Chapter 4: Namespaces and DTDs

In this chapter you will:

- Organize your elements with namespaces
- Define your elements with Document Type Definitions (DTDs)
- Validate your XML documents against DTDs
- Declare elements in a DTD
- Declare attributes in a DTD

As you learned in Chapter 3, XML has no predefined elements or attributes. Instead, you must define your own elements, attributes, and document structure. Because you have to build everything in your XML documents from the ground up, you are actually writing your own markup language. One of the biggest challenges you will face with XML is deciding which elements and attributes to use in your new markup language, and how those elements should be structured. Simply creating new elements and attributes each time you need them works fine for XML documents that you will only use once. However, you may want to design an XML document that is used many times, or that should include specific elements, attributes, and structure expected by an application that needs to access the document's data.

In this chapter, you will study namespaces and Document Type Definitions (DTDs) to learn how to organize, define, and structure the elements and attributes in your XML documents. You will also learn how to validate your XML documents against a DTD.

Organizing Elements with Namespaces

One of XML's greatest strengths is that you can define your own elements in your documents. However, it is only a matter of time before you create an XML document containing elements with names that are identical to elements in another XML document. At first, you may not realize that this can cause problems, but remember from Chapter 3 that XML documents commonly define and transfer data between Web applications. If an application accesses two separate XML documents that contain identical element names, the application will have no way of differentiating them. Or, you may combine into a single document two separate XML documents, both of which contain elements with the same names, but with different purposes.

For instance, consider an element named `name`. The `name` element could contain the name of a person, an organization, a country, and so on. Suppose you have an XML document that contains multiple `name` elements, with each element storing data with a different meaning. For instance, one of the `name` elements in the document could store customer names, while another `name` element could store product names. Without some way to uniquely identify each `name` element, an application that is accessing the XML document will have no way of knowing which `name` element to use. Or, an application may need to access the `name` elements in your XML document, along with the `name` elements in someone else's XML document. Again, an application will have no way of knowing which `name` elements to use. To solve these problems, you use **namespaces** to organize the elements and attributes of an XML document into separate collections.

Namespaces and URIs

You should already be familiar with the basics of working with the Web, but to review, a **Uniform Resource Identifier (URI)** is a generic term for identifying namespaces and addresses on the World Wide Web. A **Uniform Resource Locator**, or **URL**, is a unique address that identifies a Web page. URLs are also referred to as Web addresses. For instance, *www.course.com* is a typical URL that points to Course Technology's home page. Namespaces are identified by a URI because it is guaranteed to be unique, which means that an associated namespace will also be unique. This allows any applications that use an XML document to clearly identify its elements and attributes, resolving any conflicts with identically named elements and attributes in other XML documents.

With URL namespaces, common practice is to include an `ns` folder in the URL name. (The `ns` stands for "namespace".) Beneath the `ns` folder, you create additional folders that uniquely identify individual namespaces. For instance, the following two Course Technology URLs could be used to identify two unique namespaces:

```
http://www.course.com/ns/catalog
http://www.course.com/ns/certification
```

One potentially confusing fact about namespaces is that a URL you use to identify a namespace does not need to exist. In other words, you do not actually need to create an `ns` folder or any subfolders on your server. If you do create an `ns` folder and subfolders for namespaces on your server, you can place anything you want into the folder or you can leave it empty; it makes no difference. The point is that a URL associated with a namespace only needs to be a unique name used to identify the namespace; it does not matter if it physically exists as a resource on your Web site.

Default Namespaces

A **default namespace** is applied to all of the elements and nested elements beneath the element that declares the namespace. You select a default namespace for an entire XML document by using the `xmlns` attribute in the document's root element. The **xmlns** attribute assigns a namespace to an element; to this attribute you assign the URI that you want to use as a namespace. For instance, the following XML document contains the hardware costs associated with the renovation of a company's offices. A default namespace for the document is created by assigning the URL *www.GosselinConsulting.com/ns/renovation* to the `xmlns` attribute in the `renovation` root element.

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<renovation xmlns="http://www.GosselinConsulting.com/
            ns/renovation">
   <hardware><description>plumbing</description>
     <cost>$15,000</cost></hardware>
   <hardware><description>electrical</description>
     <cost>$11,000</cost></hardware>
</renovation>
```

You can also apply a namespace to a particular element in an XML document. In this case, the namespace will be applied to all of the element's nested elements, with the exception of elements with explicit namespace declarations (which you will study next). For instance, you may want to use separate namespaces for each of the hardware elements in the preceding code. The following code shows how to assign two default namespaces: one for the plumbing <hardware> element and one for the electrical <hardware> element:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<renovation>
   <hardware xmlns="http://www.GosselinConsulting.com/
            ns/plumbing">
     <description>plumbing</description>
     <cost>$15,000</cost></hardware>
   <hardware xmlns="http://www.GosselinConsulting.com/
            ns/electrical">
     <description>electrical</description>
     <cost>$11,000</cost></hardware>
</renovation>
```

Next you will create a simple XML document that contains weather-related elements. You will apply a default namespace to the weather document's root element.

To create a simple XML document that uses a default namespace:

1. Start your text editor and create a document.
2. Type the opening XML declaration, as follows:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
```

3. Next, type the following root element named <weather> that uses the xmlns attribute to declare a default namespace. Replace the "Your_Name" portion of the domain name with your name. This namespace assumes that the weather being reported is for San Francisco. Notice that the default namespace includes an `ns` folder in the URL name.

```
<weather
xmlns="http://www.Your_Name.com/ns/SanFrancisco">
```

4. Type the following `<weather_reading>` element, which contains other nested weather elements:

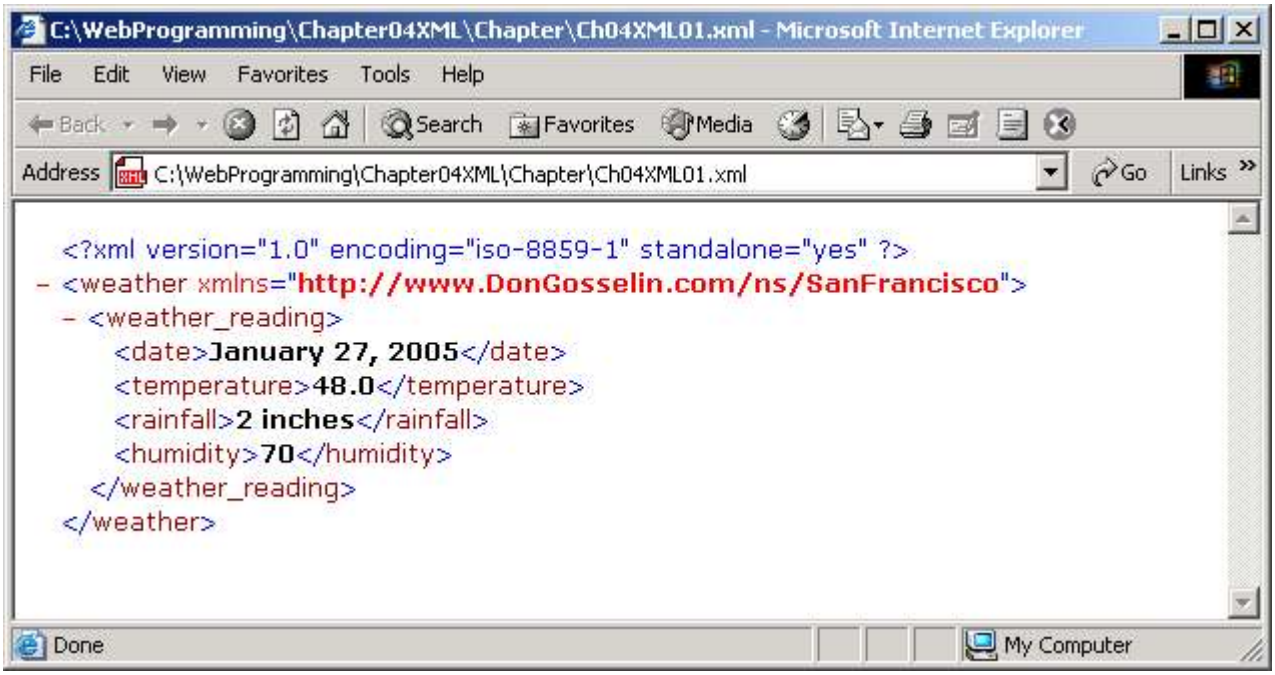
```
<weather_reading>
      <date>January 27, 2003</date>
      <temperature>48.0</temperature>
      <rainfall>2 inches</rainfall>
      <humidity>70</humidity>
</weather_reading>
```

5. Type the closing tag for the `<weather>` root element:

```
</weather>
```

6. Save the file as **Ch04XML01.xml** in the Chapter folder for Chapter 4.
7. Open the **Ch04XML01.xml** file in Internet Explorer. Your Web browser should look like Figure 4-1. If you did not create the document properly, fix the error that appears in the browser and reload the document.

Figure 4-1: Ch04XML01.xml file in Internet Explorer



8. Close your Web browser.

Explicit Namespaces

Sometimes you will want to explicitly declare a namespace for a specific element in an XML document. Namespaces that are assigned to individual elements in an XML document are called **explicit namespaces**. For instance, with the renovation XML document, you could probably get by with using a single default namespace for the `<hardware>` elements that relate to construction. However, as part of your renovation, you may want to upgrade your computer hardware. If you used `<hardware>` as the element name for your computer hardware, then you would want to use a separate namespace because computer hardware is quite different from construction hardware. To explicitly declare a namespace for a specific element in an XML document, you must assign a prefix to the namespace declaration using the following syntax:

```
xmlns:prefix="URI"
```

The following statement declares a computers prefix within the opening `<renovation>` tag for a namespace that represents computer hardware elements:

```
<renovation xmlns:computers="http://www.GosselinConsulting
      /ns/computers">
```

Usually, you place all namespace declarations, including default and explicit namespaces, within an XML document's root element. For instance, the following code

shows how to declare both a default namespace and an explicit namespace within the root element of the renovation XML document.

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<renovation xmlns="http://www.GosselinConsulting
      /ns/construction"
      xmlns:computers="http://www.GosselinConsulting
      /ns/computers">
      <hardware><description>plumbing</description>
            <cost>$15,000</cost></hardware>
      <hardware><description>electrical</description>
            <cost>$11,000</cost></hardware>
      <computers:hardware>
            <computers:description>
            personal computers</computers:description>
            <computers:cost>$25,000</computers:cost>
      </computers:hardware>
</renovation>
```

Next, you will declare an explicit namespace for weather in Los Angeles. You will also add a new `<weather_reading>` element that contains the weather information.

