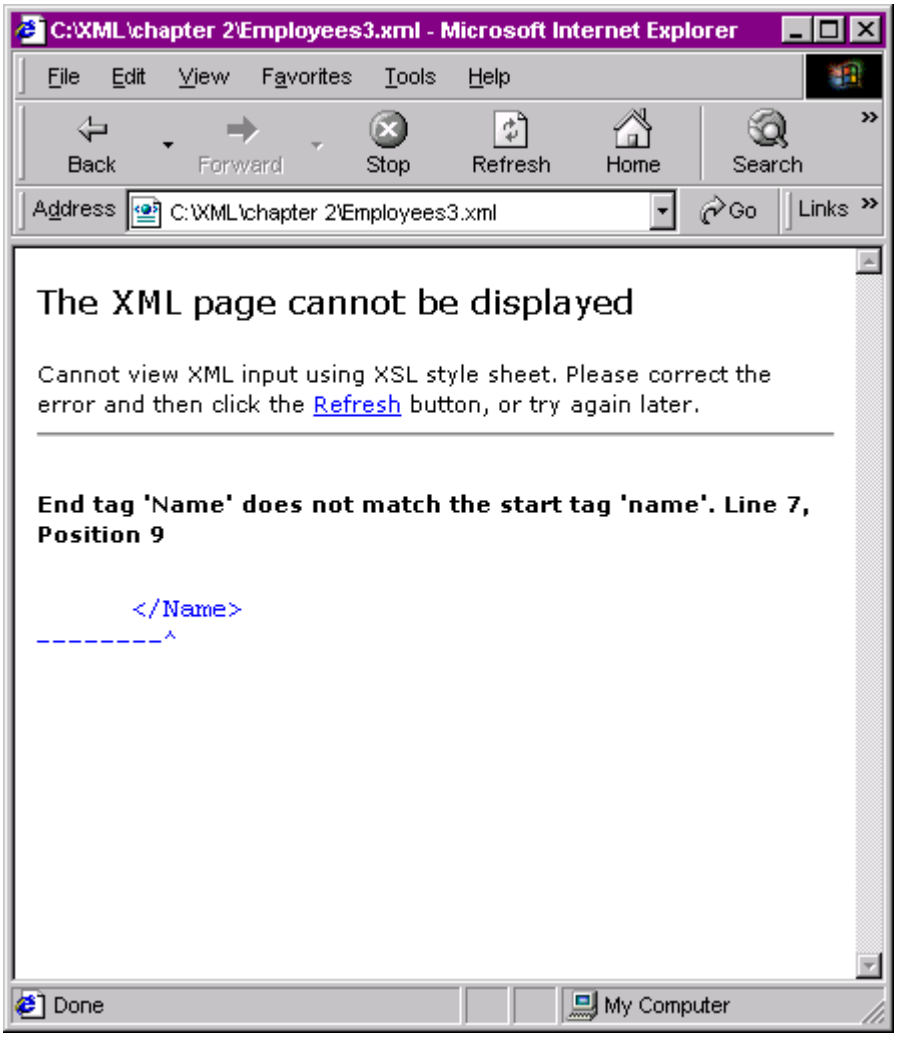
```
<name>
   <first>John</first>
   <last>Doe</last>
</Name>
```

File: Chapter 2\Employees3.htm code snippet

The above would result in an error regarding unmatched tags. The following screen captures show how both XML Notepad and IE5 react to the error.



File: Chapter 2\Employees3.htm in XML Notepad



File: Chapter 2\Employees3.htm in IE5

The reason that XML is case sensitive is very interesting. Most ASCII based text systems will convert text into upper case so that case-sensitivity is not an issue. However, XML is a portable standard, and by portable we mean language independent. This is the very reason why XML supports Unicode, and not ASCII. This makes it impossible to do any conversion with confidence since some character set conversions may behave differently than expected. XML defaults to lower case. It is recommended that you always use lower case as well.

