XML—An Executive Summary

- XML explained
- Advantages of XML
- Relating XML to HTML
- Relating XML to SGML
- Existing users and uses of XML

Chapter 1

This chapter takes a look at the world of XML from a height of approximately 30,000 feet! It is intended to serve two purposes. Firstly, it gives readers with only 20 minutes to spare right now, a single chunk of reading to familiarize themselves with XML. Secondly, it sets the scene for the more comprehensive treatment presented in the rest of the book.

Having read this chapter, you will know enough about XML to discuss it in general terms with your boss or with your colleagues. This chapter will give you some idea of the broad areas of application of XML and, I hope, trigger some thoughts about how you can apply this exciting technology in your own projects.

I hope that it will also whet your appetite to dig into the rest of the book in order to flesh out your understanding of XML. Even if you are rushing off after this chapter, intending to return later on, please take the time to peruse Chapter 2, which provides practical illustrations of some areas where XML is being successfully deployed today.

This chapter is set out as a dialog of questions and answers, not unlike an Internet FAQ (Frequently Asked Questions) list. For best results, please read the questions and answers in the order they are presented. Ready to get started?

1.1 | Can you explain XML in less than half a page?

No! Too many good ideas and too many *killer applications* of XML exist for half a page to do it justice. But if you can only read half a page right now, try this for size . . .

XML is a computer language for describing information. So too is HTML. XML improves on the HTML approach and makes the Web a better place in which to do business, to learn, and to have fun.

HTML is a great technology, and it has changed the world. However, a great deal of useful information is *lost* when data is converted into HTML—information that, if preserved, can be used to build a whole new world of computer applications on the Web.

Compare this little document:

```
<!-- HTML Snippet -->
<h1>Invoice</h1>
<p>From: Joe Bloggs
<p>To : A. Another
<p>Date : 1 Feb 1999
<p>Amount : $100.00
<p>Tax : 21 %
<p>Total Due : $121.00
```

To this little document:

```
<!-- XML Snippet -->
<Invoice>
<From>Joe Bloggs</From>
<to>A. Another</To>
```

```
<Date year = '1999' month = '2' day = '1'/>
<Amount currency = 'Dollars'>100.00</Amount>
<TaxRate>21</TaxRate>
<TotalDue currency = 'Dollars'>121.00</TotalDue>
</Invoice>
```

Now put yourself in a computer's shoes. Which little document is easier to process? Which one captures the most useful information? Which has the most potential uses?

The distinction illustrated in these two snippets is the very essence of XML. XML is all about *preserving* useful information—information that computers can use to be more intelligent about what they do with our data.

You might be thinking, "Why not just add tags for **<Invoice>**, **<TaxRate>**, etc. to HTML?" This step could certainly be taken, but where does this process end? *No* set of tags, no matter how large, will ever come close to providing all the tags we might conceivably want. There must be a better way!

Well, a better way does exist, and it is called XML. XML is not a "go-faster HTML." It is a fundamentally different technology that liberates information from the shackles of a fixed-tag set. For example, if you are describing an invoice, why not call it an **<Invoice>** rather than a level 1 heading?

In my country (Ireland), the phrase "call a spade a spade" means "Speak plainly. Tell it like it is." This phrase captures the core idea of XML rather well. It could be a mantra for XML—"call a spade a **<spade>**"!

1.2 | Where did XML get its name?

XML is an acronym. It stands for e**X**tensible **M**arkup **L**anguage. Like all the best acronyms, it is a TLA—**T**hree-**L**etter **A**cronym.

1.3 | What does it do?

XML is not a software program and thus does not *do* anything unaided. The same can be said of HTML. XML provides a standard approach for describing, capturing, processing, and publishing information. It is a language that has significant benefits over HTML, as you shall see.

1.4 | Sounds complicated

Far from it! Indeed, the opposite is the case. XML is a set of ideas—all of them quite simple ideas. However, it will take a while to present all these ideas thoroughly enough to do them justice yet simply enough to be easily digested. Trust me on this one! By the time you have reached the end of this chapter, you will know what XML does and how it does it.

1.5 | Can you explain the term “markup language”?

I can use the book you are now reading to illustrate the concept of a markup language. This book, like many books before it, began life as a collection of electronic files. These files were created in a word processor. As the content was created, the word processor stored extra information, over and above the words you are now reading. This extra information consisted of instructions to control the layout and appearance of the words themselves. Such information is collectively known as *markup.*

The term *markup* dates back to the days before electronic documents, when publishing professionals used to take basic text from

authors on paper and write markup instructions to tell the typesetter how to make the document look good on the final printed page—e.g., "insert a paragraph break here, make this word bold, double space this text," etc.

