2 XML boot camp

What this chapter covers:

- Getting started with the basics
- What are DTD, XSL, and XML files?
- Elements, entities, RDF, Schemas and more

2.1 Overview

Let's look at the goals set out for XML by the W3C as stated on their website:

- XML shall be straightforwardly usable over the Internet
- XML shall support a wide variety of applications
- XML shall be compatible with SGML
- It shall be easy to write programs that process XML documents
- The number of optional features in XML is to be kept to the absolute minimum, ideally zero
- XML documents should be human-legible and reasonably clear
- The XML design should be prepared quickly
- The design of XML shall be formal and concise
- XML documents shall be easy to create
- Terseness in XML markup is of minimal importance

To be honest, we don't think this needs explaining. What this says to us is that XML should be easy to use, legible to humans, Internet-friendly, and basically all things to all people.

2.2 XML and its derivatives is a huge topic

As you may have already seen, XML (being so extensible) is already an incredibly diverse and complex topic. You can be a guru on an issue or area and be completely ignorant of another up-and-coming area. New XML application servers, solutions, markup languages, and W3C recommendations appear almost every day. In the middle of this information overdose, *XML Programming With VB And ASP* will keep your ship on a steady course in this sea of knowledge.

We will focus on the basics of XML and assume that you can discover the more complex details for yourself. We will restrict our scope of interest to focus on XML from the perspective of a Visual Basic or ASP developer. Also, we'll try to keep the XML syntax at a minimum and keep the solutions and example code at a maximum.

XML was born from SGML and should be compatible.

You may be wondering where XML came from. XML was born from SGML, which is a complex, flexible, and just plain hard-to-work-with markup language. Needless to say, XML has taken off like a rocket because it has some of the best ingredients of SGML without many of the downsides. (We can almost hear all those SGML aficionados reading this as they open their email programs to send us emails on why SGML rules.)

As you have seen, XML was designed from the ground up to be extensible and to be simple to implement. As a result, it seems that attempt to be simple and flexible has

resulted in the massive adoption of XML in the computing industry. Unfortunately anything that is so flexible and extensible will be used for a bewildering array of uses, and eventually the number of choices and solutions begin to overlap and perhaps compete with each other.

2.2.1 Learning more about the XML syntax

To learn more about XML syntax, we recommend reading John E. Simpson's book Just XML *by Prentice Hall.*

Although we will cover a substantial amount of technical detail in this book that will definitely get you more than started, we recommend you source some kind of XML syntax reference book.

If you would like to learn about the details of the syntax of XML, we recommend reading John E. Simpson's book *Just XML*.

2.2.2 The X(ML) files

XML has three basic files: XML, DTD, and XSL.

Most XML solutions have three files, as shown in figure 2.1:

1 XML file that contains the data

2 DTD (or Schema) file that provides structure for the XML file (optional)

3 XSL that provides the *look* or user interface of the XML file (optional)

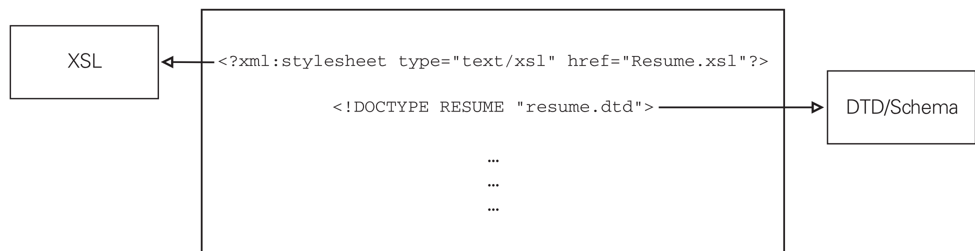

Figure 2.1 Three basic files

We have not yet explained DTDs, Schemas, or XSL, but we will soon. Figure 2.1 is only an overview to get you started. Bear in mind that XML files do not need to have DTD or XSL files, as they are optional and serve different purposes to the core XML file. However, in our examples in this book, we will build all three files.Bear in mind that XML files do not need to have DTD or XSL files, as they are optional.

Let's look at the three steps involved in building a simple XML data file that you can use for your current data. There are many ways to approach this, but we have boiled it down to three simple steps:

1 Discover (or establish) the structure of your data

2 Build the XML file that holds the data
3 Apply (or create) the XSL formatting

2.2.3 *Step 1: Discovering the structure*

This is the first step in creating an XML document. We need to look at our data and extract the important concepts. To do this, ask yourself what each section or sentence is about conceptually. In essence you are building up a vocabulary that can be reused across similar documents. To demonstrate this, let's build a sample resumé or CV. Now this is something we are all intimately familiar with, we imagine!

