Chapter 4

The Promise of XML

With this chapter, we begin discussing ebXML's background and underlying technology, starting with the *Extensible Markup Language* (*XML*). We will provide an overview of XML technologies, and discuss various features related to e-business, as well as its limitations for business purposes. This chapter is mostly technical in nature, to explain the raw technology itself and how the various components relate together. Our goal is to provide the reader enough of a sense of XML so that its overall use for business is clarified. The reader should view this chapter as a roadmap to the salient features of XML itself; however, it is not intended to be a tutorial in XML syntax.

What Is XML?

XML makes possible the entire idea of using the World Wide Web and the Internet for exchanging business messages. XML is a generic *markup language*, which means that it provides instructions that only define the message or document content, not how that content is displayed or printed. For example, the instructions can say, "This block of text contains a business shipping address."

By focusing on the content and detailing the precise business context, XML makes it possible for systems in remote locations to exchange and interpret such documents without human intervention. This ability

to automatically send, retrieve, interpret, transform, and process the data in electronic messages is of course critical to the conduct of electronic business itself.

The *World Wide Web Consortium* (*W3C*) developed XML in 1996–97, and officially released version 1.0 in February 1998.[1] While XML is widely recognized as a technology and the W3C is a highly respected organization, drawing its membership from both major software vendors and academic institutions, the W3C chooses to call its fully approved technical documents *recommendations* rather than *standards*, to avoid anti-competitive lawsuits in the U.S. Recommendations represent a consensus within the W3C as well as the approval of the W3C director, now Tim Berners-Lee. As recommendations, documents such as the XML specifications demonstrate stability and are considered ready for widespread implementation and business use.[2]

Markup: Seeing Is Believing

The World Wide Web emerged as a common communications medium once the *Hypertext Markup Language* (*HTML*) became available in the early 1990s.[3] HTML is also a recommendation of the W3C (the latest version is 4.01, December 1999), and now there is also an XHTML recommendation (February 2001). HTML provides a good example of a markup language in wide use, and makes a convincing case study for the importance of consistent standards.

You can see HTML markup by opening any web page with Internet Explorer or Netscape Communicator. Using the top-level menu in the browser, select View, Source (Internet Explorer) or View, Page Source (Netscape). What you see displayed is the internal HTML syntax that the browser uses to render the page content you see onscreen. The familiar web page with its human-readable text and images are exposed as machine-readable computer markup code. Notice that the code contains a lot of instructions in angle brackets, such as `<HTML>`, `<BODY>`, `<HEAD>`, `<TITLE>`, `<TABLE>`, and so on. (See Listing 4.1 for an example.)

Enclosing the syntax text within angle brackets creates a *tag* or *element*. Close to the top of the web page's HTML markup source is the tag <HTML>. This tag tells the web browser that the page is coded in HTML; the web browser responds by displaying the information as directed by the rest of the tags on the page. At the bottom of the page is a similar tag, </HTML>. The slash after the opening angle bracket in the tag tells the browser that it has reached the end of the HTML page. The <HTML> tag is called an *open tag*, and the </HTML> tag is a *close tag*. The markup also contains other tag pairs: <HEAD> and </HEAD>, <TITLE> and </TITLE>, <BODY> and </BODY>. These tags define parts and functions of the HTML document.

Listing 4.1 Sample of HTML Markup

```
<HTML>
<HEAD>
<TITLE>Dynamiks Research Center News
Homepage</TITLE>
<meta http-equiv="Content-Type"
content="text/html; charset=iso-8859-1">
<meta name="keywords" content="press releases,
wind tunnels, aerospace".>
</HEAD>
<BODY bgcolor="#FFFFFF">
<TABLE width="100%" border="0" cellpadding="0"
cellspacing="0">
</TABLE>
</BODY>
</HTML>
```

The power of HTML is that it's very simple to use, as the HTML software excuses most obvious mistakes by human editors—unclosed tags, orphaned tags, mistyped tags—by always displaying something, not just a blank page. This leads to very complex HTML software, but ease of use for content creators. HTML has a fixed set of markup tags and most HTML software readily understands such commonly used tags. Because HTML is a standard more or less recognized by the browser manufacturers,[4] millions of people and companies worldwide have found new and innovative ways of communicating over the web—and in many cases doing good business—without worrying about too many technical details.

XML takes a different approach, first by allowing its users to create their own tags (hence the *extensible* part of its name). As a result, XML is highly suited to describing your own particular business data in messages and exchanging those messages with trading partners. Listing 4.2 shows the XML markup of a customer's telephone number, using the XML vocabulary from version 3.0 of the xCBL syntax:[5]

Listing 4.2 Sample XML Content for a Supplier Mailing Address

```
<?xml version="1.0" encoding="UTF-8"?>

<Supplier>
 <NameAddress>
  <Name1>ABC Wholesale</Name1>
  <Address1>1222 Industrial Park Way
  </Address1>
  <City>South San Francisco</City>
  <StateOrProvince>California</StateOrProvince>
  <PostalCode codetype='ZIP'>96045</PostalCode>
  <Country>US</Country>
 </NameAddress>
</Supplier>
```

XML elements use start and end tags as in HTML. However, the elements also contain attributes such as `codetype` within the `<PostalCode>` tag. *Attributes* act as qualifiers of the elements, providing more definition or direction to the trading partners exchanging the messages. Attributes are familiar in HTML too, such as the `<FONT typeface="italic">I said hello!</FONT>` instruction, where `italic` qualifies the style of presentation font for the text. Similarly, in the case of the XML postal code number shown in Listing 4.2, the attribute tells us that this is a U.S.-style numeric-based ZIP code.

HTML uses a fixed set of tags for display of text, not for the definition of data. Listing 4.3 shows the same information as in Listing 4.2, but coded in HTML.

You may notice another characteristic of XML from this example—its readability. XML doesn't restrict tag writers to specific string lengths; tags can be labeled to confer hierarchy, context, and meaning.

