9 XML: Extensible Markup Language

LEARNING OBJECTIVES

At the end of this chapter, you will be able to:

- Define the XML and list its features.
- Describe the basic syntax of an XML document.
- Describe the key components of the XML including elements, attributes and namespaces.
- Present the user data in XML format.
- Define the purpose of the DTD and schema.
- Describe the syntax of the DTD and schema.
- Write a DTD or schema for a given XML document.
- List the applications of the XML.

In this chapter you will learn about the basics of XML and how it can be used to store information away from the mechanism of processing or formatting of such data. You will learn how to build simple XML files, and be able to manipulate and refer to them.

The primary use of XML is the description of data rather than presentation. Initially this would be to extract data away from HTML and focus on the *semantics* (the meaning!) rather than the formatting. XML has far more to it than just this though. XML is, as the name suggests, a language, or rather a meta-language, which can be used to store data and act as a mechanism to transfer information between dissimilar systems.

XML does not actually do anything by itself, it simply describes data. To have anything happen with that data it needs to be processed or transmitted or whatever it is that needs to be done.

Here we explore its basic use and capabilities before moving on to more advanced topics.

9.1 Introduction to XML

XML is a language to describe other languages. It's possible to use it to actually make up other markup languages and in this book we have met it before in the chapters describing XHTML. You can use it to describe any kind of data that you can think of, whether it be mathematical, genealogical, scientific or business data. Several languages are based on XML including SVG, XHTML, RSS, MathML, RDF, WAP and WML.

XML came about in the mid-nineties as a solution to perceived problems with the WWW. Practitioners of SGML saw that it held some answers to these problems and in 1996 activity commenced into researching the possibilities by an 11-member working group. In 1998 XML 1.0 became a W3C recommendation.

Its main purpose is to allow the sharing of data across different systems and it is particularly useful in this sense for applications that do this over the Internet.

These are the main features of XML:

1. It is in a format that both humans and machines can read. There is nothing special about XML, it's just plain text, made up of normal characters and angled brackets.

2. It supports unicode, which is capable of encoding all human languages.
3. It supports data structures well.
4. It is self-documenting.
5. It has a strict format that makes it easy for parsing to take place.
6. It can be understood and exchanged between dissimilar systems.
7. It can be useful in swapping data between different applications.

9.2 The Many Uses of XML

In a Web context XML can be used to separate the data away from HTML, which should be concerned with presentation. The HTML is then concerned with the layout of such data and it doesn't matter if that data changes. In this way you would not change your HTML if you wanted to add or amend the data.

9.3 Simple XML

An XML document has a simple and self-describing syntax. Listing 1 shows a simple example containing an item described in XML for a database containing people information.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<person>
<first>Fred</first>
<last>Bloggs</last>
<birthdate>20th March 1977</birthdate>
<birthplace>Leeds</birthplace>
<employed started="12/12/04">Leeds City Library</employed>
</person>
```

Listing 1 A simple XML document.

The first line is the XML declaration and gives the XML version and character encoding used. The next line contains the root element, `<person>`. All XML documents contain a root element, defining an element that is descriptive of that document as a whole, which in this case says 'this is a person'. The next few lines contain the data and also describe that data. For example, the next line after the root element is:

```
<first>Fred</first>
```

In this case the line contains a first name, Fred, which is wrapped in a descriptive element, `<first>`. Similarly, the next few lines describe the data containing the last name, the birth date, birth place and current employer until the root element is finally closed on the last line:

```
</person>
```

As you can see this is fairly self-descriptive and readable by you!

9.4 XML Key Components

One of the key aspects of XML is how strict the syntax is. Here we go into more detail regarding how the language is constructed.

Elements

The strict syntax of XML contains a few rules about elements that must be adhered to:

1. Elements must have a closing tag.
2. Tags are case sensitive.
3. Elements must be nested correctly.
4. XML documents must have a root element.

The closing tag in XML is important; for example, older forms of HTML allowed elements to miss out the closing tag but this would be illegal with XML and therefore XHTML! It's also important to remember that XML is case sensitive so:

```
<birthdate>20th March 1977</birthdate>
```

would not be the same as:

```
<Birthdate>20th March 1977</Birthdate>
```

In HTML it is sometimes possible to improperly nest elements inside each other:

```
<b><i>Some text</b></i>
```

whereas it should be more properly done like this:

```
<b><i>Some text</i></b>
```

In XML it is essential to nest elements in the correct order.

Attributes

There is an attribute in Listing 1 containing the person's start date at his job. Attributes can be added to elements in XML but must always be quoted. For example, this would be incorrect:

```
<employed started=12/12/04>Leeds City Library</employed>
```

To make it legal XML syntax, the attribute must be enclosed in quotes:

```
<employed started="12/12/04">Leeds City Library</employed>
```

Other Essentials

There are a few other important aspects to keep in mind when working with XML.

For example, white space is preserved in XML where as in HTML it is truncated down to just a single space. So, you may have text in HTML like this:

```
Hello World, this is a simple document
```

which will be displayed as:

```
Hello World, this is a simple document
```

XML the spaces would be kept as they are.

Another interesting XML fact is that a carriage return (CR) and linefeed (LF) combination, sometimes oduced at the end of text lines by word processors, becomes translated to just a linefeed with XML.

One thing does remain in common with HTML though: comments can be added using the triangle ckets like this:

```
<!-- here are some remarks -->
```

Namespaces

Sometimes in XML there is a danger of conflicting names between documents; for example, it would be quite easy for two different documents to contain the same named element. It's even more likely if you are sharing documents with someone else. It may be that you have used the word 'name' for an animal database and the other person has used it for a region identity. A problem like this can be resolved by adding a prefix to the elements, such as `animal:name`, or even your own name for that particular document. However, these may also not be unique enough.

It is possible instead to associate a namespace with the elements in use, providing a unique name. When a namespace is used it is not for looking up information by the parser but simply to provide the unique name. It is sometimes used though to give a Web site address about the namespace.

Namespaces usually take the form of a URL, beginning with a domain name, an optional namespaces label in the form of a directory name and finally a version number, which is also optional:

```
xmlns="http://www.mydomain.com/ns/animals/1.1"
```

Checkpoint!

- XML means Extensible Markup Language; a meta-language that describes other languages.
- Where HTML deals with the presentation and formatting of Web documents, XML is concerned with the semantics or meaning of data.
- It stores data in such a way as to be understandable and readable by both humans and computers.
- It has a strict syntax; for example, tags must be closed and nested properly. It is also case sensitive.
- It's useful for both storing information and the transmission of information.
- To help prevent name conflicts between documents it is possible to use a prefix or a namespace.

Concept Check

- *Develop* an XML document that will hold a music collection with fields for artist name, record company, date released and format (CD, Minidisk, DVD).

9.5 Document Type Definitions and Schemas

XML is particularly concerned with being *well formed* or correct in syntax. There are two ways of checking whether a document follows the expected order and structure: Document Type Definitions (DTDs) and schemas. These template schemas are checked as the parser scans the document.

DTDs

DTDs can be declared in the XML document itself, or as an external file.

In Listing 2 the first line declares the XML version and then proceeds into the DTD. `DOCTYPE` defines that this is a document of type `memo`. It is then defined as having four elements, `to`, `from`, `title` and `message`. These elements are each in turn defined as having type `PCDATA`, which stands for parsed character data, which refers to everything except markup; this includes numbers, letters, symbols and entities. A DTD describes an XML document in terms of elements, tags, attributes, entities, `PCDATA` and `CDATA`. Most of these have been described before apart from `CDATA`, which is the equivalent of a comment that will not be processed.

```
<?xml version="1.0"?>
<!DOCTYPE memo [
  <!ELEMENT memo (to,from,title,message)>
  <!ELEMENT to       (#PCDATA)>
  <!ELEMENT from     (#PCDATA)>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT message    (#PCDATA)>
]>
<memo>
  <to>Alan</to>
  <from>Sid</from>
  <title>Meeting</title>
  <message>Don't forget the meeting today at 12</message>
</memo>
```

Listing 2 A simple DTD.

The DTD can also be stored in an external file, as shown in Listing 3, allowing many documents to use the same structure check.

```
<!ELEMENT memo (to,from,title,message)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT message (#PCDATA)>
```

Listing 3 DTD as an external file.

The actual XML document can be then stored in a separate file that refers to it (Listing 4).