First we will define the words we need and then we will set them out in a hierarchical order. We will not put actual information inside this vocabulary; rather we will only define the data structure. Of course, there are many things we can put in this vocabulary, but let's only create a basic structure. A possible vocabulary for an XML document that contains a resumé could be:

```
1DPH
7HOHSKRQH QXPEHU
&RPSDQ\ (ZKLFK LV PDGH XS RI)
&RPSDQ\ QDPH
&RQWDFW SHUVRQ
&RQWDFW WHOHSKRQH QXPEHU
&RQWDFW HPDLO
3RVLWLRQ
'HVFULSWLRQ
```

Now that you know the XML file will hold all of its data in between a beginning and end tag (this is correctly referred to as an element), we can also understand the benefit of separating the structure from the data. As we mentioned before, that means that many different XML files can reuse the same structure file. This structure file is formally known as a DTD (Document Type Declaration).

DTDs: Reusing the structure in your data and industry

As XML becomes widespread, your industry association or company is likely to have one or more published DTDs that you can use and link to. These DTDs define tags for elements that are commonly used in your applications. You don't need to recreate these DTDs—you just point to them in your XML file and follow their structure when you create your XML document.

Note Schemas are a better structure document than DTDs for industry-wide data exchange. Schemas provide better datatypes, and they are already being used in document repositories such as http://www.biztalk.org

---

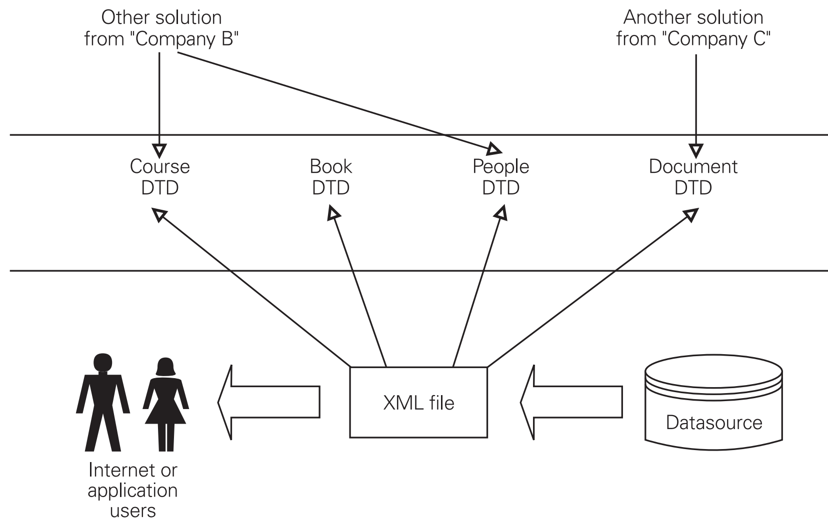

Figure 2.2 DTDs provide standards

We have already seen how we can use XML and a DTD to simplify the exchange of information between different sources. In addition, the increasingly widespread use of vertical industry DTDs means that systems can use these common DTDs to exchange information with each other, regardless of each business's internal format or storage architecture. In this way, industries will begin to reuse common formats as shown in figure 2.2.

Elements

An element consists of a start tag, an end tag, and what's between them.

As you already know, similar to HTML, XML also uses tags to define the beginning and end of each piece of data or concept. Actually, the correct word for this is an element. Each element can hold many other elements. In this way, we saw the company element holding other elements, including CONTACT NAME, CONTACT PERSON, and CONTACT TELEPHONE NUMBER.

Let's have a closer look. What consists of a start tag, an end tag, and what's between them is an element:

```
<%22.!...<7,7/(!;0/ 3URJUDPPLQJ :LWK 9% $QG $63</7,7/(!...</%22.!
```

`<%22.!` and `<7,7/(!` are both elements, even though `<7,7/(!` is within the `<%22.!` element.

Attributes

Attributes are parts (or properties) of elements.

When elements contain properties, they are called attributes. In the following example, the element is BOOK and the attribute is TITLE. The value of the attribute is XML Programming With VB And ASP.

```
<%22. 7,7/( ";O/ 3URJUDPPLQJ :LWK 9% $QG $63"!...</%22.!
```

Attributes are parts or properties of elements.

Entities

Any file, charater representation, or web resource that can be included into an XML file is an entity.

To put it simply, any file or web resource that can be *included* into an XML file can be called an entity. Any file or web resource that can be included into an XML file is an entity. Entity is also used to refer to special character representations and substitutions of text strings and includes.

Here's an example of substituting entities for text strings:

```
<!(17,7< %RRN1DPH ";O/ 3URJUDPPLQJ :LWK 9% $QG $63"!
```

Now you can use the entity &%RRN1DPH; in a document, and wherever you refer to it, the entire string of ;O/ 3URJUDPPLQJ :LWK 9% $QG $63 will be substituted. In VB, this is similar to using a constant. Also, by defining this in a DTD and keeping the DTD external, you can ensure that the same definitions are reused accurately across all your XML files.

Say we placed this in our DTD:

```
<!(17,7< ,QWURGXFWLRQ 6<67(0 "KWWS://PDUN/[POFRGH/LQWUR.[PO"!
```

We have used the 6<67(0 keyword to indicate that the file being pointed to is external to the DTD.

Then wherever we used the keyword &Introduction; the XML text in intro.xml at our website would be included.

