POET Content Management Suite

- Information life cycle
- Object data base
- Content management system components

Chapter

27

Thanks to the Web, the life cycle of information is shortening even as the volume of information delivered is growing dramatically. POET Software Corporation, `http://www.poet.com`, the sponsors of this chapter, believe that content management is the key to meeting this challenge, and that object database technology is the key to content management.

Information in general, and written content in particular, has a life cycle. Managing it has always been a challenge, and the challenge is increasing.

27.1 | Managing the information life cycle

The information content life cycle varies widely, depending on the type of information being presented. For example, a newspaper might be published on a daily basis. The information it contains is extremely time sensitive in the first iteration of its life cycle as breaking news.

The content of a newspaper becomes important again as it is archived for historical purposes and placed in knowledge bases for research purposes. In a newspaper's second cycle, the presentation of information moves from paper-based delivery to electronic delivery. The information may be indexed and cross-referenced as an aid to research that never existed in its

paper-based form. The actual content of the newspaper is not changed, but the information might be reused in a condensed or edited future reprint.

A medical dictionary has yet another life cycle. Its information needs to be timely, but is not as time-sensitive as a newspaper. Medical terms change, but traditionally this type of publication has a fairly long life cycle. Many terms need to change before a reader feels the need to reinvest in a newer edition of the dictionary. A typical print life cycle for such a dictionary can be as long as four years. It is very expensive to reprint and manage such a large body of information.

27.1.1 *Changes to the information life cycle*

Information life cycles change over time as new media and technologies change the business model of information delivery. The magazine industry is a good example of how technology can affect the delivery and creation of information.

Magazines like *Time* and *Newsweek* were revolutionary in the way they created an entirely new format for disseminating the news. Technology helped make it possible to print magazines on a weekly basis. The proliferation of special interest magazines such as *Men's Health*, and *Outside Magazine* are a direct result of the capabilities of desktop publishing and more efficient national distribution channels.

27.1.2 *The World Wide Web has changed the rules*

Very few changes in the history of information delivery have been as profound as the creation and rise of the World Wide Web. The Web has, in a very short period of time, forced us to radically rethink the information life cycle.

Essentially, there are now two different – yet complementary – life cycles, one for print delivery and one for electronic. The changes that the Web has spawned go beyond just life cycle management. Information is no longer presented in a single format: e.g. newspaper, magazine, manual, dictionary, encyclopedia.

Information needs to be able to change its presentation depending on context. Is the information being presented in an electronic or print

medium? Is the information supplemental or being cross-referenced from some other body of information?

For example: In our medical dictionary scenario, medical definitions might be available on-line to insurance claims adjusters who need to look up terms. Or, it might be available in the context of other medical publications like a drug guide.

The print life cycle for a medical dictionary might never be less than 4 years, but the life cycle of the same publication on the World Wide Web or CD-ROM might be far shorter. Web customers would most likely subscribe to this information and would want the most up-to-date information at all times. Most likely the same publication will have more than one distribution medium and a complex life cycle to manage.

Managing today's information requires more flexibility than traditional document management systems can provide. XML provides the appropriate notation for storing and editing information over its entire life cycle. What is needed are new tools and paradigms for managing and enabling XML content.

27.1.3 *Object-oriented components*

The new paradigm for content management with XML is "components". Document components are logical, hierarchical divisions of a document. Through parsing, XML documents are automatically broken down into a component object structure. In a content management system, component objects can be edited, versioned, and shared independently of the document that contains them.

Why are document components more powerful than monolithic documents? By breaking a document down into components, pieces of a single document can be managed as if they were objects. This allows authors to work collaboratively on pieces of a large document while managing the document life cycle through version control.

Document components can be nested inside one another according to the natural structure of the document itself. By reading the document type definition (DTD), a content management system has the necessary information needed to create document components according to user-defined parameters.

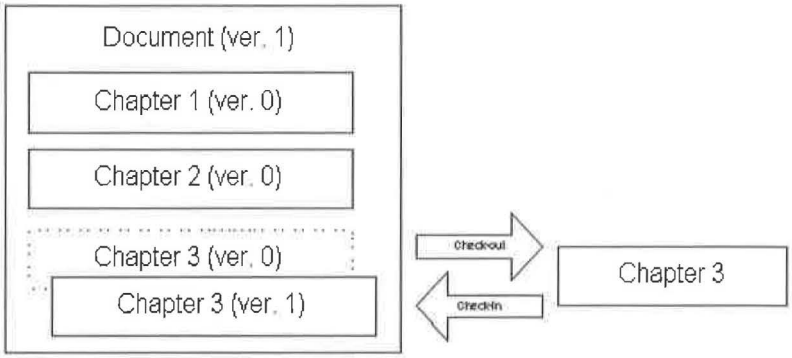

Figure 27-1 Nested document components.

As seen in Figure 27-1, a "document" component can contain "chapter" components. (These components correspond to elements in the XML document.)

A checkout can occur at any level in the hierarchy. Multiple clients can check out different components in the same large document for collaborative authoring.

The component structure of a document looks very much like a table of contents would. Chapters might contains sections and subsections, tables and figures.

Components present the user with a logical view of the document as opposed to a storage-based view of a document. It is not necessary to care how the document is actually stored, whether it be in files or in a database. The most important view of a document is one that reflects the semantic structure of the document itself. Components allow us access to the world of XML structured content in a way never before possible.

27.2 | The *POET Content Management Suite*

One approach to managing document life cycles with content management is the *POET Content Management Suite* (POET CMS).

27.2.1 *POET CMS components*

The architecture and components of *POET CMS* are shown in Figure 27-2.

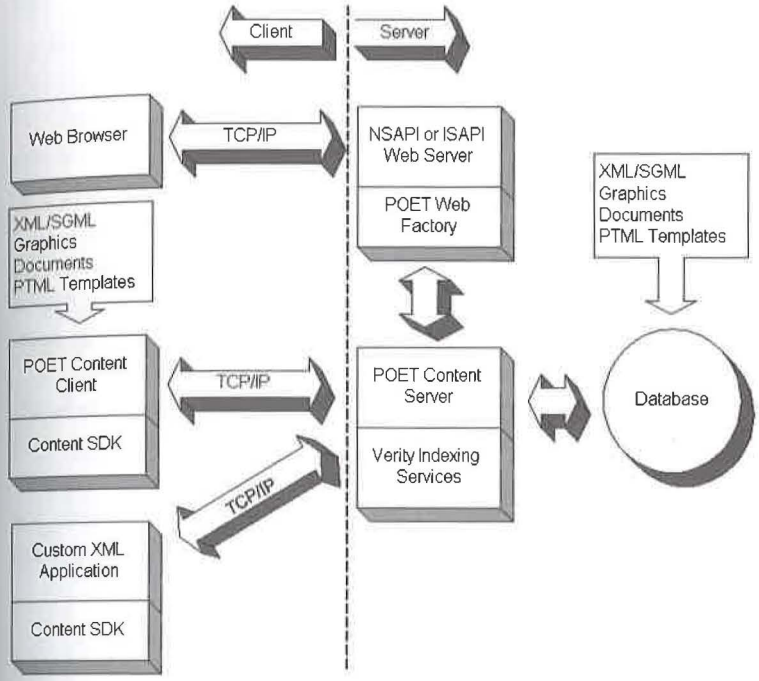

Figure 27-2 *POET Content Management Suite.*

27.2.1.1 POET Content Server

The *POET Content Server* provides the full functionality of the *POET Object Server* database product. However, it is optimized for content access and includes a suite of tools for administering the database.

The product is packaged with a content-enabled version of the *POET Web Factory* server plug-in. It uses server-side templates to dynamically assemble and deliver the latest content to the Web or the corporate intranet. It can deliver HTML, XML, other documents, and graphics, directly from a *POET* database.

27.2.1.2 POET Content Client

The *POET Content Client* provides an immediately deployable solution to managing content in an editorial or delivery environment. The product is a Win95/NT end-user application that provides project management, version control, document component exploration, viewing, querying and component sharing functionality.

27.2.1.3 POET Content SDK

The *POET Content SDK* is a development environment with several application programming interfaces (API).

The *High-level API* provides an interface for performing common tasks such as importing and validating documents, checking components in and out, managing folders and projects, pressing editions and delivering content.

The *Navigation API* provides object-oriented abstractions for traversing, accessing, and querying XML elements, attributes, and components, as well as common graphic and non-XML document files.

The *ActiveX API* provides ready-made tree, list and query controls for embedding in an application, either without programming, by scripting with VBScript and JavaScript, or by interfacing with C++ or Java.

27.2.2 *The POET CMS Architecture*

The *POET Content Management Suite* architecture allows different types of client to access the database at one time. Editors can use the *Content Client* interface while reviewers are viewing the latest draft in Web browsers. Custom applications can run at the same time. A single server can access many different databases at once, and a single client can access many different servers.

27.2.3 *Using POET CMS*

27.2.3.1 Server-side content management

Content management requires control. A database server needs to provide control over concurrent access by multiple clients, management of transactions, user rights, and multi-threaded queries.

Moreover, a content server must be able to address the challenges of structured documents effectively. Fine-grained control of locking and versioning is necessary.