The digital world we live in is positively awash with different markup languages. Firstly, you have the proprietary markup languages used by word processors, desktop publishing packages, etc. Then you have open nonproprietary markup languages such as TeX, Troff, and, of course—the most famous nonproprietary markup language of them all—HTML (**H**yper**T**ext **M**arkup **L**anguage).

1.6 | So XML is just another markup language?

No. XML is a markup language with a very important twist. Most markup languages, HTML included, are *fixed* markup languages. That is to say, they provide a certain feature set in their markup, and that set is fixed in the design of the language. HTML, for example, has a fixed set of *tags* with which you craft your documents—**<H1>**, **<P>**, **<TABLE>,** etc. XML, on the other hand, does not define *any* particular set of tags. Rather, it provides a standardized framework with which to define your own, or to use those defined by others that best fit your needs.

1.7 | What does XML look like?

It looks an awful lot like HTML! (This is not a coincidence, as you will see later on.) XML documents—just like HTML documents—

consist of a mixture of data and markup. The syntax for the markup is very reminiscent of HTML. Here is an example:

```
<Spice>
<Name>Sichuan Peppercorns</Name>
<CountryOfOrigin Country = "China"/>
<Description>Pungent, distinctive. Excellent
with slow cooked, earthy dishes.
</Description>
<Example>Sichuan Braised Chicken</Example>
</Spice>
```

With XML, you have the freedom to use pretty much any names you like to tag up your data. You can literally roll your own tags, just as I have done here!

1.8 | So XML is extensible because I can use it to make up my own tags?

Exactly. Unlike most other markup languages, XML is a flexible framework in which to create your *own* customized markup languages. All XML-based languages will share the same look and feel. They will all share a common basic syntax. Beyond that, the sky is the limit in terms of the diverse markup languages that can be built on the foundation that XML provides.

Naturally, not everyone who uses XML will feel a compelling urge to create his or her own markup language. The majority of people will simply use the XML-based markup languages created by others that best fit their purpose. Already, a number of industry standard XML-based languages exist in fields such as Push Technologies (CDF—**C**hannel **D**efinition **F**ormat), Electronic Commerce (OTP—**O**pen **T**rading **P**rotocol) and mathematics (MML—**M**athematical **M**arkup **L**anguage).

Any language based on XML consists of a set of *element types* that have been given certain names and certain meanings. Examples we have encountered so far in this chapter include Invoice, TaxRate and Spice element types. The presence of elements of various types in documents is indicated by *tags* that serve to indicate where the element starts and ends. For example the Spice element in the previous document starts with a start-tag "<Spice>" and ends with an end-tag "</Spice>".

A set of element types serves to define *types* of documents and are referred to as *Document Type Definitions*, or *DTDs* for short. Thus you will read and hear references to the CDF DTD, the OTP DTD, the MML DTD and so on.

1.9 | But why would people bother to invent their own XML-based markup language (DTD)?

The main advantage of being able to define your own markup language is that it gives you the freedom to capture and publish useful information about what your data is and how it is structured, instead of having to shoe-horn it into someone else's often ill-fitting format. This advantage is best illustrated by example. Consider a company running an e-Business selling PCs on the Internet.[1] Here is the sort of information the company needs to publish:

```
Maker : Acme PC Inc
Model : Blaster 555
Storage:
  RAM: 72 MB
  Hard Disk : 2 GB
```

1. A scenario I use extensively in this book

In order to publish this information using HTML, they need to create a document looking something like this:

```
<!-- snippet of HTML -->
<h1>Personal Computers For Sale</h1>
<h2>Maker : Acme PC Inc</h2>
<h3>Model : Blaster 555</h3>
<table border = 1>
 <tr>
   <td>Storage:</td>
 </tr>
 <tr>
   <td>RAM</td><td>72 MB</td>
 </tr>
 <tr>
   <td>Hard Disk</td><td>2 GB</td>
 </tr>
</table>
```

Opened in an HTML browser, this information looks like Figure 1–1.

The original data has been transformed into HTML for publishing purposes. In the course of that transformation, useful information

Figure 1–1 The PC for Sale information as shown in a Web browser

about what the information *really is* has been lost. The HTML version of the data knows nothing about PCs or hard disk sizes. All it knows about are heading levels, tables, italic text, etc. As a consequence, when this document is let loose on the World Wide Web, search engines and users alike see only a collection of levels, tables, italic text, etc., as in Figure 1–2.