Just as useful is being able to include graphics. Say we used:

```
<!127$7,21 JLI 6<67(0 "KWWS://PDUN/[POFRGH/JLIYLHZHU.H[H"!
<!(17,7< P\SLF 6<67(0 "KWWS://PDUN/[POFRGH/P\SLF.JLI" 1'$7$ JLI !
```

The mypic.gif would be placed wherever we use the &P\SLF; keyword.

Of course you may have been wondering about the <!127$7,21 piece. Well, that is the location of a helper program that can view the gif. The parser knows to look for this notation because at the end of the !ENTITY line, 1'7 gif was inserted, indicating that a notation for the gif is included.

To attribute or to element, that is the question!

An interesting similarity between attributes and elements is that this %22. element contains an attribute of PUBLISHER:

```
<%RRN 3XEOLVKHU "ODQQLQJ"!;O/ 3URJUDPPLQJ :LWK 9% $QG $63</%RRN!
```

But you could also change the attribute into its own element. Then you would use:

```
<3XEOLFDWLRQ!
        <%RRN!;O/ 3URJUDPPLQJ :LWK 9% $QG $63</%RRN!
        <3XEOLVKHU!ODQQLQJ</3XEOLVKHU!
</3XEOLFDWLRQ!
```

The two alternatives shown above don't look the same, but they are in fact the same in many ways. Of course, there are some subtle differences between elements and attributes. When choosing how to structure your data, consider the following:

- The order of attributes cannot be controlled
- You cannot specify different types of elements, though you can specify (to a limited extent) the contents of elements through the use of a DTD or Schema
- You can specify an attribute as an "ID" type of attribute, which can then be used from various DOM methods
- Attributes cannot contain subelements; only elements can contain other elements

Inline or standalone?

Bearing in mind that the structure or vocabulary is held in a DTD document, let's look at various ways to use this vocabulary we have identified. We have at least three possibilities:

- We can create a DTD that is *standalone*, external to the XML document, and therefore reusable across several documents
- We can create a DTD that is held within our XML file and is therefore only available to this file
- We can avoid defining a DTD file completely (since it is an optional file) and simply get on with creating an XML data file itself

In the solution in this book, we will create and use standalone and reusable resume structures.

A simple DTD (Document Type Definition)

Because a DTD defines the structure of our XML document, our DTD file could look like this:

```
<!-- 7KLV LV D FRPPHQW ZLWKLQ DQ H[DPSOH UHVXPH.GWG --!
<!(/(0(17 0<1$0( $1<!
<!(/(0(17 0<7(/(3+21(180%(5 $1<!
<!(/(0(17 &203$1<  (&203$1<1$0( , &217$&73(5621 , &217$&77(/(3+21(180%(5 ,
&217$&7(0$,/  , 326,7,21  , '(6&5,37,21  )!
<!(/(0(17 &203$1<1$0(  (#3&'$7$ )!
<!(/(0(17 &217$&73(5621  (#3&'$7$ )!
```

```
<!(/(0(17 &217$&77(/(3+21(180%(5  (#3&'$7$ )!
<!(/(0(17 &217$&7(0$,/  (#3&'$7$ )!
<!(/(0(17 326,7,21  (#3&'$7$ )!
<!(/(0(17 '(6&5,37,21  (#3&'$7$ )!
```

DTDs can be much more complex than this example—and they typically are—but this gives you a sense of what they can do. It's just a matter of structuring your data and figuring out the *parts* of your content.

An XML document only has one root element, and in this case, it is the `<5(680(!</5(680(!` element. This is not listed within the DTD file, but within this root element, we define a dataset element called `<&203$1<!` that is made up of six different parts:

```
<!(/(0(17 &203$1< (&203$1<1$0(, &217$&73(5621,
&217$&77(/(3+21(180%(5, &217$&7(0$,/, 326,7,21, '(6&5,37,21)!
```

If you are interested, HTML is rumored to have 26 or more DTDs that you can use. You can read about one of them at: http://www.w3.org/TR/REC-html40/loose.dtd

We then identify each of the parts within the dataset. What we want to do is identify what type of information each one can and will hold.

So far in this book, we have occasionally mentioned Schemas. Let's take a closer look at them and how they compare to DTDs.

Schemas and DTDs—what's the difference?

DTDs have four major things going for them:

- They are easy to learn and are a standard
- They are mature and are used in several thousand applications
- They have an enormous number of trained developers already using them from the SGML heyday
- There are many tools out there for using DTDs

Schemas were born out of the limitations to DTDs. For example:

- DTDs are not written in XML, and therefore they are not available via the DOM
- DTDs do not have support for namespaces
- DTDs are good at describing structure, but not the data contents—their data typing is poor and limited. There is no support for currencies and so on

Schemas, on the other hand:

- Are built in XML and can therefore be used via the DOM in Visual Basic or VBScript in ASP
- Provide datatypes such as float, currencies, and so on
- Provide better relationships between elements
- Enable you to create your own user-defined data types

- Indicate some form of inheritance
- Support namespaces

Since Schemas are XML documents themselves, there is a DTD (or Schema) available for evaluating if your Schema itself is valid.