It is tight control of the database that enables flexibility for users.

For example, the *POET Content Server* can operate in editorial and delivery environments concurrently. Users can edit and manage versions on the same system and database that is being used for delivery to the Web or the corporate intranet. Approved editions can be delivered at the same time that new content and versions of the documents are being created.

Moreover, access can be controlled so that different people can access different versions. Web users on the public Internet might only get formally released material, while intranet users might see the most recent version of the information.

27.2.3.2 Client-side editing and viewing

The *POET Content Client* provides a familiar explorer-style interface for managing document components. Documents and components are stored as objects that can be organized into folders, dragged and dropped between projects, checked in and out, viewed, edited, and queried. The product provides the security and power of a traditional version control system with the added power of hierarchical component versioning.

Let's take a closer look at a few of its functions.

Version control

In Figure 27-3 the main window shows the database for a very large document, *Taber's Cyclopedic Medical Dictionary*. In the left-hand window pane is an explorer-style view of the database. Folders and publications can be viewed just as if they were on the file system.

In the case of XML documents, like this one, the document itself is not the smallest unit that can be browsed. Clicking on a document will reveal a new world of document components. This particular document is broken up into the user-defined component levels "MAINENTRY" and "SUBENTRY", which are elements of the dictionary.

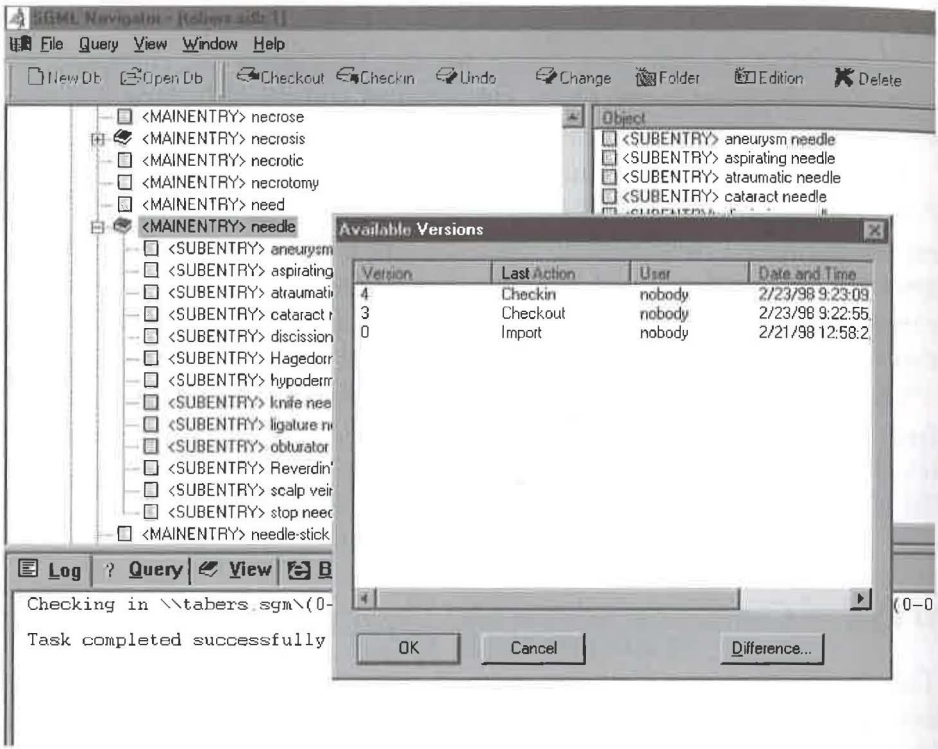

Figure 27-3 Component database with version selection dialog.

Right-clicking on the selected component, in this case the MAINENTRY "needle", pops up a menu from which various actions can be taken (Figure 27-4). Clicking on "Change Version" brings up the "Available Versions" dialog that we saw in Figure 27-3, from which a specific version of the component can be selected. Versions are listed along with the last action performed on them and the user who performed the action.

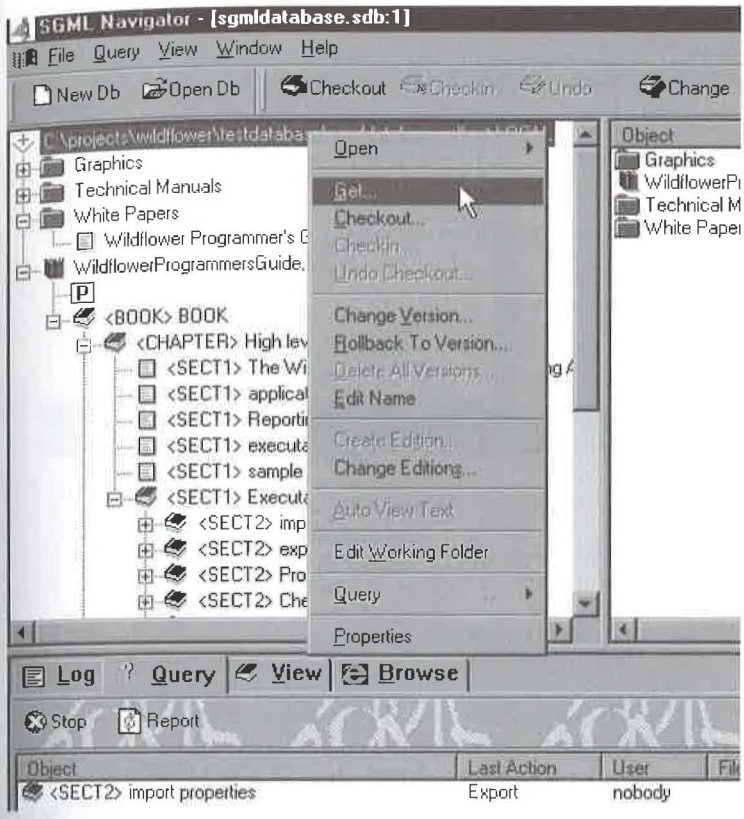

Figure 27-4 Right-click action menu in *POET CMS*.

Checking out a component

The right-click menu offers other actions as well, including the possibility of checking-out the component in order to edit it. In Figure 27-5, the MAINENTRY "neck" in the medical dictionary has been checked out. It and its subcomponents appear in red to indicate that they are checked out and cannot be modified by another user until they are checked back in. check out. All other components are still available for modification by other users.

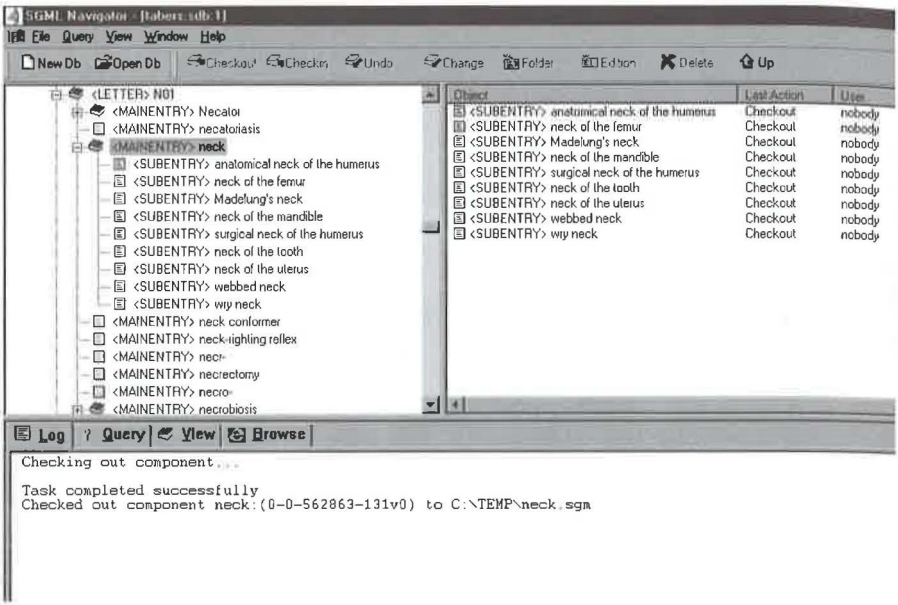

Figure 27-5 Checked-out components.

Sharing a component

A component can be shared by holding down the CTRL-SHIFT keys and dragging it into another project folder. All changes made to the component will be reflected in both projects.

In Figure 27-6 the component "Executable unit properties" is being shared with the "Technical Manuals" folder, as indicated by the curved arrow in the cursor.