```
<?xml version="1.0"?>
<!DOCTYPE memo SYSTEM "memo.dtd">
<memo>
<to>Alan</to>
<from>Sid</from>
<title>Meeting</title>
<message>Don't forget the meeting today at 12</message>
</memo>
```

Listing 4 The referring XML document.

In the above DTDs there are two different kinds of definition:

```
<!ELEMENT memo (to,from,title,message)>
```

or

```
<!ELEMENT message (#PCDATA)>
```

Table 1 DTD

Number of Occurrences	*Syntax*	*Example*
Only one	<!ELEMENT element-name (child-name)>	`<!ELEMENT memo (message)>`
One or more	<!ELEMENT element-name (child-name+)>	`<!ELEMENT memo (message+)>`
Zero or more	<!ELEMENT element-name (child-name*)>	`<!ELEMENT memo (message*)>`
Zero or one	<!ELEMENT element-name (child-name?)>	`<!ELEMENT memo (message?)>`
Either/or	<!ELEMENT element-name (child-name|child-name)>	`<!ELEMENT memo (note|message)>`

The first is an element followed by a list of children elements; the second is an element named message followed by parsable data. The other possibilities include the ability to say exactly how many elements should occur; for example, at least one, none or one occurrence, none or multiple occurrences, or one or more occurrences (Table 1).

In a similar way it is possible to declare attributes:

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

For example, in using DTD:

```
<!ATTLIST payment type CDATA "card">
```

a typical XML example could be:

```
<payment type="card" />
```

The attribute type can have other values, as shown in Table 2.
The specified default value can have several values, as specified in Table 3.
For example, to use `#REQUIRED`, the DTD would include:

```
<!ATTLIST product idcode CDATA #REQUIRED>
```

For the correct XML:

```
<product idcode="28177"/>
```

it would be an error to not include an ending tag:

```
<product />
```

Schemas

A schema performs the same function as a DTD; that is, it defines what is legal in an XML document. Schemas are largely replacing DTDs as they are extensible, richer and generally more useful. They became an official W3C recommendation in May 2001.

Table 2 DTD attribute types

Value	*Description*
CDATA	Character data
(enum1|enum2|...)	Must be from enumerated list
ID	Unique ID
IDREF	ID of another element
IDREFS	List of other IDs
NMTOKEN	XML name
NMTOKENS	List of XML names
ENTITY	Entity
ENTITIES	List of entities
NOTATION	Notation name
xml:	Predefined XML value

Table 3 Attribute required or default

Value	*Description*
value	Default value of an attribute
#REQUIRED	Attribute must be included
#IMPLIED	Doesn't have to be included
#FIXED value	This value is fixed

Going back to the memo example, the XML schema in Listing 5 is the equivalent of the DTD in Listing 2 for the XML memo document.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.myXMLnspace.com"
xmlns="http://www.myXMLnspace.com"
elementFormDefault="qualified">
<xs:element name="memo">
   <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="message" type="xs:string"/>
```

```
        </xs:sequence>
      </xs:complexType>
</xs:element>
</xs:schema>
```

Listing 5 Equivalent XML schema.

The schema in Listing 5 would be stored as `memo.xsd`. Most elements contained in this schema are simple types, as they do not contain other elements. However, the element `memo` is complex as it contains a lot of the simple types. To use this schema, a link is placed inside the original document as shown in Listing 6.

```
<?xml version="1.0"?>
<memo
xmlns="http://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.myXMLnspace.com memo.xsd">

<to>Alan</to>
<from>Sid</from>
<title>Reminder</title>
<message>Don't forget the meeting today at 12</message>
</memo>
```

Listing 6 Referring to a schema in an XML document.

The schema begins properly at its root element:

```
<xs:schema> ... </xs:schema>
```

As you can see the `<schema>` element can contain several attributes, the first part of which (`xmlns`) declares the namespace that the elements and data types (such as `schema`, `element`, `complexType`, `sequence`, `boolean` ...) used in this schema come from. All these are to be prefixed with `xs:`. The `targetNamespace` indicates where elements defined in this schema (such as `to`, `from`, `title` and `message`) come from.

Elements

Elements in XML schemas are defined in a similar way to the DTDs but with a different syntax. For example, you may have:

```
<firstname>Greg</firstname>
<lastname>Hurst</lastname>
<age>35</age>
```

which could have some schema element definitions:

```
<xs:element name="firstname" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="lastname" type="xs:string"/>
```

These are some simple element types:

1. xs:string
2. xs:decimal
3. xs:integer
4. xs:boolean
5. xs:date
6. xs:time.