While Schemas will definitely figure into your future projects, a consensus view is that the final proposal of the working group on Schemas will not look like the current implementation of Schemas within Microsoft IE5. For example, in August 1998 the Schema syntax changed substantially. It is expected that will happen again once the working draft becomes a recommendation. Therefore this book will be focusing mainly on DTDs and giving pointers to Schemas. For more information, see chapter 8, "Schemas, BizTalk, and eCommerce."

2.2.4 *Step 2: Building an XML file*

Let's create our first XML file now. We have defined our DTD and now we can create an XML document that conforms to the structure of the DTD.

In a nutshell, most XML applications will extract data from a database, mark it up with XML, and send it to the user. There are many choices of what to do to the file during the markup stage. As a result, several different types of documents can end up at the client's browser.

Overall, the pattern shown in figure 2.3 is followed.

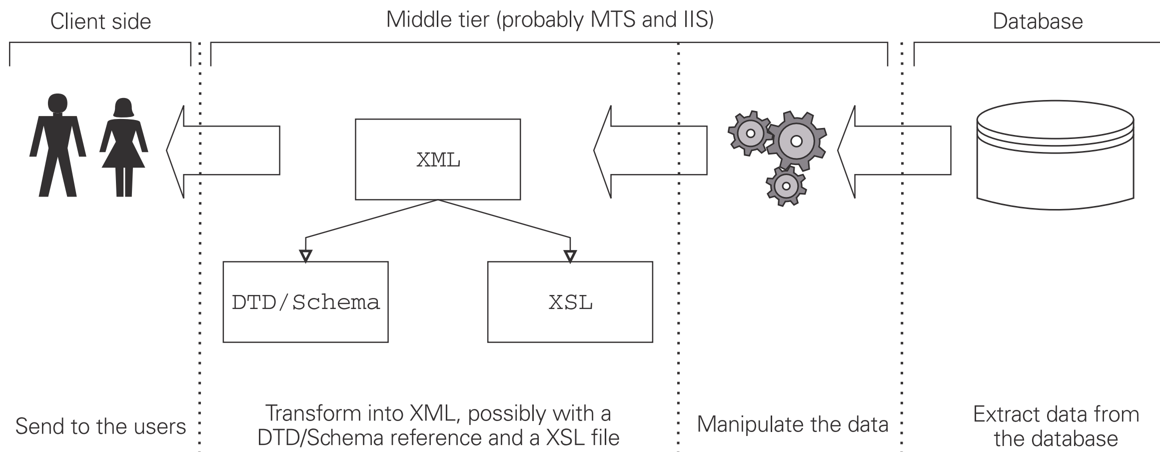

Figure 2.3 Typical development pattern

What you see below is an XML file with standard headings, followed by some (fictitious) personal details, and then followed by details on two companies we have (fictitiously) worked for. Note how there are no spaces within the tag names, as a sample

company name becomes <&203$1<1$0(!. Also note how the tag <&203$1<> holds six tags within itself (just as our DTD said it would).

Our resume.xml file looks like this:

```
<"[P0 YHUVLRQ "1.0" HQFRGLQJ "87)-8""!
<!-- 7KLV LV D VDPSOH ILOH IRU WKH 5HVXPH ;0/ DSSOLFDWLRQ ZH DUH JRLQJ WR
EXLOG --!
<"[P0:VW\OHVKHHW W\SH "WH[W/[VO" KUHI "UHVXPH.[VO""!
<!'2&7<3( 5(680( 6<67(0 "UHVXPH.GWG"!
<5(680(!
  <0<1$0(!0DUN :LOVRQ</0<1$0(!
  <0<7(/(3+21(180%(5!001 123456</0<7(/(3+21(180%(5!
    <&203$1<!
      <&203$1<1$0(!,QGLJR ZHE GHVLJQ</&203$1<1$0(!
      <&217$&73(5621!6DQG\ 5RELQVRQ</&217$&73(5621!
      <&217$&77(/(3+21(180%(5/!
      <&217$&7(0$,/!LPSRUWDQW#LQGLJRZHE.FRP</&217$&7(0$,/!
      <326,7,21!
        <52/(!:HEPDVWHU</52/(!
        <'(6&5,37,21!7KLV ZDV DQ H[FLWLQJ SURMHFW ZKHUH ZH ZRUNHG RQ
          LQWHJUDWLQJ...</'(6&5,37,21!
      </326,7,21!
    </&203$1<!
    <&203$1<!
      <&203$1<1$0(!3XUSOH SHRSOH HDWHUV ZHE GHVLJQ</&203$1<1$0(!
      <&217$&73(5621!5REL 6DQG\VRQ</&217$&73(5621!
      <&217$&77(/(3+21(180%(5!(003) 123456</&217$&7-7(/(3+21(180%(5!
      <&217$&7(0$,/!LPSRUWDQW#SSZHE.FRP</&217$&7(0$,/!
      <326,7,21!
        <52/(!6HQLRU ZHE GHYHORSHU</52/(!
        <'(6&5,37,21!, GHYHORSHG WKHLU LQWUDQHW VROXWLRQ...</'(6&5,37,21!
      </326,7,21!
    </&203$1<!
</5(680(!
```