Tip:	Always use lowercase instead of mixed case. This results in consistency throughout your code and makes it more portable.

Comments

Yes, XML has comments, and it is always a good idea to use them. Why comment? If you are a programmer, you already know why. If not, understand that at some point, someone is going to have to look at some XML document you did and figure out why you did it that way (beware, it may just be you!). Getting in a good habit of commenting what you do will save you lot's of time and frustration in the long run.

The syntax for comments is:

```
<!-- This text is a comment -->
```

The start of a comment is represented by the <!-- characters, while the end of the comment is represented by the -->.

Warning:	Do not use spaces in the start and end tags of a comment. It will result in a misinterpretation of your document and may even result in unexpected behavior.

You can put comments anywhere inside of your XML document, as long as it is outside other markup and not within an element. The would result in an error:

```
<name <!-- employee name -->>
```

Comments can not be placed with elements. The following is not a comment, but in fact part of an element value. Do not be surprised to find that this line causes different behaviors in different XML processors.

```
<another>This is <!-- oops --> another error</another>
```

Anything within a valid comment block will be ignored by an XML processor. In the following example, the <name>, <first> and <last> tags are all ignored.

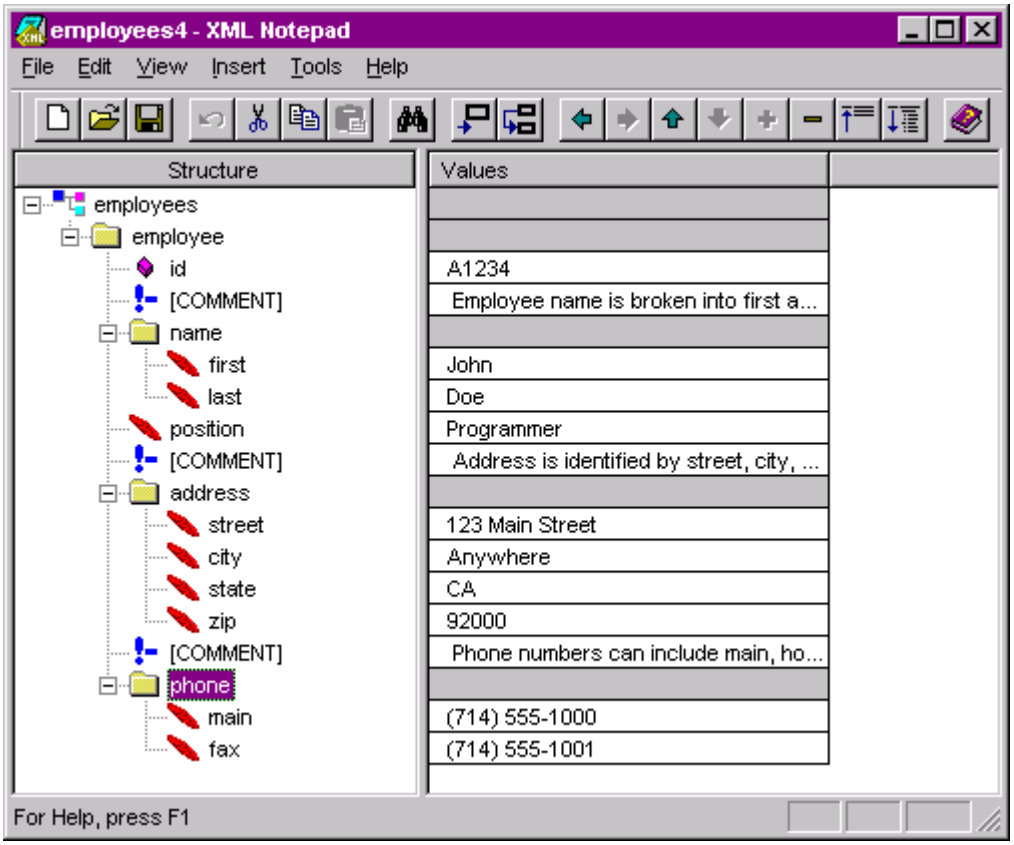
```
<!-- We have commented out the name elements.
```

```
<name>
   <first>John</first>
   <last>Doe</last>
</name>
-->
```