To add an explicit namespace for an element:

1. Return to the **Ch04XML01.xml** file in your text editor.
2. Modify the `<weather>` root element so it declares an explicit namespace for Los Angeles, as follows:

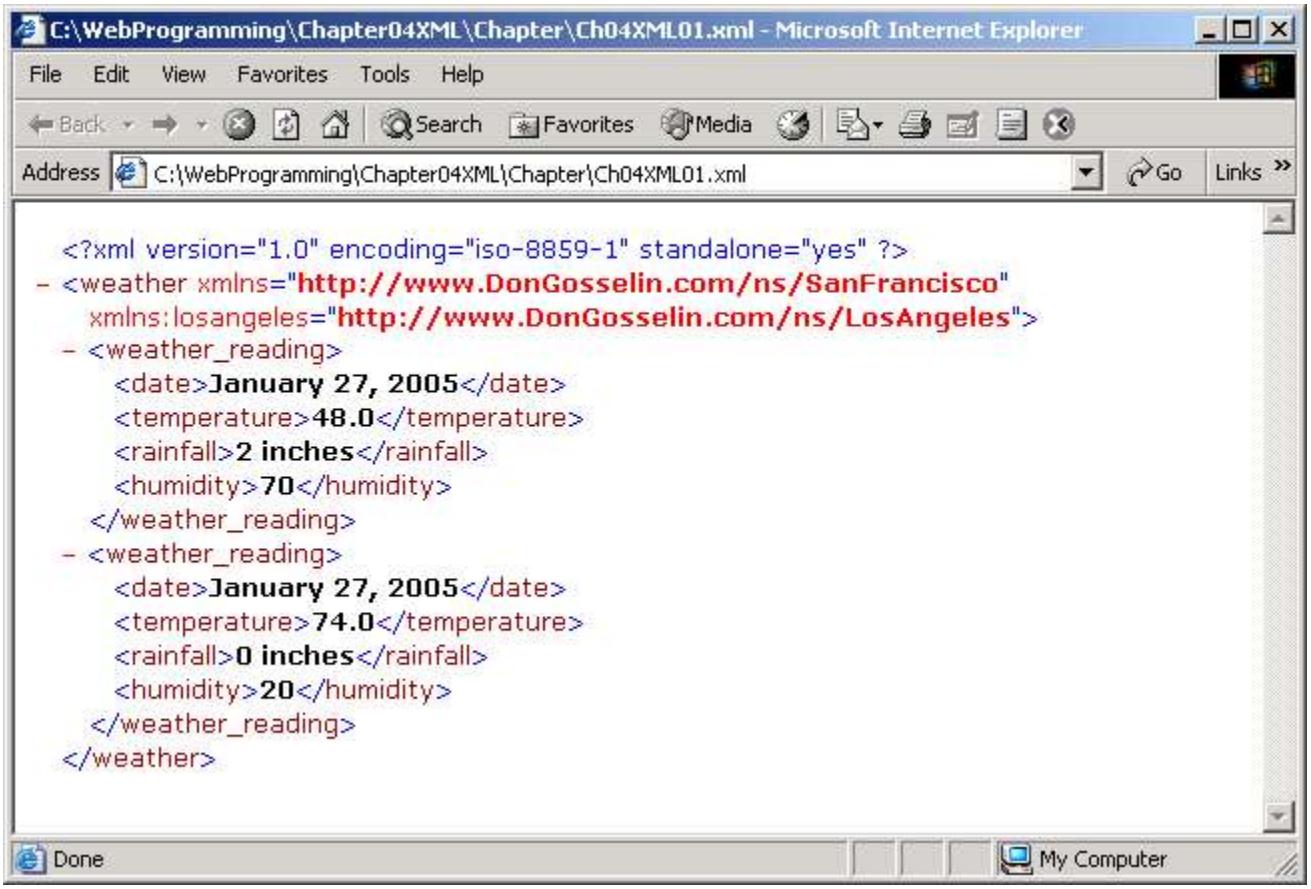
```
<weather xmlns="http://www.DonGosselin.com/ns/SanFrancisco"
      xmlns:losangeles="http://www.DonGosselin.com/
            ns/LosAngeles"> (so, it doesn't matter here that "L"
            and "A" are in caps?)
```

3. Above the closing `</weather>` element, add the following new `<weather_reading>` element:

```
<weather_reading>
      <date>January 27, 2003</date>
      <temperature>74.0</temperature>
      <rainfall>0 inches</rainfall>
      <humidity>20</humidity>
</weather_reading>
```

4. Save the **Ch04XML01.xml** file, and then open it in Internet Explorer. The file should look like Figure 4-2.

Figure 4-2: Ch04XML01.xml file in Internet Explorer after adding an explicit namespace and new `<weather_reading>` element



5. Close your Web browser.

To explicitly assign a namespace to a specific element, you must place the namespace's prefix and a colon in an element's opening and closing tag. The following code shows how to assign the computers prefix to a `<hardware>` element and its nested elements for the computer hardware equipment. Notice that the prefix and colon are applied to both the opening and closing tags for each element.

```
<computers:hardware>
      <computers:description>personal computers
      </computers:description>
      <computers:cost>$25,000</computers:cost>
</computers:hardware>
```

Any elements that do not contain an explicit namespace declaration will belong to the default namespace. This means that you must explicitly assign a namespace to any nested elements, even if the element that contains them declares an explicit namespace. For instance, in the following code, the `<description>` and `<cost>`

elements belong to the default namespace, even though the <hardware> element that contains them declares an explicit namespace:

```
<computers:hardware>
      <description>personal computers</description>
      <cost>$25,000</cost>
</computers:hardware>
```

Next, you will assign the explicit losangeles namespace to the new <weather_reading> element in the weather XML document.

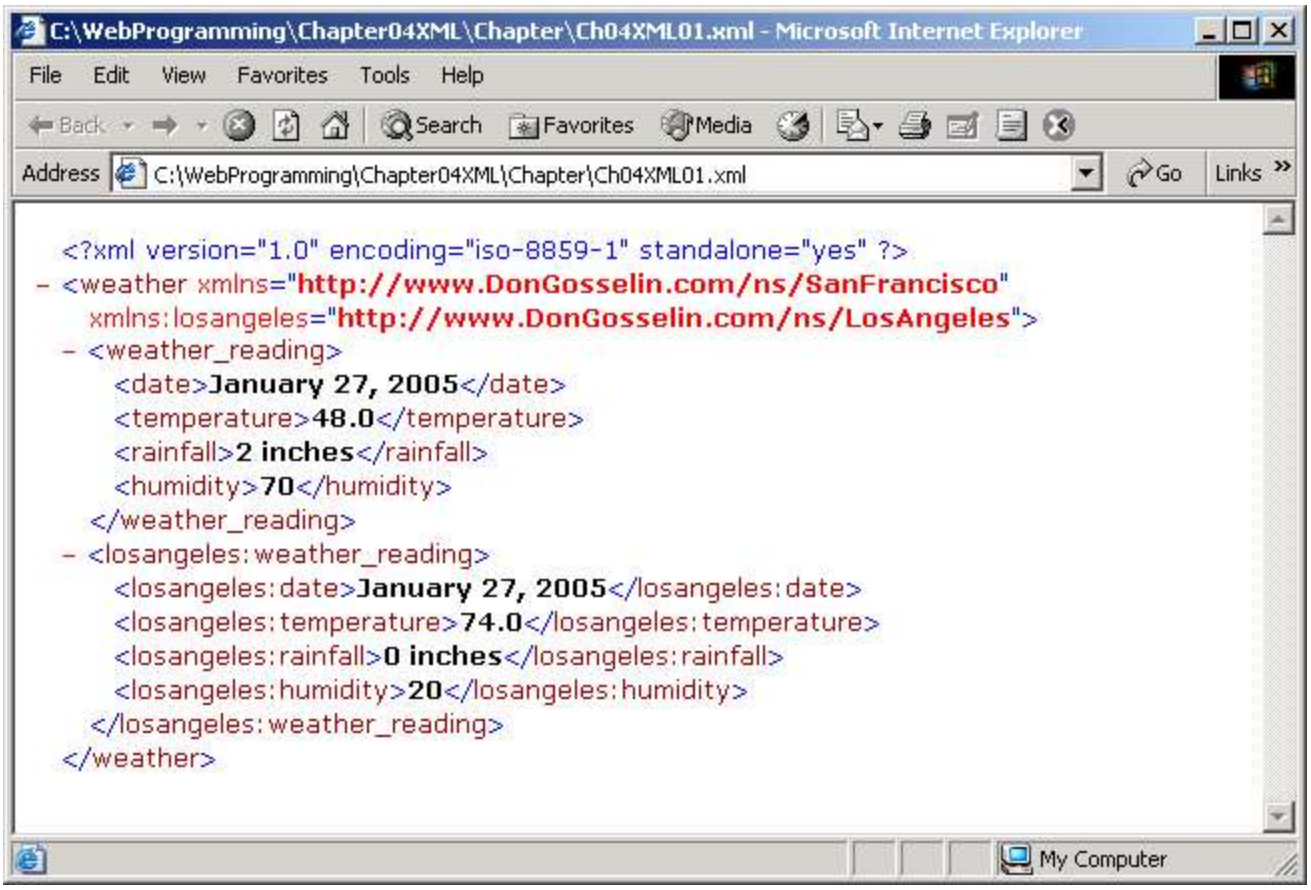
To assign an explicit namespace to an element in an XML document:

1. Return to the **Ch04XML01.xml** file in your text editor.
2. Modify the new <weather_reading> element and its nested elements so that each element is explicitly assigned the losangeles namespace, as follows:

```
<losangeles:weather_reading>
      <losangeles:date>January 27, 2003</losangeles:date>
      <losangeles:temperature>74.0</losangeles:temperature>
      <losangeles:rainfall>0 inches</losangeles:rainfall>
      <losangeles:humidity>20</losangeles:humidity>
</losangeles:weather_reading>
```

3. Save the **Ch04XML01.xml** file, and then open it in Internet Explorer. The file should look like Figure 4-3.

Figure 4-3: Ch04XML01.xml file in Internet Explorer after adding explicit namespace declarations



4. Close your Web browser.

Tip: Remember that the prefix is only a way of referring to a namespace within an XML document—the namespace itself is still identified by a unique URI. Namespaces with the same prefix but different URIs are considered to be separate namespaces. However, namespaces with different prefixes but the same URI are considered to be the same namespace.