Listing 4.3 HTML Content for a Supplier Mailing Address

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<HTML>
<HEAD>
   <META HTTP-EQUIV="Content-Type"
   CONTENT="text/html; charset=iso-8859-1">
   <META NAME="Author" CONTENT="Alan Kotok">
   <META NAME="GENERATOR"
   CONTENT="Mozilla/4.06 [en]C-gatewaynet
   (Win95; I) [Netscape]">
</HEAD>
<BODY>

<ADDRESS>
Supplier name and address:</ADDRESS>

<ADDRESS>
Name: ABC Wholesale</ADDRESS>

<ADDRESS>
Address: 1222 Industrial Park Way</ADDRESS>

<ADDRESS>
City: South San Francisco</ADDRESS>

<ADDRESS>
State: California</ADDRESS>

<ADDRESS>
Zip: 96045</ADDRESS>

<ADDRESS>
County: USA</ADDRESS>

<BR> 
</BODY>
</HTML>
```

XML, Where Past Is Prologue

A review of XML's background shows some further thinking behind the development of XML, as well as its current readiness as a tool for business.

XML is a subset of the *Standard Generalized Markup Language* (*SGML*), a markup language first conceived in the late 1960s. A committee of the *Graphic Communications Association* (*GCA*) determined the need for standard page-composition instructions sent

from publishers of books and journals to printing plants. Individual printers at the time had their own means of marking up the text with codes that translated into font sizes or effects, such as boldface or italics. They recognized that a standard means of marking up the text would make it possible for any publisher to communicate in the same way with any printer, and save the publishers the headaches of reconciling one form of markup with another.

HTML's large set of features was designed to handle the demanding requirements of scientific and technical documentation and went well beyond the needs of people and companies to display text and images or exchange business messages.

The GCA committee proposed separating the information content from the presentation format and developing a generic code to represent the format, rather than trying to decipher each printer's specific coding scheme. The generic code would be represented in a set of descriptive tags. The tags would indicate where the information for the heading of the document resided—identification of the author, date, title, and other general details—as opposed to the body of the document that contained the intellectual product.

By 1969, Charles Goldfarb, then working at IBM, led a research project to build on the GCA committee's ideas for a *Generalized Markup Language* (*GML*) for text editing and formatting to enable electronic document sharing and retrieval. GCA, working first with the *American National Standards Institute* (*ANSI*) and then with the *International Organization of Standards* (*ISO*), moved GML from an IBM proposal into a recognized international standard—the *Standard Generalized Markup Language* (*SGML*), ISO 8879, in 1986.[6]

Based on this standard, the U. S. Department of Defense, the Internal Revenue Service, and other such organizations with large numbers of complex documents were able to invest in systems to help them manage their electronic publishing operations. The European Particle Physics Laboratory in Geneva (which uses the organization's original French acronym, *CERN*) became another major user of SGML. While on staff at CERN, Tim Berners-Lee developed HTML as an application of SGML in the late 1980s and early 1990s.[7]

The development of SGML predated the emergence of the Internet, at least as we know it in the year 2001. What attracted Berners-Lee and many of the other web pioneers to the Internet was its decentralized nature and a design that allowed any kind of computing platform to plug in, as long as it complied with the Net's protocols. The public availability of the Internet created the potential for anyone to exchange such marked-up documents with ease.[8] HTML transformed the Internet from islands of hard-to-find content into one homogeneous whole that's visible through a web browser interface.

Meanwhile, companies, agencies, and organizations with large electronic publishing operations—usually technical, scientific, engineering, financial, or legal—found SGML useful in managing their documents and re-purposing the content in those documents. Because of its nurturing in the publishing world, however, SGML contains complexity that the average user finds intractable. Its large set of features was designed to handle the demanding requirements of scientific and technical documentation and went well beyond the needs of people and companies to display text and images or exchange business messages.[9]

Figure 4.1 shows the timeline for development of XML and the other main markup languages, as well as EDI and ebXML. The development of the web-based markup languages, both HTML and XML, came about in part to provide an alternative to the highly complex and feature-rich SGML. With HTML, the ability to write web pages with simple and inexpensive tools (free, in many cases) makes everyone with a web connection a potential publisher.

And the numbers seem to point out that the world has responded accordingly. According to Whois.Net, more than 32 million domain names with `.com`, `.net`, or `.org` extensions were registered as of November 2000, and NetCraft's domain search engine lists nearly 1,000 domains with *XML* somewhere in the name.[10]

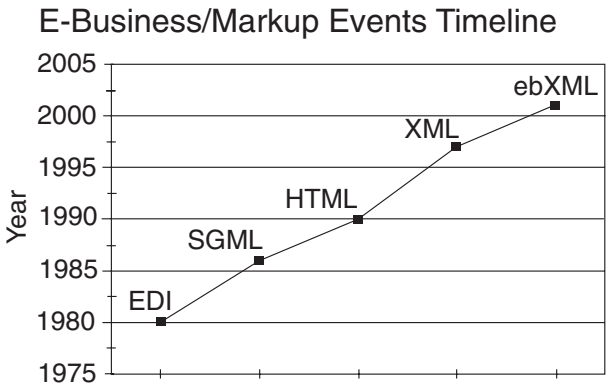

Figure 4.1
Evolution of markup technologies.

While HTML offered an effective way of presenting images, text, and multimedia content on the Internet, it still didn't meet many of the critical needs of business information dissemination. HTML has a fixed set of tags. While easy to learn, it's too generalized, and in particular doesn't provide any means to interpret the context of the information within a web page.

XML aimed to bridge the gap. Figure 4.2 shows how the needs can be viewed as four interrelated technologies: the Internet delivering content, HTML presenting it, XML identifying the data content, and Java and similar programming tools providing the process control.

Having identified the need, the World Wide Web Consortium committee convened in 1996, and, led by Jon Bosak of Sun Microsystems and Tim Bray of Textuality, designed XML for electronic publishing. The group focused on creating a simpler form of markup to overcome the obstacles to broad adoption shown by SGML. They therefore set 10 design objectives for the new language:

- XML shall be straightforwardly usable over the Internet.
- XML shall support a wide variety of applications.
- XML shall be compatible with SGML.

- It shall be easy to write programs that process XML documents.
- The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
- XML documents should be human-legible and reasonably clear.
- The XML design itself should be prepared quickly by the W3C team.
- The design of XML shall be formal and concise.
- XML documents shall be easy to create.
- Terseness in XML markup is of minimal importance.[11]

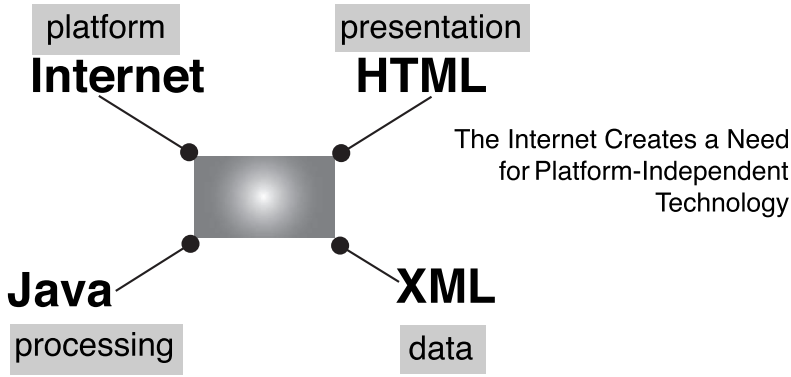

Figure 4.2
The role of XML: the four-legs-of-the-table metaphor.

XML Validation and Parsing

An XML document by itself is just arbitrary text. To describe the actual rules to be followed in creating your particular type of XML content, you need an additional mechanism.

One of the features of SGML that carried over into XML version 1.0 is the concept of a *schema*, which describes the layout of a document, also known as the *Document Type Definition* (*DTD*). The role of both schemas and DTDs is to allow the author to define the structure permitted for any given XML document, including the relationships among elements in the document. Think of schemas or DTDs like the instructions that come with a Lego™ bricks model for assembling the pieces in the correct order.

The terms *schema* and *DTD* are often used interchangeably, but they have specific meanings. A schema is a generic term for document or data structures with a predetermined set of rules. A DTD is one type of schema, specified in SGML and XML 1.0.

The DTD also provides a way of testing the structure of a document against the prescribed structure in the DTD, a process called *validation*. This validation step, designed as a quality check for documents, also can be used to check the structure of business messages sent using XML.

The combination of its extensibility, structure, and validation makes XML useful not only for electronic publishing but for business messages sent between companies.

The XML schema or DTD therefore performs two roles. It acts as a blueprint to allow someone who has no prior knowledge of your particular XML to create that content. It also allows software to check content to make sure it is correctly structured.

But XML allows for sending documents without such validation being required. The XML creators allowed for XML documents that are correctly tagged, but that don't have a schema DTD, and thus can't be tested for any structural validity. These documents are referred to as *well-formed documents*, indicating that they meet the basic XML markup syntax rules.

A *valid XML document* is both well-formed *and* meets the additional requirements of the schema DTD.[12] Again, the Lego model is instructive; if you lose the printed directions, you can probably still build an interesting model, but you won't know if it exactly matches the original design.