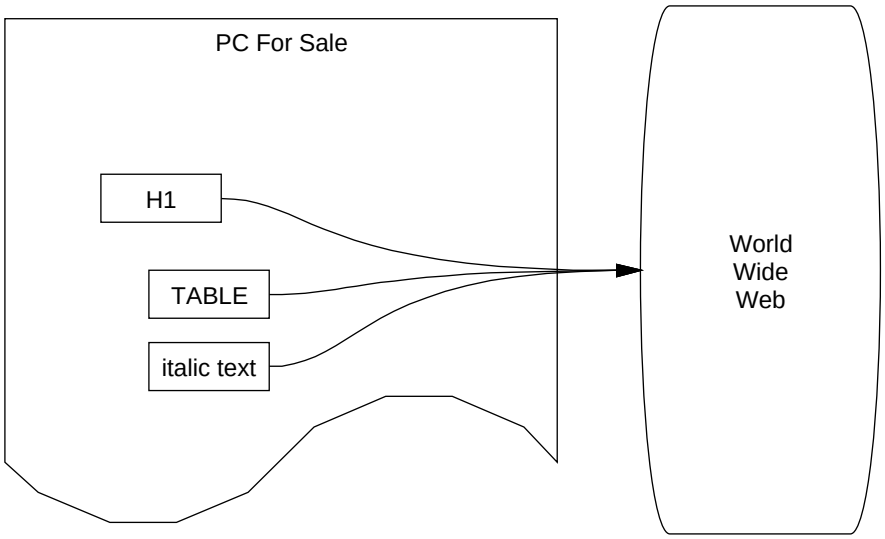
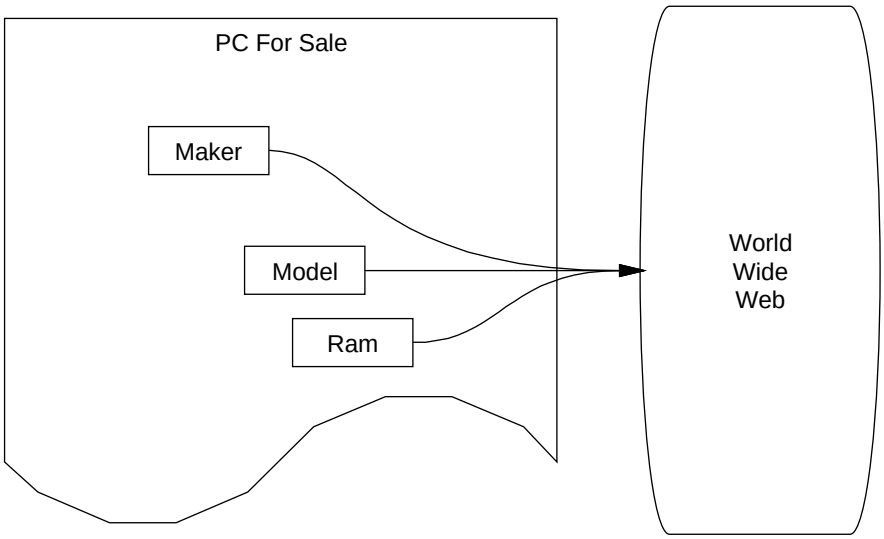


Figure 1–2 HTML document linked to the World Wide Web

Ignoring the details (and perhaps suspending disbelief!) for a moment, let your mind chew over the possibilities arising if the company could publish this on the Web instead:

```
<!-- Snippet of an XML document -->
<PcForSale>
 <Maker>Acme PC Inc</Maker>
 <Model>Blaster 555</Model>
 <Storage>
  <Ram Units = "MB">72</Ram>
  <HardDisk Units = "GB">2</HardDisk>
 </Storage>
</PcForSale>
```

I think you will agree that this representation opens up some pretty interesting possibilities. This document can have a much richer interface to the Web, an interface that presents all sorts of possibilities about how it might be put to use, as in Figure 1–3.



igure 1–3 XML document linked to the World Wide Web

By keeping information about what the pieces of data *really* are—i.e., a hard disk capacity, a PC model name, etc.—you can contemplate:

- Letting the browser do the work to format the data on the user's screen. Perhaps allowing users to choose between a variety of "looks" or presentation formats for the same data.
- Letting the user's browser perform calculations on the data, and manipulate and display the results in a variety of ways.

