Chapter Three: XML Namespaces

The need of Namespaces in XML

Because of the nature of XML, it is possible for any company or individual to create XML document types that describe the world in their own terms. If your company feels that an **<order>** should contain a certain set of information, while another company feels that it should contain a different set of information, both companies can go ahead and create different document types to describe that information. Both companies can even use the name **<order>** for entirely different uses if desired.

However, if everyone is creating personalized XML vocabularies, you'll soon run into a problem: Only so many words are available in human languages, and a lot of them are going to be snapped up by people defining document types. How can you define a **<title>** element to be used to denote the title in a person's name (such as Dr. or Mrs.) when XHTML already has a **<title>** element used to describe the title of an HTML document? How can you then further distinguish those two **<title>** elements from the title of a book?

If all of these documents were to be kept separate, this still would not be a problem. If you saw a **<title>** element in an XHTML document, you'd know what kind of title it referred to, and if you saw one in your own proprietary XML document type, you'd know what that meant too. Unfortunately, life isn't always that simple, and eventually you'll need to combine various XML elements from different document types into one XML document. For example, you might create an XML document type containing information about a person, including that person's title, but also containing the person's résumé, in XHTML form. Such a document may look similar to this:

```
<?xml version=”1.0”?>
<person>
  <name>
    <title>Sir</title>
    <first>Yosseph</first>
    <middle>Getachew</middle>
    <last>Abebe</last>
  </name>
<position>Vice President of Marketing</position>
 <resume>
  <html>
  <head><title>Resume of Yosseph Getachew</title></head>
  <body>
     <h1>Yosseph</h1>
     <p>Yosseph's a great guy, you know?</p>
  </body>
   </html>
 </resume>
</person>
```

To an XML parser, there isn't any difference between the two **<title>** elements in this document. If you do a simple search of the document to find Yosseph Getachew's title by looking for <title> elements, you might accidentally get Resume of Yosseph Getachew instead of "Sir". Even in your application, you can't know which elements are XHTML elements and which aren't without knowing in advance the structure of the document. That is, you'd have to know that there is a **<resume>** element, which is a direct child of **<person>**, and that all of the descendents of **<resume>** are a separate type of element from the others in your document. If your structure ever changed, all of your assumptions would be lost.

Using Prefixes

The best way to solve this problem is for every element in a document to have a completely distinct name. For example, you might come up with a naming convention whereby every element for your proprietary XML document type gets your own prefix, and every XHTML element gets another prefix.

You could rewrite the previous XML document to something like this:

```
<?xml version="1.0"?>
<pers:person>
      <pers:name>
         <pers:title>Sir</pers:title>
         <pers:first>Yosseph</pers:first>
         <pers:middle>Getachew</pers:middle>
         <pers:last>Abebe</pers:last>
      </pers:name>
    <pers:position>Vice President of Marketing</pers:position>
  <pers:resume>
         <xhtml:html>
         <xhtml:head><xhtml:title>Resume of Yosseph Getachew</xhtml:title></xhtml:head>
         <xhtml:body>
         <xhtml:h1>Yosseph Getachew</xhtml:h1>
        <xhtml:p>Yosseph'ss a great guy, you know?</xhtml:p>
        </xhtml:body>
        </xhtml:html>
     </pers:resume>
</pers:person>
```

> *This is just an example to illustrate the theory: If you try to view this document in Internet Explorer, IE will give you an error about an "undeclared namespace." You'll see why as we investigate the namespace syntax in more detail.*

This is a bit uglier, but you can immediately tell what kind of title you are talking about: a **<pers:title>** or an **<xhtml:title>**. Doing a search for **<pers:title>** will always return Sir. You can always immediately tell which elements are XHTML elements, without having to know in advance the structure of your document.

The drawback to doing this is that you're no longer using proper XHTML elements. Browsers that are able to display XHTML understand the <p> element, but they don't understand the <xhtml:p> element, so if you wrote an application to read this XML document and it wanted to display the XHTML portions in a browser, it would have to rename all of the elements first, to get rid of the xhtml prefix.