Defining Elements with DTDs

The XML documents you have created so far have been well formed, but they have not been valid. When an XML document conforms to an associated DTD, it is said to be **valid**. When an XML document does not conform to an associated DTD, it is said to be **invalid**. As you learned in Chapter 3, a DTD is a set of rules that define the elements and attributes you can use in an XML document. A DTD also defines how the elements should be structured in an XML document. You can think of a DTD as the place where you define your own markup language. An XML document must use only the elements and attributes defined in an associated DTD, and be structured according to the DTD's

rules, or it will not be valid. Later in this section, you learn how to use a validating parser to check whether your XML documents conform to their associated DTDs.

Tip: An XML document can be well formed but invalid if it does not conform to its associated DTD. Most non-validating parsers (such as a Web browser) will render an invalid but well-formed XML document. The only problem with this scenario is that an application may not function properly if it expects an XML document to use specific elements and attributes, and be structured in a certain way.

Document Type Declarations

You use the `<!DOCTYPE>` tag to create a **document type declaration**, which defines the structure of a DTD. The syntax for the <!DOCTYPE> tag is as follows:

```
<!DOCTYPE root_element [ element_declarations ]>
```

Caution: Do not be confused by a Document Type Definition (DTD) and a document type declaration. The acronym DTD is only used with Document Type Definitions, while the term “document type declaration” refers to the DTD’s elements and structure, which are defined within the `<!DOCTYPE>` tag.

You can create two types of DTDs: internal and external. An **internal DTD** is defined within an XML document. Use an internal DTD when you want to define the elements, attributes, and structure for a single XML document, or when you are first developing and testing your DTD. When you create an internal DTD, you place the document type declaration after the XML declaration. The following code shows an example of an XML document with an internal DTD that a museum might use to catalog a collection of artworks:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
<!DOCTYPE artwork [
      <!ELEMENT artwork (artist+, title, date, medium)>
      <!ELEMENT artist (#PCDATA)><!ELEMENT title (#PCDATA)>
      <!ELEMENT date (#PCDATA)><!ELEMENT medium (#PCDATA)>
]>
<artwork>
      <artist>Rembrandt van Rijn</artist>
      <title>Lucretia</title><date>1666</date>
      <medium>oil on canvas</medium>
</artwork>
```

For now, do not worry about how the `<!ELEMENT>` tags in the document type declaration are structured—you will study them in the next section. Instead, focus on how the XML document is structured. Notice how the standalone attribute in the XML declaration is

assigned a value of “no” because the document requires a DTD to be rendered correctly. Also notice how the <artwork> root element is defined within the document type declaration.

Next, you will create a human resources XML document with an internal DTD.

To create an XML document with an internal DTD:

1. Start your text editor and create a document.
2. Type the opening XML declaration, as follows. Be sure to assign the standalone attribute a value of “no”.

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
```

3. Next, declare the following internal DTD, which defines several elements that would be used in a human resources document. Again, do not worry about how the <!ELEMENT> tags are structured. You will study them in the next section.

```
<!DOCTYPE human_resources [
      <!ELEMENT human_resources (employee+)>
      <!ELEMENT employee (first_name, last_name,
            position, department)>
      <!ELEMENT first_name (#PCDATA)>
      <!ELEMENT last_name (#PCDATA)>
      <!ELEMENT position (#PCDATA)>
      <!ELEMENT department (#PCDATA)>
]>
```

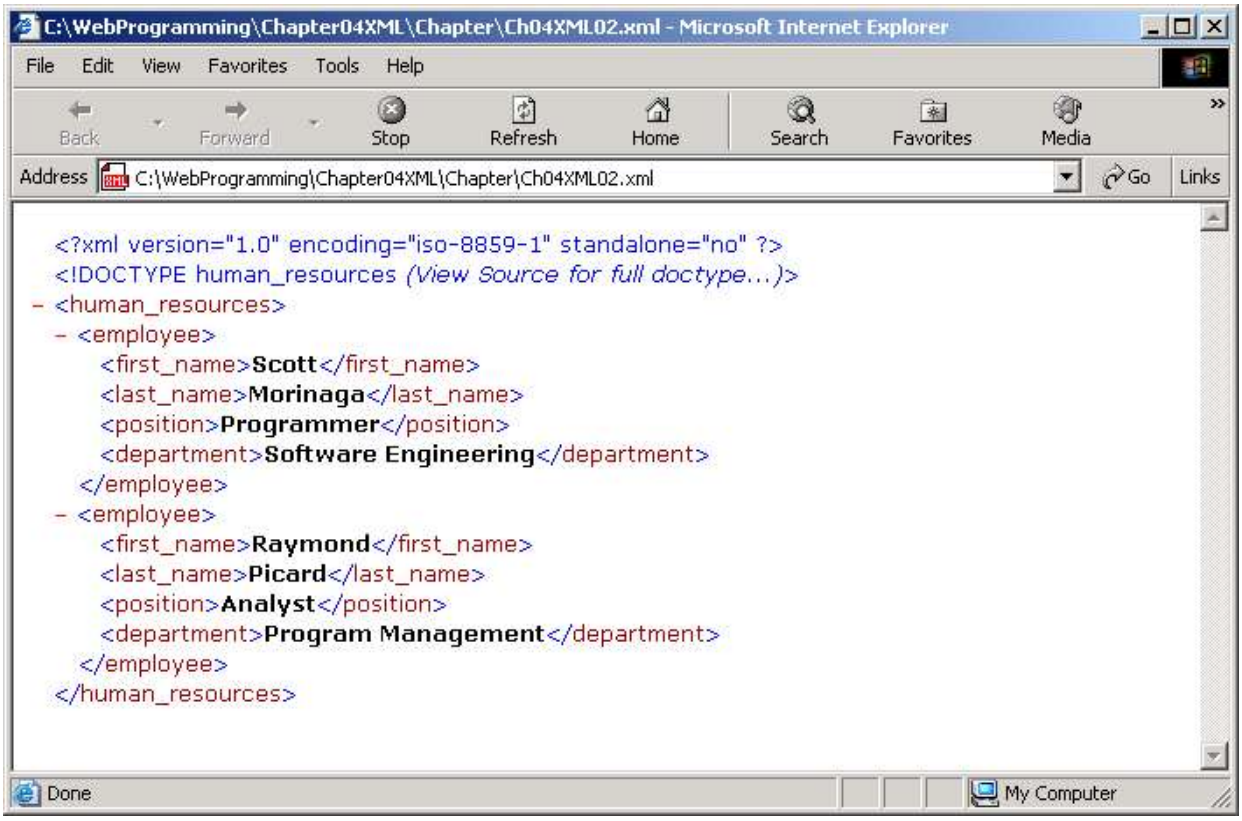
4. Next, add the following XML document, which declares two employees:

```
<human_resources>
      <employee><first_name>Scott</first_name>
            <last_name>Morinaga</last_name>
            <position>Programmer</position>
            <department>Software Engineering</department>
      </employee>
      <employee><first_name>Raymond</first_name>
            <last_name>Picard</last_name>
            <position>Analyst</position>
            <department>Program Management</department>
      </employee>
</human_resources>
```

5. Save the file as **Ch04XML02.xml** in the Chapter folder for Chapter 4.

6. Open the **Ch04XML02.xml** file in Internet Explorer. Your Web browser should look like Figure 4-4. If you did not create the document properly, fix the error that appears in the browser and reload the document.

Figure 4-4: Ch04XML02.xml file in Internet Explorer



7. Close your Web browser.

Although an internal DTD is useful when you are first developing and testing a DTD, most of the DTDs you create will usually be external DTDs that can be shared by multiple XML documents. An **external DTD** is defined in a separate document with an extension of .dtd. One of the main differences between declaring an internal DTD and an external DTD is that you do not include an XML declaration or document type declaration in the external DTD file. Also, you do not place the element declarations inside brackets ([ ]). For instance, the following code shows how you declare an external DTD for the artworks example:

```
<!ELEMENT artwork (artist+, title, date, medium)>
<!ELEMENT artist (#PCDATA)><!ELEMENT title (#PCDATA)>
<!ELEMENT date (#PCDATA)><!ELEMENT medium (#PCDATA)>
```

To declare that an XML document uses an external DTD, you place the document type declaration within the XML document using the following syntax:

```
<!DOCTYPE root_element SYSTEM or PUBLIC "DTD file">
```

You use either the SYSTEM or the PUBLIC attribute in an external document type declaration. The SYSTEM attribute declares that the DTD file is located on a local computer, network server, or corporate intranet. The PUBLIC attribute declares that the DTD is publicly available on the Internet. With either the SYSTEM or the PUBLIC attribute, you can use a URL for the location of the DTD file. For simplicity, this chapter assumes that the DTD files you use are local, so you will use a SYSTEM attribute along with a local filename for your DTD files. For instance, if the artworks DTD were named Artworks.dtd, then you would place the following document type declaration in an XML document that uses the DTD:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
<!DOCTYPE artwork SYSTEM "Artworks.dtd">
<artwork>
      <artist>Rembrandt van Rijn</artist>
      <title>Lucretia</title><date>1666</date>
      <medium>oil on canvas</medium>
</artwork>
```

Next, you will modify the human resources document so the DTD is defined in a separate document as an external DTD.

To define a DTD in a separate document as an external DTD:

1. Return to the **Ch04XML02.xml** file in your text editor.
2. Modify the <!DOCTYPE> declaration so that it uses the SYSTEM attribute to reference an external DTD named HumanResources.dtd. Before you modify the <!DOCTYPE> declaration, cut the <!ELEMENT> tags to your Clipboard–you will need them when you create the external DTD file. Your modified <!DOCTYPE> declaration should appear as follows:

   ```
   <!DOCTYPE human_resources SYSTEM "HumanResources.dtd">
   ```

3. Save the **Ch04XML02.xml** file.
4. Create a document in your text editor and type or cut and paste the element declarations that were included in the internal DTD:

   ```
   <!ELEMENT human_resources (employee+)>
   <!ELEMENT employee (first_name, last_name,
         position, department)>
   ```