- Allowing intelligent searching of the information, e.g., "find all PCs for sale on the Web with disk drive capacity greater than 2 GB."
- Intelligently checking that all the pieces of information required for a proper entry on the PC selling Web page are actually there, e.g., "all PCs must have a RAM size element and can optionally have a hard disk size element."
- Performing complex queries on the data either for your own management purposes or as a service to customers, e.g., "how many laptop PCs with built-in CD-ROM drives were sold last month in Arkansas?"
- Building rich links between different types of information—for example, linking a sales invoice (itself perhaps an XML document!) with the particular makes/models of PCs it references.
- Standardizing a set of XML element types for an entire industry, such as PC vendors. Users and vendors alike would benefit from the standardization. Software "robots" could trawl the Web to find the perfect PC for you, based on criteria you specify. Vendors would be able to easily contrast their offerings with those of the competition via "tick sheets" and so forth.
- Avoiding the need to "dumb down" data into HTML prior to publishing. This activity often involves complex software and is frequently error prone. With XML, the data can be stored and published in the *same* format. You don't need either batch or on-the-fly translation into HTML (although XML allows you to continue doing that if you so wish).

1.10 | Is some philosophical stuff going on here that I need to know?

Yes. The core philosophy of XML has come about as a result of a long and thoughtful analysis of what is really meant by the term "document" in the digital world. By and large, documents consist of three distinct components, namely:

- Data content—the words themselves
- Structure—the document type and the organization of its elements, i.e., memo, contract, cooking recipe. Also, what kind of elements it can contain and in what order they can occur.
- Presentation—the way the information is presented to the reader, on a piece of paper, a browser screen or via voice synthesis. Also, which fonts or voice inflections are used for each element type and so on.

The central idea of XML is that *significant* benefits accrue to the document owner if these three aspects of a document are kept separate and made explicit in a computer system. Now compare and contrast the treatment of these three strands of a document in traditional word processors to their treatment in XML (see Figure 1-4).

A word processor—especially a WYSIWYG word processor—entwines content and presentation in a very tight embrace. Using such tools, we create documents with a specific output device in mind—typically paper of a particular width and height. As we create the content, we are ever watchful of the appearance of the result; we inextricably bind that content to a particular presentation. Indeed, being able to do so is the very essence of the WYSIWYG (What You See Is What You Get) philosophy. As for structure—capturing what the information really is—the concept is hardly present

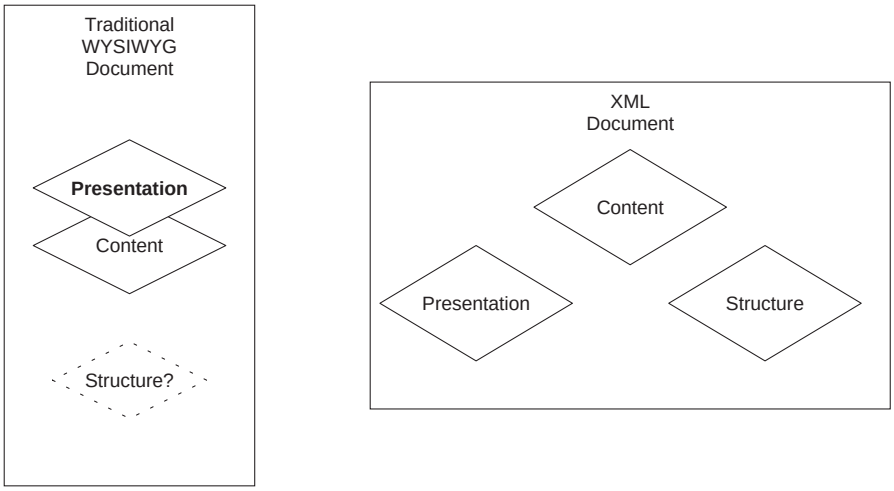


Figure 1–4 Content, structure, and presentation in traditional and XML documents

at all. The only structural information stored relates to the creation of the final paper output—details about page margins, font sizes, and so on.

This approach is in stark contrast to that espoused by the XML approach. The inherent structure of documents such as procedure manuals, invoices, and tax returns is considered just as important as the content itself. Presentation information is also, naturally, important but is kept well separated from the content. In XML, you create document content by concentrating on what the information really is and how it is structured (see Figure 1–5). We defer issues to do with presentation, leaving them to be dealt with at the point where someone needs to *look* at the document. I will come back to presentation in "But how do I make XML look nice in a browser?" later on in this chapter. For now, suffice it to say that an XML document can be made to look arbitrarily beautiful without intertwining the formatting information with the core content of the document.