We’re sure the following four lines in the XML header caught your attention:

```
<";0/ YHUVLRQ "1.0" HQFRGLQJ "87)-8""!
<!-- 7KLV LV D VDPSOH ILOH IRU WKH UHVXPH ;0/ DSSOLFDWLRQ ZH DUH JRLQJ WR
  EXLOG --!
<"[P0:VW\OHVKHHW W\SH "WH[W/[VO" KUHI "UHVXPH.[VO""!
<!'2&7<3( 5(680( 6<67(0 "UHVXPH.GWG"!
```

The following line tells the reader (for example, a web browser) which version of XML and which text encoding you are using (UTF-* is the default):

```
<"[P0 YHUVLRQ "1.0" HQFRGLQJ "87)-8""!
```

<" and "! are delimiters that tell the reader that there is a PI (processing instruction), which in this case is the version of XML that is being used.

The characters `<!--` *begin a comment, and the characters* `-->` *end the comment.*

Note in the following line that the characters `<!--` begin a comment and the characters `--!` end the comment:

```
<!-- 7KLV LV D VDPSOH ILOH IRU WKH UHVXPH ;0/ DSSOLFDWLRQ ZH DUH JRLQJ WR
EXLOG --!
```

This comment will go unprocessed and not be displayed. It can even hold characters such as < and `!`.

This line tells the reader that there is a document called a stylesheet, which will explain to the reader (for example, a Web browser) how to display our resume.xml file:

```
<"[PO:VW\OHVKHHW W\SH "WH[W/[VO" KUHI "UHVXPH.[VO""!
```

A version of XSL has shipped with IE5, but be aware that at the time of writing the XSL specification is undergoing radical change, and the final various XSL specifications have not been recommended at the W3C.

CSS (Cascading Stylesheet) provides a way to define a look (such as *use bold and italics on the heading in font size 10*) across several HTML pages. With an external XSL (eXtensible Stylesheet Language), you have an easy way to define a look (or presentation) for more than just one XML file.

This line tells the reader that there is a document against which the structure of this XML file should be checked:

```
<!'2&7<3( 5(680( 6<67(0 "UHVXPH.GWG"!
```

This new document is called a DTD. A DTD for a document need not always be an external file. The DTD for our resume.xml can be held within itself. However, since one of the useful points of a DTD is that it provides a way to use a common structure across a company or across an industry, it makes sense to provide it in an external or separate file. Using the keyword `6<67(0` means the DTD can be found outside of the current XML document.

Validation and parsing

Using the keyword `SYSTEM` *means the DTD can be found outside of the current XML document.*

Since our resume.xml contains a reference to a DTD, when our resume.xml document is parsed, its structure is compared with the resume.dtd to ensure that the structure is correct (assuming, of course, that the parser intends to validate the document). This comparison process is called validation and is performed by a tool called a parser.

IE5 doesn't automatically validate HTML files on the web.

Although the specification calls for programs to automatically validate XML files when displaying the XML file, IE5 does not. It is assumed that web surfers don't need to know if the file structure is incorrect. Logically, the onus is on the developer to get the implementation of the XML file right before releasing it to the public.

To have IE5 validate your XML file, you need to use code or scripting. This book covers this issue and many other code examples.

The difference between well-formed and valid documents

Valid means your XML file does not break any of the rules imposed on XML and correctly uses all external references—such as a DTD.

Before we go any further, let's clarify an important concept—the differences between *well-formed* and *valid*. You will find many explanations of these two terms, but for our needs, here is the simplest interpretation:

> XML files that follow the standard syntax for XML are *well-formed*. XML files that are well-formed and also correctly follow the structure of a DTD are considered *valid*.

A few things to remember about XML and DTDs

Valid is a superset of well-formed, and it is easier to be well-formed than it is to be valid.

- XML files do not need to have DTDs unless the developer is sure that a specific structure must always be followed
- DTDs can be held internally within the XML document. Alternatively, if you need to share a common structure across several XML files, you can make the DTD standalone. In thiscase, you use the 6<67(0 keyword
- XML files that follow the standard syntax for XML are well-formed
- XML files that are well-formed and also correctly follow the structure of a DTD are considered valid
- XML is by design case-sensitive (though tool builders may not adhere to this)
- Every XML document can only have one root element

What does the syntax look like if the structure goes deeper than just one level, as we have above? In addition, how do we show no information between tags?