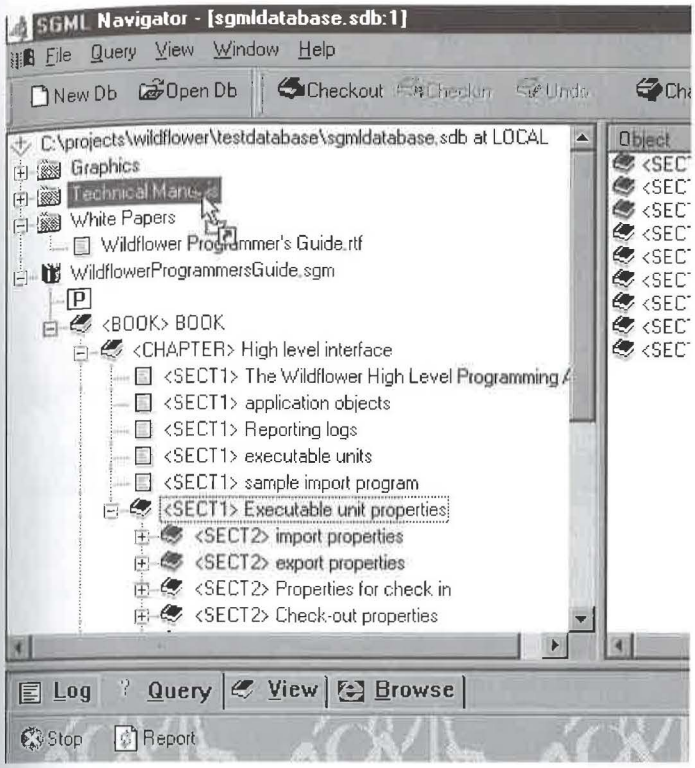

Figure 27-6 Sharing a component.

Viewing a document

An object, such as an XML (or full SGML) component, graphic object, or word processing document, can be viewed by dragging it onto the browser toolbar. Depending on the object type, the browser will either view it directly or launch an external program to view it.

In Figure 27-7 we see the *Softquad Panorama* viewer in the bottom window, viewing the component "microscope".

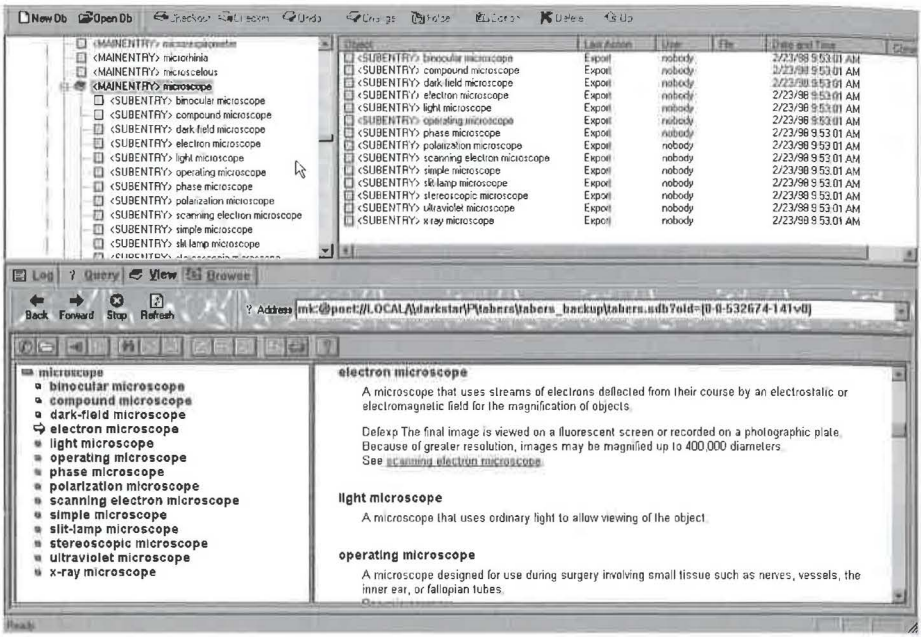

Figure 27-7 Viewing a component.

Full-text search

POET CMS supports several forms of searching. Figure 27-8 illustrates the results of a full text search for the word "migraine". The "Query" tab of the main window lists all of the components that contain the word.

The search was conducted by means of a dialog box, in which the search string was entered. The dialog offers the option of restricting the search to the current version of components that contain the word, or returning all versions that contain it.

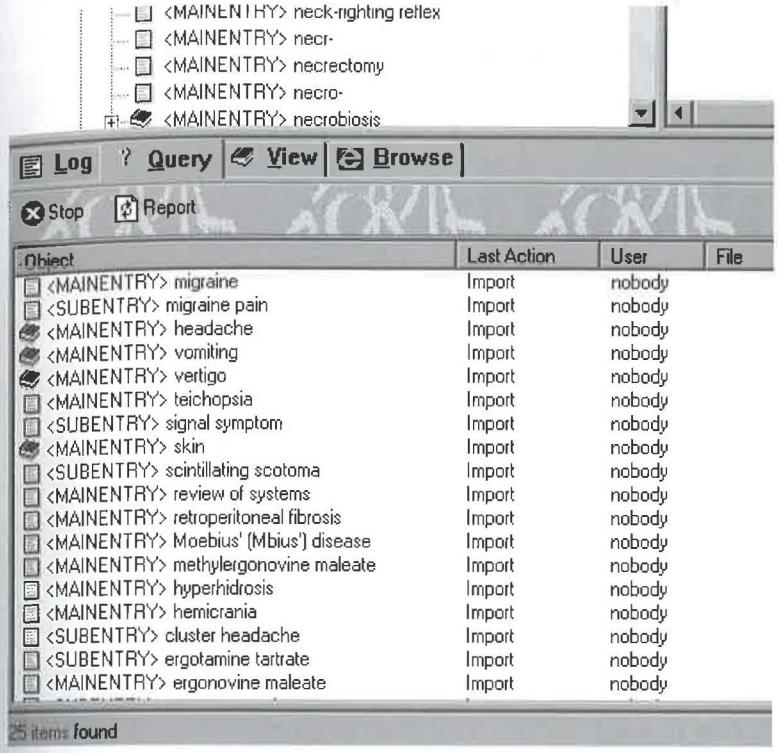

Figure 27-8 Result of querying for components containing "migraine".

The query results can be can be used as a basis for checkout, viewing, and editing by right-clicking on a component as if it were listed in the main pane of the window.

Analysis *The decreasing life cycles of information brought about by the World Wide Web make a persuasive argument for XML and content management. It is true that content management was originally conceived for large-scale technical publishing. However, it appears likely to play a significant role in middle-tier Web applications as well, particularly as the value of maintaining persistent metadata on the middle tier becomes better appreciated.*

HoTMetaL Application Server

- Middle-tier server tool
- Familiar HTML base
- WYSIWYG development environment
- Free trial version on CD-ROM

Chapter

28

The "leading edge" of the Web today is actually in the middle. That is, on the middle tier, where servers gather data in any format from any source location and serve them up in XML. The friendly folks at SoftQuad Inc., `http://www.sq.com`, believe that this highly technical domain could be a lot friendlier, and have sponsored this chapter to explain how.

The landscape of the Web is rapidly changing. No longer are static Web pages enough to give an organization the business edge it requires in today's highly competitive world. No matter how well designed and link-intensive your Web page is, it won't make the grade unless it can provide the interactive services that the new generation of "Nintendo" trained, computer-savvy, business-oriented Web users now demand.

Remaining competitive on the Web can be overwhelming for organizations that have struggled just to create today's static Web pages. How can they cope with emerging requirements to provide a "personal experience" on their Web site? This is especially difficult if the organization lacks the technical resources it takes to add interactivity to their site.

28.1 | Dynamic descriptive markup

The solution lies with a new generation of XML-based software for the middle tier that relies on descriptive markup, rather than procedural scripts

written in JavaScript, VBscript or C++. Products like SoftQuad's HoTMetaL Application Server (*HoTMetaL APPS*) seek to introduce new functionality painlessly by leveraging existing user expertise in HTML.

With *HoTMetaL APPS*, for example, Web development is done directly in XML markup, using an extended set of HTML tags. Because the source markup is XML, the product can provide a "what-you-see-is-what-you-get" (WYSIWYG) view of the page as well as the tagged view. (Figure 28-1) Developers new to dynamic Web pages may find it easier to adapt to this familiar paradigm than to deal with scripting code.

Figure 28-1 WYSIWYG view of a dynamic Web page in *HoTMetaL APPS*.

28.2 | How *HoTMetaL APPS* works

The product has two components. The first is the development interface. The second is the application server itself, which runs on several UNIX platforms as well as IIS, Windows NT, and Apache.

In order to allow the user to work in the familiar HTML Web page development paradigm, the development interface is based on the vendor's HoTMetaL Pro HTML editor, with additional features known as the *Power Pack*. The latter includes support for special "MV" tags (described below), plus documentation and sample applications.

Also included is the *HoTMetaL Personal Server*, which runs under *Windows 95* or *Windows NT. HoTMetaL Personal Server* is a completely self-contained environment that can be used for staging applications before they go to the Web.

A free trial version of *HoTMetaL APPS* can be found on the CD-ROM accompanying this book. It works with existing Web browsers and, for those who have scripting language skills, JavaScript and DHTML can be integrated into its Web pages as well.