```
<!ELEMENT first_name (#PCDATA)><!ELEMENT last_name (#PCDATA)>
<!ELEMENT position (#PCDATA)><!ELEMENT department (#PCDATA)>
```

5. Save the file as **HumanResources.dtd** in the Chapter folder for Chapter 4.
6. Reopen the **Ch04XML02.xml** file in Internet Explorer. The file should look the same in your Web browser as it did in Figure 4-4.
7. Close your Web browser.

Validating XML Documents against DTDs

When you open an XML document in a non-validating parser such as Internet Explorer, it only checks to see if the document is well formed; it does not check to see if the document is structured according to an associated DTD. A **validating parser**, on the other hand, checks to see if an XML document is well formed and also compares the document to a DTD to ensure that it adheres to the DTD's rules. There are numerous XML parsers on the market, both validating and non-validating. The one you choose is completely up to you, but keep in mind that you must use a validating parser if you want to ensure that an XML document complies with the rules of any given DTD.

Tip: You can find a comprehensive list of validating and non-validating parsers by searching for "XML parsers" on a search engine such as Google.

The book's CD includes an evaluation copy of Altova's xmlspy 5, a popular XML development tool. xmlspy 5 is a large and comprehensive program with many features that are far too advanced for this chapter's purposes. However, xmlspy 5 is an excellent tool to use as both a validating and non-validating parser. You will need a validating parser for the rest of the exercises in this chapter, so be sure to install xmlspy 5 (or some other validating parser) before you continue. The instructions in this chapter assume you are using xmlspy 5, but feel free to use whatever validating parser you like.

Next, you will validate the human resources XML document against its DTD.

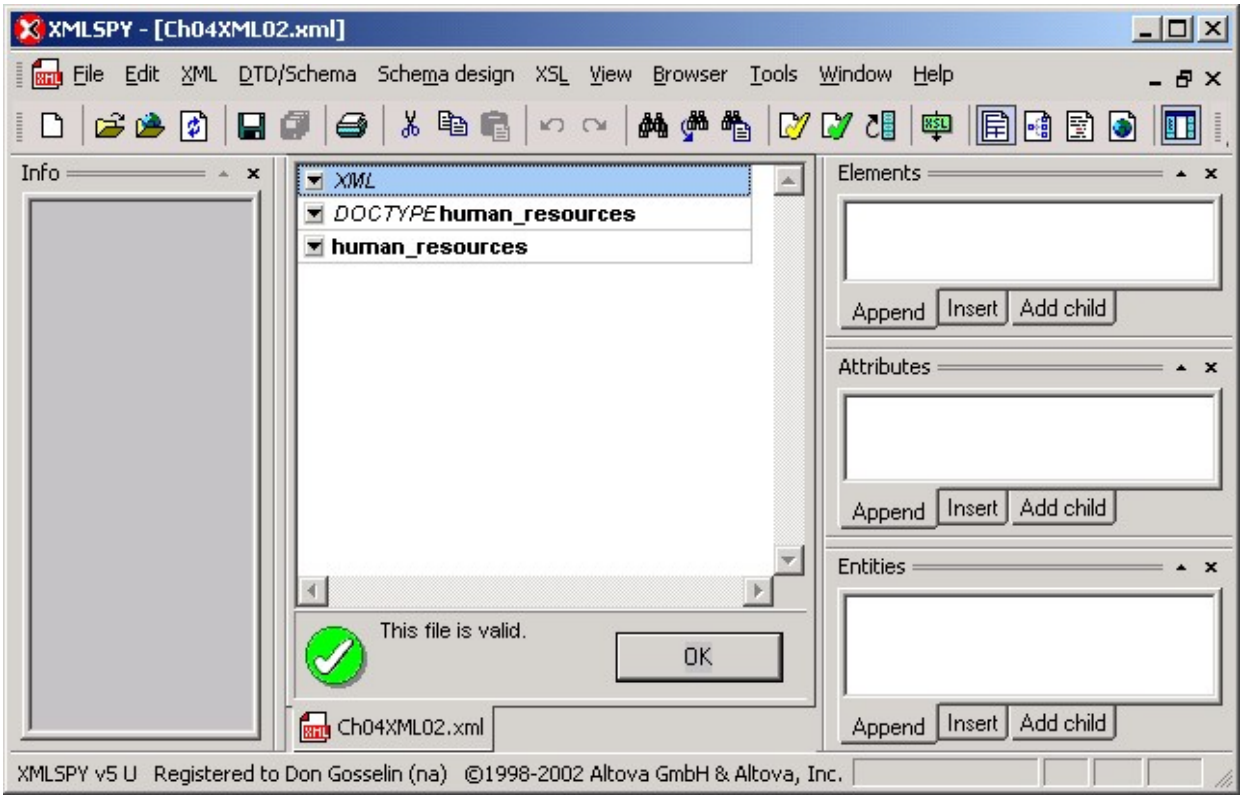
To validate an XML document against its DTD:

1. Start **xmlspy 5** and open the Ch04XML02.xml file by clicking **File** on the menu bar and then clicking **Open and browsing for the file**. By default, xmlspy 5 validates a file when you first open it, although you can change this setting by clicking **Tools** on the menu bar and then clicking **Options**. If xmlspy 5 does not automatically validate the Ch04XML02.xml file when you first open it, then click **XML** on the menu bar and then click **Validate**.

Tip: The XML menu also contains a “Check well-formedness” command that you can use to check if an XML document is well formed, but not valid.

2. If your Ch04XML02.xml file is valid, then you should receive a “This file is valid” message box, as shown in Figure 4-5. Click **OK** to close the message box. If your file is not valid, then you will receive a message box that points you to the error. You can fix the error directly in xmlspy 5, and then click the **Recheck** button to revalidate the file.

Figure 4-5: Ch04XML02.xml after being validated in xmlspy 5



Tip: The easiest way to edit an XML file in xmlspy 5 is to click **View** on the menu bar and then click **Text view**, which opens the XML file in a simple text editor window.

3. Once your Ch04XML02.xml file is valid, close it by clicking **File** on the menu bar and then clicking **Close**. Click **Yes** if you are prompted to save changes to the file.

Declaring Elements in a DTD

As you know, elements are the main building blocks of XML documents. You use an **element declaration** in a DTD to define an element’s name and the content it can

contain. You create an element declaration using the <!ELEMENT> tag with the following syntax:

```
<!ELEMENT name content>
```

While a DTD's element declarations determine the names of the elements you can use in an XML document, they also declare the content (if any) that can be stored in a particular element, along with the elements that must be structured.

The root element must be the first element declaration to follow the document type definition in an internal DTD, or it must be the first element declaration in an external DTD. The root element also cannot be an empty element. (You will learn how to define empty element declarations shortly.) One of the simplest ways to define the root element is to use the ANY keyword, which declares that an element can contain any type of content. For instance, the following statement declares the root element for the artworks DTD using the ANY keyword:

```
<!ELEMENT artwork ANY>
```

You need to understand, however, that it is considered very bad form to include the ANY keyword in any final DTDs because it essentially prevents an element from having any enforceable structure. For any element, including the root element, it is much more preferable to define the exact content that the element can accept. However, when you first start developing a DTD, you may find the ANY keyword useful as a placeholder until you determine the exact element structure that will appear in your DTD. Once you finish developing your DTD, remember to replace the ANY keyword with the element structure to which you want users of your DTD to adhere.

Next, you will start creating a DTD that defines elements a shipping company may use when shipping a package.

To create a DTD with an element declaration:

1. Create a document in your text editor.
2. Declare the following <shipping> root element using the ANY keyword:

```
<!ELEMENT shipping ANY>
```

3. Save the file as **Shipping.dtd** in the Chapter folder for Chapter 4.

For the rest of this section, you will study other types of content and element structure you can define with an element declaration.

Character Data Elements

You can create a simple element that stores only character data by placing the keyword `#PCDATA` inside parentheses in an element declaration. `PCDATA` stands for “parsed character” data and declares that an XML parser should parse the content of the element. This type of element can only contain character data and not other types of elements. For instance, the following statements declare parsed character elements in the artworks DTD:

```
<!ELEMENT artist (#PCDATA)><!ELEMENT title (#PCDATA)>
<!ELEMENT date (#PCDATA)><!ELEMENT medium (#PCDATA)>
```

Next, you will add parsed character elements to the shipping DTD.

To add parsed character elements to a DTD:

1. Return to the **Shipping.dtd** file in your text editor.
2. Add the following parsed character elements to the end of the file:

```
<!ELEMENT package ANY><!ELEMENT sender (#PCDATA)>
<!ELEMENT recipient (#PCDATA)><!ELEMENT weight (#PCDATA)>
<!ELEMENT cost (#PCDATA)>
```

3. Save the **Shipping.dtd** file in the Chapter folder for Chapter 4.

Next, you will create an XML document that conforms to Shipping.dtd and validate it using xmlspy 5.

To create and then validate an XML document that conforms to a DTD:

1. Create a document in your text editor.
2. Type the opening XML declaration, as follows.

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
```

3. Add the following `<!DOCTYPE>` declaration that uses the `SYSTEM` attribute to reference the Shipping.dtd file.

```
<!DOCTYPE shipping SYSTEM "Shipping.dtd">
```