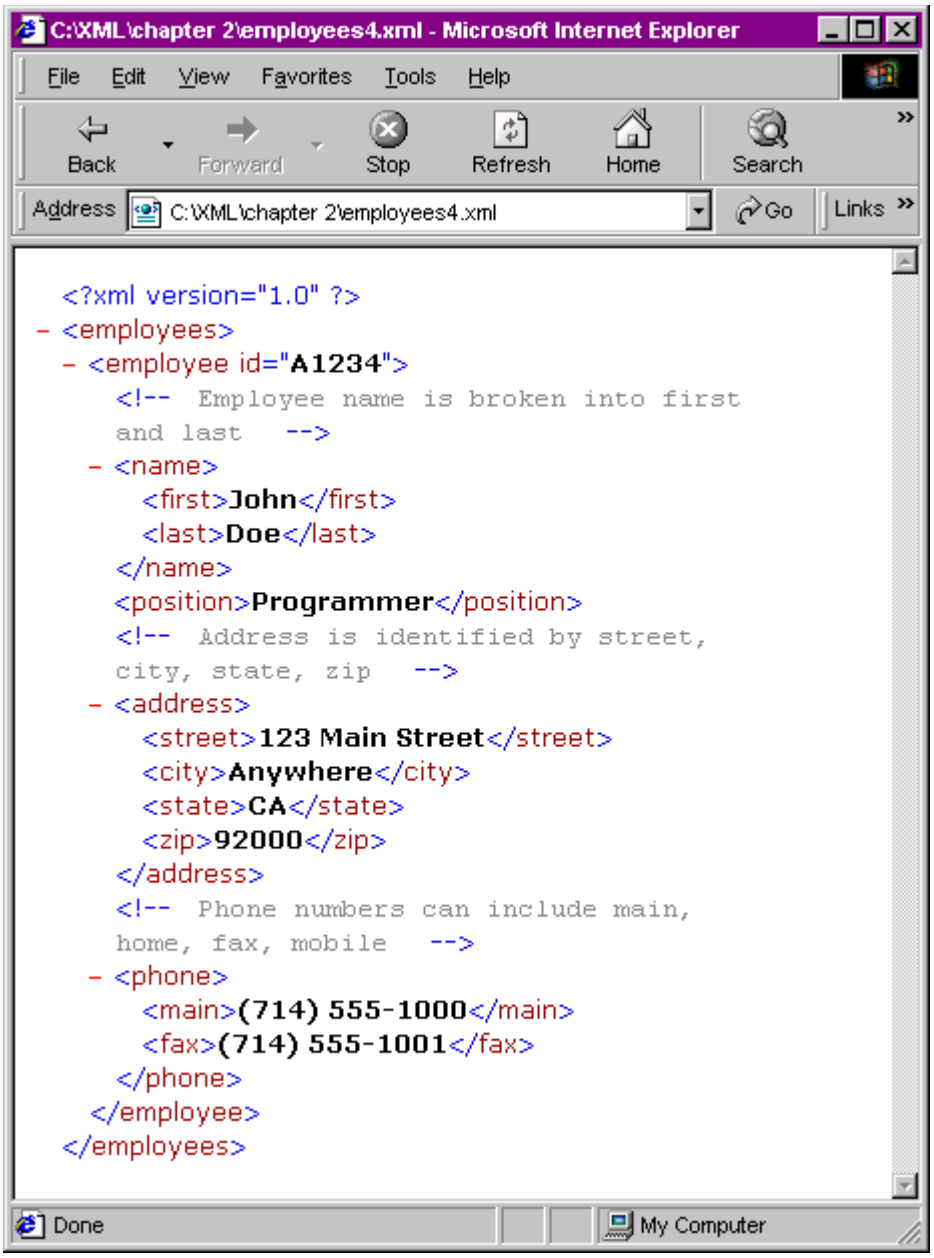
Do not use comments for anything other than documenting your code. It may seem advantages to stick something in a comment that a specific application could use later, but that is not what comments are for. In fact, it might not give you the results you expect.