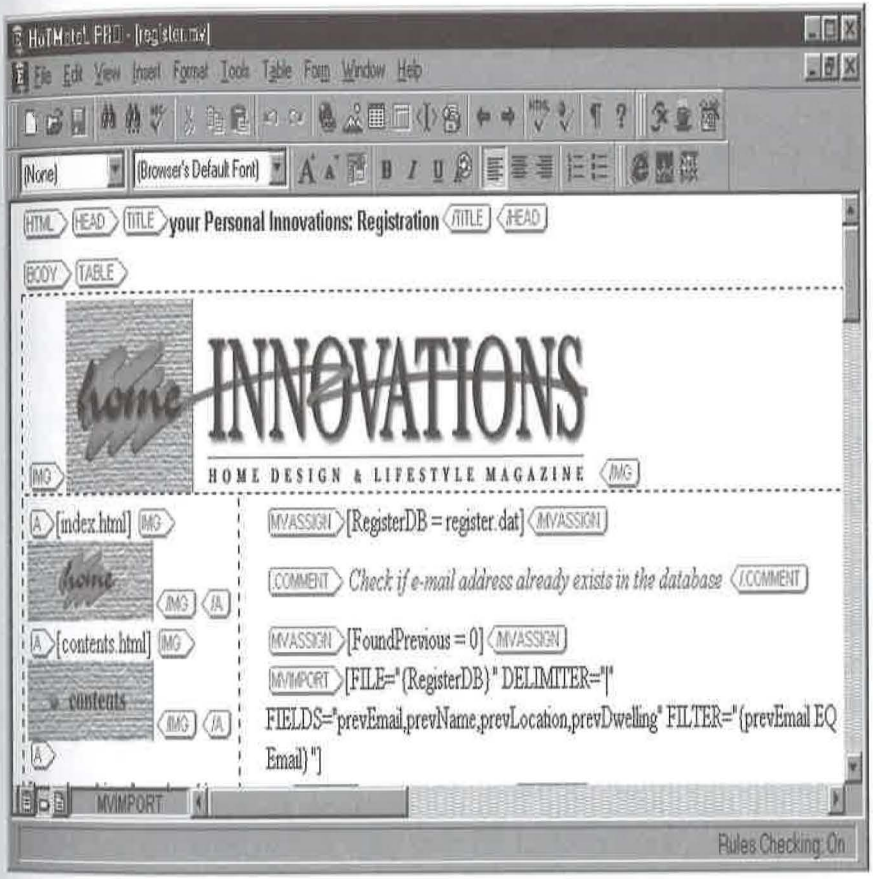

Figure 28-2 *HoTMetaL APPS* development interface.

28.2.1 *Middle-tier server tags*

The major difference between an HTML editor and the *HoTMetaL APPS* development interface (Figure 28-2) is support for a set of special "Middle-tier serVer", or "MV" tags. Normally, XML and HTML tags such as "H1" and "TABLE" identify elements of those types and are rendered by the client browser.

MV tags, however, though resembling normal tags in syntax and method of use, are interpreted by *HoTMetaL APPS* and are never seen by the browser. Instead, they dynamically control the generation of the page that the browser eventually receives.

28.2.1.1 Data access tags

One group of MV tags enables the dynamic inclusion of data from databases and other sources. They include:

MVOPEN

Opens a specified data source.

MVINPUT

Dynamically pulls data from a specified source into the Web page.

MVCLOSE

Closes a specified data source.

28.2.1.2 Conditional logic tags

This set of MV tags enables the Web site developer to perform conditional processing. That is, the generated browser page can depend on conditions occuring in the data or other changing information. Some examples are:

MVIF

Causes following text to be processed if a specified statement is true.

MVELSE
Causes alternative text to be processed when the MVIF statement is false.

Note that the text that is processed because of the above tags can include both MV tags to be processed on the server and ordinary text to be included in the generated page.

28.2.2 *Guided construction of dynamic pages*

When creating dynamic Web pages with a procedural scripting language, the developer must know the language commands and parameters and where they can be used. In contrast, markup-based development is governed by an XML document type definition that allows entry of tags only where appropriate.

In *HoTMetaL APPS*, for example, MV tags are entered in the Web page via the "add markup" pull-down menu in the same manner as normal tags can be entered. The available tag selection dynamically changes based on document context, so adding tags in the right spot is automatic.

Moreover, since the programming parameters for MV tags are specified as attributes, the *HoTMetaL APPS Attribute Inspector* can guide the developer in their correct use. For example, in Figure 28-3, the *Attribute Inspector* indicates required parameters in boldface.

Figure 28-3 The *Attribute Inspector*

28.3 | Functionality can be friendly

The business requirement to engage the customer in the Web site experience is growing each day. No longer is it enough simply to present informa-

Figure 28-4 Dynamic XML-based Web Pages

tion to those who visit your site. You must include them in the business process (Figure 28-4).

XML is the enabling technology that will make such interaction pervasive. And there is no need for it to be limited to those with large budgets or huge technical staffs. Products like HoTMetaL Application Server offer the non-programmer the potential to build highly interactive Web sites in a tag-based environment, where today's Webmasters are already comfortable and proficient.

Junglee Virtual DBMS

- Middle-tier server tool
- Virtual database technology
- Wrappers and extractors

Chapter

29

One view of the middle tier is that it should appear to be a single relational database, one whose data can be delivered by packaging it as XML documents. This chapter is sponsored by Junglee Corporation, `http://www.junglee.com`, and was prepared by Anand Rajaraman, STS Prasad, and Debra Knodel.

With the explosive growth of the Internet, corporate intranets, and the World Wide Web, a vast resource of data is now available. This data is scattered across Web sites, file systems, database systems and legacy applications. Writing applications to query and combine data from this wide variety of sources is a complex and difficult process.

29.1 | Why virtual database technology?

All of the middle-tier approaches described in this book can integrate data from multiple sources and deliver it to the client in the form of XML documents. Some users, however, may find a compelling logic in going even further, as Junglee Corporation has.

Their *Virtual Database Management System* (VDBMS) is designed to make the World Wide Web and other external data sources behave as a single relational database – as part of your enterprise or Web application infrastructure. The relational database view makes it possible to execute

powerful queries using the industry-standard *Structured Query Language* (SQL), and results can be presented as relational tables or XML documents, as required by the application.

The VDBMS leverages the power of XML to deliver results in a form that can be manipulated by the browser without round trips to the Web server. For example, XML allows browser-side sorting and filtering of data, and presentation of the data to suit specific user preferences based on style sheets. Figure 29-1 illustrates the core functionality of the VDBMS.

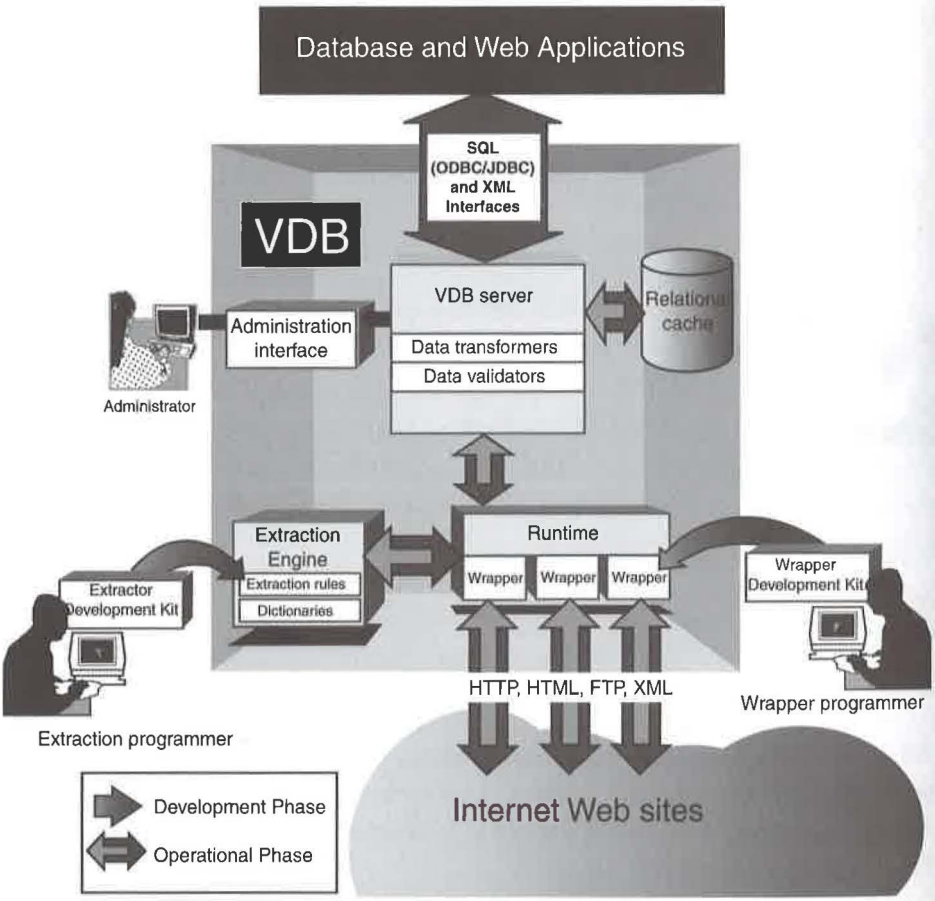

Figure 29-1 The *Virtual Database Management System* (VDBMS).

29.2 | How the VDBMS works

The *Virtual Database Management System* is an integrated Java-based system that enables you to develop and operate a "virtual database", a relational view over large collections of Web sites and other data sources. Database and Internet applications can access a virtual database using SQL, through ODBC and JDBC interfaces.