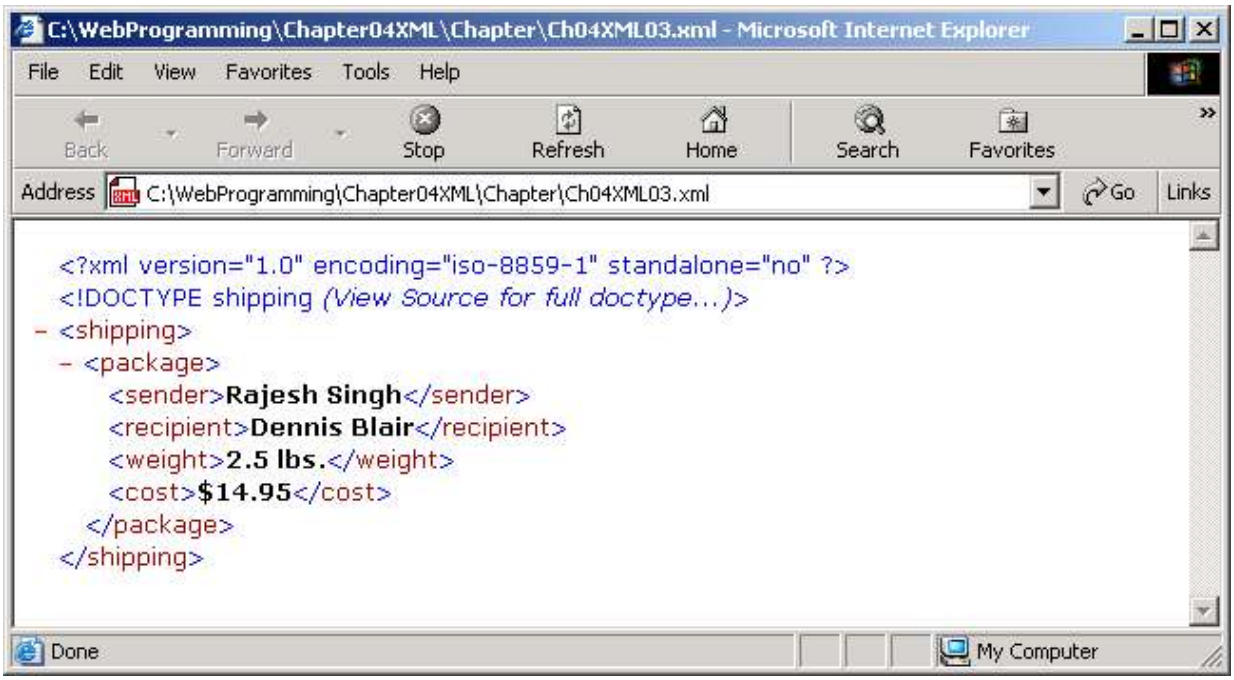
4. Add the following root and body elements that conform to the elements you declared in the Shipping.dtd file:

```
<shipping>
      <package><sender>Rajesh Singh</sender>
```

```
                <recipient>Dennis Blair</recipient>
                <weight>2.5 lbs.</weight>
                <cost>$14.95</cost></package>
    </shipping>
```

5. Save the **Ch04XML03.xml** file in the Chapter folder for Chapter 4.
6. Validate the **Ch04XML03.xml** file with xmlspy 5. Once the file is valid, close xmlspy 5.
7. Open the **Ch04XML03.xml** file in Internet Explorer. Your Web browser should look like Figure 4-6.

Figure 4-6: Ch04XML03.xml in Internet Explorer



8. Close your Web browser.

Empty Elements

You should be familiar with empty elements, which do not require ending tags and therefore do not include content. A number of elements in HTML do not have corresponding ending tags, including the `<hr>` tag, which inserts a horizontal rule into the document, and the `<br>` tag, which inserts a line break. The example of an empty XML tag you saw in the last chapter was in a document containing automobile data. The document included an empty `<photo>` element with a single attribute that stored the name of an image file containing a photograph of the automobile. Using the same example, you might want to create a `<photo>` element for the artworks DTD that stores the name of an image file containing a photograph of a particular artwork.

You create an empty element declaration in a DTD by using the keyword EMPTY in the content portion of an element declaration. For instance, the following statement declares an empty <photo> element for the artworks DTD:

```
<!ELEMENT photo EMPTY>
```

When you use an empty element in XML, you can either use an opening and closing tag or just use the opening tag by adding a single slash (/) before the tag's closing bracket to close the element. For instance, both of the following statements are valid for using the empty <photo> element in an XML document:

```
<photo></photo>
<photo/>
```

Keep in mind that even though some empty HTML elements can include content, such as the <img> element, empty XML elements cannot. The following statement would result in an invalid XML document because content is placed within the opening and closing tags of the empty <photo> element:

```
<photo>Rembrandt's "Lucretia"</photo>
```

Next, you will add an empty <account> element to Shipping.dtd and to the Ch04XML03.xml file.

To add an empty element to a .dtd file and an .xml file:

1. Return to the **Shipping.dtd** file in your text editor.
2. Add the following empty declaration for the <account> element:

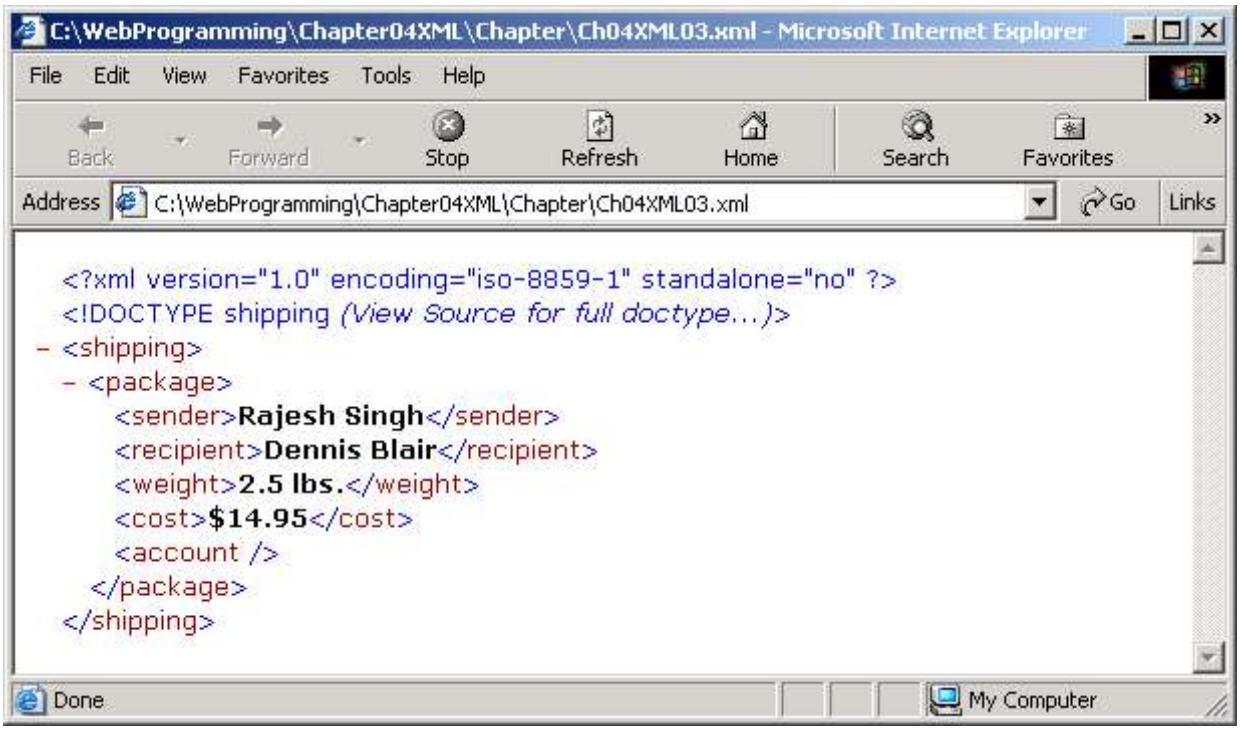
```
<!ELEMENT account EMPTY>
```

3. Save the **Shipping.dtd** file.
4. Return to the **Ch04XML03.xml** file in your text editor.
5. Add the following empty <account> element above the closing </package> tag:

```
<account/>
```

6. Save the **Ch04XML03.xml** file.
7. Validate the **Ch04XML03.xml** file with xmlspy 5 and then open it in Internet Explorer. Your Web browser should look like Figure 4-7.

Figure 4-7: Ch04XML03.xml in Internet Explorer after adding an empty element



8. Close your Web browser.

Tip: An empty XML element is essentially useless unless it includes attributes. Later in this section, you will learn how to create attribute declarations in your DTDs.

Element Sequences

One of the most important aspects of DTDs is their ability to define the number and order of elements you can add to an XML document. This lets a DTD determine exactly how to structure XML documents that conform to it. For instance, with the artworks DTD, you would want to give an XML document the ability to add multiple artists, and to list multiple works for each artist. However, you would only want to allow each artwork to be given a single title, date, and medium. Or, you may want XML documents that conform to the artworks DTD to nest the elements in a specific order to conform to the requirements of a Web application. You define the number of elements and the order in which they can be added to an XML document using the symbols in Table 4-1.

Table 4-1: Symbols for Defining Content in Element Declarations

Symbol	Description

( )	Groups expressions in the content portion of an element declaration
,	Determines the sequence in which elements must appear
+	Requires that at least one instance of the element be included
?	Allows zero or one instance of an element
*	Allows zero or more instances of an element
|	Allows one element from a group of elements to be included

To define an element sequence, you place the elements you want to include in the sequence within parentheses in the content portion of an element declaration. This essentially determines how elements can be nested within an XML document that conforms to the DTD. For instance, you may have an element named `<employee>` within a DTD with a root element of `<company>`. If you want the `<employee>` element to contain a single `<name>`, then you add the following element declarations to your DTD:

```
<!ELEMENT company ANY><!ELEMENT employee (name)>
<!ELEMENT name (#PCDATA)>
```

An XML document that conforms to the DTD in the previous code must include an `<employee>` root element that contains a single `<name>` element. If you want an element to include multiple nested elements, but in a specific order, you separate the element names with a comma. The following code, for instance, shows the same code as the previous example, but includes three required elements for the `<employee>` element: `<first_name>`, `<last_name>`, and `<position>`.

```
<!ELEMENT company ANY>
<!ELEMENT employee (first_name, last_name, position)>
<!ELEMENT first_name (#PCDATA)><!ELEMENT last_name (#PCDATA)>
<!ELEMENT position (#PCDATA)>
```

If you created an XML document that conformed to the preceding DTD, then the document could only contain a single `<employee>`. To require an XML document to include one or more instances of a particular element, you follow the element name with

the + symbol. Similarly, you use the ? symbol to allow an XML document to include zero or one instance of an element, and you use the * to allow an XML document to include zero or more instances of an element. The following code shows another example of the company DTD. This time, however, the content portion of the <company> element declaration includes the employee element name, followed by the + symbol, which allows XML documents that conform to the DTD to create one or more <employee> elements. Also, the <employee> element includes a new <middle_name> element that is optional because its name is followed by the ? symbol in the <employee> element declaration. Finally, the code includes new <phone>, <fax>, and <mobile> elements that are followed by * symbols, which means you can include zero or more instances of each of these elements.

```
<!ELEMENT company (employee+)>
<!ELEMENT employee (first_name, middle_name?,
      last_name, position, phone*, fax*, mobile*)>
<!ELEMENT first_name (#PCDATA)><!ELEMENT middle_name (#PCDATA)>
<!ELEMENT last_name (#PCDATA)><!ELEMENT position (#PCDATA)>
<!ELEMENT phone (#PCDATA)><!ELEMENT fax (#PCDATA)>
<!ELEMENT mobile (#PCDATA)>
```

The | symbol is useful in that it allows one element from a group of elements to be included. You must enclose the group of elements and | symbols within another set of parentheses. For instance, the following code shows another version of the artworks DTD. In this case, the <medium> element has been replaced with two new elements: <painting> and <sculpture>. The <artwork> root element requires an XML document to include either the <painting> or the <sculpture> elements.

```
<!ELEMENT artwork (artist+, title, date, medium,
      (painting | sculpture))>
<!ELEMENT artist (#PCDATA)><!ELEMENT title (#PCDATA)>
<!ELEMENT date (#PCDATA)><!ELEMENT painting (#PCDATA)>
<!ELEMENT sculpture (#PCDATA)>
```

Next, you will add element sequences to the Shipping.dtd file.