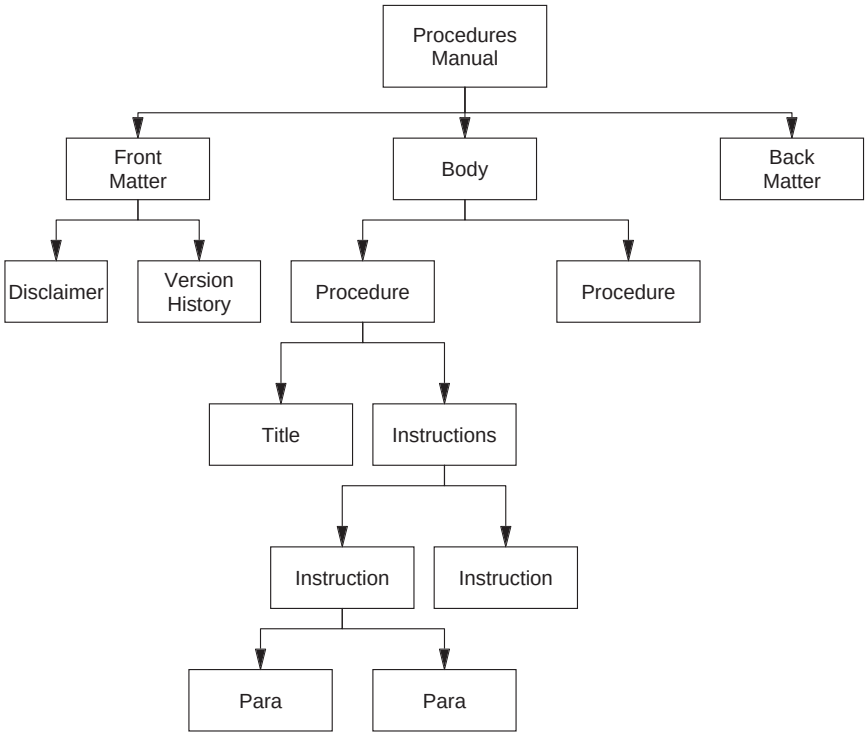
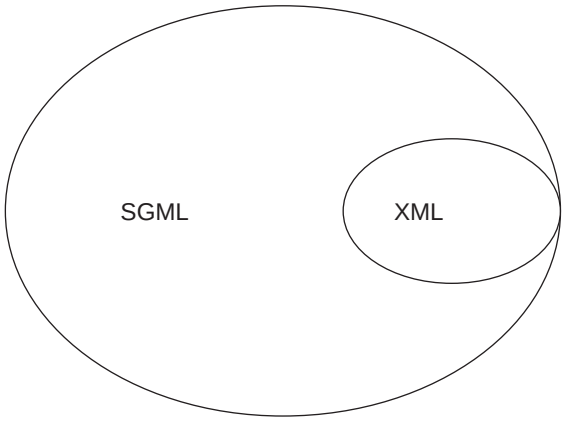


Figure 1–5 Representing the logical structure of a document

1.11 | Ah! So that is what they mean by “structured documents”!

In a word, yes.



igure 1–6 The relationship of XML to SGML

1.12 | Did someone just sit down and, you know, “invent” XML?

XML was certainly invented but not out of thin air[1]. Since 1986, an international standard has existed for doing what XML does. In fact, that standard does a lot more than XML does, in many respects. Its name is SGML—Standard Generalized Markup Language—ISO 8879.

SGML is a very powerful, very general standard, but with that power comes increased complexity. XML is a subset of SGML intended to make SGML “light” enough for use on the Web. XML is a *proper subset* of SGML. That is to say, all XML documents are valid SGML documents. However, not all SGML documents are valid XML documents (see Figure 1–6).

SGML has been used very successfully over the years in industries such as technical publishing, pharmaceuticals, aerospace and so on. Some major SGML initiatives are listed in Table 1.1.