The VDBMS provides a comprehensive set of tools for transforming the Internet into a database:

- Wrapper Development Kit (WDK)
- Extractor Development Kit (EDK)
- VDB Server and Data Quality Kit
- Administration Interface

29.2.1 *Wrapper Development Kit (WDK)*

Wrappers are Java programs designed to extract data from data sources (such as Web sites) on demand, and present the data in a tabular format. Figure 29-2 shows the wrapper development process.

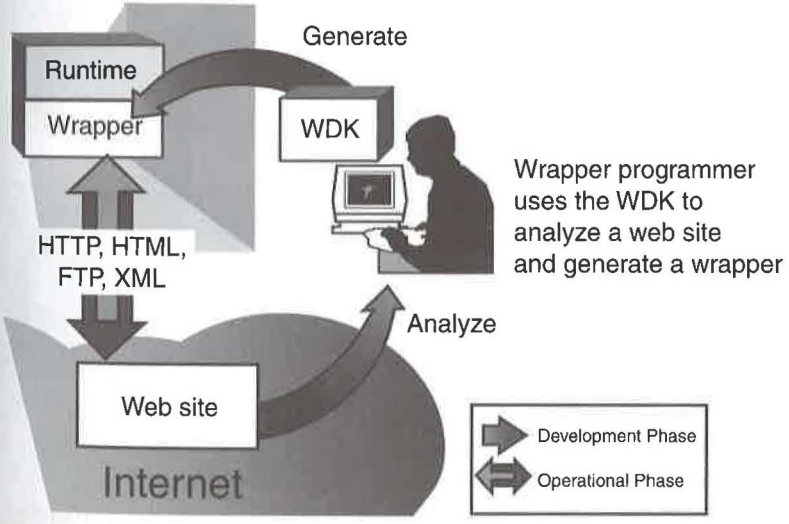

Figure 29-2 The wrapper development process.

The *Wrapper Development Kit* provides *wrapper frameworks*, which are collections of Java classes. Using these frameworks, the wrapper programmer can easily customize data retrieval. The WDK provides a high-level abstraction for network access, HTML parsing, pattern-matching and relational data output, allowing the programmer to focus on the core issues of data manipulation.

29.2.2 *The Extractor Development Kit (EDK)*

Data integration often involves extracting structured data from "unstructured" text; that is, text whose computer representation intermixes style information with the abstract data. To do this, a wrapper uses a program called an extraction rule, a set of rules and dictionaries created by the programmer using the *Extractor Development Kit* (EDK). Figure 29-3 illustrates the extractor development process.

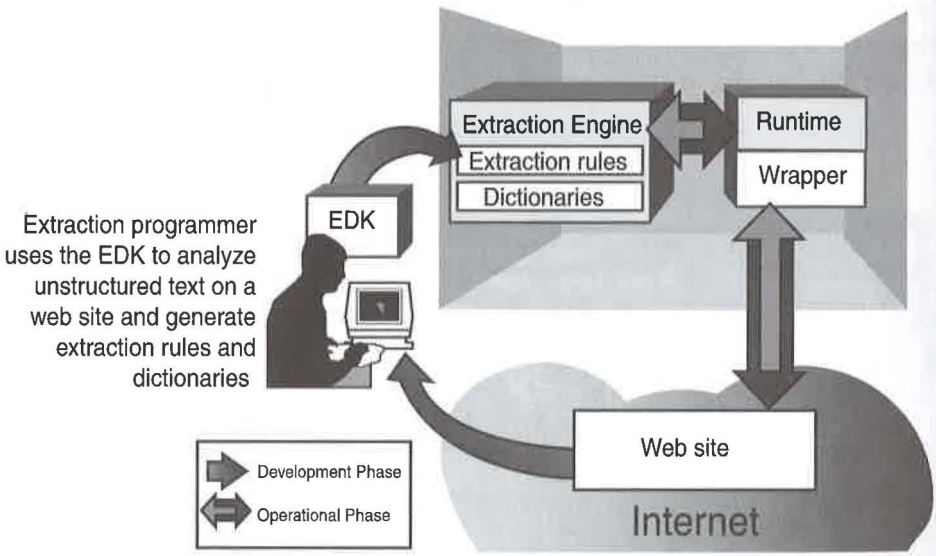

Figure 29-3 The extractor development process

Extraction rules are expressed in a high level language called the Junglee Extraction Language (JEL). JEL allows programmers to specify complex

textual patterns and linguistic structures to identify the context in which specific terms are used. Individual terms are listed in the dictionaries, and can be tagged with flags and values using the EDK compilers. Extraction rules and dictionaries are interpreted by the EDK Extraction Engine.

29.2.3 *VDB Server and Data Quality Kit*

The *VDB Server* combines a collection of wrappers – and the necessary extractors – and presents them as a coherent relational database that can be queried using SQL through JDBC or ODBC. The VDB server can present results as tables or XML documents, as required by the application.

The VDB Server can be configured with a relational cache (see Figure 29-4), to improve query performance over Web data sources. The cache can be preloaded and refreshed depending on the application requirements.

Virtual databases often deal with highly irregular data from sources that are outside the control of the VDB administrator, and are subject to large-scale changes without notice. For this reason data transformation and data validation are key issues.

The *Data Quality Kit* provides the capability to set up data transformers that map attribute values from different sources to a common representation and vocabulary. In addition, data validators can be set up to monitor conditions or enforce constraints at various levels – row, column, or table.

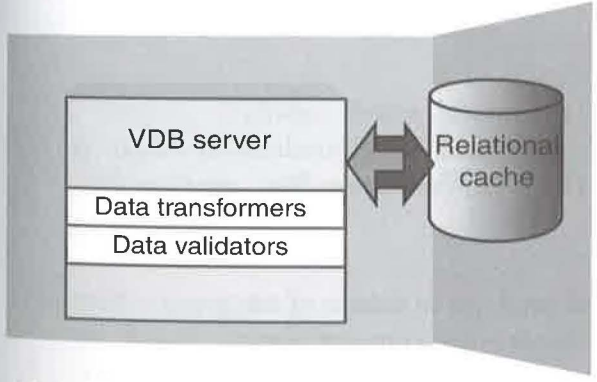

Figure 29-4 The VDB Server

Stability tests are unique to data from wrappers. Since wrappers often export data from Web sites that are subject to change, these tests are a vital first line of defense against corrupt data. These tests compare statistics of the data in a table against historical statistics for that table and report large deviations from historical trends.

29.2.4 *Administration interface*

The administration utility is used in conjunction with the VDB server to register or unregister each data source and the associated wrapper used to access it. Registration enables the data source to become visible as a set of one or more tables in the virtual database.

The registration command permits the system administrator to set up authentication mappings between a VDB server user and the corresponding name to be used when accessing the data source. Registration also allows the system administrator to distribute the work load on the system over a number of workstations on the LAN.

29.3 | Applications of VDB technology

The increasing use of the Internet to deliver information and provide sources leads to innovative applications of VDB technology, based on its capability to normalize disparate Web sites. Two such applications are described in this book.

online recruitment

Job seekers can utilize a powerful search capability through a virtual database of jobs aggregated from hundreds of corporate Web sites. See Chapter 15, "The Washington Post", on page 202.

Web commerce

Online shoppers can find products in dozens of categories – from hundreds of online merchants – and compare features, availability and pricing among merchants. See Chapter 9, "Comparison shopping service Web site", on page 132.

But many more come to mind, including:

corporate procurement
Enable employees to find and compare products and services from approved suppliers.

engineering design databases
Provide engineers with integrated catalogs of design components from manufacturers, searchable by specific attributes.

information management
Integrate content from Web sites, news feeds, digital libraries and document databases to deliver value to the knowledge worker.

Free XML software

- Editors
- Browsers
- Parsers
- ... and More!

Chapter

30

The best things in life are free ... and 55 of them are on our CD-ROM!

Although XML is a new specification, it has a long history of free software for the SGML family of standards (including HyTime and DSSSL) to draw on. As a result, we could be selective about what we included.

The CD-ROM has over 55 free software titles, collected for The XML Handbook by Lars Marius Garshol. There is other useful software as well, as we'll see.

30.1 | What do we mean by "free"?

To meet our requirements for genuinely free software, a product must:

- Allow the user to do useful processing of the user's own documents.
- Have no time limit on its use.

These requirements are easily met by software that is written only for free distribution. In the case of shareware and "lite" versions of commercial soft-

ware, though, the free version might have less function than the full-fledged offering, or it may have a restriction that it is only for personal use, or some similar legal constraint to prevent commercial abuse. It can be shareware that gently nags, as long as it doesn't expire.

Many of the programs are available under the GNU General Public License. That license meets all of our criteria and is available on the CD.

We have included executable files for both Win32 and Macintosh when available. In the Unix world, we only provide binaries for Linux, not every variant under the Sun.

Anything else doesn't meet our definition of free, but free isn't everything.

Demos, time-limited "trialware", and software that won't save the output or is otherwise unsuitable for production work, can still be useful. It can save you hours of product research and phone calls, and provide definitive answers about whether a product meets your requirements.

Our sponsors have provided lots of software in the CD-ROM Sponsor Showcase. Some of their goodies will surprise and delight you, so be sure to take a look.

And remember that the product descriptions only apply to the release being discussed. If you've tried out the software on the CD-ROM and you like it, be sure to visit the author's web site for the latest version.