To add element sequences to a .dtd file:

1. Return to the **Shipping.dtd** file in your text editor.
2. Modify the <shipping> root element declaration so that it can contain multiple <package> elements, as follows:

```
<!ELEMENT shipping (package+)>
```

3. Next, modify the <package> element as follows, so that it must contain the <sender>, <recipient>, <cost>, and <account> elements. Make the <weight> element optional by following it with a question mark.

```
<!ELEMENT package (sender, recipient, weight?,
      cost, account)>
```

4. Save the **Shipping.dtd** file.
5. Validate the **Ch04XML03.xml** file with xmlspy 5 and then open it in Internet Explorer. Your Web browser should look the same as it did in Figure 4-7.
6. Close your Web browser.

Mixed Content Elements

Mixed content elements contain both character data and other elements. A mixed content element is useful when you want to specify the elements that can be nested within the element, but also allow the element to contain character data. Mixed content elements also allow you to define elements that should be nested within another element, but do not require XML documents to follow a specific element sequence. This can be useful when the XML documents that conform to your DTD do not need to be structured as rigidly as they would with element sequences. The syntax for creating a mixed content element is as follows:

```
<!ELEMENT name (#PCDATA | element | element | ... )* >
```

You must use the preceding syntax to create a mixed content element. Specifically, the #PCDATA keyword must be the first item in the option list. Also, you must place the * symbol after the option list to indicate that the mixed content element is optional and that an XML document can contain more than one instance of it. (Recall that the * symbol allows an XML document to create zero or more instances of an element.) Understand that the preceding syntax is required for the DTD to be well formed.

As an example of a mixed content element, consider a DTD that defines travel information elements. You may not need to require XML documents that conform to the DTD to use a specific sequence of elements, and you may want to allow the document to have different types of travel comments. Therefore, you could create the DTD's root element as a mixed content element, as follows:

```
<!ELEMENT travel (#PCDATA | destination | airline |
travel_dates | cost)* >
<!ELEMENT destination (#PCDATA)><!ELEMENT airline (#PCDATA)>
<!ELEMENT travel_dates (#PCDATA)><!ELEMENT cost (#PCDATA)>
```

The following code shows an XML document that uses the travel information DTD. Notice that the code does not include the `<airline>` element. However, it does include character data that specifies the transportation method for getting to Napa Valley.

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
<!DOCTYPE travel SYSTEM "Travel.dtd">
<travel>
      <destination>Napa Valley, California</destination>
      Transportation: We drove from Portland, Oregon
      <travel_dates>May 4</travel_dates>
      <cost>$850</cost>
</travel>
```

Keep in mind that with mixed content elements you can include as many or as few of the elements in the option list as you like. For instance, with the travel information DTD, you can include only some character data between the `<travel>` root element, and the XML document will still be valid:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>

<!DOCTYPE travel SYSTEM "Travel.dtd">

<travel>Our trip was cancelled.</travel>
```

Or, you can include multiple instances of the same element, as follows:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
<!DOCTYPE travel SYSTEM "Travel.dtd">
<travel><destination>Paris, France</destination>
<destination>London, England</destination>
<destination>Rome, Italy</destination></travel>
```

Declaring Attributes in a DTD

As you know, you use attributes to provide additional information about an element. Attributes are placed before the closing bracket of the starting tag, and separated from the tag name or other attributes with a space. The value assigned to an attribute must be in quotations. It's important to understand that many attributes can also be created as an element. For instance, the following code shows a `<company>` element containing a name attribute that stores the name of the company:

```
<company name="Course Technology">
nested elements
</company>
```

However, the name attribute in the preceding code could just as easily be created as an element, as follows:

```
<company><name>Course Technology></name></company>
```

In general, elements should contain information that will be displayed. Attributes, on the other hand, should contain information about the element. For instance, because the company name in the preceding code would probably be displayed, it should be created as an element. By comparison, you may want to record a tax ID number for the `<company>` element that you do not need displayed. In this case, you could create a `tax_id` attribute, as follows:

```
<company tax_id="12-3456789">
<name>Course Technology></name>
</company>
```

You use an **attribute declaration** in a DTD to declare all of the attributes that are allowed or required for a particular element. You create an attribute declaration using the `<!ATTLIST>` tag with the following syntax:

```
<!ATTLIST element-name
attribute-name attribute-type default-value
attribute-name attribute-type default-value
...
>
```

As you can see in the preceding syntax, the element name to which the attribute declaration applies immediately follows the `<!ATTLIST` portion of the declaration. Then, you create a list of attribute names, types, and default values that are allowed or required for the element.

Attribute Types

Just as you can specify an element's content, you can also specify the values that can be assigned to an attribute by declaring its **type**. Although you can create several types of attributes, the type you will study in this chapter is the CDATA type. The **CDATA attribute type** can accept any combination of character data, with the exception of tags and elements. For instance, the following code declares a CDATA attribute type named `name` for the `<company>` element. The "Course Technology" portion of the attribute declaration is the default value for the attribute, and will appear automatically if an XML document does not declare the name attribute in a `<company>` element.

```
<!ATTLIST company
name CDATA "Course Technology"
>
```

Next, you will add an attribute declaration for the empty <account> element to the Shipping.dtd file.

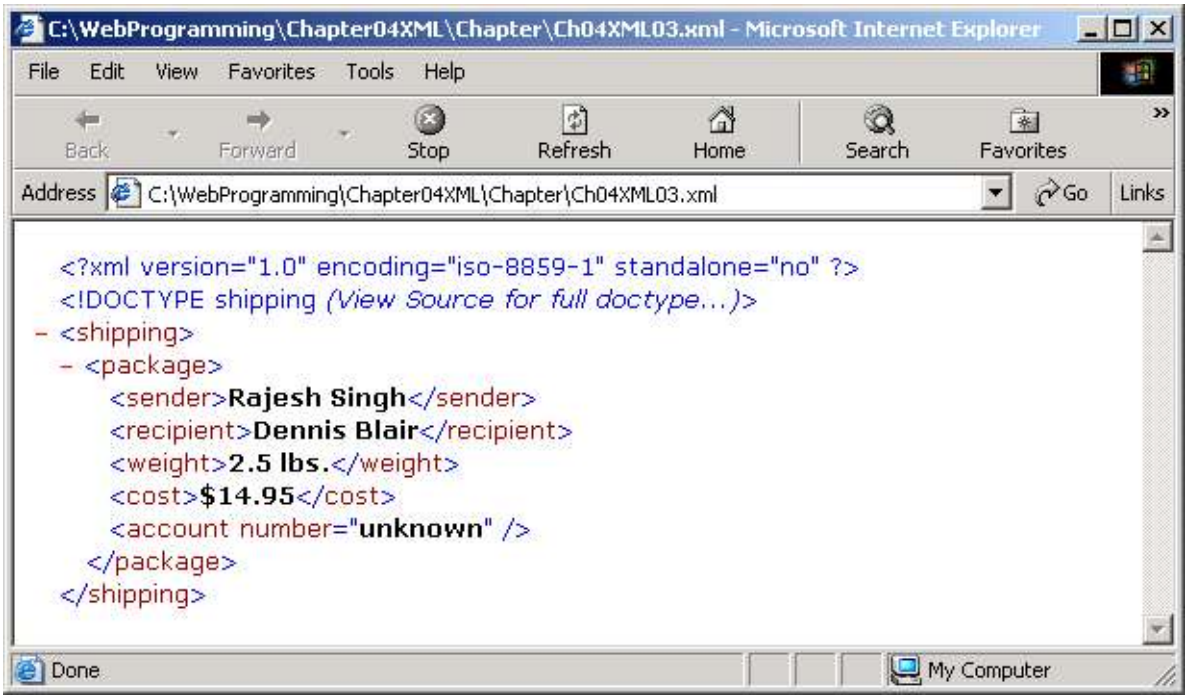
To add an attribute declaration for an empty element to a .dtd file:

1. Return to the **Shipping.dtd** file in your text editor.
2. At the end of the file, add the following declaration for an attribute named “number” in the <account> element. The attribute declaration includes a default value of unknown.