By separating these elements using a prefix, you have effectively created two kinds of elements in your document: **pers** types of elements and **xhtml** types of elements. Any elements with the **pers** prefix belong to the same "category" as each other, just as any elements with the **xhtml** prefix belong to another "category." These "categories" are called *namespaces*.

Note that namespaces are concerned with a ***vocabulary***, not a ***document type***. That is, the namespace distinguishes which names are in the namespace, but not what they mean or how they fit together. It is simply a "bag of names."

A namespace is a purely abstract entity; it's nothing more than a group of names that belong with each other conceptually.

The concept of namespaces also exists in certain programming languages, such as Java, where the same problem exists. How can you name your Java variables whatever you want and not have those names conflict with names already defined by others, or even by the Java library itself? The answer is that Java code is broken up into packages, whereby the names within a package must be unique, but the same name can be used in any package.

Why Doesn't XML Just Use These Prefixes?

Unfortunately, there is a drawback to the prefix approach to namespaces used in the previous XML: Who will monitor the prefixes? The whole reason for using them is to distinguish names from different document types, but if it is going to work, then the prefixes themselves also have to be unique. If one company chose the prefix **pers** and another company also chose that same prefix, the original problem still exists.

In fact, this prefix administration would have to work a lot like it works now for domain names on the Internet. A company or individual would go to the "prefix administrators" with the prefix they would like to use. If that prefix weren't already being used, they could use it; otherwise, they would have to pick another one.

To solve this problem, you could take advantage of the already **unambiguous** Internet domain names in existence and specify that URIs must be used for the prefix names.

A URI (Uniform Resource Identifier) is a string of characters that identifies a resource. It can be in one of two flavors: URL (Uniform Resource Locator) or URN (Universal Resource Name).

For example, if I want to write XML for Wiley publishing inc., which owns the domain name ***www.wiley.com***, I could incorporate that into the prefix. Perhaps the document might end up looking like this:

```
<?xml version=”1.0”?>
<{http://www.wiley.com/pers}person>
<{http://www.wiley.com/pers}name>
<{http://www.wiley.com/pers}title>
Sir
</{http://www.wiley.com/pers}title>
<!--etc...-->
```

We have solved our problem of uniqueness. Because Wiley owns the ***www.wiley.com*** domain name, I know that nobody else will be using that ***http://www.wiley.com/pers*** prefix in their XML documents, and if I want to create any additional document types, I can just keep using our domain name, and add the new namespace name to the end, such as ***http://www.wiley.com/***other-namespace.

If you visit http://www.wiley.com/pers, you'll notice that there is no document at that location. The Wiley website will give you an error message instead. Does this mean that our namespace is broken? Actually, not at all. The URL we're using is simply used as a name, for the namespace; the XML parser won't try to pull back any resources from that location, or use it for any purpose other than naming the namespaces in the document. We'll talk about this more in a bit, but for now you can remember the following:

Even though it looks like a URL, a namespace name is only used as a name, not a location.

It's important to note that we need more than just the **www.wiley.com** part of the URI; we need the whole thing. Otherwise, there would be a further problem: Different people could have control of different sections on that domain, and they might all want to create namespaces. For example, the company's HR department could be in charge of **http://www.wiley.com/hr** and might need to create a name space for names (of employees), and the sales department could be in charge of **http://www.wiley.com/sales**, and also need to create a namespace for names (of customers). As long as we're using the whole URI, we're fine—we can create both namespaces (in this case, **http://www.wiley.com/hr/names** and **http://www.wiley.com/sales/names**, respectively). We also need the protocol (http) in there because there could be yet another department—for example, **ftp://www.wiley.com/hr** and **ftp://www.wiley.com/sales**.

The only drawback to this solution is that our XML is no longer well formed. Our names can now include a myriad of characters that are allowed in URIs but not in XML names: / characters, for example. In addition, for the sake of this example, we used {} characters to separate the URL from the name, neither of which is allowed in an XML element or attribute name.

What we really need to solve all of our namespace-related problems is a way to create three-part names in XML: One part would be the name we are giving this element, the second part would be a URI associated with the name, for the element's namespace, and the third part would be an arbitrarily chosen prefix that refers to a URI, which specifies the namespace to which this element belongs. In fact, this is what XML namespaces provide.