The combination of its extensibility, structure, and validation makes XML useful not only for electronic publishing but for business messages sent between companies. The ability to define the elements exchanged between companies and the structure of the elements means that trading partners can define messages in advance and thus process the messages automatically on receipt. Having validation means that trading partners can test the messages against the associated schema DTD, and thus provide a form of quality assurance.

To validate an XML message with a DTD, the message needs to be read and interpreted, in a process called *parsing*. A software component called a *parser* reads the XML message and interprets the XML tags it finds. A *validating parser* tests the message against the predefined rules of the schema DTD and then reports any errors.

To help provide software programmers using parsers with a standard connection between the message and the parser, the W3C developed the *Document Object Model* (*DOM*), independent of software languages or computing platforms.[13] XML documents have a nested structure that resembles a tree with a trunk and branches. The DOM represents the XML message as an inverted hierarchical tree, starting with the root element and branching out from there. By defining this logical structure in a common application program method, parsers and other software packages can manipulate messages consistently. Software developers call this kind of tool an *Application Program Interface* (*API*).

Microsoft's web browser, Internet Explorer (IE), displays XML documents using the DOM. If you open a well-formed XML document with IE 5.0 or higher, you'll see the document hierarchy clearly portrayed. The W3C approved Level 1 of the DOM in 1998, but had some enhancements approved as of November 2000.[14] Under the hood, IE 5.0 provides an automatic visual display of an XML document with another technology called the *Extensible Stylesheet Language* (*XSL*), and a default stylesheet.

One limitation of the DOM approach is that the whole XML document must be stored in memory at the same time. Obviously, this doesn't work for high transaction-volume or large-sized business information flows. In a process befitting the free and open nature of the Internet, members of the XML Developers mailing list (XML-DEV) developed an event-based programming interface called *Simple API for XML* (*SAX*), while waiting for the W3C to finish work on the more complex DOM specifications. SAX therefore allows programmers to process just fragments of XML content at high speed.[15]

Therefore, SAX is an event-based rather than a tree-based API. The event-based approach looks for tags and content meeting some conditional criteria that identifies the fragment within the overall information stream. The SAX API then passes that fragment to a custom event handler (software program) that the programmer has defined. SAX lets systems access and query only those parts of XML documents without loading them entirely into memory, thus working faster and more efficiently. All the major vendors providing XML parser implementations support SAX.[16]

XML's Global Reach and Accessibility

Although XML is a creation of the W3C, companies don't need the web to send and receive XML messages. XML's first design objective makes XML straightforwardly usable over the Internet, not just the web. As a result, trading partners can exchange documents with email messages or *File Transfer Protocol* (*FTP*) downloads, as well as over the web. With XML, the means of transporting the messages is independent of the message content.

XML Works with Non–English Character Sets

We often take the ASCII-English alphabet codeset for granted, but we forget that most of the world uses alphabets and characters not based on simple Latin (Roman) characters.

Since the Internet made the information technology business truly a worldwide endeavor, the designers of XML added an important XML feature, namely the ability to support non–English character sets. In North America and Western Europe, we often take the ASCII-English alphabet codeset for granted, but we forget that most of the world uses alphabets and characters not based on simple Latin (Roman) characters. XML supports the Unicode standard, a system for representing text characters for computer processing of all the known 50,000 written languages on the planet.

The latest version of Unicode (3.0) matches up to the international standard for character sets, ISO/IEC 10646-1:2000. It uses pairs of two bytes or 16 bits to represent characters, which allows for encoding most of the world's known character sets, including scientific and mathematical symbols. As a result, Unicode provides codes for more than 65,000 characters.[17]

With the worldwide nature of business today, this ability to represent non–English characters has become vital for many businesses.[18] Fortunately, the design of XML is backwardly compatible with today's ASCII 8-bit encoding, so regular ASCII editors and tools work just fine handling and creating what are labeled as "UTF-8 encoded" XML documents.

XML Works with Java

While the development of the Java[19] programming language preceded the development of XML, the two technologies now complement each other. Java is a high-level language used extensively in distributed applications over the web. Sun Microsystems developed Java to run on any computing platform. Programs written in Java are first compiled into an intermediate form called *bytecodes*—machine codes that are interpretable on most computing platforms.[20]

In 1997, Jon Bosak of Sun Microsystems, one of the creators of XML, wrote a white paper describing ways that the two technologies could work together. Bosak pointed out that "XML gives Java something to do." He described potential applications of XML in which the processing is distributed among client and server sites rather than centralized in a single server, using Java applets. For example, a design engineer could download XML data from a manufacturer's web site, and then use distributed Java code to try the circuits in various configurations.[21]

Matthew Fuchs notes several affinities between XML and Java that make them a productive partnership. Java uses a simple and predictable package structure that follows the structure of a typical Windows or UNIX filesystem. As a result, when sharing data with XML documents, programmers can easily route the data to the correct location thanks to this property of Java.

XML supports the use of style sheets that contain the instructions for presenting data on screens, in print, and in audio formats. Style sheets provide the formatting details for visual display or printing, such as page size, margins, and fonts.

Another feature of Java loads code dynamically at runtime, which allows for *applets*—pieces of Java code that browsers can download and run locally rather than relying on a full program at a remote site. This ability allows applets to run code that can process XML documents locally at much higher speeds and with much less overhead.

Fuchs also names Java Beans technology as an innovation that works well with XML. *Java Beans* are a set of application program interfaces that work as components with other software.[22]

XML Works with Style Sheets

Early in this chapter we discussed how markup languages such as XML separate content from its presentation format. Since business-to-business exchanges involve sending data from one computer to another, they don't require a human-readable version at either end of the exchange. But many business processes need to present the exchanged data in some human-readable presentation form. XML supports the use of *style sheets* that contain the instructions for presenting data on screens, in print, and in audio formats.[23]

Style sheets provide the formatting details for visual display or printing, such as page size, margins, and fonts. They are used frequently in word processing (but often called *templates*); for example, many organizations have a standard fax cover page template.

Style sheets have important business uses and do much more than just make data look pretty. For example, documents formatted for North American customers generally need to be printed on standard letter-size pages (8 1/2 inches by 11 inches), while other parts of the world commonly use the A4 size (210 x 297 mm).

While HTML by itself offers some ways to present text and images on a web page, its features are limited and don't provide enough power or flexibility for professional print content designers.

Another problem with HTML is consistency and reuse. This problem has been addressed in advanced word processing products by the use of *styles*; the user can apply a paragraph style, a table of contents style, an indented list style, and so on. HTML also has a need to use consistent style of text display size, font, and layout. HTML needs a separate style system, called *Cascading Style Sheets* (*CSS*), to display web page content. CSS can also be used to display XML content in the same way.[24] The method CSS uses is very simple, but requires that the content of the XML already be structured in a way that matches the output layout.

The W3C has developed an even more powerful style sheet for XML documents, called the *Extensible Style Sheet Language* (*XSL*), that gets around the restrictions of CSS and has two sets of core features:

- **Transformation** changes XML data according to predefined program rules.
- **Formatting** provides presentation rules and instructions to display the XML content as HTML, or some other provided target markup.