1. Someone did sit down and invent the original form of SGML—Charles F. Goldfarb, the editor of this series.

Table 1.1 Some SGML Initiatives

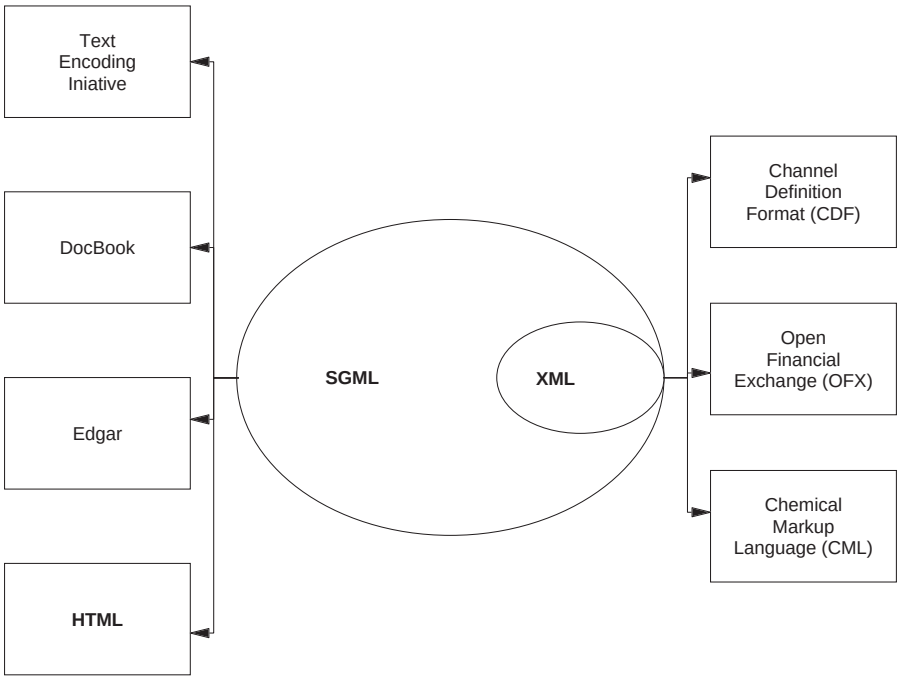
Name	*Industry*
ATA	Aviation (Air Transport Authority)
DocBook	Technical manuals
Text Encoding Initiative (TEI)	Encoding of literature
J2008	Automotive maintenance
Edgar	Financial reports for public companies
HTML	Hypertext Markup Language
Pinnacles (PCIS)	Semiconductor data

1.13 | Is something wrong with SGML?

No. The complexity of implementation that is a by-product of SGML's power has had the effect of limiting its user base to big companies that need all that power. Organizations with tens of thousands of pages of information are typical SGML users. Having said that, the ideas that SGML embodies are just too good and useful to be restricted to such a niche. Hence XML—a simplified SGML that retains most of the inherent power of SGML in a simple, tidy, easy-to-use, easy-to-implement form.

1.14 | Can you draw me a picture of how all these languages are related?

Sure! In Figure 1–7, rectangle boxes indicate *applications*, and ellipses indicate *framework languages* or *meta-languages*, if you like. From it, you can see that XML is a simplified version of SGML; and CML, CDF, and so on are XML applications; whilst HTML, Edgar, and Docbook, etc., are SGML applications.



igure 1–7 The relationships among SGML, XML, and HTML

1.15 | Can the structure of an XML document be checked somehow?

Yes. XML includes a mechanism for defining rules that control how documents are structured. In jargon, these are called *Document Type Definitions,* or DTDs for short. In a DTD, you can arrange for XML documents to be automatically checked in various ways. Here are some examples.

- A person's name consists of an optional title, a given name, and a surname.
- A TV timetable contains one or more channels. Each channel contains one or more time slots. Each time slot has a program title and an optional description.

These effects can be achieved in the Document Type Definition by listing the element types you wish to use in your document and indicating the structural order in which they can occur. A utility program called an XML Parser is then able to test whether or not the document meets the prescribed rules (see Figure 1–8).

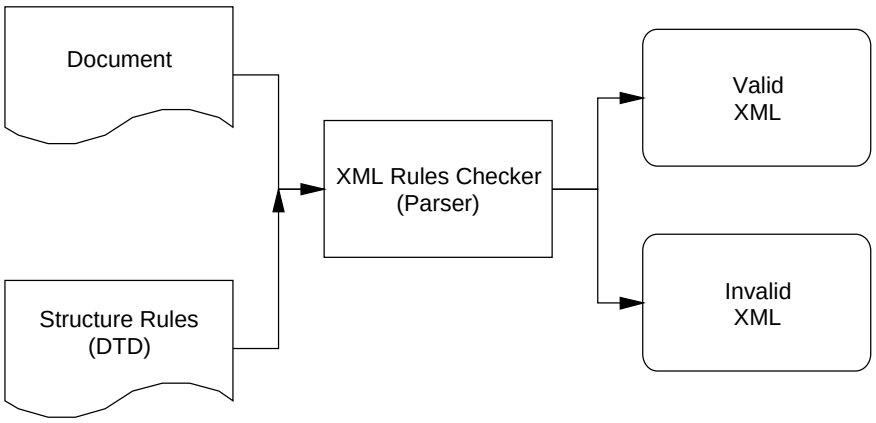


Figure 1–8 Checking the structure of an XML document with an XML Parser

1.16 | What if I do not want my structure checked?