30.2 | The best XML free software

30.2.1 *Parsers and engines*

30.2.1.1 Xlink engines

xmllink, 17.Mar.98 release

Bert Bos

Information on web:
http://www.w3.org/XML/notes.html
Software on web:
http://www.w3.org/XML/xmllink.zip

Software included on CD-ROM (Java):
./freesw/others/xmllib.tar.gz

This is a Java XML parser which also implements Xlink and an experimental proposal for SQL-like typing. There are command-line utilities for showing how links are interpreted and ESIS-like output.

30.2.1.2 XSL engines

Sparse, 28.Feb.98 release

Jeremie Miller

Information on web:
http://www.jeremie.com/Dev/XSL/index.phtml
Software on web:
http://www.jeremie.com/Dev/XSL/sparse.js
Software included on CD-ROM (JavaScript):
./freesw/others/sparse.zip

This is an XSL processor written in JavaScript. According to the author, it "is still of alpha quality and missing most of the features of the XSL Proposal".

The Microsoft XSL Processor, 7.Jan.98 release

Microsoft

Information on web:
http://www.microsoft.com/xml/xsl/msxsl.htm
Software on web:
http://www.microsoft.com/xml/xsl/downloads/msxsl.zip

The Microsoft XSL processor is a command-line utility that can process XSL style sheets and deliver HTML output. This processor can also be used as an ActiveX control from MSIE 4.0. It requires MSIE 4.0.

docproc, 07.Feb.98 release

Sean Russell

Information on web:
http://javalab.uoregon.edu/ser/software/docproc/index.xml
Software on web:
http://javalab.uoregon.edu/ser/software/distributions/docproc_full.jar
Software included on CD-ROM (Java):
./freesw/others/docproc.zip

docproc is a Java servlet that can be used to automatically convert XML documents to HTML and serve them on the web. docproc also comes as an application that can be run separately from any web server. docproc uses XSL, with Pnuts for scripting (Pnuts is similar to ECMAScript). The parser used by docproc is Lark.

xslj, 0.4

Henry Thompson

Information on web:
http://www.ltg.ed.ac.uk/~ht/xslj.html
Software on web:
ftp://ftp.cogsci.ed.ac.uk/pub/XSLJ/
Software included on CD-ROM (Source):
./freesw/xslj/xslj-0.4.tar.gz
Software included on CD-ROM (Win 32):
./freesw/xslj/xslj-0.4-bin-win32.tar.gz

xslj is an XSL processor with support for nearly all of the XSL proposal. It converts the XSL stylesheets to DSSSL stylesheets for further processing with Jade, and so requires Jade to be installed.

30.2.1.3 DSSSL engines

Jade, 1.1 (General SGML/XML tool)

James Clark

Information on web:
http://www.jclark.com/jade/
Software on web:
ftp://ftp.jclark.com/pub/jade/
Software included on CD-ROM (Source):
./freesw/jade/jade1_1.zip
Software included on CD-ROM (Win32):
./freesw/jade/jadew1_1.zip

Jade is James Clark's excellent DSSSL engine, which is really a general SGML tool for conversion from SGML to other SGML DTDs or to output formats like RTF and TeX. Jade can process XML documents and can also output XML.

DAE SDK

Copernican Solutions

Information on web:
http://www.copsol.com/products/dae/
Software on web:
http://www.copsol.com/products/dae/downloadreq.html
Software included on CD-ROM (Java):
./freesw/others/dae_install.class

This is a collection of Java tools from Copernican Solutions, which includes a DSSSL engine, a grove API, a Scheme interpreter (Kawa) and an XML parser.

DAE Server SDK

Copernican Solutions

Information on web:
http://www.copsol.com/products/daeserver/
Software on web:
http://www.copsol.com/products/daeserver/downloadreq.html
Software included on CD-ROM (Java):

./freesw/others/daeserver_install.class

This is an extended version of the DAE SDK that has been integrated with the Jigsaw web server from the W3C.

30.2.1.4 SGML/XML parsers

SGMLSpm, 1.03ii (General SGML/XML tool)

David Megginson

Information on web:
http://home.sprynet.com/sprynet/dmeggins/software.html
Software on web:
http://home.sprynet.com/sprynet/dmeggins/SGMLSpm-1.03ii.tar.gz
Software included on CD-ROM (Perl):
./freesw/others/sgmlspm-1.03ii.tar.gz

SGMLSpm reads ESIS output (from parsers like SP) and offers an event-based interface to the parser. As long as the parser can parse XML this also works for XML.

SP, 1.3 (General SGML/XML tool)

James Clark

Information on web:
http://www.jclark.com/sp/
Software on web:
http://www.jclark.com/sp/howtoget.htm
Software included on CD-ROM (Win32):
./freesw/sp/sp1_3.zip
Software included on CD-ROM (Source):
./freesw/sp/sp-1.3.tar.gz

SP is an SGML/XML parser, and is fast, complete, highly conformant and very stable. SP has been the parser of choice for most of the SGML community for many years and has been embedded in many other applications.

The SP package includes the SX program, which can adapt arbitrary SGML documents to XML automatically.

30.2.1.5 XML parsers

Windows Foundation Classes, Release 34

Sam Blackburn

Information on web:
http://ourworld.compuserve.com/homepages/sam_blackburn/wfc.htm
Software on web:
http://ourworld.compuserve.com/homepages/sam_blackburn/wfc.zip
Software included on CD-ROM (Win32):
./freesw/others/wfc.zip

WFC is a collection of C++ classes for Windows programming. Included are classes to parse and create XML documents.

RXP, beta6

Richard Tobin

Software on web:
ftp://ftp.cogsci.ed.ac.uk/pub/richard/rxp.tar.gz
Software included on CD-ROM (Source):
./freesw/others/rxp.tar.gz

RXP is a non-validating parser written in C. It is distributed as C source and must be compiled before use. It supports Unicode and comes with a command-line application that prints out the parsed document.

Dan Connolly's XML parser, 1.8

Dan Connolly

Information on web:
http://www.w3.org/XML/9705/hacking
Software on web:

http://www.w3.org/XML/9705/xml.py
Software included on CD-ROM (Python):
./freesw/others/dc_parser.zip

This is a simple well-formedness (not complete) parser written in Python that handles both XML and HTML and can output lout.

XML-Toolkit, 0.7

David Schere

Information on web:
http://csmctmto.interpoint.net/didx/xml.html
Software on web:
http://csmctmto.interpoint.net/didx/archive/xmltoolkit.tar.gz
Software included on CD-ROM (Python):
./freesw/others/xmltoolkit.tar.gz

The XML-Toolkit provides a non-validating XML parser, a WIDL implementation and the parser can also be used in a client/server model.

LTXML, 0.9.5

Edinburgh Language Technology Group

Information on web:
http://www.ltg.ed.ac.uk/software/xml/
Software on web:
http://www.ltg.hcrc.ed.ac.uk/software/research_xml.html
Software included on CD-ROM (Source):
./freesw/ltxml/ltxml-0.9.5.tar.gz
Software included on CD-ROM (Linux):
./freesw/ltxml/ltxml-0.9.5-bin-linux.tar.gz
Software included on CD-ROM (Mac):
./freesw/ltxml/ltxml-0.9.5-bin-mac.sea.hqx
Software included on CD-ROM (Win32):
./freesw/ltxml/ltxml-0.9.5-bin-win32.zip

LTXML is a set of tools (including a parser) written in portable C. Included are: a program to strip out all XML markup, an XML normalizer

(mainly useful for well-formedness checking), an ESIS outputter, an element occurrence counter, a tokenizer, a down-translation tool, a grep tool, a sorting tool, some linking tools as well as some other minor utilities. The executables are mainly intended to be pipelined to produce various kinds of output, but provide a C API that can be used to extend them for other purposes.

expat, 19980405

James Clark

Information on web:
http://www.jclark.com/xml/expat.html
Software on web:
ftp://ftp.jclark.com/pub/xml/expat.zip
Software included on CD-ROM (C source):
./freesw/others/expat.zip

expat is written in C, and is the parser previously known as XMLTok. It is used in Mozilla 5.0 and in a Perl parser module written by Larry Wall. expat does no validation, but aims to be a fully conforming well-formedness parser. This is a beta release.

Tcl Support for XML, 1.0a1

Steve Ball

Information on web:
http://tcltk.anu.edu.au/XML/
Software on web:
http://tcltk.anu.edu.au/XML/XML-1.0a1.tcl.gz
Software included on CD-ROM (tcl):
./freesw/others/tclxml.zip

TCLXML is a validating XML parser written entirely in tcl, but is an alpha release that has not been tested much. The parser offers several ways to access the in-memory document structure after parsing, but only one is documented.

Xparse, 0.91

Jeremie Miller

Information on web:
http://www.jeremie.com/Dev/XML/index.phtml
Software on web:
http://www.jeremie.com/Dev/XML/xparse.js
Software included on CD-ROM (JavaScript):
./freesw/others/xparse.zip

Xparse is a very simple XML parser for use in web pages. It reads an XML document and produces a document tree consisting of elements, PIs, comments and character data.