Our new resume1.xml file looks like the following. Note that the <326,7,21! tag now contains the <52/(! and <'(6&5,37,21! tags.

```
<"[PO YHUVLRQ "1.0" HQFRGLQJ "87)-8""!
<!-- 7KLV LV D VDPSOH ILOH IRU WKH 5HVXPH ;0/ DSSOLFDWLRQ ZH DUH JRLQJ WR
   EXLOG --!
<"[PO:VW\OHVKHHW W\SH "WH[W/[VO" KUHI "UHVXPH.[VO""!
<!'2&7<3( 5(680( 6<67(0 "KWWS://PDUN/[POFRGH/UHVXPH.GWG"!
<5(680(!
    <0<1$0(!ODUN :LOVRQ</0<1$0(!
    <0<7(/(3+21(180%(5!001 123456</0<7(/(3+21(180%(5!
        <&203$1<!
            <&203$1<1$0(!,QGLJR ZHE GHVLJQ</&203$1<1$0(!
            <&217$&73(5621!6DQG\ 5RELQVRQ</&217$&73(5621!
            <&217$&77(/(3+21(180%(5/!
            <&217$&7(0$,/!LPSRUWDQW#LQGLJRZHE.FRP</&217$&7(0$,/!
            <326,7,21!
               <52/(!:HEPDVWHU</52/(!
               <'(6&5,37,21!7KLV ZDV DQ H[FLWLQJ SURMHFW ZKHUH ZH
                   ZRUNHG RQ LQWHJUDWLQJ...</'(6&5,37,21!
            </326,7,21!
        </&203$1<!
```

```
    <&203$1<!
       <&203$1<1$0(!3XUSOH SHRSOH HDWHUV ZHE GHVLJQ</&203$1<1$0(!
       <&217$&73(5621!5RELQ 6DQG\VRQ</&217$&73(5621!
       <&217$&77(/(3+21(180%(5!(003) 123456</&217$&77(/(3+21(180%(5!
       <&217$&7(0$,/!LPSRUWDQW#SSZHE.FRP</&217$&7(0$,/!
       <326,7,21!
        <52/(!6HQLRU ZHE GHYHORSHU</52/(!
        <'(6&5,37,21!, GHYHORSHG WKHLU LQWUDQHW VROXWLRQ...</'(6&5,37,21!
       </326,7,21!
    </&203$1<!
</5(680(!
```

We declared the DTD with:

```
<!'2&7<3( 5(680( 6<67(0 "KWWS://PDUN/[POFRGH/UHVXPH.GWG"!
```

You can see we inserted the keyword `6<67(0` and followed it with the name of an external (to the document) DTD file.

The format of a DTD declaration is:

```
<!'2&7<3( QDPH H[WHUQDO'7'SRLQWHU >LQWHUQDO'7'VXEVHW@!
```

To show an empty tag, you could use:

```
<&217$&77(/(3+21(180%(5! </&217$&77(/(3+21(180%(5!
```

This is the shortened version:

```
<&217$&77(/(3+21(180%(5/!
```

We also saw the following new line.

```
<"[PO:VW\OHVKHHW W\SH "WH[W/[VO" KUHI "UHVXPH.[VO""!
```

To `6<67(0` *or not to* `6<67(0`*, that is the question! Making the DTD standalone means you can reuse the same DTD across several XML files.*

Up to this point we have been dealing with two files: the XML file and the DTD file. We know that the data is held in the XML file and structures are held in the DTD file, but what we have not seen yet is how this data will be displayed. That is the role of a stylesheet. We will focus on stylesheets a bit later. For the moment, we will concentrate on understanding a more complex DTD for the resume1.xml file.

CDATA and #PCDATA

In some instances you may be working with restricted characters, or you may want to have script or HTML passed directly through without the parser accessing the data. There are several element declarations you can use; two of the most popular—CDATA and #PCDATA—are described below.

In a very elegant way, XML has managed to separate the data from structure and the display. All three will typically be in separate files and can be reused across an organization or an industry.

In the examples above, we handled the restricted character of < by changing it to `&JW;`, and we then placed it into a CDATA section.

character data (CDATA):

By design this should receive no parsing, but this depends of course on the parser, which could look for the use of `@@!` and other XML-reserved sets of characters. The contents are treated as characters to be passed to the application.

parsed character data (#PCDATA):

For example, `&JW;` will be parsed into < on output. (The `#` ensures that this tag is not seen as a tag that the designer has created.)

Namespaces provide uniqueness

We know that XML is a language for creating markup languages, and the whole point of XML is to enable users to be able to create unique tags that identify their information in more meaningful ways than simply applying the basic set of HTML tags to all documents. While this gives users great flexibility, it poses problems for interchange and software integration. What happens when two documents make use of the same tag names in different contexts?

Table 2.1 shows all your alternative element declarations.

Table 2.1 The XML document content model

Type	Element Declaration	Description
Parsed character data	#PCDATA	The contents should be ignored by the parser
Character data	CDATA	Certain parts of the contents shuld be converted to characters
Empty data	EMPTY	Declares that an element cannot contain any contents
Any data	ANY	This type of element can contain any contents allowed by the DTD
Mixed	#PCDATA|x|y|z	Provides a set alternative

The W3C definition of namespaces is at http://www.w3.org/TR/REC-xml-names/

For example, within a single XML document, the tag `<7,7/(!` may refer to the name of the employer whom you worked for:

```
<&217$&73(5621!
<7,7/(!'U.</7,7/(!
<6851$O(!$QGUHZV</6851$O(!
</&217$&73(5621!
```

It could also refer to the title of the document itself:

```
<7,7/(!ODUN :LOVRQ· V UHVXPH</7,7/(!
```

In addition, if you have a section where you listed books or training courses, you may have:

```
<&2856(6$1'%22.6!
<7, 7/(! %RRN1</7, 7/(!
<7, 7/(! &RXUVH1</7, 7/(!
<7, 7/(! &RXUVH2</7, 7/(!
<7, 7/(! %RRN2</7, 7/(!
</&2856(6$1'%22.6!
```

What you need is to explain how each instance of the <7, 7/(! tag is different from the other ones. How can you help a software program know what each one is? That's the basic reason why current search engines are not perfect. They have to guess which context applies to which words.