No problem. With XML, it is perfectly okay not to rigorously check the structure of your documents against a DTD. As long as the elements *nest* properly within each other, creating a tree-like structure, the document is known as a *well formed* XML document. Well formed documents are particularly suited for Internet use because they can be processed with simple XML tools. These tools are small and light enough to be used in everything from browser applets to credit-card swipers to laboratory equipment.

1.17 | But how do I make XML look nice in a browser?

In XML, presentation and content are kept separate for reasons I discussed in "Is some philosophical stuff going on here that I need to know?" earlier in this chapter. Making XML look nice—either in a browser or on a sheet of paper—is the responsibility of an XML subsidiary standard called XSL—**X**ML **S**tyle **L**anguage.

You may be familiar with the concept of a style sheet in a word-processor, or you may have come across a style-sheet standard for HTML called Cascading Style Sheets (CSS). The core idea is to capture details about how the various elements in a document should look and then to store them in a separate document, rather than intertwine them with the content of the document. Separating the two allows the presentation to be changed by simply changing the style sheet. XSL is the proposed style-sheet language for XML. It is more powerful than CSS yet broadly compatible with it. In the same way that XML is a subset of the SGML International Standard (ISO 8879),

XSL is a simplified subset of the International Standard style language known as DSSSL (ISO/IEC 10179)—see Figure 1–9.

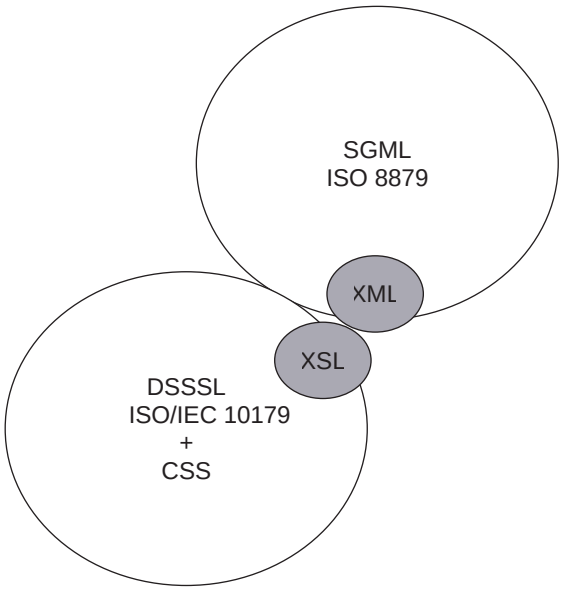


Figure 1–9 The relationship between XML, XSL, and ISO Standards

1.18 | What about hypertext?

With HTML, we are used to the idea of having hypertext-linking functionality built directly into the language. In HTML, the famous **<A>** element serves this purpose. As you know, XML does not predefine any elements, so how do you go about specifying hypertext links? Just as in the case of presentation information, XML delegates the task of capturing hypertext information to a subsidiary standard known as XLL—e**Xtensible L**ink **L**anguage (see Figure 1–10).

