Piecing Together the XML Jigsaw

As XML is only just coming of age, it is a very interesting time to be learning about it. However, it can also be quite a challenging time to come to XML as the standards are being argued over and new proposals are being made. Hopefully we will guide you towards what can sometimes appear to be a moving target. In this chapter we will show you how the various pieces of the XML jigsaw fit together to create a working XML application.

We also give you a taste of some of the new proposals, so you know which areas are the ones to keep your eyes on. In this chapter we will discuss:

- Document Type Definitions (DTDs)
- The difference between well-formed and valid documents
- Style Sheets for adding formatting to your XML documents
- Parsers
- Linking in XML
- Displaying XML in a browser
- The emerging proposals for Namespaces
- The new emerging proposals for XML Schemas including XML-Data and DCDs
- Why you should be getting so excited about XML

Bits and Pieces

As we saw in the introduction, to use XML practically and to be able to view it over the Web we need a number of pieces that together make up the XML jigsaw. We will give you an overview of these pieces, so that when we come to look at them in detail in the following chapters you will know how they all fit together. To demonstrate them we will keep coming back to the CD Library example we met earlier. Here it is again to remind you, remember that the full XML file would contain many more CDs between the `<cdlib>` root tags marked up in exactly the same way:

```
<cdlib>

     <cd>
     <artist>Arnold Schwarzenegger</artist>
     <title>I'll Be Bach</title>
     <format>album</format>
     <description>Arnie plays Bach's Brandenburg Concertos 1-3 on the Hammond
     Organ</description>
     </cd>

     <cd>
          ...
     </cd>

</cdlib>
```

Document Type Definition (DTD)

So, we have seen that you can have your XML document nicely marked up with handy tags that you can understand and which provide information about their content. But we need to declare the rules for the language we have created somewhere. We referred to this as having to:

- Declare what constitutes markup
- Declare exactly what our markup means

Practically speaking, this means that we have to give details of each of the elements, their order and say what attributes they can take. The XML specification defines these rules using a DTD, or Document Type Definition. When a DTD is sent with an XML file the user agent can then expect a document that conforms to the DTD.

However, as XML keeps changing we need to be aware of the other proposals put in for alternative ways of providing this information, which all come under the banner of **XML Schemas**. We will cover DTDs first, as they are part of the original core XML 1.0 specification, before coming back to these other options, which are just at the proposal stage, later in this chapter.

The DTD can either be an external file or it can be declared internally within the **document type declaration**. If the DTD is in an external file we link it to our document in the following way:

```
<!DOCTYPE cdlib SYSTEM "cdlib.dtd">
```

HTML has a DTD, but you don't get to see it every day because it is contained within most popular Web browsers. You can view it at `http://www.w3.org/TR/REC-html40/loose.dtd`. According to the HTML standard, when authoring HTML documents you are supposed to include the following line of code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 //EN">
```

It tells the user agent the location of HTML's DTD, however it is often left out because, practically speaking, it is not necessary and if you are using browser specific tags which deviate from the specification it may cause unpredictable results.

Creating your own markup language using a DTD need not be excessively complicated. Here is the internal DTD for the CD Library example. As you can see, it is a very simple DTD. We go into more detail in the next chapter, however this is enough to make our library example functional.

```
<!DOCTYPE cdlib [
<!ELEMENT cdlib (cd+)

<!ELEMENT cd (artist+, title+, format?, description?)>
<!ELEMENT artist (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT format (#PCDATA)>
<!ELEMENT description(#PCDATA)>

]>
```

Briefly, the `<!DOCTYPE cdlib [` line is used to distinguish it from other DOC types. It also gives the same name as the root element of the document. While `<!ELEMENT>` is used to declare elements in the format:

```
<!ELEMENT name (contents)>
```

where `name` gives the name of the element, and `contents` describes what type of data can be included and which elements are allowed to be nested inside that element. The element `<cd>` has to include the elements `<artist>` and `<title>` (they are forced to be included by the use of the `+` symbol which means one-or-more), while the `<format>` and `<description>` elements are optional (denoted by the `?` symbol).

The elements included in `<cd>` are then defined. These can take almost any ordinary text without markup (we explain the exceptions in the next chapter).

> **It's very easy to get confused between Document Type Definitions and Document Type Declarations... To clarify, just remember that a document type declaration either refers to an external document type definition, or else it contains one in the form of markup declarations as in the example we have just seen.**

As XML grows it is likely that there will be an ever increasing number of popular DTDs that we will be able to download and use, without always having to go to the trouble of writing our own. It is widely expected that industry standards for marking up certain types of document will soon appear. You can see this happening already with examples like Channel Definition Format (CDF) and Chemical Markup Language (CML) which we look at later in this chapter. But even if you are not planning to write your own DTDs an understanding of how they work will still be very important when writing XML applications.

Well-formed and Valid Documents

The XML specification defines two types of document, **well-formed** and **valid** ones. To be well-formed a document must conform to the following three rules:

- The document must contain at least one element
- The document must contain a root element, that is a unique opening and closing tag that surrounds the whole document
- All other elements contained within the document must be nested with no overlap between elements

So, looking again at our example of a record library, the following is a well-formed XML document:

```
<cdlib>
     <cd>
          <artist>Arnold Schwarzenegger</artist>
          <title >I'll Be Bach</title>
          <format>album</format>
          <description>Arnie plays Bach's Brandenburg Concertos 1-3 on the
          Hammond Organ</description>
     </cd>
</cdlib>
```

It contains more than the one required element. It has a root element in the form of the `<cdlib>` element – this can be compared to the opening `<HTML>` and closing `</HTML>` tags in HTML documents. And its sub-elements, or child elements, nest without any overlap. So it meets the criteria for being well-formed.

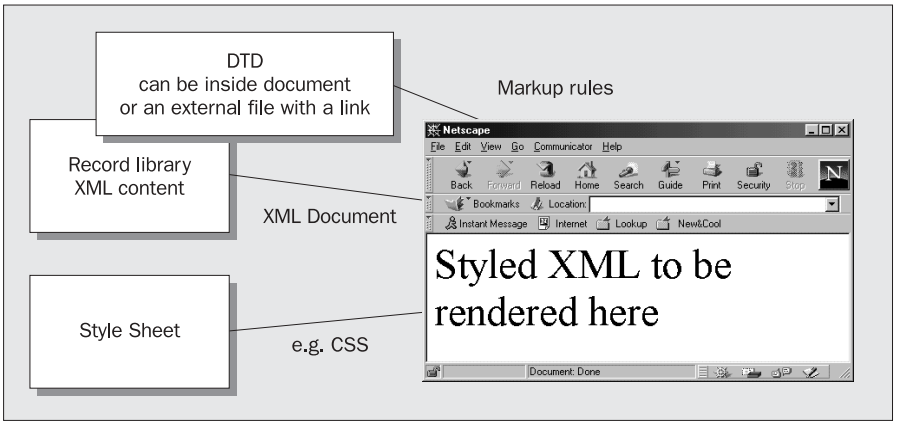
Valid documents on the other hand should not only be well-formed, they should also have a Document Type Definition which the well-formed document conforms to. This means that it must only use elements that have been declared in the order specified, and take the allowed types of content defined in the DTD.

The concept of a valid document is borrowed from SGML, however documents in SGML *must* be valid – there is no concept of SGML documents just being well-formed. We go into the area of well-formed and valid documents in detail in Chapter 2. However, simply put, XML was designed for the Web, and as long as the well-formed document can be meaningfully used - whether displayed on the browser or used by some other user agent - there is no need to send a DTD, it's just extra traffic. It may sound strange, but it is possible to make use of an XML document even without its DTD in certain instances, despite the user agent not knowing exactly what the tags you created mean.

Although XML is far simpler than SGML it is actually stricter, which is why you don't *need* a DTD with an XML file. This is the case because XML's strictness allows an XML processor to infer what rules apply from a well-formed document. It does this by constructing a tree of all the nested elements, and establishing the relationships between all of the various parts. As SGML doesn't require closing tags it would be impossible to do this with an SGML document if there wasn't a DTD. But, as long as the XML document can be used/displayed with functionality it is not always necessary to use the extra bandwidth by sending a DTD.

This doesn't mean that you should skip the DTD section of Chapter 2 too quickly, it's still advisable for a set of tags created in XML to conform to a DTD. This is because, if several people are creating or using documents that need to be compatible, the DTD sets out the rules that have to be obeyed. This helps maintain the structure and feel of a multi-author project. DTDs also allow you to use a piece of software called a validating parser to make sure that they are not violating the rules of the DTD. And if that means less work for you, and the other authors, then all the better.

Having seen the DTD, the following diagram shows how it can link to the XML document, and how the document can then be formatted for a browser when linked to a style sheet. It is to style sheets that we will turn next.



Remember that, at the time of writing, the main browsers had very little support for XML. Although it is likely that both Netscape Communicator 5 and Microsoft IE 5 will feature a high degree of support for XML, there are ways we can display XML in non-compliant browsers - which we come back to after our discussion of style sheets.

Style Sheets

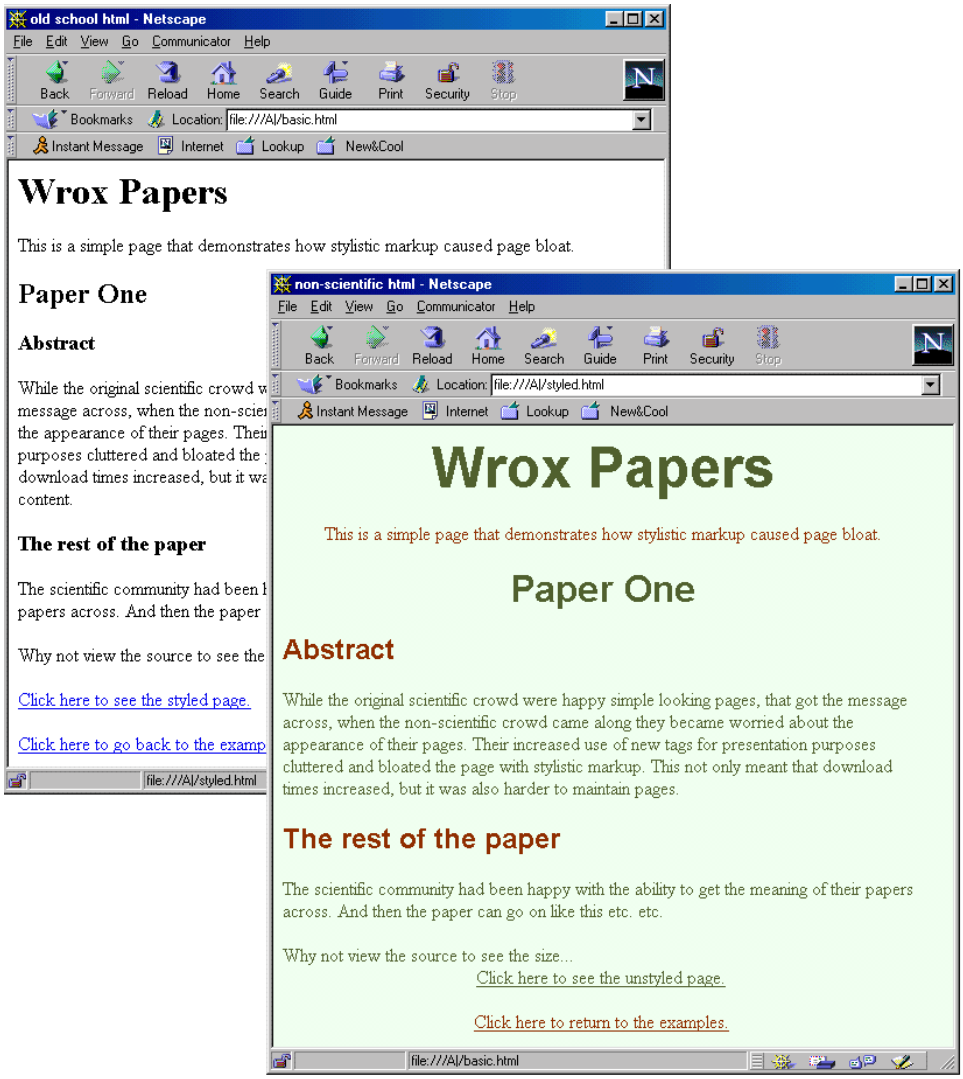
Unlike most of the things we have talked about so far, style sheets are by no means a recent invention. They are as old as printing. Style sheets are made up of rules that declare how a document should be displayed. Even in the days of manual typesetting, book publishers would have a written set of instructions that told the printer how their house style was to be represented. The printer would then use this to mark up the publisher's manuscript. As our XML documents do not contain details of how the contents are to be displayed (maintaining the distinction between content and presentation), we use style sheets when displaying our documents to a Web browser so that we can present them in an attractive or practical manner.

Advantages of Using Style Sheets

As we have seen, style sheets are needed when we want to specify how our XML documents are to be presented. Apart from being a necessary addition, they offer a number of general advantages:

- Improve the clarity of documents
- Can help reduce download time, network traffic and server load
- Allow you to present the same source in different ways for different purposes
- Allow you to change the presentation of several files by just altering the one file that contains the rules of how it is to be displayed

As we saw in the introduction, when the non-scientific crowd began to create Web pages, their concern over the appearance of their pages led to the creation of many new and heavily used stylistic tags and attributes. As a result the size of files soared - known as **page bloat** - which made it harder to read the documents' code, thus making maintenance of pages more difficult. Take the following pages for example:



The use of stylistic markup in the second page increases the file size and decreases legibility of code. You can view the full pages and their source on our Web site at:

`http://webdev.wrox.co.uk/books/1525`

But here is the code so that you can see the difference in file size:

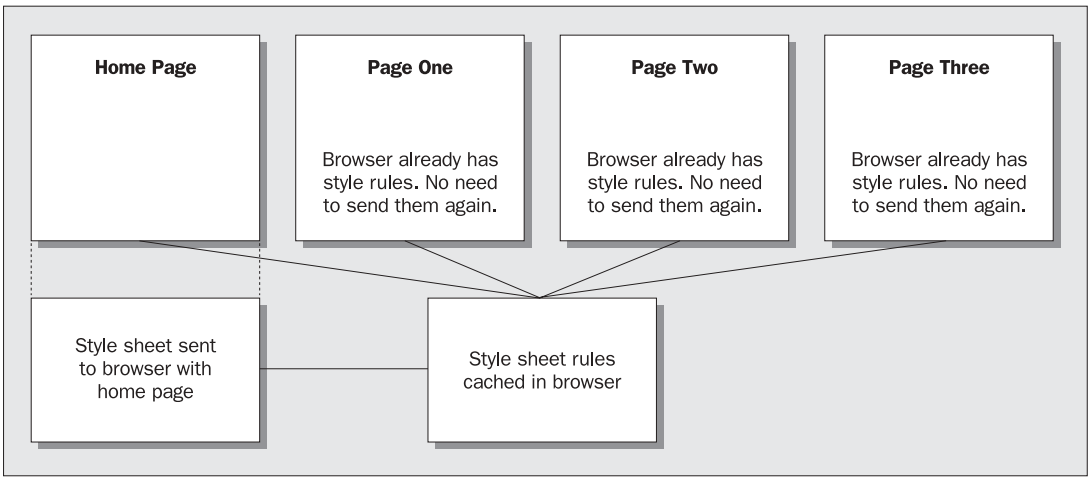
```
<HTML>
<HEAD>
<TITLE>old school html :/TITLE>
</HEAD>