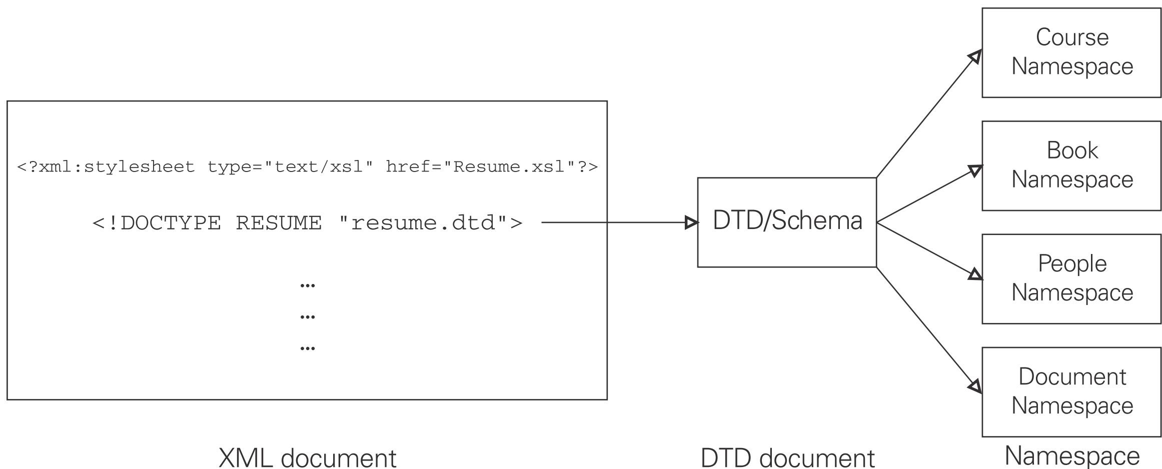

Figure 2.4 Namespaces ensure uniqueness

Enter XML namespaces. Namespaces allow tags like <7, 7/(! to have uniqueness and a specific context for each instance of <7, 7/(!. Then when you refer to a <7, 7/(! tag, you can include a reference to the context. The uniqueness is provided by pointing to a URI (Universal Resource Indicator), which is similar to but broader than a URL. And because each URI is unique, that makes each namespace unique.

Note It is important to understand that there is no document at the URI indicated by the URI. The URI is only used to ensure uniqueness for each instance of a tag.

While the human reader can distinguish between the different interpretations of the `<7,7/(!` element, a computer program does not have the context to tell them apart. Namespaces solve this problem by associating a namespace with a tag name. For example, the titles can be written as (see figure 2.4):

```
<%RRN,QIR:7,7/(!(DWLQJ OU. %LJJOHVZRUWK</%RRN,QIR:7,7/(!
<$XWKRU,QIR:7,7/(!OLQL PH</$XWKRU,QIR:7,7/(!
```

The name preceding the colon refers to the name of the namespace and ensures that identically named elements are not confused. Both humans and computers can tell which `<7,7/(!` is being referred to.

Now, in the DTD for our own document, we can refer to these namespaces. The default declaration declares a namespace for all elements within scope. The scope below is from

```
<%22.! WR </%22.!
```

All tags within this scope reference the stated namespace:

```
<"[PO YHUVLRQ "1.0" HQFRGLQJ "87)-8""!
<%22. [POQV "XUQ:%RRN/RYHUV.RUJ:%RRN,QIR"!
<7,7/(!%RRN1 </7,7/(!
 <35,&( FXUUHQF\ "86 'ROODU"!34.95</35,&(!
</%22.!
```

You can also declare a default namespace at the start of your XML document; any tags without prefixes are assumed to be in the default namespace.

But this is not complex enough. In the example above, we did not show two `<7,7/(!` elements, right? What we need to do is to declare several namespaces and somehow use them within the elements. The following example declares `%22.` and `&2856(` to be shorthand for the full names of their respective namespaces. Now, each element can have the same name but reference different namespaces!

Note The W3C definition of XML namespaces is: An XML namespace is a collection of names, identified by a URI, which are used in XML documents as element types and attribute names. XML namespaces differ from the namespaces conventionally used in computing disciplines in that the XML version has internal structure and is not, mathematically speaking, a set.