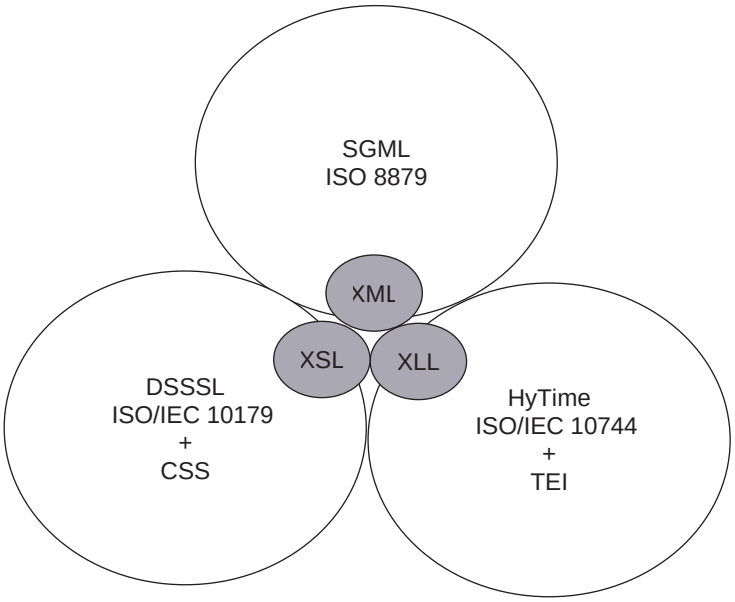


Figure 1–10 Adding XLL to the family of XML standards

The XLL standard draws heavily upon an existing standard for expressing hypertext links in SGML documents, known as HyTime (yet another ISO standard—ISO/IEC 10744). It also draws on the wealth of experience in dealing with complex hypertext linking that has built up over the years in the SGML-based Text Encoding Initiative.

1.19 | So XML is based on truly international standards?

Right down to its toes! XML is derived from SGML (ISO 8879), XSL is derived from DSSSL (ISO/IEC 10179), and XLL is derived from HyTime (ISO/IEC 10744). On top of that, the native character set of XML is Unicode (ISO/IEC 10646). Any system that declares itself to be XML-compliant must be able to handle the Uni-

code character set. Unicode, with its multiple byte characters, supports a wide variety of languages and alphabets. Support for Unicode is becoming increasingly common; it is directly supported in programming languages such as Java and in operating systems such as Windows NT and AIX.

1.20 | Where does all this leave HTML and the concept of a browser?

In pretty good shape, actually! XML can usefully be seen as a generalization of the information-publishing paradigm pioneered by HTML and the World Wide Web. HTML is not threatened by XML. In fact, a direct comparison of the two is meaningless. XML is a *framework* for making markup languages; HTML is an *example* of a markup language. If HTML is like a slice of bread, XML is like a bakery. If HTML is a red rose, XML is a greenhouse.

In fact, not only are HTML and XML not in competition, but HTML may well one day become an XML application!

1.21 | Why not just let people invent proprietary languages—why base them on XML?

The primary reason is to ensure that your information can leverage the existing set of open standards, tools, and expertise available for XML. For too long now, the lowest common denominator in the document world has been *plain text*. XML builds on top of plain text,

allowing layers of useful information to be captured along with basic data, and it does so in a completely application-independent, vendor-independent fashion.

With the range of uses to which XML is being put and the range of XML software (much of it free!) that is available, it will become increasingly difficult for developers to cost-justify inventing yet another proprietary syntax. This difficulty is a good thing. Every proprietary syntax creates another isolated island of information and another doubling of the world's already bulging set of file conversion utilities!

1.22 | Where does XML fit in with other information technology standards?

XML is, first and foremost, a document technology. Having said that, the XML approach to information as a threesome of content, structure, and presentation cuts right across the world of Information Technology. XML can be gainfully applied to database modeling, for example. Also, nothing, in principle, can prevent XML from being used as a graphics file format! It would not be terribly space-efficient, but that is not the point. The point is that XML's modeling power has significant depth. Who knows where it will be applied outside its initial realm of documents? It may well emerge as a base interchange framework for all electronic data.

1.23 | If XML is so clever, how come the Web was not designed that way in the first place?

The Web started out with modest aims—to allow easy dissemination of information amongst a group of scientists at CERN. It has expanded somewhat since then! Given the original requirements, HTML was a very good design, easy to read and easy to write, both for humans and computers.

However, the scope of the Internet has extended at such a pace that the sheer simplicity of HTML—on one hand the Web's greatest asset—is becoming one of its greatest weaknesses. To extend the Web into areas now envisaged for it—such as electronic commerce, health-care information, on-line voting and the like—the Web needs a more extensible, robust, and formally defined standard. We owe HTML a great debt of gratitude. It has opened our eyes to a whole new vista of possibilities for Information Technology. It still has a significant role to play as a standard *display* format. However, it will become just one of a family of markup languages in everyday use on the World Wide Web—the vast majority of them based on XML.

1.24 | Okay. Sounds good, but let's cut to the chase. Who out there is using XML and for what purposes?

Here are some corporate names you may recognize:

- Microsoft
- Netscape

- Sun Microsystems
- Adobe
- IBM
- Corel
- Hewlett-Packard

Here are some XML application areas you may recognize:

- Online Banking
- Push Technology
- Web Automation
- Database Publishing
- Software Distribution

Would you care to see some proof? Proceed to Chapter 2, where I take a closer look at some of these real-world XML applications.