The transformation features of XSL give it extra power over CSS. With XSL, you can add or remove elements from an XML file, rearrange the elements, and make decisions about the display of the elements.[25] Figure 4.3 shows an example of XSL stylesheets displaying XML data on an HTML page.

The next need is to locate information consistently within an XML document. What if you need the second occurrence of `Address`, not the first? Associated with *XSL Transformations* (*XSLT*) is the *XML Path Language*, also called *Xpath*, which permits the location of any part of the tree branch path of an XML document hierarchy to be specified. Figure 4.3 gives an example of this capability, to find and select `Database/People/Person`.

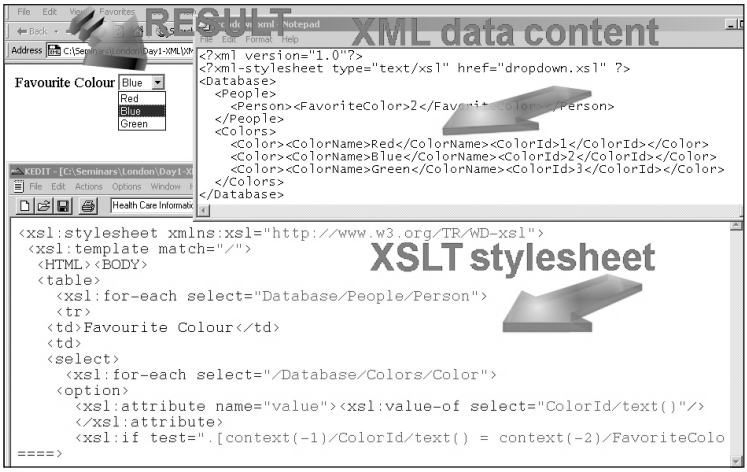

Figure 4.3
How a style sheet works to display XML as HTML.

A transformation using XSLT needs to address specific components in an XML document, and Xpath provides that ability. It works like a pair of programming tweezers to find and return the exact piece of an XML document desired. XSLT can also change the hierarchical structure of an XML document using a set of predefined XSLT syntax rules that dynamically inspect and traverse the document structure.[26] [27] [28]

Using either CSS or XSL style sheet references with an XML document provides a way to visually display XML content to end users, or to morph XML documents for input processing by business application software.[29]

Building XML Messages from Processes to Data

This section looks at the process for building business messages with XML. As we discussed earlier in this chapter, XML lets trading partners define their own elements and tags, taking advantage of XML's extensible nature—the *X* in XML. But XML messages also represent the structure of those elements, following their prescribed relationships in the hierarchy. The message schema DTD captures the names of the

elements as represented by the tags and their hierarchical structure. Messages exchanged among trading partners therefore must represent the rules and practices of a business or industry, as captured in the schema DTD.

For example, in Chapter 3, "ebXML at Work," the Marathoner running store case study points out how retailer and manufacturers can exchange product identifiers and precise inventory levels, so that manufacturers can compare inventory levels to predefined reorder points and decide whether they need to ship more product. Before any of these exchanges can happen, however, the retailer and the manufacturers—or, better yet, the entire industry—need to agree on common terminology and structure of the messages. With this common set of rules, shoe manufacturers and retailers can use the same basic set of messages, which promotes the use of packaged software and makes it possible for the parties to develop their systems faster and for less money.

We call this common set of rules a *data model* because, like a schematic drawing, it offers a skeleton view of the messages, specifies the order of the elements in a message, and shows how the various elements relate to one another in a hierarchy. The term comes from the database world, where database design needs to meet the users' business requirements as efficiently as possible, yet still allow for future growth. The *logical model* defines the information fields and their relationships in a database (much like a schema DTD in an XML message), while the *physical model* details field sizes and datatypes, such as alphanumeric or date formats.[30] In fact, defining an XML schema of information is analogous to creating traditional row-and-column layouts for a database design system.

The XML syntax is not just about interpreting the content. The business process is a vital component of the content and is helped along by XML.

Determine Processes

As shown in the case studies in Chapter 3, the parties identify business processes or actions taken by the companies to achieve their business goals. For example, the travel agency case proposes a process to decide on a tour package. This process has contingencies built in for continued bids and best-and-final offers if the customers don't want to accept one of the first offers. By working out these larger processes, the trading partners can agree on the overall conduct of the business, before trying to determine the individual messages.

A tool called *use cases* can help identify these processes. Use cases describe scenarios in which users interact with each other and the systems under development. Each scenario describes the accomplishment of a specific task or achievement of a goal. They also identify the players, steps in the process, and the messages or even the data exchanged. By describing these situations in a storytelling mode, use cases often uncover the processes underlying business practices.[31]

By working out larger processes, trading partners can agree on the overall conduct of the business, before trying to determine the individual messages.

One of the ebXML development activities involves identifying similarities in business processes across industries. While each industry has its own language and culture, using these common processes helps speed the work and improves the chances for interoperability among industries.[32]

Determine Message Flows

Each process contains a set of individual messages exchanged among the trading partners. In Chapter 3, the running store case listed a series of messages in the process of reporting inventory levels and replenishing the stock:

- Periodic inventory report sent from the store to the manufacturer
- Ship notice sent from the manufacturer to the store with the shipment details
- Receiving report sent from the store to the manufacturer once inventory is accepted

Industries defining their processes can identify the individual messages contained in those processes, as well as how and when the companies send and receive the messages. These messages may resemble EDI transaction sets (see Chapter 5, "The Road Toward ebXML," for a discussion of EDI), as in the running store case, or look nothing like EDI transactions, as in the travel agency case.

Identify Data in the Messages

Once industries identify the messages, they next need to identify the sets of business data that go into those messages. Industry organizations that have previously developed EDI transactions can use this work as the basis for identifying data for XML messages. Newer business processes must rely on information analysis between companies to determine the content required, often replacing older, paper-based documents. But the objective is to improve the way companies do business—not necessarily to follow the current EDI transactions or old paper-process documents. Industries sometimes use this exercise to test traditional assumptions and practices, which can cut out captured or exchanged data that's no longer needed. On the other hand, this process can generate *more* pieces of data needed by trading partners to meet their business requirements.

When applying this process to XML, industry groups develop XML vocabularies that put these groups of data into definable messages, also identifying the structure of the data in the messages. To aid understanding and reuse, the XML structure should link related and most-used pieces of information together as logical blocks. The messages thus embody the rules and practices of doing business in a particular industry, defined in terms of XML. In this way, industries can design common groups of data with common structures as industry-wide rules for processing XML messages.

XML vocabularies can represent more than vertical industries. Vocabularies can also define business functions found in multiple industries, or entire frameworks that provide interoperability across industries and functions. One of these frameworks is ebXML itself, which provides the underpinning for global business, not just an industry sector.[33]

Business Schema DTDs

As discussed earlier, DTDs, as specified for XML, contain the rules for both constructing and structurally validating XML messages. We'll now describe schemas in more detail to give you an understanding of how this key piece of the XML technology is used to enable consistent electronic business.

XML vocabularies can define business functions found in multiple industries, or entire frameworks that provide interoperability across industries and functions.

DTDs assemble information into *elements* with connected attributes. Elements are the basic building blocks of XML messages, and therefore the basic components of DTDs. Elements can contain other elements expressed in a hierarchy (*compound elements*), or they can stand alone as simple containers for character data. Compound elements for parent/child blocks can be referenced together. When the modeling process identifies the data in proposed XML messages, most of these data items will become elements, identified as such in DTDs. In XML messages themselves, elements are marked up as tags within the now familiar angle brackets (<>). Element definitions can indicate the frequency with which the elements occur—once or more than once—and whether they're required or optional.[34]

Then *attributes* provide additional description or qualification for elements. Using the language metaphor often applied to XML, one can think of elements as nouns and attributes as adjectives. The XML document example presented earlier and the following DTD fragment identify the `PostalCode` as an element, with the `codetype` and its use as an attribute of that element:

```
<PostalCode codetype='ZIP'>96045
</PostalCode>
```

```
<!— DTD definition for element and
attribute —>

<!ELEMENT PostalCode (#PCDATA) >
<!ATTLIST PostalCode
                  codetype CDATA #IMPLIED >
```

With the schema DTD syntax, the attributes also provide a limited form of data typing, which means that they describe the kind of data allowed for that element. Attributes can contain *strings* (character data), enumerated lists, or references to other components in the document called *tokens*.

Enumerated lists restrict the attribute to only permitted character strings. For example, an attribute to identify smoking preferences for hotel reservations would have the following as its enumerated listing: `SMOKING or NONSMOKING`. Attributes can likewise indicate a default response, used routinely unless the customer requests otherwise. Returning to the hotel example, the `NONSMOKING` response could serve as the default, unless the customer specifically requests `SMOKING`.[35] While schema DTD datatyping is deliberately simplistic but thereby more easily understood, the new W3C extended schema datatyping is extensive and sophisticated.[36]

The Entity Referencing System

Entities are rather misnamed. They're really aliases or substitution strings, intended to identify the reusable objects in a schema DTD, providing handy shortcuts and helping to ensure consistency in the rules expressed by the DTD. These reusable objects can consist of text strings, such as legal boilerplate, or more complex data element and attribute combinations, defined in advance and recalled when needed. Entities can be internal to the DTD or stored as fragments externally.[37]

Entities also help when placing a character inside a character data or CDATA section of an XML document that would cause confusion with the processing of the XML, such as `&`, `<`, `>`, and `"`.

Consider the telephone number in the following example. The boldfaced element **<Telephone>** is a substitution string declared as an entity in the schema DTD `telephone-usa.xml`, and then included as needed in XML documents based on that DTD. The OpenTravel Alliance uses this technique in its customer profile, which specifies several telephone numbers (customer, emergency contact, travel agency, and so on). The use of this technique simplifies the schema DTD and guarantees that all telephone numbers in the valid messages are defined consistently.[38]