Warning:	The XML specification does not require that comments be passed to an application. Some XML processors may strip the comments out before the application sees them.

It is important to comment your XML documents. Even though XML is designed to be readable, everything is relative. Even XML can look pretty complicated when you take advantage of all of its capabilities.



File: Chapter 2\Employees4.xml in XML Notepad



File: Chapter 2\Employees4.xml in IE5

Structure of an XML Document

Let's get started by reviewing the structure of our example breaking this document down and see at how it really works.

The XML Declaration

The first line in the example document is a *processing instruction* that identifies the document as an XML document type. Processing instructions are special types of instructions, and we will go into detail on how they are used later. Let's break the line down into parts.

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

The first part is the processing instruction code. This is the <? and ?> characters. The ? is the identifier that is used to specify this markup type. The second point of note is the "xml" statement. The third is the "version" attribute, which defines the version of XML that this document complies with. This instruction also specifies whether the document is stand-alone (as is the case with this example), or requires a separate DTD in order to make sense of the data contained therein.

Tip:	While the XML declaration is optional, it is a very good idea to always include it for maximum portability.

The Root Element

Each XML document must have a root element, and there can be only one. A root element is the element that encapsulates all other elements in the XML document.

In our example, <employees> is the root element. Notice that all other elements are located with the <employees> and </employees> tag.

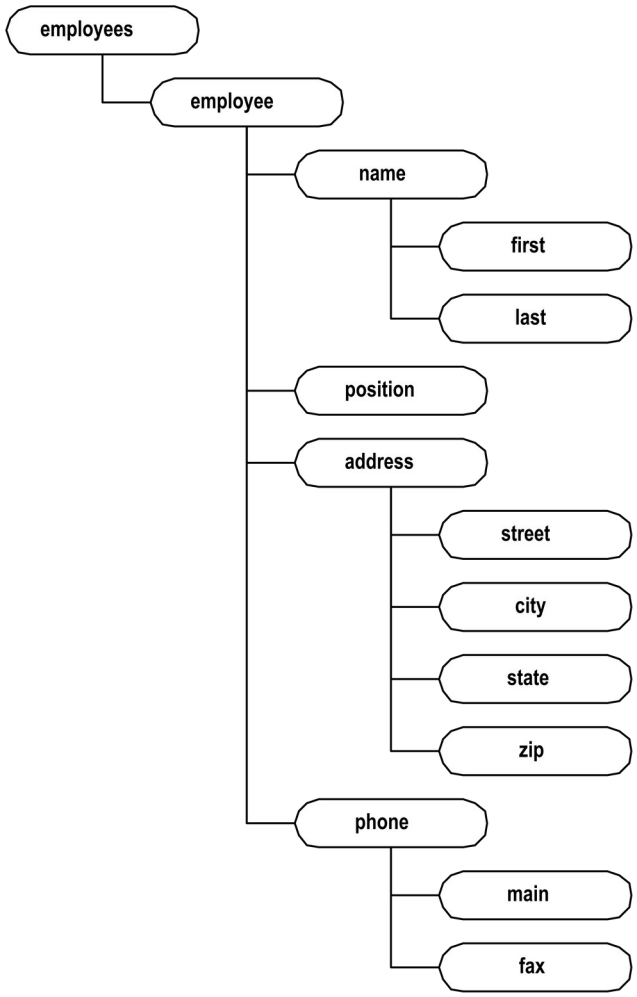
```
<?xml version="1.0"?>
<employees>
   . . . (rest of document omitted)
</employees>
```

The Logical Structure

In theory, there are two types of structure in XML. The first is the *logical* structure, the second is the *physical* structure. The logical structure has

nothing to do with the physical entities associated in an XML document, but instead has to do with the order of the elements that it contains. The logical structure is independent of the physical structure because the logical structure includes all external entities that may be referenced by an XML document.