Now we can use the following tags in our XML document because we have included those namespaces in the DTD:

```
<"[PO YHUVLRQ "1.0" HQFRGLQJ "87)-8""!
<%22. [POQV:EN "XUQ:%RRN/RYHUV.RUJ:%RRN,QIR"
         [POQV:&2856( "XUQ:7UDLQLQJ.RUJ:&RXUVHV"!
```

```
<&2856(:7,7/(!/HDUQLQJ ;0/</&2856(:7,7/(!
<EN:7,7/(!%RRN1</EN:7,7/(!
<EN:7,7/(!'U.1R</EN:7,7/(!
<EN:7,7/(!0\ UHVXPH</EN:7,7/(!
</%22.!
```

What you can see above is that we defined a namespace of EN and &2856(6 within the root tag of BOOK. We have then prefixed the <7,7/(! element with the namespace <&2856(! and the other <7,7/(! element with the EN namespace. Now humans and machines can easily differentiate between the two elements.

2.2.5 Step 3: Using style in your design

W3C XSL Working Draft is at http://www.w3.org/TR/WD-xsl

You can find out more about the working draft of one of the modules of the new CSS3 at http://www.w3.org/TR/CSS3-selectors

XSL is a language for expressing stylesheets. In the past CSS provided the *look and feel* for HTML documents on the web. However, now CSS2 adds support for *paged media*, which mainly relates to paper or transparencies. While CSS continues to evolve in this way, and many XML pundits still use CSS as their preferred stylesheet over XSL, we will be delving into XSL as our preferred stylesheet for this book.

However, before we get going on the Microsoft XSL implementation, be aware that the XSL specification has already expanded and changed since IE5 and its objects were shipped. Although you can use XSL in Microsoft IE5, it is undergoing major changes at the W3C. They call it "modularization." Let's describe the W3C changes that you will be encountering in the future before we discuss the XSL implementation, which you can find in the Microsoft XML Objects in IE5.

Here we describe some of the W3C changes that you will encounter in the future, not the current Microsoft IE5 XSL implementation.

XSL can be used in two ways. It can be used to transform (this is called XSL Transformations, or XSLT) and for rendering (by using the XSL Formatting Objects). Although there are an increasing number of tools available, there is at the time of writing no browser implementation for native XML/XSL. For example, in IE5, your XML documents must first be converted to HTML (using either CSS or XSL) before IE5 can display it. Netscape 5.0 (also named Mozilla) shows great promise of working with XML natively and being totally standards-compliant.

XSLT: a transforming stylesheet

XSLT is a modular part of XSL, as there is a separate XSL specification. Both are a part of the W3C *Style Sheets* activity, and XSLT is expected to become recommended before XSL does. It is not a general-purpose display language as XSL is; rather XSLT is designed primarily for the kinds of transformation that are needed when transforming one XML document into another XML or HTML document.

Note If you are interested in having a look at Mozilla, then for a bit of fun go to http:/

/www.mozillazine.org and check out their developers, the discussions, and all kinds of interesting uses they are putting XML to (such as extensible and custom user interfaces). You can download the latest development version of Mozilla at http://www.mozilla.org

---

The latest working draft of XSLT can be found at http://www.w3.org/TR/xslt

The intention of XSLT is neatly demonstrated in this way. As you may already know, XML DOM objects can be referred to as a *tree*. In this analogy, an XSLT describes rules for transforming a source tree into a result tree. A *pattern* is matched against the source tree, and this creates the result tree, with the result tree being separate from the source tree. That's the important thing! It is intended to be a separate tree that can have a completely different structure from the source tree. The pattern that is applied may filter, order, or add structure to the new result tree.

We can summarize by saying that XSLT is designed primarily for the kinds of transformation that are needed when transforming one XML document into another XML or HTML document. It is not a general-purpose language like XSL for specifying the display (or rendering) of XML documents.

Note The W3C Style Sheets activity can be found at http://www.w3.org/ Style/Activity

---

The Microsoft implementation of XSL

XSLT is not a part of the Microsoft IE5 objects, as this proposal appeared after the browser was shipped. However, the good news is that the Microsoft IE5 objects provide all of the functionality you are looking for.

More good news is that this book does not focus on the (evolving) W3C XSL standards, but on the Microsoft IE5 implementation of XSL as objects. This means you can read this book, learn from its examples, and get going immediately regardless of the changes to underlying specification. The Microsoft IE5 objects provide a rich set of methods to use on your next project, and we will show you how to use them.

2.2.6 RDF (Resource Description Framework)

RDF is good at providing a way to express complexity and complex data (metadata).

RDF has been described as an *application* of XML, an *API* to XML, and also as a way of associating metadata to anything that has a URI or a web address.

Either way, it inherits all the benefits of XML, and it goes one better by providing *machine-readable* descriptions of objects. Eventually all web resources will be described either in or through RDF, and thereafter even the relationships between objects will be explained as well. We can just see those search engines revving to go!

RDF is good at providing a way to express complexity and complex data (metadata) and it will be a good way to allow applications to exchange their metadata. RDF is tar-

geted at application areas such as: the description of Internet resources, site-maps, content rating for websites, electronic commerce, EDI, collaborative applications, or services.

Note W3C activity for RDF can be found at http://www.w3.org/Metadata/Activity. Also, for the Dublin Core for A Simple Content Description Model for Electronic Resources, check http://purl.oclc.org/dc/for.