Complex elements can be built that contain other elements and attributes. The following parts of a schema use a complex type named productinfo. This in itself contains a sequence of elements, hence the complex type. The productinfo complex type can then be used for elements:

```
<xs:element name="food" type="productinfo"/>
<xs:element name="magazines" type="productinfo"/>
<xs:element name="clothes" type="productinfo"/>
<xs:complexType name="productinfo">
  <xs:sequence>
    <xs:element name="item" type="xs:string"/>
    <xs:element name="itemcode" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Attributes

To define an attribute a similar style is adopted; for example for the XML element:

```
<firstname lang="English">Greg</firstname>
```

the schema definition could be:

```
<xs:attribute firstname="lang" type="xs:string"/>
```

The data types used are the same as those defined as above. It is possible to have an attribute assigned automatically should no other be given; in other words it is possible to give a default value:

```
<xs:attribute name="lang" type="xs:string" default="English"/>
```

If an attribute is to be attached that cannot be changed then a fixed value can be ascribed:

```
<xs:attribute name="lang" type="xs:string" fixed="English"/>
```

Attributes can also be made optional or required:

```
<xs:attribute name="lang" type="xs:string" use="optional"/>
<xs:attribute name="lang" type="xs:string" use="required"/>
```

This should give some idea of XML schemas. There is much more to learn in this area so you could try the W3C Web site for more information or w3schools.

9.6 Well Formed?

XML documents should be well formed and utilize rules set out in DTDs or schemas to define them. A parser can use these rule sets to work out whether a document is correct or not. Parsers can be stand alone software, part of a browser or an object in a programing language. JavaScript, for example, includes the ability to access parsers, such as Microsoft's XMLDOM, utilize them and report any errors generated. Some Web sites will let you paste XML into a text area and will check it for you, while others will verify a specified site is valid. For example, if you visit

```
http://www.w3schools.com/dom/dom_validate.asp
```

you can check whether XML is correct using both methods mentioned earlier.

Checkpoint!

- There has to be some way of verifying that an XML document is correct or well formed.
- There are two ways to do this, DTDs and schemas.
- Schemas are probably better and more commonly used.
- Both give a set of rules for the parser to verify the XML code.
- They are composed of descriptions of the order, sequence and description of the various elements, attributes and other parts of the document.

Concept Check

- Try and develop a DTD for your XML music collection document.
- Develop a schema for your XML music collection document.
- Find a browser that will parse and validate your document.
- Find out more about the JavaScript mechanism for accessing and using an XML parser.

9.7 Using XML with Applications

So, if you have managed to get your data into order with XML what can you do with it? Well, it's possible to look at XML with a browser and it will show it in a hierarchical form, which it has derived by scanning through it with its own parser.

It's also possible to utilize XML inside HTML. For example, XML can be stored as a separate file as we have seen. It can also be embedded within an HTML document using the unofficial `<xml>` tag. It is also possible to bind XML to HTML elements using *data islands*, although this does not work with all browsers.

Listing 7 shows an example of using a data island. First the XML file with the data island is loaded using the `<xml>` element then the `<table>` element is bound to the data island using the `datasrc` attribute. This code will output all the entries in that file that are in the fields `ARTIST` and `TITLE`.

Many applications now use XML as a storage or transfer mechanism. For example, you could easily make a database of books and store the data as XML. It is possible to use parsers to scan through the

```
<html>
<body>
<xml id="cdcat" src="cd_catalog.xml"></xml>
<table border="1" datasrc="#cdcat">
<tr>
<td><span datafld="ARTIST"></span></td>
<td><span datafld="TITLE"></span></td>
</tr>
</table>
</body>
</html>
```

Listing 7 Using data islands.

tags and hierarchy to extract the information your application requires. Java, JavaScript and the .NET programing languages among others all contain ready-built parsers for this, although is also possible to write your own!

Summary

1. XML is a language that can be used to describe other markup languages.
2. Its main purpose is to give meaning to data.
3. It can be used as both a store and a transfer mechanism for information.
4. It is readable by humans as it is simply held in plain text format.
5. The syntax is strict and case sensitive.
6. Conflicts between names in XML documents can be resolved with prefixes or namespaces.
7. XML can be verified with a parser that uses an associated DTD or schema.
8. DTDs and schemas consist of a set of rules that determine the correct order and type of included parts of the document.
9. XML can be used with many applications, which can use it to store or transfer data; for example, to store information on a music collection or library of books.
10. XML can be utilized in many programing and scripting languages that already contain components to manipulate such documents.

Key Words and Phrases

Data islands A way of including information stored in an XML file inside an HTML page.

DTD Document Type Definition, a document containing a set of rules that determine how an XML file should be structured. Can be inadequate for some XML documents, the alternative being the schema.

Enumerated An ordered sequence, or list, of information.

Schema Another way of providing a description for an XML document so it can be checked.

XML The Extensible Markup Language, a markup language that can be used to define other markup languages. Used to give meaning and a description of data rather than providing formatting as HTML would.