The following diagram illustrates the logical structure of our employee example.



File: Chapter 2\Employee.xml Logical Diagram

A key concept in XML is the idea of *element relationship*. Elements are said to be related to each other in one of three ways: *parent*, *child, or sibling*. This is not an either/or model as you will see. Relationship is relative to the way that you are referring to an element.

Parents

Parent elements are elements that contain other elements. An example of a parent is the <name> element. <name> has two children: <first> and <last>. An element is a parent to the elements that it contains.

```
<name>
   <first>John</first>
   <last>Doe</last>
</name>
```

Child

Child elements are elements that are contained within a parent element. Using our previous example, both <first> and <last> are child elements to <name>.

```
<name>
   <first>John</first>
   <last>Doe</last>
</name>
```

Siblings

Siblings are elements that share a parent are called *siblings*. While <first> and <last> are children of <name>, they are also siblings to each other. Another example of siblings include <name>, <position>, <address> and <phone>.

XML and Databases

XML is *hierarchical* in nature. The concepts of parent, child, and sibling elements are not new to data storage and databases in general. Those who have used hierarchical databases like IMS and CICS will easily understand the logical structure of an XML document. For those who are familiar with newer databases that utilize the relational model, XML may seem a little odd.

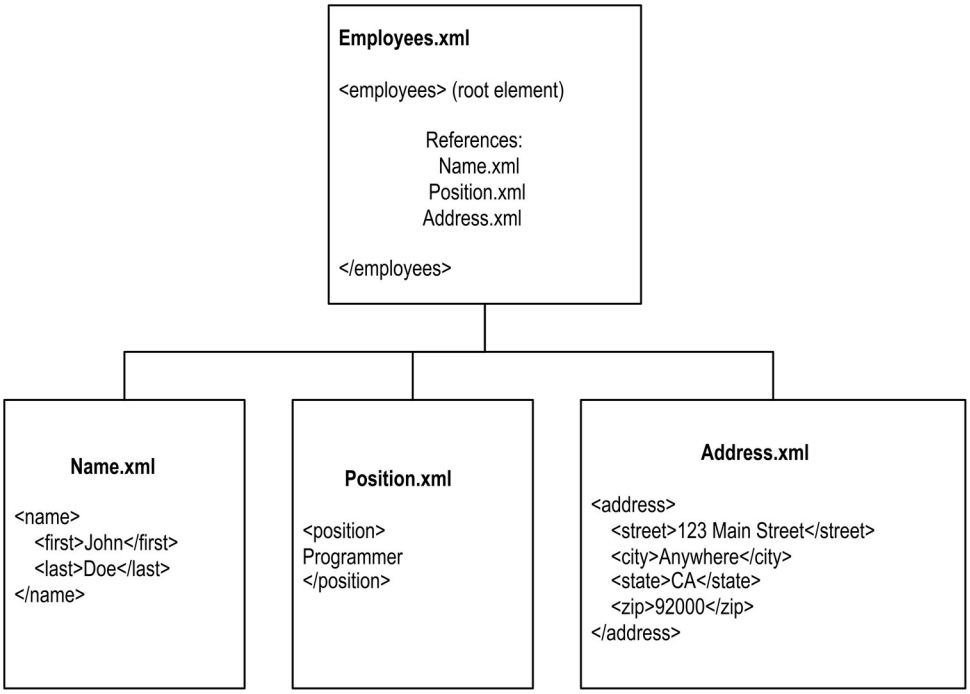
When we get into *content modeling*, you will see that it is easy to model both types of databases using XML.

The Physical Structure

Understanding the logical structure is fundamental to using XML within your software applications. Understanding how to manipulate the physical structure is fundamental to creating and maintaining XML documents.