```
<?xml version="1.0"?>
<!DOCTYPE Cust.Telephone SYSTEM
'http://xml.org/telephone-usa.xml' []>
<Cust.Telephone PhoneTech="Voice"
PhoneUse="Home">
    < Telephone CountryAccessCode="1">
        < Phone.AreaCityCode>703
        </Phone.AreaCityCode>
        < Phone.Number>555-9999
        </Phone.Number>
    </ Telephone>
</ Cust.Telephone>
```

Example of Building a Data Model and XML Equivalent

Using a traveler's customer profile, we can show an example of a DTD and how it helps build and validate an XML message.

Table 4.1 shows the pieces of information in a scaled-down traveler profile database, showing three levels in the data hierarchy, as well as the content of each level—element, text, or attribute—as well as single/multiple occurrences, requirement indicator, and allowable options.

The control information identifies the creator of the profile (a travel agency, for the purpose of this exercise), whether it's a new record or an update, whether the customer has given permission to share the data in the profile, and a date/time stamp that most systems can generate routinely.

Table 4.1: Traveler Profile Database Structure

Data level 1	Data level 2	Data level 3	Content	Occurs	Required?	Options
Control info			Element	Single	Yes	
	Share permission?		Attribute			Yes No
	Agency		Element	Single	Yes	
		Agency name	Text	Single	Yes	
		Agency ID	Text	Single		
	New/Update		Text	Single	Yes	New Update
	Date-time		Text	Single	Yes	
Traveler ID			Element	Multiple	Yes	
	Traveler name		Element	Single	Yes	
		Title	Text	Multiple		
		Family name	Text	Single	Yes	
		Given names	Text	Multiple		
	Address		Element	Multiple	Yes	
		Address type	Attribute			Mailing Delivery
		Number/street	Text	Single	Yes	
		Room/floor	Text	Multiple		
		City name	Text	Single	Yes	
		Postal code	Text	Single	Yes	
		State/Province	Text	Multiple		
		Country	Text	Single		
	Telephone		Element	Multiple	Yes	
		Telephone use	Attribute			Work Home
		Country access	Text	Single		
		Area/city code	Text	Single	Yes	
		Tel. number	Text	Single	Yes	
	Email		Element	Multiple		
		Email type	Attribute			Work Personal
		Email address	Text	Single		
Form of payment			Element	Multiple	Yes	
	Payment type		Attribute			Credit card Debit card
	Payment detail		Element	Multiple	Yes	
		Card number	Text	Single	Yes	
		Exp. date	Text	Single	Yes	
		Name on card	Text	Single	Yes	

continues

Table 4.1: Continued

Data level 1	Data level 2	Data level 3	Content	Occurs	Required?	Options
Travel preferences			Element	Multiple		
	General		Element	Multiple		
		Smoking section	Text	Single		Smoking Non-smoking
		Meal preferences	Text	Multiple		
		Special needs		Multiple		
	Loyalty programs		Element	Multiple		
		Program type	Attribute			General Airline Hotel Rental car
		Program name	Text	Single		
		Program ID	Text	Single		
	Airline		Element	Multiple		
		Departure airport	Text			
		Seat selection	Text			Aisle Center Window
	Hotel		Element	Multiple		
		City section	Text			Downtown Suburbs Airport
		Room type	Text			Single Double
	Car rental		Element	Multiple		
		Car type	Text			Compact Midsize Full SUV Truck
		Child seat	Text	Single		Yes No

The DTD for this database structure (`Traveler.dtd`) is found on this book's web site (`www.ebxmlbooks.com`). Please note that this DTD example is meant only to illustrate how a DTD works, and should not be used for normal business messages.

From this database structure, a travel agency wants to create a traveler profile record for a traveler, with the following specific data and preferences:

Administrative control data

- Agency name: GoGo Travel
- Agency ID code: ZZY98234
- Purpose of record: new
- Date/time: 21 June 2001, 3:55 pm
- Permission to share data in profile? No

Traveler identification

- Traveler's name: Ms. Phoebe P. Peabody-Beebe
- Address (delivery): 312 Sycamore St., Buffalo, NY 14204
- Telephone (work): 716-555-9999
- Email: Phoebe@PeabodyBeebe.com

Payment data

- Type of payment: Credit card
- Card number: 0000111122223333
- Expiration date: 12/2002
- Name on card: Phoebe P Peabody-Beebe

Preferences

- Nonsmoking
- Meal type:Vegetarian
- Loyalty program—airlines: US Airways, no. 24680
- Loyalty program—car rental: National Car Rental, no. 54321
- Loyalty program—general: AmEx Membership Miles, no. 09876
- Departure airport (IATA code): BUF
- Airline seat preference: Aisle
- Hotel, city section preference: downtown
- Hotel room preference: single
- Car type preference: Compact

Listing 4.4 gives a validated XML document for these entries based on the rules presented in `Traveler.dtd`.

Listing 4.4 Sample XML Document Based on Traveler.dtd

```
<Traveler>
  <Control>
        <Agency>
              <AgencyName>Go-Go Travel
              </AgencyName>
              <AgencyID>ZZY98234</AgencyID>
        </Agency>
        <Purpose>New</Purpose>
        <DateTime>20010621t15:55:00</DateTime>
  </Control>
  <TravelerID Share="No">
        <TravelerName>
              <Title>Ms</Title>
              <Family>Peabody-Beebe</Family>
              <Given>Phoebe</Given>
              <Given>P.</Given>
        </TravelerName>
        <Address AddressType="Deliver">
              <NumberStreet>312 Sycamore St
              </NumberStreet>
              <City>Buffalo</City>
              <PostalCode>14204</PostalCode>
              <StateProv>NY</StateProv>
        </Address>
        <Telephone PhoneUse="Work">
              <AreaCity>716</AreaCity>
              <PhoneNumber>555-9999
              </PhoneNumber>
        </Telephone>
        <Email>
              <EmailAddress>
               Phoebe@PeabodyBeebe.Com
              </EmailAddress>
        </Email>
  </TravelerID>
  <Payment>
        <PayDetail>
           <CardNumber>
            0000111122223333
           </CardNumber>
           <ExpDate>12/2002</ExpDate>
           <NameOnCard>
            Phoebe P Peabody Beebe
           </NameOnCard>
        </PayDetail>
  </Payment>
```