```
<!ATTLIST account
      number CDATA "unknown"
>
```

3. Save the **Shipping.dtd** file.
4. Validate the **Ch04XML03.xml** file with xmlspy 5 and then open it in Internet Explorer. Your Web browser should look like Figure 4-8. Notice that because no number attribute was declared for the <account> element, the default value of “unknown” is added automatically.

Figure 4-8: Ch04XML03.xml after adding an attribute declaration



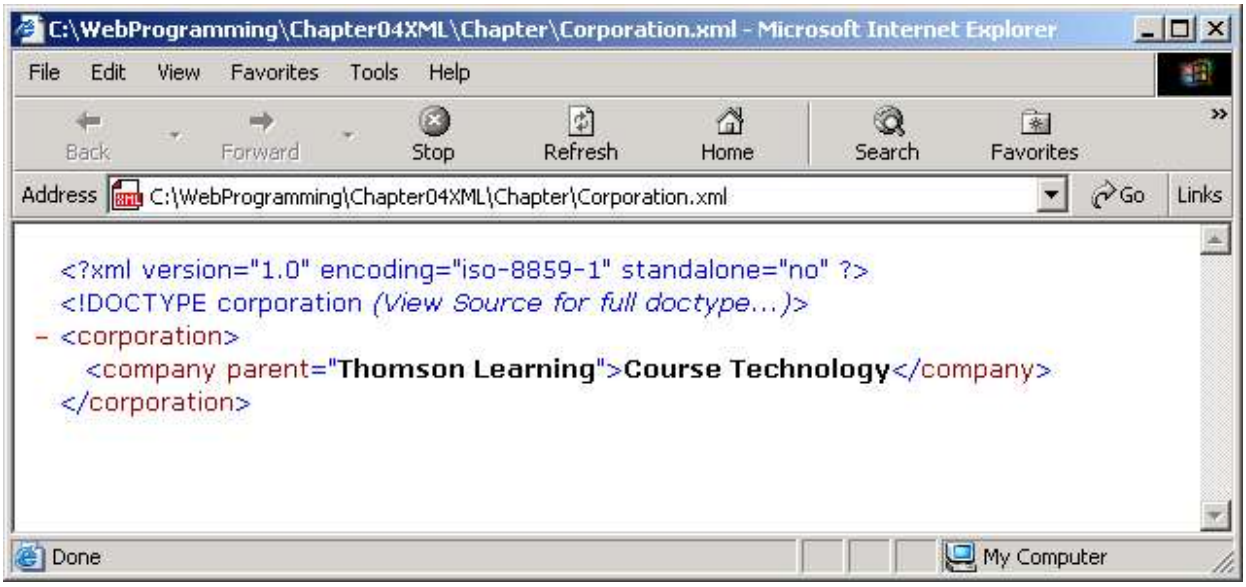
5. Close your Web browser.

Attribute Defaults

You can declare a default value for an attribute by placing the value in quotations, as shown in the `name` attribute declaration in the preceding section. If you use an element in an XML document and exclude an attribute that has a default value, then the default value will automatically be used by a program that is accessing the XML document. For instance, the following code shows a simple internal DTD that defines the `<company>` element and a `parent` attribute. A default value of "Thomson Learning" (Course Technology's parent company) is assigned to the `parent` attribute. Notice that even though the `<company>` element does not include the `parent` attribute, the default value of "Thomson Learning" is automatically added when you open the document in Internet Explorer, as shown in Figure 4-9.

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
<!DOCTYPE corporation [
<!ELEMENT corporation (company+)>
<!ELEMENT company (#PCDATA)>
<!ATTLIST company
  parent CDATA "Thomson Learning" >
]>
<corporation><company>Course Technology</company>
</corporation>
```

Figure 4-9: Output of an XML document with a default attribute



If you do not want to include a default value for an attribute, then you can use one of the attribute defaults in Table 4-2.

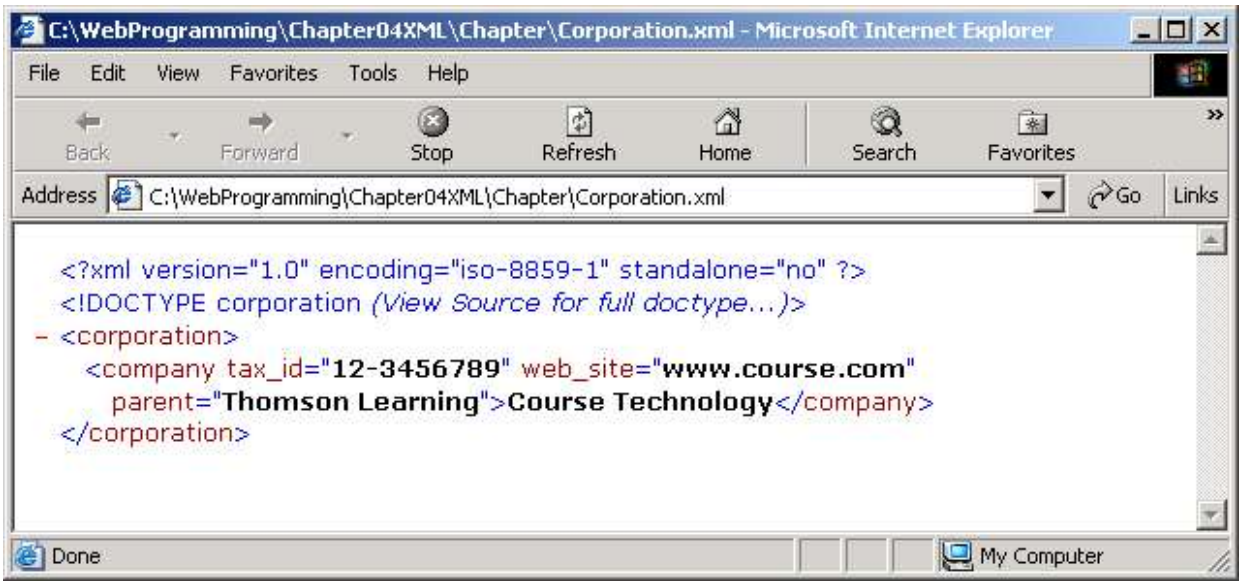
Table 4-2: Attribute Defaults

Default	Description
#REQUIRED	An XML document must assign the attribute a value each time it is used.
#FIXED	This assigns a default value to an attribute that cannot be modified.
#IMPLIED	The attribute is not required and there is no default value.

The following code shows an attribute declaration for the <company> element that uses the three values listed in Table 4-2. Notice that the #FIXED attribute also declares a default value. Even though the <company> element does not include the #FIXED parent attribute, the default value of "Thomson Learning" is automatically added when you open the document in Internet Explorer, as shown in Figure 4-10.

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
<!DOCTYPE corporation [
<!ELEMENT corporation (company+)>
<!ELEMENT company (#PCDATA)>
<!ATTLIST company
  tax_id CDATA #REQUIRED
  web_site CDATA #IMPLIED
  parent CDATA #FIXED "Thomson Learning" >
]>
<corporation>
    <company tax_id="12-3456789" web_site="www.course.com">
Course Technology</company>
</corporation>
```

Figure 4-10: Output of an XML document with multiple attribute declarations



Because each shipped package must include an account number, you will now modify the Shipping.dtd file so that the number attribute of the `<account>` element is required.

To modify the example .dtd file so the element's number attribute is required:

1. Return to the **Shipping.dtd** file in your text editor.
2. Modify the declaration for the number attribute so that it uses the `#REQUIRED` attribute default instead of the default value of "unknown", as follows:

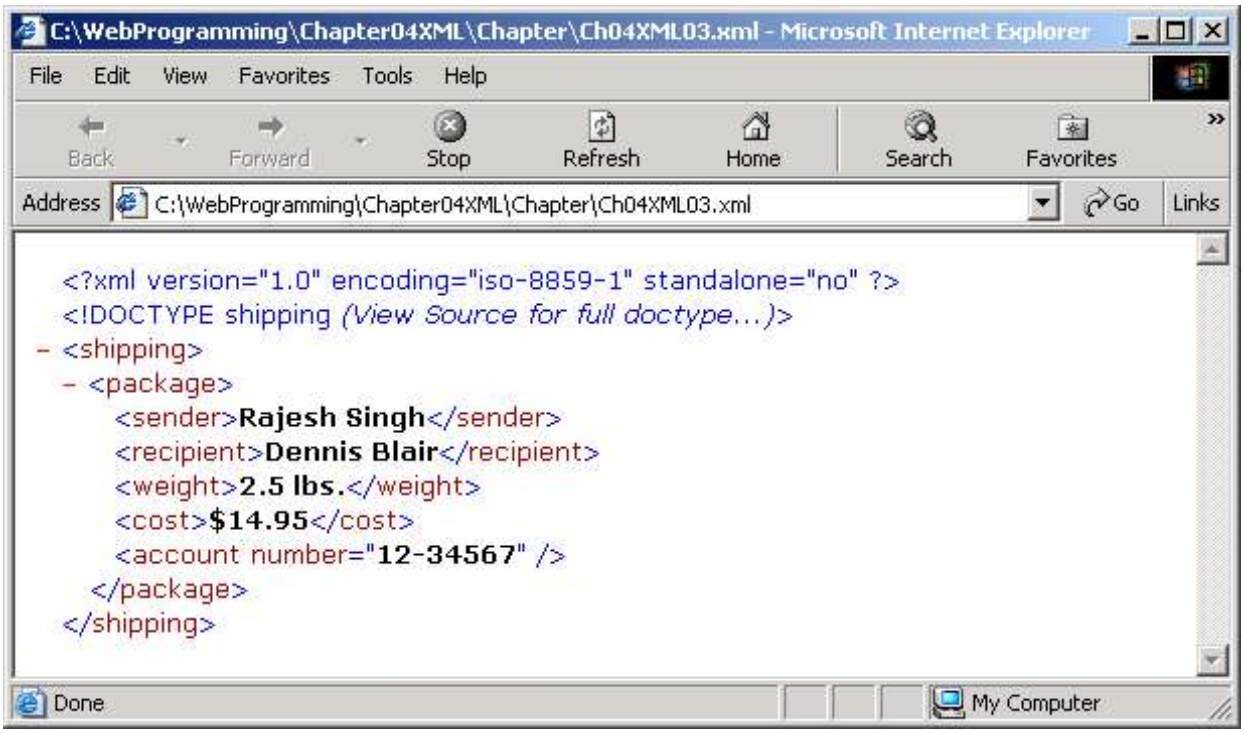
```
<!ATTLIST account
      number CDATA #REQUIRED
>
```

3. Save and Close the **Shipping.dtd** file.
4. Return to the **Ch04XML03.xml** file in your text editor.
5. Modify the `<account>` element so that it assigns a value to the number attribute as follows.