XP, 0.2

James Clark

Information on web:
http://www.jclark.com/xml/xp/
Software on web:
ftp://ftp.jclark.com/pub/xml/xp.zip
Software included on CD-ROM (Java 1.1):
./freesw/others/xp.zip

XP is written to be fully-conforming and as fast as possible. The emphasis is on server-side production use. There is no validation, only well-formedness checking. Even though 0.2 is an alpha release it is very stable and extremely fast.

DataChannel XML Parser (DXP), 1.0 beta1a

DataChannel

Information on web:
http://www.datachannel.com/products/xml/DXP/
Software on web:
http://www.datachannel.com/products/xml/reg1.htm
Software included on CD-ROM (Java):

./freesw/others/DXP.zip

DXP is based on NXP, which was one of the first XML parsers, written by Norbert Mikula. It is validating and written for server-side use. There are both ESIS and SAX interfaces and support for SGML Open entity catalogs.

A full DOM interface and sophisticated error checking may appear in the full 1.0 version, so you may want to check the web site.

XML::Parse, 25.Mar.98 release

Larry Wall

Information on web:
ftp://www.wall.org/pub/larry/
Software on web:
ftp://www.wall.org/pub/larry/xmlparser-0.0.tar.gz
Software included on CD-ROM (Perl):
./freesw/others/xmlparser-0.0.tar.gz

This is a Perl wrapper around James Clark's XMLTok C parser (now known as expat). It is intended to be compiled into the Perl interpreter. It is undocumented and is not a final, official release.

PyXMLTok, 13.Mar.98 release

Jack Jansen

Information on web:
http://www.python.org/pipermail/1998q1.doc-sig/03eba1bbb5c3.html
Software on web:
ftp://ftp.cwi.nl/pub/jack/pyxmltok.tar.gz
Software included on CD-ROM (Python):
./freesw/others/pyxmltok.tar.gz

This is James Clark's XMLTok C parser module (now called expat) wrapped up as a Python module. This means that by compiling this into the Python interpreter (which is much easier than it sounds) one can have a fast C parser available from within Python. The interface is non-standard (i.e. not SAX).

Lark, 1.0 final beta

Tim Bray

Information on web:
http://www.textuality.com/Lark/
Software on web:
http://www.textuality.com/Lark/lark.tar.gz
Software included on CD-ROM (Java):
./freesw/others/lark.tar.gz

Lark was one of the two first XML parsers to appear, written by XML spec co-editor Tim Bray, but was non-validating for a long time. Tim Bray has now added Larval, a validating parser, to the package.

Lark is fast, small and thread-safe. Larval is in version 0.8 and not yet finished. The interface is non-standard, but there is a SAX driver in the SAX package.

Microsoft XML Parser in Java, 1.8

Microsoft

Information on web:
http://www.microsoft.com/workshop/author/xml/parser/
Software on web:
http://www.microsoft.com/xml/parser/msxml.tar.gz

This parser is validating, and has a non-standard interface, although a SAX driver is available in the SAX driver package. The parser implements the complete November 17, 1997, XML Working Draft, but has not been updated since the final recommendation. Lots of examples and documentation are bundled with the parser.

XML for Java, 9.Feb.98 release

IBM alphaWorks

Information on web:
http://www.alphaworks.ibm.com/formula/xml
Software on web:
http://www.sil.org/sgml/xml4j-19980206.zip

Software included on CD-ROM:
./freesw/others/xml4j-19980206.zip

This parser was written by Kento Tamura and Hiroshi Maruyama of the Tokyo Research Laboratory, IBM Japan. It is a validating parser that conforms to the 08.Dec.97 XML Working Draft and has not been updated since the release of the XML Recommendation.

The parser builds a tree structure object from an XML document, and can generate an XML document from a tree structure. There is also a SAX interface.

Use of this product is subject to the IBM alphaWorks license distributed with the software. However, the 90-day time limit is waived for purchasers of this book.

Ælfred, 1.1

Microstar

Information on web:
http://www.microstar.com/XML/index.htm
Software on web:
http://www.microstar.com/XML/aelfred-1.1.zip
Software included on CD-ROM (Java):
./freesw/others/aelfred-1.1.zip

Ælfred is designed to be small and fast, and is especially intended for use in Java applets (uses only two .class files). It has a non-standard interface, but comes with a SAX driver. Ælfred also handles a large number of different Unicode encodings.

Ælfred reads the DTD, but does not validate the document.

xmlproc, 0.30

Lars Marius Garshol

Information on web:
http://www.stud.ifi.uio.no/~larsga/download/python/xml/xmlproc.html
Software on web:
http://www.stud.ifi.uio.no/~larsga/download/python/xml/xmlproc.zip
Software included on CD-ROM (Python):

./freesw/others/xmlproc.zip

xmlproc is a validating parser written in Python. It implements most of the XML Recommendation, but not all. Some validation checks are not performed, illegal characters are accepted in some places, and some input transforms are not performed.

xmlproc has a non-standard interface, but comes with a SAX driver.

xmllib, 0.1

Sjoerd Mullender

Information on web:
http://www.python.org/doc/lib/
node162.html#SECTION001210000000000000000
Software on web:
http://www.cwi.nl/ftp/sjoerd/xmllib.tar.gz
Software included on CD-ROM (Python):
./freesw/others/xmllib.tar.gz

The xmllib parser is part of the Python 1.5 distribution. The version included here is a newer version than the one in the standard distribution. xmllib is non-validating, but a fairly complete well-formedness parser with a simple and intuitive interface. A SAX driver is available in saxlib.

30.2.1.6 XML middleware

XPublish, 1.0

Media Design in*Progress

Information on web:
http://interaction.in-progress.com/xpublish/index
Software included on CD-ROM (Mac):
./freesw/others/xpublish10.sit

XPublish is a web content management system based on XML. XPublish lets you write your documents in XML (and hybrids of HTML and XML) and publish them on the web as HTML. The XPublish editor is an Emacs

clone, with Lisp as an extension language. XPublish also helps you create a DTD and simplifies XML to HTML conversion.

Frontier, 5.0.1

Userland Software

Information on web:
http://www.scripting.com/frontier5/
Software on web:
http://www.scripting.com/frontier5/downloads/default.html
Software included on CD-ROM (Mac):
./freesw/frontier/frontier5.0.1.sit.hqx
Software included on CD-ROM (Win32):
./freesw/frontier/frontier501.zip

Frontier is a scripting environment for web content management. It works with a lot of different data sources, including XML. Frontier is rather unique and difficult to describe, so be sure to check the information at the Frontier web site. An XML package called blox is also included.

blox, 1.0b7

Technology Solutions

Information on web:
http://www.techsoln.com/frontier/blox/
Software on web:
http://www.techsoln.com/frontier/blox/download/
Software included on CD-ROM (Mac):
./freesw/frontier/blox.sit
Software included on CD-ROM (Win32):
./freesw/frontier/blox.zip

blox is an XML tool suite for Frontier that includes a parser and tools for manipulating XML documents.

PyDOM, 0.1

Stephane Fermigier

Information on web:
http://www.math.jussieu.fr/~fermigie/python/PyDOM/
Software on web:
http://www.math.jussieu.fr/~fermigie/python/PyDOM/dom-0.1.zip
Software included on CD-ROM (Python):
./freesw/others/dom-0.1.zip

PyDOM can build the DOM tree from a supplied XML parser (Dan Connolly's), an ESIS outputter, any SAX-compliant parser, the Python HTML parser (htmllib) or the Python SGML parser (sgmllib).

This online version of this list was produced by a Python script that used PyDOM for navigation through the parsed document. PyDOM is preliminary software and may have some bugs.

XML::Grove, 0.03

Ken MacLeod

Information on web:
http://bitsko.slc.ut.us/~ken/perl-xml/
Software on web:
http://bitsko.slc.ut.us/~ken/perl-xml/XML-Grove-0.03.tar.gz
Software included on CD-ROM (Perl):
./freesw/others/xml-grove-0.03.tar.gz

XML::Grove uses XML::Parse to build a tree structure from the parsed document that programs can access and change. Similar to DOM, that is, but based on ISO standards.

saxlib, 0.92

Lars Marius Garshol

Information on web:
http://www.stud.ifi.uio.no/~larsga/download/python/xml/
Software on web:
http://www.stud.ifi.uio.no/~larsga/download/python/xml/saxlib.zip

Software included on CD-ROM (Python):
./freesw/others/saxlib.zip

saxlib is a Python translation of the SAX parser interface. It has drivers for the xmllib parser and for the XML-Toolkit parser. There are also two demo applications: saxdemo.py, which produces canonical XML output and saxtimer.py, which measures the time used to parse a document with an empty document handler.

SAX, 27.Jan.98 release

David Megginson

Information on web:
http://www.microstar.com/XML/SAX/java-implementation.html
Software on web:
http://www.microstar.com/XML/SAX/sax-java-19980127.zip
Software included on CD-ROM (Java):
./freesw/others/sax-java-19980127.zip

SAX is a simple event-based API for XML parsers. It is not an official standard, since it was developed by the participants of the xml-dev mailing list instead of a standards body. However, SAX is very much a de facto standard, since it is supported by at least 9 parsers.

This library contains the Java implementation of SAX, but no drivers.