```
  <Preferences>
        <General>
              <Smoking>Non-smoking</Smoking>
              <MealPref>Vegetarian</MealPref>
        </General>
        <Loyalty LoyalType="Airline">
              <LoyalName>US Airways
              </LoyalName>
              <LoyalID>24680</LoyalID>
        </Loyalty>
        <Loyalty LoyalType="Car Rental">
              <LoyalName>National Car
               Rental</LoyalName>
              <LoyalID>54321</LoyalID>
        </Loyalty>
        <Loyalty LoyalType="General">
              <LoyalName>Amex Member
               Miles</LoyalName>
              <LoyalID>09876</LoyalID>
        </Loyalty>
        <Airline>
              <DepartAirport>BUF
              </DepartAirport>
              <SeatSelect>Aisle</SeatSelect>
        </Airline>
        <Hotel>
              <CitySection>Downtown
              </CitySection>
              <RoomType>Single</RoomType>
        </Hotel>
        <CarRent>
              <CarType>Compact</CarType>
        </CarRent>
  </Preferences>
</Traveler>
```

This message referencing the `Traveler.dtd` contains all of the required data, uses tags that match the element names in the DTD, presents the elements and tags in the order prescribed by the DTD, and therefore conforms as a valid structure to that DTD. Notice that the example doesn't have any data for child seat preferences listed under the XML car rentals section, but does have three different loyalty programs listed. The rules expressed in the DTD allow for such variations. However, if a message left out the traveler's name, a validating parser would return an error message accordingly.

XML Schema

The generic name for DTDs is *schemas*, a term borrowed from the database world. DTDs represent data only in a hierarchy, which works fine for documentation; remember that the W3C borrowed DTDs from SGML, designed for electronic documentation and the predecessor to XML.

However, many business databases use other kinds of structures—such as relational databases or object-oriented classes and properties—some of which don't always lend themselves to a hierarchical model. In some cases, particularly when working with a simple data structure, data architects have been able to adapt object-oriented structures or relational data models to the kind of hierarchies represented in DTDs. But business doesn't always deal a simple hand, and technologists need more robust and flexible tools than the DTD to be prepared for these more complex conditions.

The W3C has developed *XML Schema*, a major enhancement to XML that offers extended tools for representing information structures and objects, as well as providing extended datatypes beyond those in DTDs. In May 2001, XML Schema reached full recommendation status.[39]

Software and systems supporting XML Schema will need to resist the temptation to cover all of the bells and whistles, since they build in more complexity and cost than is needed.

XML Schema provides more power for defining the structure, content, and semantics of XML documents. The W3C specifications document has three parts:

- Methods for describing the structure of data
- Definition of datatypes
- A primer, explaining its features[40]

The first part of the specification deals with structures, documenting the meaning, use, and relationships of the components of an XML document, such as elements, attributes, and entities. It provides the rules for validating XML documents, based on the rules described in the schemas. It also allows for referencing partial or multiple schemas, thus providing a great deal more flexibility and power than DTDs.[41]

The second part of XML Schema covers datatypes and addresses the need for defining more kinds of data in the rules used to validate XML documents. This part of the specification identifies a group of basic (or *primitive*) datatypes such as strings, integers, dates, and sequences. The specification describes features of a datatype system, including acceptable ranges of values and valid representations of the data (such as whole numbers or scientific notation).

The specification identifies datatypes derived from those built into the basic XML recommendations, such as character data (CDATA), tokens, and entities. And it defines various components of datatypes to allow for the development of unanticipated datatypes.[42]

This greater flexibility comes with a price, however. While it's tempting to use many of these new features, many business applications require just a few of them at any time. For example, being able to validate dates and times will be a significant addition to XML's ability to support business. Few businesses, however, will need the ability to create entirely new datatypes. Software and systems supporting XML Schema will need to resist the temptation to cover all of the bells and whistles, since they build in more complexity and cost than is needed.[43]

As an alternative, work on *RELAX NG* is being developed by an OASIS Technical Committee and eventually for submission to ISO. RELAX NG is designed as a simpler and more accessible approach to providing schema functionality for XML documents.[44]

Other Details

XML Schema incorporates one of the first enhancements to the XML specification, called *XML Namespaces*. With XML Namespaces, schemas can address multiple XML vocabularies in a single document. Namespaces provide for uniqueness in element names by combining the namespace prefix (mapped to a uniform resource identifier, like a web address), and the local part or element or attribute name.[45]

Put simply, XML Namespaces allow different companies or industries to avoid name clashes where they both use the same word with different meanings or contexts, but with the same tag name. An example is the word *stock*, which has at least six possible meanings. An obvious example is using formats such as `billing:address` and `supplier:address` to clarify that *address* is being used in two different contexts.

Is XML Ready for Business?

The major inhibitor to the use of XML for business is its inability to provide for interoperability among the various vocabularies written for exchanging business messages.

While the XML family of technologies goes a long way to build up the features needed for exchanging business messages, XML markup technology by itself can't do all that's needed. The major inhibitor to the use of XML for business is its inability to provide for interoperability among the various vocabularies written for exchanging business messages. The number of these vocabularies is expanding rapidly; a survey in 2000 showed these vocabularies doubling between February and August 2000.[46] Unless a way is found to allow businesses using these vocabularies to understand messages from other vocabularies, the promise of XML as a data exchange technology will go unfulfilled.