How XML Namespaces Work

The XML Namespaces Recommendation introduces a standard syntax for declaring namespaces and identifying the namespace for a given element or attribute in an XML document.
To use XML namespaces in your documents, elements are given qualified names. These qualified names consist of two parts: the local part, which is the same as the names we have been giving elements all along, and the namespace prefix, which specifies to which namespace this name belongs.

```
<pers:person xmlns:pers=”http://www.wiley.com/pers”/>
```

The key is the **xmlns:pers** attribute (xmlns stands for XML Namespace). Here you are declaring the pers namespace prefix and the URI of the namespace that it represents (**http://www.wiley.com/pers**). We can then use the namespace prefix with our elements, as in **pers:person**. As opposed to our previous prefixed version, the prefix itself (**pers**) doesn't have any meaning—its only purpose is to point to the namespace name. For this reason, we could replace our prefix (**pers**) with any other prefix, and this document would have exactly the same meaning. (The prefix does, however, have to follow the same naming conventions as element names.)

This prefix can be used for any descendants of the **<pers:person>** element, to denote that they also belong to the **http://www.wiley.com/pers** namespace, as shown in the following example:

```
<pers:person xmlns:pers=”http://www.wiley.com/pers”>
   <pers:name>
      <pers:title>Sir</pers:title>
   </pers:name>
</pers:person>
```

Notice that the prefix is needed on both the start-tags and end-tags of the elements. They are no longer simply being identified by their names, but by their qualified names.

Only elements that are specifically prefixed are part of a namespace. For example, consider this document:

```
<pers:person xmlns:pers=”http://www.wiley.com/pers”>
    <first/>
</pers:person>
```

The **<first>** element is not part of the same namespace as the **<person>** element because it doesn't have a namespace prefix. In fact, in this case, the **<first>** element is not in a namespace at all.

Internally, when this document is parsed, the parser simply replaces any namespace prefixes with the namespace itself, creating a name much like the names we used earlier in the chapter. That is, internally a parser might consider **<pers:person>** to be similar to **<{http://www.wiley.com/pers\person>.** For this reason, the **{http://www.wiley.com/pers\person}** notation is often used in namespace discussions to talk about fully qualified names. Just remember that this is only for the benefit of easily discussing namespace issues; it is not valid XML syntax.
Example:

```
<?xml version=”1.0”?>
<pers:person xmlns:pers=”http://www.wiley.com/pers”
          xmlns:html=”http://www.w3.org/1999/xhtml”>
   <pers:name>
         <pers:title>Sir</pers:title>
         <pers:first>Yosseph</pers:first>
         <pers:middle>Getachew</pers:middle>
         <pers:last>Abebe</pers:last>
      </pers:name>
    <pers:position>Vice President of Marketing</pers:position>
   <pers:resume>
         <html:html>
         <html:head><html:title>Resume of Yosseph                    Getachew</html:title></html:head>
         <html:body>
         <html:h1>Yosseph Getachew</html:h1>
        <html:p>Yosseph’ss a great guy, you know?</html:p>
        </html:body>
        </html:html>
     </pers:resume>
</pers:person>
```