```
<account number="12-34567"/>
```

6. Save and Close the **Ch04XML03.xml** file.
7. Validate the **Ch04XML03.xml** file with xmlspy 5 and then open it in Internet Explorer. Your Web browser should look like Figure 4-11.

Figure 4-11: Ch04XML03.xml after modifying the attribute declaration



8. Close your Web browser.

Chapter Summary

- You use namespaces to organize the elements and attributes of an XML document into separate collections. Uniform Resource Identifiers, such as Uniform Resource Locators, are used to identify namespaces.
- A default namespace is applied to all of the elements and nested elements beneath the element that declares the namespace.
- The xmlns attribute assigns a namespace to an element. Namespaces that are assigned to individual elements in an XML document are called explicit namespaces.
- When an XML document conforms to an associated DTD, it is said to be valid.
- You use the <!DOCTYPE> tag to create a document type declaration, which defines the structure of a DTD.
- An internal DTD is defined within an XML document. An external DTD is defined in a separate document with an extension of .dtd.

- A validating parser checks to see if an XML document is well formed and also compares the document to a DTD to ensure that it adheres to the DTD's rules.
- You use an element declaration in a DTD to define an element's name and the content it can contain. Use an attribute declaration in a DTD to declare all of the attributes that are allowed or required for a particular element.
- An attribute's type determines the values that you can assign to the attribute. For example, the CDATA attribute type can accept any combination of character data, with the exception of tags and elements. You can declare a default value for an attribute by placing the value in quotations.

Review Questions

1. Which of the following statements is true?

   a. If an application accesses two separate XML documents that contain identical element names, the application can automatically tell them apart without the use of namespaces.

   b. You must include an `ns` folder in the URL name you want to use as a namespace.

   c. You are not allowed to place any files within an `ns` folder that is part of a URL name.

   d. The URL you use to identify a namespace does not need to exist.

2. A _____ namespace is applied to all of the elements and nested elements beneath the element that declares the namespace.

   a. default

   b. standard

   c. implied

   d. built-in

3. The _____ attribute assigns a namespace to an element.

   a. `namespace`

   b. `xmlns`

c. `xml`

d. `ns`

4. Namespaces that are assigned to individual elements in an XML document are called _____ namespaces.

   a. local

   b. nested

   c. explicit

   d. child

5. How do you explicitly assign a namespace to a specific element?

   a. You must place the namespace's prefix and a colon in an element's opening tag.

   b. You must place the namespace's prefix and a colon in an element's closing tag.

   c. You must place the namespace's prefix and a colon in an element's opening and closing tag.

   d. You cannot explicitly assign a namespace to a specific element.

6. When an XML document conforms to an associated DTD, it is said to be _____.

   a. valid

   b. well formed

   c. intrinsic

   d. correct

7. Which tag do you use to create a document type declaration?

   a. `<!DECLARATION>`

   b. `<!DOC>`

   c. `<!TYPE>`

   d. `<!DOCTYPE>`

8. Inside which symbols do you place the element and attribute declarations in an internal DTD?

   a. ( )

   b. < >

   c. { }

   d. [ ]

9. Which attribute do you use in an external document type declaration to declare that the DTD file is located on a local computer, network server, or corporate intranet?

   a. `PUBLIC`

   b. `PRIVATE`

   c. `SYSTEM`

   d. `LOCAL`

10. Which of the following statements about the root element is false?

    a. The root element must be the first element declaration to follow the document type definition in an internal DTD.

    b. The root element must be the first element declaration in an external DTD.

    c. You can declare the root element with the `ANY` keyword.

    d. The root element can be empty.

11. What is the correct syntax for declaring a `<name>` element that stores only character data?

    a. `<!ELEMENT name (PCDATA)>`

    b. `<!ELEMENT name (#PCDATA)>`

    c. `<!ELEMENT name #PCDATA>`

    d. `<!ELEMENT name CDATA>`

12. What is the correct syntax for declaring an empty `<sales>` element?

a. `<!ELEMENT sales>`

b. `<!ELEMENT />`

c. `<!ELEMENT sales EMPTY>`

d. `<!ELEMENT EMPTY sales>`

13. Which element sequence symbol requires that at least one instance of the element be included?

a. +

b. ?

c. *

d. |

14. Which symbol must you place after the option list in a mixed content element?

a. +

b. ?

c. *

d. |

15. Which attribute default value assigns a value that cannot be modified?

a. `#REQUIRED`

b. `#FIXED`

c. `#IMPLIED`

d. `#STATIC`

Hands-on Exercises

Exercise 4-1

In this exercise, you will create an XML document that includes default and explicit namespaces.

1. Create a document in your text editor and type the opening XML declaration.

2. Type the opening tag for a root element named **<mail_order>**. Use the `xmlns` attribute to include a default namespace:

```
<mail_order xmlns="http://www.MailOrderCatalogs.com/
      ns/clothing">
```

3. Create the following nested <catalog> element for a clothing catalog:

```
<catalog>
<merchandise>clothing</merchandise>
<customers>children</customers>
<pages>83</pages>
</catalog>
```

4. Modify the **<mail_order>** root element so it includes an explicit namespace, as follows:

```
<mail_order xmlns="http://www.MailOrderCatalogs.com/
      ns/clothing"
xmlns:automotive="http://www.MailOrderCatalogs.com/
      ns/automotive">
```

5. At the end of the document, add the following <catalog> element and its nested <catalog> element, along with explicit namespace declarations for each element:

```
<automotive:catalog>
<automotive:merchandise>auto parts</automotive:merchandise>
<automotive:customers>mechanics</automotive:customers>
<automotive:pages>77</automotive:pages>
</automotive:catalog>
```

6. Type the closing tag for the **<mail_order>** root element.

7. Save the XML document as **Ch04XMLEX01.xml** in the Exercises folder for Chapter 4. (students may need to create this folder)

8. Validate the **Ch04XMLEX01.html** (xml) document in Internet Explorer. If you receive any parsing errors, fix them and then reopen the document.

Exercise 4-2

In this exercise, you will create an XML document that includes three explicit namespaces.

1. Create a document in your text editor and type the opening XML declaration.

2. Type the opening tag for a root element named **<coffee_house>**.

3. Within the <coffee_house> root element, create the following nested elements for different types of coffee:

```
<coffee>
      <name>Kona</name>
      <price>$18.95</price>
</coffee>
<coffee>
      <name>Sumatran</name>
      <price>$7.95</price>
</coffee>
<coffee>
      <name>Columbian</name>
      <price>$5.95</price>
</coffee>
```

4. Type the closing tag for the **</coffee_house>** root element.

5. Create three explicit namespaces, one for each of the <coffee> elements. Assign the explicit namespaces to the appropriate elements for each <coffee> element.

6. Save the XML document as **Ch04XMLEX02.xml** in the Exercises folder for Chapter 4.

7. Validate the **Ch04XMLEX02.html** (xml) document in Internet Explorer. If you receive any parsing errors, fix them and then reopen the document.

Exercise 4-3

In this exercise, you will declare elements in a DTD that will store university information. You will also create and validate an XML document against the universities DTD.

1. Create a document in your text editor.

2. Create the following DTD that declares elements for the universities DTD:

```
<!ELEMENT universities (university+)>
<!ELEMENT university (name, location)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT location (#PCDATA)>
```

3. Save the DTD document as **Ch04XMLEX03.dtd** in the Exercises folder for Chapter 4.

4. Create another document in your text editor and type the following XML document that uses the universities DTD:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
            (need to include the DOCTYPE declaration for corresponding
dtd file here)
<universities>
      <university>
            <name>Harvard University</name>
            <location>Cambridge, MA</location>
      </university>
      <university>
            <name>Yale University</name>
            <location>New Haven, CT</location>
      </university>
      <university>
            <name>Columbia University</name>
            <location>New York, NY</location>
      </university>
</universities>
```

5. Save the XML document as **Ch04XMLEX03.xml** in the Exercises folder for Chapter 4.

6. Use xmlspy 5 to validate the **Ch04XMLEX03.html** (xml) document against the **Ch04XMLEX03.dtd** file. If you receive any parsing errors, fix them and then open the document in Internet Explorer.

Exercise 4-4

In this exercise, you will add an attribute declaration to the DTD you created in the last exercise.

1. Open the **Ch04XMLEX03.dtd** file in your text editor and immediately save it as **Ch04XMLEX04.dtd**.

2. In the **Ch04XMLEX04.dtd** file, add the following declaration for a `name` attribute that will be used in the `<university>` element:

```
<!ATTLIST university
  name CDATA #REQUIRED
>
```

3. Delete the `<name>` attribute declaration. (Users also need to delete “name” from the parenthetical mandatory element list in the university element declaration)

4. Save and close the **Ch04XMLEX04.dtd** file.

5. Open the **Ch04XMLEX03.xml** file in your text editor and immediately save it as **Ch04XMLEX04.xml**.

6. Modify the three `<university>` elements so they include a name attribute with the name of the university. Also, delete each `<name>` element.

7. Save and close the **Ch04XMLEX04.xml** file.

8. Use xmlspy 5 to validate the **Ch04XMLEX04.html (xml)** document against the **Ch04XMLEX04.dtd** file. If you receive any parsing errors, fix them and then open the document in Internet Explorer.

Exercise 4-5

In this exercise, you will create a DTD for an existing XML document.

1. Create a document in your text editor and type the following XML document.

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
(Users will need to enter the DOCTYPE declaration here for the dtd file
they are about to create)
<travel>
      <transportation mode="airplane">
            <destination>Paris</destination>
            <depart_date>June 1</depart_date>
            <carrier company="United" />
      </transportation>
      <transportation mode="train">
            <destination>New Orleans</destination>
            <depart_date>April 15</depart_date>
            <railroad company="Amtrak" />
      </transportation>
      <transportation mode="automobile">
            <destination>Vancouver</destination>
            <depart_date>August 3</depart_date>
      </transportation>
</travel>
```

2. Save the XML document as **Ch04XMLEX05.xml** in the Exercises folder for Chapter 4.

3. Create another document in your text editor and create a DTD for the **Ch04XMLEX05.xml** document.

4. Save the DTD document as **Ch04XMLEX05.dtd** in the Exercises folder for Chapter 4.

5. Use xmlspy 5 to validate the **Ch04XMLEX05.html** document against the **Ch04XMLEX05.dtd** file. If you receive any parsing errors, fix them and then open the document in Internet Explorer.

Web Programming Projects

In the following projects, use xmlspy 5 to validate each XML document against its associated DTD. Save the documents in the Projects folder for Chapter 4.

Project 4-1

Create an accounts receivable DTD. Include elements such as `<vendor>`, `<date>`, and `<amount>`. Also, include empty elements for different payment options, such as check, credit card, and cash, but allow only one payment option to be selected. Create unique attributes for each payment option, such as a check number attribute for the check element. Also create an XML document that uses the accounts receivable DTD. Save the DTD document as **Ch04XMLProject01.dtd** and the XML document as **Ch04XMLProject01.xml**.

Project 4-2

Create a DTD that contains elements you would find in a business memo. Include elements such as sender, recipient, subject, salutation, and paragraph. Add at least one empty element and one attribute with a default value - but do not use an attribute default. Be sure to set up the element sequence so that XML documents must add each element in the proper order. Also, allow XML documents to include multiple `<paragraph>` elements. Create an XML document that uses the elements and attributes in the DTD. Save the DTD document as **Ch04XMLProject02.dtd** and the XML document as **Ch04XMLProject02.xml**.

Project 4-3

Create a DTD that contains elements you would find in a resume. Include elements such as your name and position desired. Create any other nested elements that you deem appropriate, such as `<references>` or `<special_skills>` elements. Be sure to set up the element sequence so that XML documents must add each element in the proper order. Also, allow XML documents to include multiple `<employment>` and `<education>` elements. Save the XML document as **Ch04XMLProject03.xml** in the Projects folder for Chapter 4.