To achieve this interoperability, companies using XML need to have a common set of methods with translation among the different industry syntaxes. Also needed is a way of relating the XML messages to overall business processes that give context to the messages and the data contained within them. XML by itself also has no inherent provisions for security and privacy, although the W3C has undertaken important initiatives in these areas, notably with digital signatures and privacy preferences using extensions to the base XML specifications.

At the same time, no solution can just pile on all of these requirements without keeping an eye on the impact it will have on achieving the desired objectives—keeping within the scale and price range of the millions of smaller businesses that so far are left out of the data-exchange experience. This is the challenge laid at the feet of ebXML.

Endnotes

[1] A second edition, issued in October 2000, incorporates error corrections from the original February 1998 version.

[2] "W3C Technical Reports and Publications," World Wide Web Consortium, 3 November 2000, www.w3.org/TR/.

[3] Tim Berners-Lee with Mark Fischetti, *Weaving the Web* (New York: Harper Collins, 1999), pp. 44–45.

[4] We say "more or less recognized" because the leading browser makers have added their own features to HTML for competitive advantage. The Web Standards Project (www.webstandards.org) seeks to reduce or eliminate this variation.

[5] xCBL business language specifications, Version 3.0, January, 2001, www.xCBL.org. The xCBL syntax is derived from SimplEDI, developed originally by a UN/EDIFACT working group.

[6] Charles Goldfarb, "A Brief History of the Development of SGML," (SGML Users Group, 11 June 1990), www.sgmlsource.com/history/sgmlhist.htm.

[7] Berners-Lee, p. 4.

[8] Berners-Lee, pp. 16–17.

[9] Berners-Lee, pages 41–42.

[10] "Domain Stats", 4 November 2000, www.whois.net/, and Netcraft, www.netcraft.com/?host=codes.

[11] "1.1, Origin and Goals," Extensible Markup Language (XML) 1.0, W3C Recommendation 10 February 1998, www.w3.org/TR/1998/REC-xml-19980210.

[12] "2.8, Prolog and Document Type Declaration," Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation 6 October 2000, www.w3.org/TR/2000/REC-xml-20001006.

[13] The DOM applies to more than XML documents. The discussion here focuses only on the DOM's XML aspects.

[14] "Document Object Model Activity Statement," World Wide Web Consortium, 25 June 2001, www.w3.org/DOM/Activity.

[15] David Megginson, "SAX: History and Contributors" (undated), www.megginson.com/SAX/SAX1/history.html.

[16] David Megginson, "What is an Event-Based Interface?" (undated), www.megginson.com/SAX/event.html.

[17] "The UNICODE Standard: A Technical Introduction," UNICODE Consortium, December 2000, www.UNICODE.org/UNICODE/standard/principles.html.

[18] Note that XML documents don't require 16 bits per character, but can be stored in 8-bit configurations known as *Unicode Transformation Format* (*UTF-8*).

[19] Java is a registered trademark of Sun Microsystems.

[20] Sun Microsystems, Inc., "About the Java Technology," The Java Tutorial, undated, http://java.sun.com/docs/books/tutorial/getStarted/intro/definition.html.

[21] Jon Bosak, "XML, Java, and the future of the Web," 10 March 1997, www.ibiblio.org/pub/sun-info/standards/xml/why/xmlapps.htm.

[22] Matthew Fuchs, "Why XML Is Meant for Java," *Web Techniques*, July 1999, www.webtechniques/1999/06/fuchs/. Reprinted in XML.com, 16 June 1999, www.xml.com/pub/a/1999/06/fuchs/fuchs.html.

[23] "What are Style Sheets," World Wide Web Consortium, 13 July 2001, www.w3.org/Style/.

[24] Matt Rotter, Charity Kahn, and Paul Anderson, "Get Started With Cascading Style Sheets: How Cascading Style Sheets Work," CNet Networks Inc., 2 November 2000, http://builder.cnet.com/webbuilding/pages/Authoring/CSS/.

[25] Jan Egil Refsnes, "Introduction to XSL," XML101 (undated), www.xml101.com/xsl/xsl_intro.asp.

[26] XML Path Language (XPath) Version 1.0, W3C Recommendation, 16 November 1999, World Wide Web Consortium, www.w3.org/TR/xpath.

[27] XSL 1.0 became a recommendation of W3C in February 2001.

[28] "Extensible Style Sheet Language (XSL)." W3C User Interface Domain, World Wide Web Consortium, 13 July 2001, www.w3.org/Style/XSL/.

[29] A good resource for anything involving XSLT is the web site www.xslt.com.

[30] Stephen Knilans, "Data Modeling Simplified: What Is Data Modeling?" 14 July 2001, Seasoned Software Inc., http://quick-tips.com/local-search/pages/Detailed/2750.html.

[31] Karl E. Wiegers, "Listening to the Customer's Voice," Process Impact, March 1997, www.processimpact.com/articles/usecase.html.

[32] "ebXML Business Process Project Team" (undated), www.ebxml.org/project_teams/business_process/.

[33] Alan Kotok, "Even More Extensible: An Updated Survey of XML Business Vocabularies," XML.com, 2 August 2000, www.xml.com/pub/2000/08/02/ebiz/extensible.html.

[34] Jan Egil Refsnes, "DTD-Elements," XML 101 (undated), www.xml101.com/dtd/dtd_elements.asp.

[35] "3.3, Attribute List Declarations," Extensible Markup Language (XML) 1.0 (Second Edition), World Wide Web Consortium Recommendation, 6 October 2000, www.w3.org/TR/REC-xml.

[36] "XML Schema Part 2: Datatypes," W3C Recommendation, 2 May 2001, www.w3.org/TR/xmlschema-2/.

[37] Jan Egil Refsnes, "DTD-Entities," XML 101 (undated), www.xml101.com/dtd/dtd_entities.asp.

[38] OpenTravel Alliance Message Specifications—version 2001A, www.opentravel.org/opentravel/Docs/OTA_v2001A.pdf.

[39] "XML Schema," World Wide Web Consortium, 25 May 2001, www.w3.org/XML/Schema.

[40] "World Wide Web Consortium Issues XML Schema as a Candidate Recommendation," World Wide Web Consortium, 24 October 2000, www.w3.org/2000/10/xml-schema-pressrelease.

[41] "XML Schema Part 1: Structures," W3C Candidate Recommendation, 24 October 2000, www.w3.org/TR/xmlschema-1/.

[42] "XML Schema Part 2: Datatypes," W3C Candidate Recommendation, 2 May 2001, www.w3.org/TR/2000/CR-xmlschema-2-20001024/datatypes.html.

[43] David R.R. Webber and Alan Kotok, "Less Is More in E-Business: The XML/edi Group," XML.com, 10 November 1999, www.xml.com/pub/a/1999/11/edi/index.html.

[44] See www.oasis-open.org/committees/relax-ng/.

[45] "Namespaces in XML," World Wide Web Consortium, 14 January 1999, www.w3.org/TR/REC-xml-names/.

[46] Alan Kotok, "Even More Extensible—An updated survey of XML business vocabularies," XML.com, 2 August 2000, www.xml.com/pub/2000/08/02/ebiz/extensible.html.