How It Works

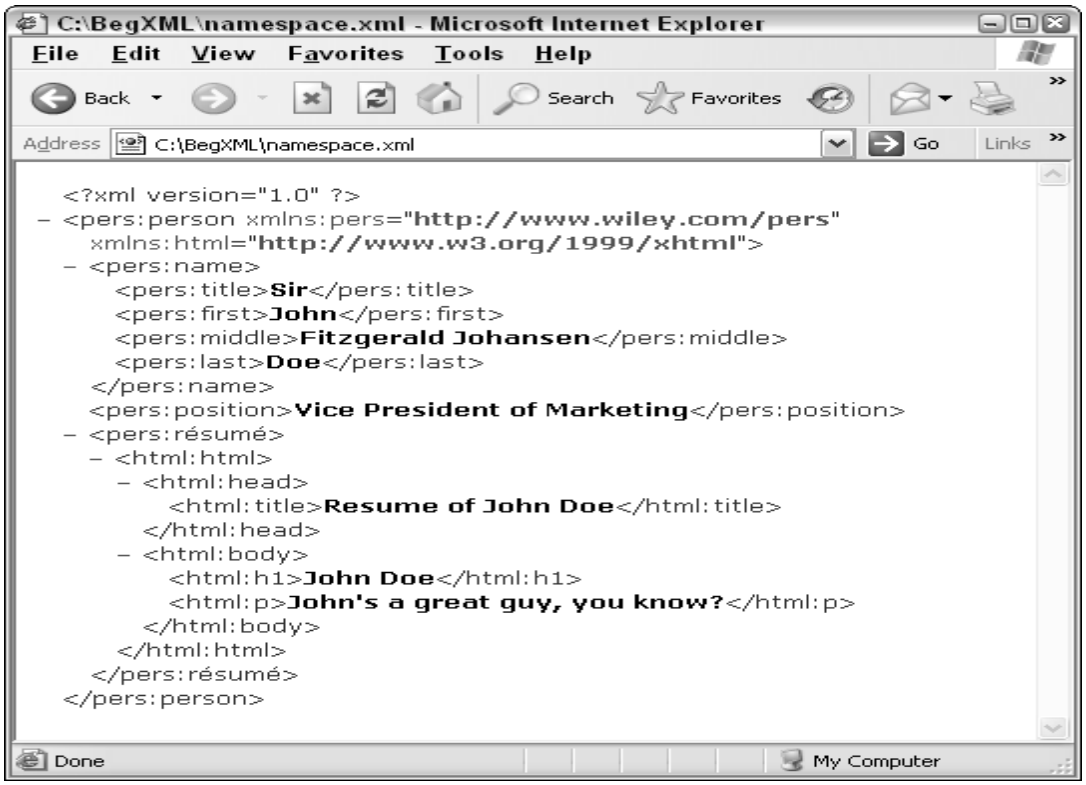
You now have a document with elements from two separate namespaces, which you defined in the highlighted code; and any namespace-aware XML parser will be able to tell them apart. (The fact that the file opens fine in Internet Explorer indicates that the parser bundled with this browser understands namespaces properly; if it didn’t, the document might raise errors instead.)

The xmlns attributes specify the namespace prefixes you are using to point to your two namespaces:

```
<pers:person xmlns:pers=”http://www.wiley.com/pers”
       xmlns:html=”http://www.w3.org/1999/xhtml”>
```

That is, you declare the **pers** prefix, which is used to specify elements that belong to the **“pers”** namespace, and the html prefix, which is used to specify elements that belong to the **XHTML** namespace. However, remember that the prefixes themselves mean nothing to the XML parser; they are replaced with the URI internally. You could have used **pers** or **myprefix** or **blah** or any other legal string of characters for the prefix; it’s only the URI to which they point that the parser cares about—although using descriptive prefixes is good practice!

When you view in a browser with XML parser, it looks like the following:



Default Namespaces

Although the previous document solves all of our namespace-related problems, it's just a little bit ugly. You have to give every element in the document a prefix to specify the namespace to which it belongs, which makes the document look very similar to the first prefixed version. Luckily, you have the option to create *default namespaces*.

A default namespace is just like a regular namespace except that you don't have to specify a prefix for all of the elements that use it.

Using default namespaces, our document might look more like this:

```
<person xmlns="http://www.wiley.com/pers">
  <name>
    <title>Sir</title>
  </name>
</person>
```

Notice that the **xmlns** attribute no longer specifies a prefix name to use for this namespace. As this is a default namespace, this element and any elements descended from it belong to this namespace, unless they explicitly specify another namespace. Therefore, the **<name>** and **<title>** elements both belong to this namespace.