If you will recall, the physical storage of an XML document is called an entity. An entity can reference another entity (i.e. one XML document references another XML document) and the result is a single logical structure of elements. This means that we could physically breakup our employee example into separate XML documents, make reference to them from a primary document,

and treat them as a single logical entity. The following is an example of how we could do this.



Being able to break a document's logical structure into multiple physical parts and reference them individually or from other documents is extremely powerful. It can also get you into a lot trouble. Let's say for arguments sake that Position.xml contained a </employees> tag. This would ruin the logical structure and cause the XML processor to be unable to process the document. The lesson here is that you need to be very careful in defining the contents of your physical structure so that your logical structure can remain intact.

> **Tip:** A good rule of thumb is to not have partial element definitions in your documents. In other words, do not begin an element in one document, and end it in another.

This is a good place to recall the need for clearly commenting your XML documents!

Synchronous Structures

It is absolutely critical to keep structures synchronous. In other words, your tags have to match up in correct order. Let's say that you are putting some HTML inside of an XML document and the HTML has some basic tags in it.

```
<text>This is <emph>italicized</emph> text.</text>
```

This is perfectly valid XML. However, if the <emph> tag was omitted like the following statement, the XML processor would not know how to handle it.

```
<text>This is italicized</emph> text<text>
```

Since there is no beginning <emph> tag, the XML processor encounters the closing tag and does not know how to handle it. The rule of XML is to always make sure that your tags match up so that the logical flow of the XML document is synchronous.

Code Listings

The following are the complete code listings for all of the samples used in this lesson.

Chapter 2\Employees.xml

```
<?xml version="1.0"?>
<employees>
    <employee id="A1234">
        <name>
            <first>John</first>
            <last>Doe</last>
        </name>
        <position>Programmer</position>
        <address>
            <street>123 Main Street</street>
            <city>Anywhere</city>
            <state>CA</state>
            <zip>92000</zip>
        </address>
        <phone>
            <main>(714) 555-1000</main>
            <fax>(714) 555-1001</fax>
        </phone>
    </employee>
</employees>
```

Chapter 2\Employees2.xml

```
<?xml version="1.0"?>
<employees>
   <employee id="A1234" id="abcdefg">
      <name>
         <first>John</first>
         <last>Doe</last>
      </name>
      <position>Programmer</position>
      <address>
         <street>123 Main Street</street>
         <city>Anywhere</city>
         <state>CA</state>
         <zip>92000</zip>
      </address>
      <phone>
         <main>(714) 555-1000</main>
         <fax>(714) 555-1001</fax>
      </phone>
   </employee>
</employees>
```

Chapter 2\Employees3.xml

```
<?xml version="1.0"?>
<employees>
   <employee id="A1234">
      <name>
         <first>John</first>
         <last>Doe</last>
      </Name>
      <position>Programmer</position>
      <address>
         <street>123 Main Street</street>
         <city>Anywhere</city>
         <state>CA</state>
         <zip>92000</zip>
      </address>
      <phone>
         <main>(714) 555-1000</main>
         <fax>(714) 555-1001</fax>
      </phone>
   </employee>
</employees>
```

Chapter 2\Employees4.xml

```
<?xml version="1.0"?>
<employees>
   <employee id="A1234">
      <!-- Employee name is broken into first
           and last -->
      <name>
         <first>John</first>
         <last>Doe</last>
      </name>
      <!-- This is a comment within the <name>
           element -->
      <position>Programmer</position>
      <!-- Address is identified by street, city,
           state, zip -->
      <address>
         <!-- This is another comment -->
         <street>123 Main Street</street>
         <city>Anywhere</city>
         <state>CA</state>
         <zip>92000</zip>
      </address>
      <!-- Phone numbers can include main, home, fax,
           mobile -->
      <phone>
         <main>(714) 555-1000</main>
         <fax>(714) 555-1001</fax>
      </phone>
   </employee>
</employees>
```