<BODY>



<H1>Wrox Papers</H1>


This is a simple page :hat
demonstrates how styli stic
markup caused page blo at.



<H2>Paper One</H2>


<H3>Abstract</H3>


While the original sci entific
crowd...
...see their content.

<H3>The rest of the pa per</H3>



The scientific communi :y had
been
...
```

```
<HTML>
<HEAD>
<TITLE>non-scientific html< TITLE>
</HEAD>

<BODY BGCOLOR="#F5FFFA"
ALINK="#F4A460" LINK="#9932 0"
VLINK="#556B2F" TEXT="#556B F">

<H1 ALIGN=CENTER><FONT FACE ARIAL
SIZE=7>Wrox Papers</FONT></ 1>

<DIV ALIGN=CENTER><FONT
COLOR="#993200">This is a s mple
page that demonstrates how  tylistic
markup caused page
bloat.</FONT></DIV>

<H2 ALIGN=CENTER><FONT FACE ARIAL
SIZE=6>Paper One</FONT></H2

<H3><FONT FACE=ARIAL SIZE=5
COLOR="#993200">Abstract</F NT></H3>

While the original scientif c
crowd...
...see their content.

<H3><FONT FACE=ARIAL SIZE=5
COLOR="#993200">The rest of the
paper</FONT></H3>

The scientific community ha  been
...
```

Any Web site or company intranet that spans several pages, while maintaining a consistent appearance, requires the same style rules to be repeatedly sent to the browser with each page. Because browsers cache the data they receive from Web pages, this repeated sending of the same rules is a waste of bandwidth, download time and server load. Style sheets put all the style rules in one separate document so that they only need to be sent once, then each of the site's pages can link to the same style sheet which is stored in the browser's cache – how net-environmentally friendly.

In addition, because all the style rules are kept in one file, it is then possible to change the appearance of the whole site by just altering the style sheet rather than the individual style rules on each page.

If you use the same data in several pages, each of which present the data in different ways, all you have to change is the line which links to different style sheets. An example of where this may be useful could be where your site has different sections, but has common information that has to be presented in a style relevant to that section. Alternatively, if you wanted to provide a large text version for people with sight difficulties, you could offer the same page styled with bigger fonts for those who would have trouble reading the normal size font.



The content pages contain a reference to the style sheet, so that the user agent knows where to get it from - rather like the way links to an image are used. There are several style sheet languages including:

- CSS Cascading Style Sheets
- XSL Extensible Stylesheet Language
- DSSSL Document Style Semantics and Specification Language
- XS also known as DSSSL-0

The two that really concern us here are CSS and XSL. DSSSL was the official styling language for SGML and is extremely powerful, however it is also extremely complicated and has not been widely adopted. XS or DSSSL-0 was intended to be a simplified form of DSSSL, although it was still seen as too complicated to ever become a success on the Internet. So let's take a quick look at the other two.

Cascading Style Sheets

The Cascading Style Sheets Level 1 specification was released by the W3C in late 1996. It was supported to a degree in both Communicator 4 and IE4, although both of the main browser manufacturers have pledged to fully support CSS1 in their forthcoming Communicator 5 and IE5 browsers.

> *CSS Level 1 will eventually be superceded by CSS Level 2 (the recommendation was released May 1998). However, it may be some time before the browser manufacturers fully support CSS Level 2 seeing how they haven't fully implemented Level 1 yet.*

Cascading Style Sheets are already finding their way onto the Web and got their name because, when several style sheets are present, a cascade is formed with properties being taken from all of the sheets, any conflicts being resolved according to a set of rules. CSS are simple to construct, and once they have been created there is no limit to the number of pages that can use them. HTML documents just need one simple line of code to link to a CSS:

```
<LINK REL="stylesheet" TYPE="text/css" HREF="example.css">
```

The W3C are currently addressing the way in which XML documents will link to CSS. There is a note on the subject available from:

`http://www.w3.org/TR/NOTE-xml-stylesheet`

Although one popular way to do this at the moment is:

```
<?xml-stylesheet href="cdlib.css" type="text/css"?>
```

Here is an example of a cascading style sheet that could be used with the CD library we have been looking at:

```
artist {
      display: block;
      font-family: Arial, Helvetica;
      font-weight:bold;
      font-size: 20pt;
      color: #9370db;
      text-align: center;
      }

title {
      display: block;
      font-family: Arial, Helvetica;
      font-size: 20pt;
      color: #c71585;
      }

format {
      display: block;
      font-family: Arial, Helvetica;
      font-size: 16pt;
      color: #9370db;
      }

description {
      display: block;
      font-family: Arial, Helvetica;
      font-style: italic;
      font-size: 16pt;
      color: #FF1010;
      }
```

When eventually viewed through a browser our CD Library with this style sheet would present a list of CDs as shown in the following screen shot. This was taken using a beta version of IE5, if you have a copy of IE5 you can try it from our Web site at: `http://webdev.wrox.co.uk/books/1525`

There are many advantages to using CSS, and such style sheets are discussed further in Chapter 7. However, CSS uses a fixed set of markup, you cannot create your own new tags – it is not extensible. While CSS can be used for presentation of XML documents, there is another more powerful option – XSL.

Extensible Stylesheet Language

While many XML authors will be content with the functionality of Cascading Style Sheets to display simply structured XML documents, XSL offers all the same advantages that CSS offers with additional functionality, and is likely to be used where more powerful formatting capabilities are required.

XSL draws on both DSSSL and CSS, (and DSSSL-0). It also uses ECMAScript, which was derived from JavaScript.

XSL is made up of two parts:

- An XML vocabulary for specifying formatting semantics
- A language for transforming XML documents

XSL not only allows the user to specify how parts of a document should look (font, size, color, alignment, borders etc.) just like CSS, but is also extensible, so it allows users to create their own new formatting tags and properties.

XSL also offers users further control over the presentation of their documents. For example, it can add rules that order the presentation of sections. If you go back to the CD library example, these rules would allow you to specify different orders for the presentation of the data, so you could have either `<artist>` first, or `<title>`. The order of the data doesn't matter, because the tags can be interpreted and re-ordered.

At the time of writing XSL was at the First Draft stage of the W3C process. It was originally a joint proposal from Microsoft, Inso Corporation and ArborText, however the Working Draft 1.0 supercedes the proposal. To keep up to date with the development of XSL, keep an eye on the W3C site's XSL page, located at

`http://www.w3.org/Style/XSL/`

CSS vs. XSL

It is likely that both CSS and XSL will be present on the Web for the foreseeable future because they address different needs. XSL allows the author control over complex formatting where the contents of a document might be displayed in multiple places; for example the singer of a track in our CD library example might also appear in a dynamically generated table displaying their back catalog, or used as a page header. Meanwhile, CSS is intended for dynamic formatting of online documents for multiple media not just visual browsers, but also audio, braille, hand-held devices etc.

	CSS	XSL
Can be used with HTML	Yes	No
Can be used with XML	Yes	Yes
Is a transformation language	No	Yes
Syntax	CSS	XML

For more information on Style Sheets, see Professional Style Sheets for HTML and XML from Wrox Press (ISBN 1-861001-65-7).

Parsers

I don't want to get embroiled in philosophical rambling here. But, when we have stored our XML in a plain text format that we consider to be human-readable, we can't really say that computers can 'read' the file - they interpret it. It is the job of a parser to help the computer interpret the XML file by putting it in a format they can use. XML applications in turn help both us and parsers interpret the file because of the information they contain on the nature of the text chunks within the tags.

The XML specification refers to two components for practical use of XML: the **XML processor** and the **application**. Parsers fulfil the role of the XML processor, they load the XML and related files, check that the XML is either well-formed or valid (depending on the type of parser), and build a document tree structure that can be passed on to the application. It is this tree structure which can then be practically used by the computer. The application is the part which processes the data that is in the document tree created by the processor.

In practice the parser is generally just a component for programmers to call upon when building their application. As such IE5 integrates an XML parser called MSXML.

Remember that we said XML documents do not always need a DTD? It is the ability of parsers to form a document tree without a DTD that allows XML files to be functional on their own.

HTML browsers combine the role of the processor and the application. Because HTML is the set of rules for how the document is to be displayed, there are strict rules that say how the files can be interpreted. However, by separating the DTD and the style sheet from the XML file, applications can use the same document for different purposes. So the type of application is not just restricted to being a browser, it could be a word-processor, braille printer etc. It may not even be an application that is used directly by humans; it may be an automated tool, for example a system which creates letters when certain events happen – such as when you go over your overdraft limit. It could even add a charge for the letter to your account.

While this is a central use for parsers, they also perform another very useful function. We have already seen that, because XML is a far stricter language than SGML, parsers can be used as a tool to help XML authors check their documents, and to help XML editing software to make sure that they produce compliant documents. While it may seem that authoring XML is a lot more demanding than writing HTML, XML's advantages in processing ability and flexibility outweigh the disadvantages of its rigidity – closing tags and carefully watching syntax. When you have just finished writing a long and complicated XML document, only to find that it doesn't work, the task of going through the code to find out where the error is can seem daunting. So before you actually use an XML document it is wise to run it through a parser, which checks the document for you and points out mistakes – such as the absence of a closing tag.

> *Although, as we have seen declining numbers of people writing HTML "by hand" with just Notepad, turning instead to authoring programs, so too can we expect a number of XML editors to appear in the near future.*

Primarily there are two types of parser. The simpler **non-validating** parsers just check for well-formedness and can be as small as 30-40 Kb, while the more complicated **validating** parsers also check for validity using the DTD.

There are several parsers freely available across the Internet, and you are strongly advised to download at least one. If you can, download more than one, as different parsers handle the reporting of errors in different ways. Let's take a quick look at some of the parsers that are available.

> *An added advantage of making sure that the document accurately follows the specification is that there is no need to put lots of code into the user agent so that they can handle poorly written documents. The current major Web browsers are notorious for including code that allows them to display HTML that is not directly compliant with the HTML specification.*

Lark by Tim Bray

Lark, written by Tim Bray (one of the editors of the XML specification), was one of the best of the early tools for checking your work. It is a non-validating XML processor written in the Java programming language - to use it you need a Java Virtual Machine.

> *You get a Java Virtual Machine in Microsoft Visual J++ for Windows, the Macintosh Runtime for Java SDK, OS/2 Warp version 4, or you can download the Sun Java Development Kit (JDK) or the lighter Java Runtime Environment (JRE) from*
> `http://java.sun.com/`

Although Lark does not have a visual user interface it is compact and does a reliable job. It efficiently builds document trees and matches tags, however it doesn't check that the document is valid – by comparing the document to its DTD. Tim has also written a validating XML parser, based on the same code as Lark, it is called Larval. You can download a free copy of both from:

`http://www.textuality.com/Lark/`

XP by James Clark

XP is a non-validating XML 1.0 parser written in Java. It can check for a document's well-formedness. XP can be downloaded from:

`ftp://ftp.jclark.com/pub/xml/xp.zip`

Microsoft XML Parser in Java

Microsoft's XML parser written in Java is a parser which checks for well-formedness of documents and optionally permits checking of the documents' validity. Once parsed, the XML document is exposed as a tree through a simple set of Java methods, which Microsoft are working with the World Wide Web Consortium (W3C) to standardize. If you use this parser it is worth keeping up to date with its development as Microsoft are constantly updating it. For more details and to download a version visit:

`http://www.microsoft.com/xml/`

There are, of course, many more parsers available on the Internet and we haven't enough space here to cover them all. To find out more why not have a look on your favorite search engine?

Linking in XML

With all the new things you'll have to learn, you'll be glad to know that the type of linking you learnt in HTML is still effective in XML. You can still use a link such as:

```
<a href= "http://webdev.wrox.co.uk">Wrox Web Developer</a>
```

Of course, because we're writing XML now, the `<a>` element has to be declared in the DTD, as does the attribute `href`, even though `href` and `HREF` are reserved keywords in the linking specification. However, as we're using XML we might as well use a more descriptive tag, such as:

```
<webdevlink href="http://webdev.wrox.co.uk">Wrox Web Developer</webdevlink>
```

This would then be declared in the DTD like so:

```
<!ELEMENT webdevlink (#PCDATA)>
<!ATTLIST webdevlink
      xml:link CDATA #FIXED "simple"
      href CDATA #REQUIRED
>
```

`xml:link` is a reserved XML keyword used as an attribute to define links; it can take the value `"simple"` or `"extended"`. Because the `xml:link` attribute's value is `#FIXED`, `xml:link` does not need to be included in each instance of the element; it is implied in each `<webdevlink>` tag, which is why we can use:

```
<webdevlink href="http://webdev.wrox.co.uk">Wrox Web Developer</webdevlink>
```

instead of having to put:

```
<webdevlink xml:link="simple" href="http://webdev.wrox.co.uk">Wrox Web
Developer</webdevlink>
```

The proposals for linking in XML reached working draft status in March 1998, and as you may have already worked out, they are expressed in XML. Originally known as XLink, then XML-Link, **Extensible Linking Language (XLL)** is the current term for the linking languages under development. XLL is based upon on SGML, HyTime and the Text Encoding Initiative (TEI) – the latter two are used as linking methods in SGML.

If you are happy with plain links, like those in HTML, there will be little more to learn on this topic. However, the linking capabilities in XML are far more powerful. Beware, you will probably find that you will see lots of uses for the new types of link and will want to learn all about it. Using XLL you can:

- Create your own link elements
- Use any element as a linking element
- Create bi-directional links with one-to-many and many-to-one relationships
- Specify traversal behavior – how users get between links
- Create link databases to specify and manage links outside of the documents to which they apply
- Aggregate links
- Transclusion – the link target appears to be part of the link's source document

There are two basic link types:

- **Inline links** that are specified at the point where the link is initiated
- **Out-of-line links** that are stored in an intermediary file – a link database

Linking is not covered in the XML specification. Instead there are two separate specifications for linking in XML:

- **XLinks** for linking separate XML documents to other XML documents
- **XPointers** are to be used for addressing the internal structures of XML documents providing specific references to elements, character strings, and other parts of XML documents, even those without an explicit ID attribute

Any element can be an XLink, and elements including links are referred to as **linking elements**. Because the linking specifications are written in XML, you can create your own tags for links that describe the link as shown in the `webdevlink` example. The link then describes the relationship between the objects or parts of data objects.

The specification for XLinks is available from:

```
http://www.w3.org/TR/1998/WD-xlink-19980303
```

XPointers are used in conjunction with URIs to specify a part of a document (or a sub-resource). They can be used to link to a specific part in the whole of a document, or create a link that just takes part of the document (as opposed to all of it). Any link that addresses part of a document must be in the form of an XPointer. However, it is not necessary to include explicit ID attributes in the XPointer language in the same way that they are needed in HTML and they can provide for specific reference to elements, character strings, and other parts of XML documents.

The XPointers specification is available from:

```
http://www.w3.org/TR/1998/WD-xptr-19980303
```

Using HTML, if you wanted to link to a specific part of a document the document itself has to be changed using named anchors. This is not necessary using XLL with XPointers. XLL will also allow custom applications – without a human user being present – to establish connections between documents, and parts of documents.

XML Comments

Comments in XML are the same as they are in HTML. They are placed inside these tags:

```
<!-- comment -->
```

Good use of comments is almost an art form. You develop your ability to use them as you develop as a programmer. They are absolutely essential – when you go back to a document after a couple of months, what seemed so obvious then suddenly is not so clear. Good use of comments also means that others can use your documents with greater ease. This is particularly important when creating DTDs because you may want others to use your DTD. The correct placing of comments in XML documents is covered in Chapter 2.

The New Kids on the Block

By now, you'll have got the idea that XML and the other related specifications are far from being stable and finished. In this section we introduce some of the new specifications that are creating a stir in the XML community. This doesn't mean that any of the things they claim to be improvements on will disappear, far from it. Once you have built a solid understanding of building XML applications with this book, you will be in a strong position to watch for the development of these specifications. We shall be looking at proposals for:

- XML Namespaces
- XML-Data
- Document Content Declarations

XML Namespaces

One of the advantages of XML that we mentioned in the introduction was that it is possible to create compound documents from several separate sources. To save us the work it is likely that software modules will be used to create these compound documents. There are, however, two problems these software modules must overcome in order to create compound documents:

- Recognizing the markup they are to process (tags and attributes)
- Coping with name 'collisions' in markup

The first problem is self-explanatory; the tools have to know what part of the document they are addressing. The second problem is less obvious. While the ability to use documents from multiple sources can be very useful, there is the risk that, with everyone creating their own tags, there will be a collision of names used for tags or attributes. If some people are merrily creating their own tags while others are following a standard, it will not be long before two files use the same tagname, for example, to describe different things. In our CD Library example we used the element `<format>` to describe whether the CD was an album or a single. It could be equally possible that another similar document uses a `<format>` tag to describe whether the recording was on CD, vinyl, cassette, DVD, MiniDisc, etc. The **XML namespaces** proposal was designed to counter these two problems.

To get over these hurdles the document constructs must have globally unique names. One way of doing this is by defining a unique namespace for element types and element attributes.

Here is the W3C definition of XML namespaces:

> 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.

Namespaces associate a prefix with a URI using the following syntax:

```
xmlns:[prefix]= "[URI of namespace]"
```

So we could use:

```
xmlns:wroxcds="http://webdev.wrox.co.uk/books/1525"
```

to declare the `wroxcds` prefix, with elements from that domain uniquely identifying those tags within the compound document.

Now this may all sound like a lot of extra work, so let me put your mind at rest. Without going into too much detail at this early stage, once the namespace has been defined earlier in the document instance, the `format` element doesn't become much more complicated and will end up looking like this:

```
<wroxcds:format>album</wroxcds:format>
```

As you may have guessed, the namespaces proposal is still subject to change in the W3C process.

The XML Namespaces Working Draft came out on August 2nd 1998. We come back to the topic of namespaces in Chapter 4, you can also keep an eye out at:

`http://w3.org/TR/1998/WD-xml-names`

XML Schemas

Schemas define the characteristics of classes of objects. So in XML, a schema is simply a definition of the way that the document is marked up. The DTD is actually a good example of a schema in that the DTD defines a class of documents and the element and attribute types in that class.

At the time of writing the following **XML Schema** proposals were still at the Note stage of the W3C proceedings, although they were generating a lot of interest in XML circles:

- XML-Data
- Document Content Description for XML

An XML Schema Work Group has since been formed by the W3C, whose job will be to look at and review the various XML Schema proposals, though it is too early to tell what they will decide to do with them. Their first step is to create a requirements document which will be used to drive the following discussion. So it is unlikely that these schemas will stay in their present form. However, we have covered them in this book to introduce the concept of schemas written using XML syntax, which as you will see have several advantages over traditional DTDs.

As this suggests, XML Schemas have arisen because of weaknesses in the traditional DTD that we saw earlier (and come back to in the next chapter). The criticisms of traditional DTDs circle around the following characteristics:

- It uses a different syntax to XML
- It is difficult to write good DTDs
- It is limited in its descriptive powers
- They are not extensible
- They do not describe XML as data well
- There is no support for the Namespaces proposal

We come back to discussion of this subject in Chapter 3, for the moment let's just take a quick look at what XML-Data and DCDs are.

XML-Data

XML-Data was a proposal submitted to the W3C on 11th December 1997 as a way of describing schemas using XML syntax. As you'll see in the next chapter DTDs use a modified form of a notation called **Extended Backus-Naur Form** notation (**EBNF**), and simplicity is *not* one of its strong points. So the idea of declaring schemas in XML (as XML-Data does) has been gratefully received.

When we have relatively simple documents, the DTD may seem like a perfectly adequate way of representing schemas. However, there is also a move on the Web to see XML as data, not just a way of marking up documents, and when doing so, we benefit from the ability to relate different types of schemas to our data.

`http://www.w3.org/TR/xml-data`

or the Microsoft site at:

`http://www.microsoft.com/xml`

Document Content Description for XML – DCD

This schema for XML documents is even more recent and was only submitted in July 1998. DCD also uses XML syntax. The basic syntax is introduced in Chapter 3, along with references of where you can find out more information on XML Schemas. It uses a modified Resource Description Framework (RDF) syntax for its description and includes a subset of the XML-Data submission in an RDF compliant way.

You can find the note at:

`http://www.w3.org/TR/NOTE-dcd`

Learning XML-Data and DCD syntax will provide a solid basis for understanding alternative XML Schemas written in XML syntax. So, you will be equipped to watch out for their progression if you so desire.

Displaying XML

This is one area of XML development where newcomers often get confused, so we're going to spend a little time here making your options clear. Hopefully, you will already be excited about the possibilities that XML can offer you, but you still need to know how XML files can be displayed over the Web. In looking at style sheets we have already covered half of the ground, so you know a little about how style rules are applied to the XML file – this section focuses on displaying the XML file in a browser.

Viewing HTML files over the Web is something we have all got very used to, it seems so simple. All you need to do is type in a URL and press *enter*, or click on a hyperlink and up comes your page. So that we can understand the process of outputting XML, let's just have a quick look at how it's done in HTML.

Outputting HTML

This is a simplified description of how the browser gets and displays the page, however it should suffice for our purposes:

- An HTML document is requested and returned to the client by HTTP (or the file system if it is not requested over the Internet).
- The browser strips the document of its tags and makes an array of the element's contents.
- The browser must work out what each tag means, look for styling associated with it – remember that the browser already has a knowledge of HTML (its DTD).
- It must then display the contents on the screen.

XML Browsers

Unfortunately things are not so simple with XML. At the time of writing the major browser manufacturers' products (up to version 4) did not allow direct viewing of XML in the same way you could view HTML – you couldn't just open an XML file in your browser and view the contents as you intended. Netscape and Microsoft have promised XML support in their Communicator 5 and IE5 browsers, but the level of support is still a little unclear – we discuss this in more detail in the later chapters. (Early beta versions of IE5 show strong support for the core XML 1.0 Specification, the Document Object Model, Namespaces, CSS and XSL.) In the mean time, don't panic, this does not mean that the `.xml` and `.xsl/.css` files you create cannot be viewed in Communicator 4 and IE4. What it means is that they have to be converted into HTML to be viewed - either statically (before the document is viewed), or dynamically (as it is being requested). But, if browsers could display XML a similar process would need to be undertaken.

- The browser would have to get the XML file, strip it of its tags and make an array of the elements' content.
- Because the tags in XML are not predefined, the browser would then have to look for information on how to style each element in a style sheet.
- To display the document, the HTML browser has a display window. However, an XML browser may be able to convert the document into a different format for display in another application – such as a rich text format displayed in a word processor – as well or instead.

Because an XML browser would not have any built-in knowledge of how the XML document is to be displayed, the styling process is – as we have seen – absolutely necessary. For now let's focus on how XML can be converted to HTML for viewing in a normal Web browser.

Static Conversion

Static conversion takes place before the file is put on the Web server. It involves using a program that takes the XML file and style sheet, then marries them to create an HTML file that can be viewed through the browser. It is the HTML file which is then put on the server for viewers to request.

One example of this type of tool is the Microsoft MSXSL Command Line Utility, this marries XML and XSL files, creating an HTML file. Unfortunately this only works with Windows 95/98 and Windows NT (x86 only) and IE4. It can be downloaded from:

```
http://www.microsoft.com/xml/xsl/downloads/msxsl.asp
```

Briefly here's what it does. From the command line you type:

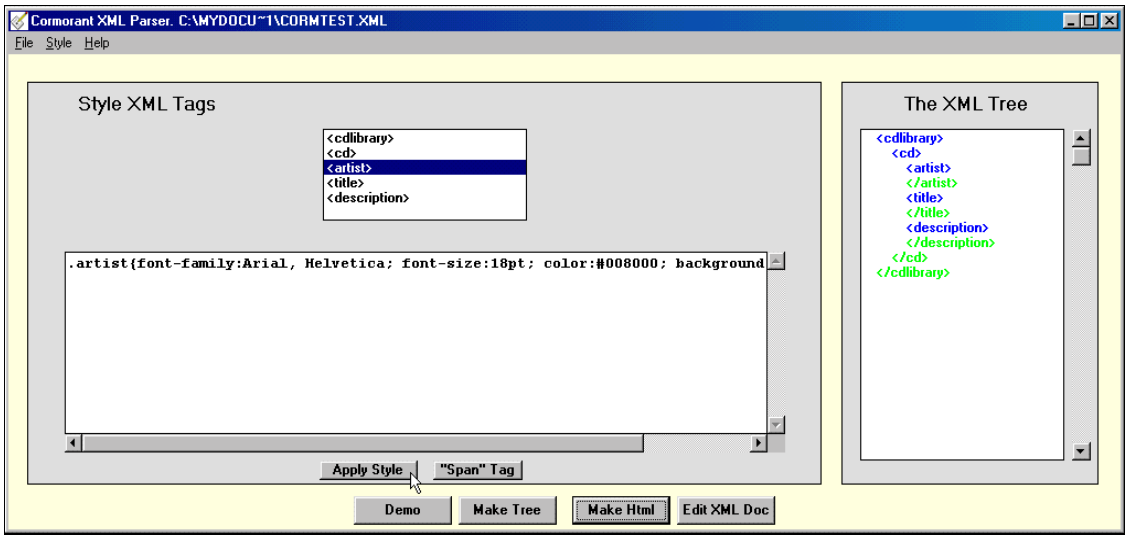
```
C:\xslproc\msxsl –i xmlfile.xml –s xslfile.xsl –o result.htm
```

For those blissfully unaware of what the command-line is (and sometimes ignorance is bliss!), in Windows 9x go to Start | Run and type in exactly the above, substituting the correct file names and the directory the files are in. You have to make sure that the `.xml` and `.xsl` files are in the same folder as the processor for this to work.

The `-i` is the input XML file, `-s` is the XSL style sheet and `–o` is the output in HTML. The resulting HTML file can then be put on the Web server or opened in an HTML browser and the style rules will have been applied.

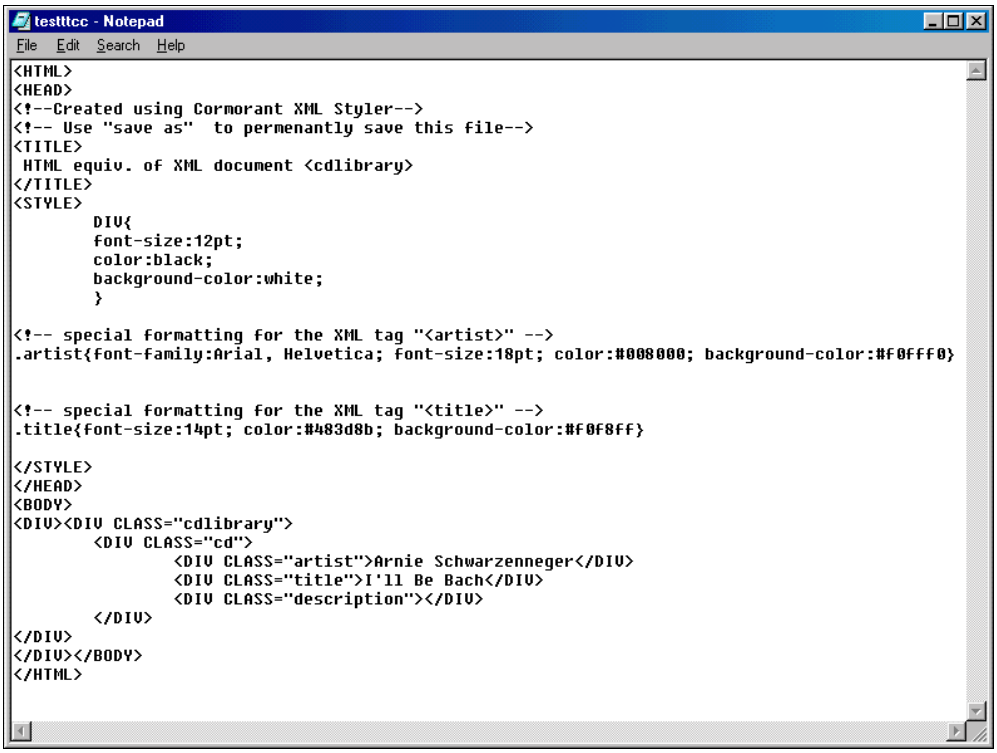
A less common approach is to run a special program, load in your XML file, then manually apply style rules to it. An example of this is the Cormorant XML Parser, written by one of our authors – Frank 'Boomer' Boumphrey. It is freely available from our Web site at:

```
http://webdev.wrox.co.uk/books/1525
```

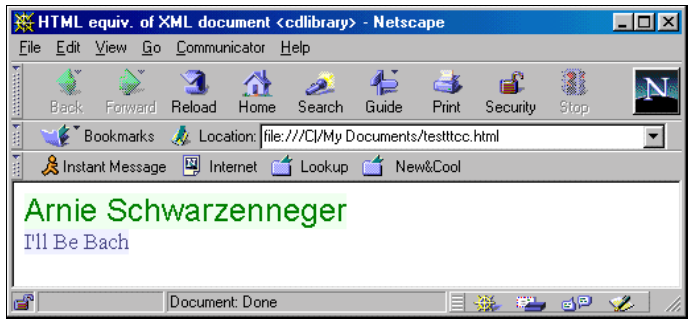


After opening an XML file (shown in the right-hand pane – The XML Tree), style rules can be applied to each tag in the left-hand pane, by clicking on the appropriate tag and typing in the style rules between the empty brackets. Then you click on the Apply Style button to make your amendments. Finally clicking the Make HTML button opens Notepad with the HTML file in it.

Saving this as an `.html` file allows you to view the original XML in an HTML browser.



And here we see the displayed HTML document:



While the static approach is fine if you just want to prepare static XML documents for the Web, it is not very helpful if you want to harness the full power of XML. Creating dynamic pages, that use the data in your XML files, serving different files to different users requires a dynamic conversion technique.

Dynamic Conversion

The dynamic conversion of XML files allows you to serve different pages to different users. There are a number of dynamic conversion techniques available that can use:

- MSXSL Command Line Utility
- ActiveX Control
- Java Applet
- JavaScript program

to convert the `.xml` and `.css`/`.xsl` files to HTML on the fly. Again, these options are freely available. Rather than cover them all here, we will just give you an idea how one of these options works - we will take a quick look at the MSXSL ActiveX Control.

MSXSL ActiveX Control

The MSXSL ActiveX Control is based on the same code as the MSXSL command-line utility that we have already seen. Again, it carries the disadvantage that it can only be used with Windows 9x/NT and IE4+. However, the advantage is that you don't have to know a lot about Java or JavaScript to get it to work.

Simply put, the ActiveX Control can be embedded into a basic HTML page. The ActiveX Control then downloads the XML file and its style sheet and converts them into HTML. Some simple JavaScript holds the result of the parsing in the HTML page. So the HTML page is really just a holder for the ActiveX Control and a wrapper for the returned code. The same ActiveX Control can be used on the Web server.

You can download the MSXSL ActiveX Control from:

```
http://www.microsoft.com/workshop/c-frame.htm#/xml/default.asp
```

While this may be the easiest option, if you want cross-browser compatibility, you'll have to look further into the other two options (JavaScript and Java Applets). Or dip into the possibility of creating the HTML server-side, something we look at in Chapter 9.

XML in the Real World

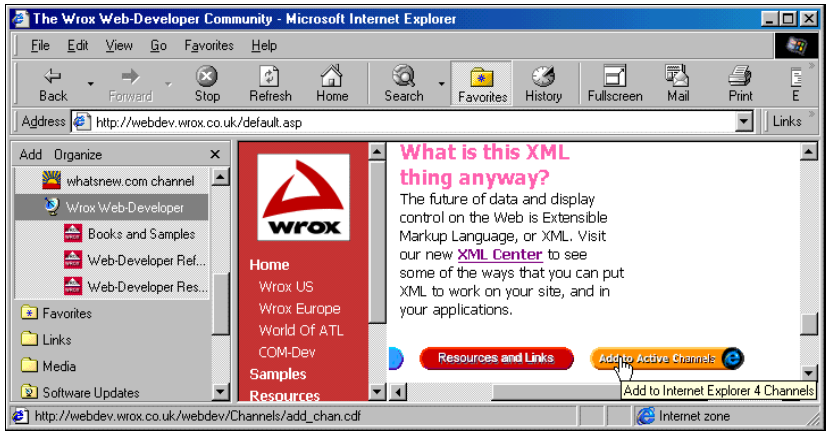
So far we have been talking about XML in rather general terms, so let's have a look at some examples where XML is already in use.

Channel Definition Format

When push technologies started to take off, browser manufacturers needed a way to describe the content of what was being pushed at your machine. Channel Definition Format (CDF) is an XML-based markup language which allows Web site authors to let subscribers know when the Web site changes, to varying degrees. CDF was introduced in IE4 and has significantly helped boost the profile of XML. Documents sent in the CDF format follow the CDF DTD.

CDF files are just linked to the `.html` or `.asp` files in a site. They remain separate, so there is no need to re-write your site in order to add CDF.

Note that CDF is different to the method that Netscape use. Here is our WebDev Channel in IE4. We explore channels further in Chapter 13.

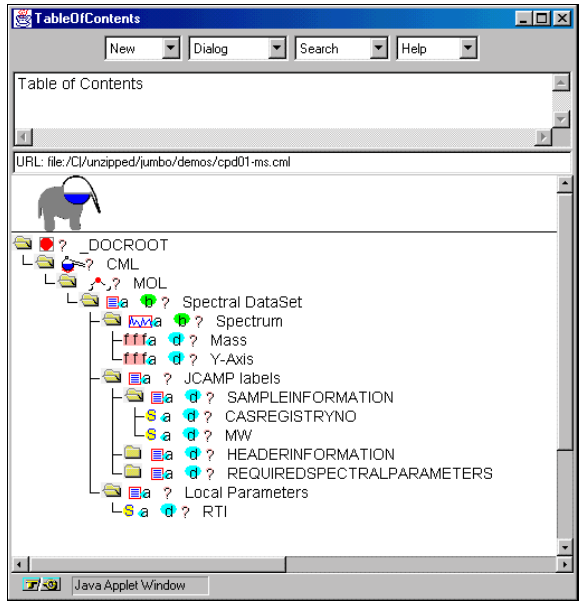


Chemical Markup Language

The Chemical Markup Language (CML) was designed by Peter Murray-Rust as a way of supporting the management of molecular information. Because of the subject area's complexity, the information could not adequately be rendered using normal HTML. Although CML was originally an SGML application, it has moved onto XML as a standard for its development.

CML is the nearest thing to an industry standard in the XML community so far. It allows researchers to capture chemical data in a form that can be reused. By using terms that are common in chemistry such as molecules, atoms, bonds, crystals, formulas, sequences, symmetries, reactions, etc., it makes the use of the XML application very straightforward for chemists. The way it handles objects means that documents can easily be worked on by a computer – searched and indexed. And because CML is written in XML it is platform independent, unlike the common binary formats used in the molecular sciences.

Professor Murray-Rust also created JUMBO, which was the first general-purpose XML browser, written in Java (although it is not suitable for displaying XML pages in the same way that the more popular Web browsers do). Jumbo is shown here demonstrating a document in CML:



JUMBO can be downloaded from:

```
http://www.vsms.nottingham.ac.uk/vsms/java/jumbo/
```

Open Financial eXchange

Open Financial eXchange (OFX) is the result of a collaboration between CheckFree, Intuit and Microsoft to develop a language that allows financial transactions to take place securely over the Web using the OFX DTD. Again it was originally an SGML application, although it now uses XML syntax. The proposal also includes security considerations to give peace of mind when sending credit card numbers over the Internet supporting Secure Sockets Layer (SSL) and the public/private encryption methods underlying SSL. It is designed so that it can be embedded in larger applications.

Among the activities which it supports are: consumer and small business banking, consumer and small business bill payment, bill presentment, and investments, including stocks, bonds and mutual funds. At the time of OFX was at version 1.5. It is intended to encourage the provision of online financial services. You can find out more about OFX at their Web site:

```
http://www.ofx.net/ofx/ab_main.asp
```

Summary

In this chapter we have taken a closer look at the pieces which go together to make up the XML jigsaw. We should have prepared you for the coming chapters that address these sections in more detail, after which we go on to creating some more complex XML applications. We have looked at:

- Well-formed and Valid Documents
- Document Type Definitions
- Document Type Declarations
- Cascading Style Sheets
- Extensible Stylesheet Language
- Parsers
- Namespaces
- XML Schemas
- How to view XML documents
- Some existing XML applications

Hopefully you will already be convinced of the advantages XML can offer. While XML may still be changing, the following chapters should give you a solid grounding in creating XML applications of your own. The later chapters will also encourage you that it is not too soon to be writing XML, and the examples will prove that it is possible to create significant XML applications despite a slight feeling of instability due to the evolving nature of such a new language.