You can declare more than one namespace for an element, but only one can be the default. This allows you to write XML like this:

```
<person xmlns="http://www.wiley.com/pers"
          xmlns:xhtml="http://www.w3.org/1999/xhtml">
    <name/>
    <xhtml:p>This is XHTML</xhtml:p>
</person>
```

In the preceding example, all of the elements belong to the **http://www.wiley.com/pers** namespace, except for the **<p>** element, which is part of the XHTML namespace. (You declared the namespaces and their prefixes, if applicable, in the root element so that all elements in the document can use these prefixes.) However, you can't write XML like this:

```
<person xmlns="http://www.wiley.com/pers"
          xmlns="http://www.w3.org/1999/xhtml">
```

This tries to declare two default namespaces. In this case, the XML parser wouldn't be able to figure out to what namespace the **<person>** element belongs (not to mention that this is a duplicate attribute, which, as you saw in Chapter 2, is not allowed in XML).

Example:

```
<?xml version="1.0"?>
<person xmlns="http://www.wiley.com/pers"
       xmlns:html="http://www.w3.org/1999/xhtml">
  <name>
          <title>Sir</title>
          <first>Yosseph</first>
          <middle>Getachew</middle>
          <last>Abebe</last>
   </name>
   <position>Vice President of Marketing</position>
   <résumé>
      <html:html>
           <html:head><html:title>Resume of Yosseph Getachew</html:title></html:head>
        <html:body>
          <html:h1>Yosseph Getachew</html:h1>
          <html:p>Yosseph's a great guy, you know?</html:p>
         </html:body>
      </html:html>
   </résumé>
</person>
```

Declaring Namespaces on Descendants

So far, when we have had multiple namespaces in a document, we've been declaring them all in the root element, so that the prefixes are available throughout the document. For example, in the previous example, we declared a default namespace, as well as a namespace prefix for our HTML elements, all on the **<person>** element.

However, you don't have to declare all of your namespace prefixes on the root element; in fact, a namespace prefix can be declared on any element in the document. You could also have written the previous XML like this:

```
<person xmlns="http://www.wiley.com/pers">
    <name/>
    <xhtml:p xmlns:xhtml="http://www.w3.org/1999/xhtml">
    This is XHTML</xhtml:p>
</person>
```

Example:

```
<?xml version="1.0"?>
<person xmlns="http://www.wiley.com/pers">
  <name>
          <title>Sir</title>
          <first>Yosseph</first>
          <middle>Getachew</middle>
          <last>Abebe</last>
   </name>
   <position>Vice President of Marketing</position>
   <résumé>
     <html xmlns="http://www.w3.org/1999/xhtml">
          <head><html:title>Resume of Yosseph Getachew</html:title></html:head>
        <body>
          <h1>Yosseph Getachew</html:h1>
          <p>Yosseph's a great guy, you know?</html:p>
        </body>
     </html>
```

Canceling Default Namespaces

Sometimes you might be working with XML documents in which not all of the elements belong to a namespace. For example, you might be creating XML documents to describe employees in your organization, and those documents might include occasional XHTML comments about the employees, such as in the following short fragment:

```
<employee>
 <name>Yosseph Getachew</name>
  <notes>
     <p xmlns="http://www.w3.org/1999/xhtml">I've worked
      with <name>Jane Doe</name> for over a <em>year</em>now.</p>
   </notes>
</employee>
```

In this case, you have decided that anywhere the employee's name is included in the document it should be in a **<name>** element, in case the employee changes his or her name in the future, such as if Yosseph Getachew gets married and becomes Ephrem Getachew. (In this case, changing the document would then be a matter of looking for all **<name>** elements that aren't in a namespace and changing the values.) In addition, because these XML documents will be used only by your own application, you don't have to create a namespace for it.

However as shown in the preceding code, one of the **<name>** elements occurs under the **<p>** element, which declares a default namespace, meaning that the **<name>** element also falls under that namespace. Therefore, if you searched for **<name>** elements that had no associated namespace, you wouldn't pick this one up. The way to get around this is to use the **xmlns** attribute to cancel the default namespace by setting the value to an empty string, as shown in the following example:

```
<employee>
<name>Yosseph Getachew</name>
  <notes>
    <p xmlns="http://www.w3.org/1999/xhtml">I've worked with <name xmlns="">Yosseph Getachew</name> for over a <em>year</em> now.</p>
  </notes>
</employee>
```

Now the second <name> element is not in any namespace. In this case, you're not declaring that the element is part of a namespace— you're trying to declare that it's not part of any namespace, which is the opposite of what you've been doing so far.