SAX drivers for Lark and MSXML, 13.Mar.98 release

David Megginson

Information on web:
http://www.microstar.com/XML/SAX/drivers.html
Software on web:
http://www.microstar.com/XML/SAX/sax-java-drivers-19980313.zip
Software included on CD-ROM (Java):
./freesw/others/sax-java-drivers-19980313.zip

This is a package consisting of two SAX drivers for Java parsers that do not support SAX natively, namely Lark and the Microsoft parser.

SAXDOM, 6.Apr.98 release

Don Park

Information on web:
http://www.docuverse.com/personal/saxdom.html
Software on web:
http://www.docuverse.com/personal/saxdom040698.zip
Software included on CD-ROM (Java):
./freesw/others/saxdom040698.zip

SAXDOM is a Java implementation of the Document Object Model that uses any SAX client (just use the SAX package and any parser you like) to build the DOM document tree. SAXDOM will continue to change as SAX and the DOM evolve. You may want to check the website to see if there is a new version.

30.2.2 *Editing and composition*

30.2.2.1 XML editors

PSGML, 1.0.1 with XML patch (General SGML/ XML tool)

Lenart Staflin
David Megginson

Information on web:
http://www.lysator.liu.se/projects/about_psgml.html
Software on web:
ftp://ftp.lysator.liu.se/pub/sgml/
Software included on CD-ROM:
./freesw/others/psgml-xml.zip (Elisp source)

Emacs is easily one of the most powerful (if not the most powerful) text editors in the world. It has an internal Lisp programming language, which means that new modes are easy to write, and as a consequence Emacs has usage modes for most programming languages, as well as a web browser with CSS (and budding DSSSL) support and a world-class news reader.

The user interface is quite unlike most modern editors, but Emacs comes with internal documentation that can help you out.

PSGML is a full SGML mode that has been patched to support XML. It reads the DTD, can use an external parser to validate documents, does syntax coloring, and a lot of other things.

Visual XML, beta 1

Pierre Morel

Information on web:
http://www.pierlou.com/visxml/
Software on web:
http://www.pierlou.com/visxml/download.htm
Software included on CD-ROM (Java):
./freesw/others/visual-xml-b1.zip

Visual XML is an XML editor written in Java with JFC (Swing). It lets you edit a tree view of the XML document. This is a test version and may have bugs.

XED, 19.Mar.98 release

Henry Thompson

Information on web:
http://www.ltg.ed.ac.uk/~ht/xed.html
Software on web:
ftp://ftp.cogsci.ed.ac.uk/pub/ht/
Software included on CD-ROM (Win32):
./freesw/xed/xed.zip
Software included on CD-ROM (Solaris 2.5):
./freesw/xed/xed-solaris2.5.tar.gz

XED is a simple XML editor written in C, Python and Tk. It tries to ensure that the author cannot write a document that is not well-formed and reads the DTD in order to be able to suggest valid elements to be inserted at any point in the document.

The document is shown as text, not as a tree view.

XML < PRO > , 1.0b

Vervet Logic

Information on web:
http://www.vervet.com/
Software on web:
http://www.vervet.com/beta.html

XML < PRO > is a tree-based XML editor, with validation.

Amaya, 1.2a

World Wide Web Consortium

Information on web:
http://www.w3.org/Amaya/
Software on web:
http://www.w3.org/Amaya/User/BinDist.html
Software included on CD-ROM (Win32):
./freesw/amaya/amaya-1.2a.exe
Software included on CD-ROM (Java (Linux)):
./freesw/amaya/amaya-java-LINUX-1.1c.tar.gz
Software included on CD-ROM (Linux):
./freesw/amaya/amaya-LINUX-ELF-1.2a.tar.gz

Amaya is the W3C testbed browser, and is an HTML browser (and editing tool) with CSS support. It also supports the MathML XML DTD and can edit and display presentational MathML graphically.

30.2.3 *Control information development*

30.2.3.1 XSL editors

ArborText XML Styler, 2.0

ArborText

Information on web:

http://www.arbortext.com/xmlstyler/
Software on web:
http://www.arbortext.com/xmlstyler/registration.html
Software included on CD-ROM (Win32):
./freesw/xmlstyler/xmlstyler2c.exe
Software included on CD-ROM (Java):
./freesw/xmlstyler/xmlstyler2c.zip

XML Styler lets you create XSL style sheets for your XML documents using a visual editor. The Windows version allows previewing with the XSL ActiveX control in MSIE 4.0.

30.2.3.2 DTD editors

tdtd, revision 0.4

Tony Graham

Information on web:
ftp://ftp.mulberrytech.com/pub/mulberrytech/tdtd/
Software on web:
ftp://ftp.mulberrytech.com/pub/mulberrytech/tdtd/tdtd.zip
Software included on CD-ROM:
./freesw/others/tdtd.zip (Elisp files)

This is an Emacs major mode for editing DTDs. It does syntax coloring and has some convenience macros for inserting commonly-typed constructs.

30.2.3.3 DTD documenters

perlSGML, 18.Sep.97 (General SGML/XML tool)

Earl Hood

Information on web:
http://www.oac.uci.edu/indiv/ehood/perlSGML.html
Software on web:

XML < PRO > , 1.0b

Vervet Logic

Information on web:
http://www.vervet.com/
Software on web:
http://www.vervet.com/beta.html

XML < PRO > is a tree-based XML editor, with validation.

Amaya, 1.2a

World Wide Web Consortium

Information on web:
http://www.w3.org/Amaya/
Software on web:
http://www.w3.org/Amaya/User/BinDist.html
Software included on CD-ROM (Win32):
./freesw/amaya/amaya-1.2a.exe
Software included on CD-ROM (Java (Linux)):
./freesw/amaya/amaya-java-LINUX-1.1c.tar.gz
Software included on CD-ROM (Linux):
./freesw/amaya/amaya-LINUX-ELF-1.2a.tar.gz

Amaya is the W3C testbed browser, and is an HTML browser (and editing tool) with CSS support. It also supports the MathML XML DTD and can edit and display presentational MathML graphically.

30.2.3 *Control information development*

30.2.3.1 XSL editors

ArborText XML Styler, 2.0

ArborText

Information on web:

http://www.arbortext.com/xmlstyler/
Software on web:
http://www.arbortext.com/xmlstyler/registration.html
Software included on CD-ROM (Win32):
./freesw/xmlstyler/xmlstyler2c.exe
Software included on CD-ROM (Java):
./freesw/xmlstyler/xmlstyler2c.zip

XML Styler lets you create XSL style sheets for your XML documents using a visual editor. The Windows version allows previewing with the XSL ActiveX control in MSIE 4.0.

30.2.3.2 DTD editors

tdtd, revision 0.4

Tony Graham

Information on web:
ftp://ftp.mulberrytech.com/pub/mulberrytech/tdtd/
Software on web:
ftp://ftp.mulberrytech.com/pub/mulberrytech/tdtd/tdtd.zip
Software included on CD-ROM:
./freesw/others/tdtd.zip (Elisp files)

This is an Emacs major mode for editing DTDs. It does syntax coloring and has some convenience macros for inserting commonly-typed constructs.

30.2.3.3 DTD documenters

perlSGML, 18.Sep.97 (General SGML/XML tool)

Earl Hood

Information on web:
http://www.oac.uci.edu/indiv/ehood/perlSGML.html
Software on web:

http://www.oac.uci.edu/indiv/ehood/tar/perlSGML.1997Sep18.tar.gz
Software included on CD-ROM (Perl):
./freesw/others/perlSGML.1997Sep18.tar.gz

perlSGML is a collection of Perl tools for working with full SGML, but they also work with XML. Included are DTD documentation tools, a DTD diff tool and several useful related libraries.

30.2.4 *Conversion*

30.2.4.1 General S-converters

OmniMark LE, 4.0e2 (General SGML/XML tool)

OmniMark Technologies

Information on web:
http://www.omnimark.com/develop/omle40/
Software on web:
http://www.omnimark.com/develop/omle40/download/omle40e2.exe

OmniMark is a well-known tool in the SGML industry, where it is much used for SGML conversions and also non-SGML conversions due to its superb regular expression and SGML support. This prerelease version of OmniMark LE (a free, but limited version of the program) is the first version of OmniMark to support XML.

30.2.4.2 Specific N-converters

RDF for XML, 9.Apr.98 release

IBM alphaWorks

Information on web:
http://www.alphaWorks.ibm.com/formula/rdfxml
Software included on CD-ROM (Java):

./freesw/others/rdf.zip

RDF for XML is an RDF implementation for building, querying and manipulating RDF structures as well as reading them from and writing them to the XML representation. This implementation follows the 16.Feb.98 RDF working draft.

RDF for XML requires the XML for Java XML parser.

Use of this product is subject to the IBM alphaWorks license distributed with the software. However, the 90-day time limit is waived for purchasers of this book.

30.2.4.3 General N-converters

DataChannel XML Generator, 0.1 beta1

DataChannel

Information on web:
http://www.datachannel.com/press_room/xml_gen/
Software on web:
http://www.datachannel.com/products/xml/dxp/xmlgenerator.zip
Software included on CD-ROM (Java):
./freesw/others/xmlgenerator.zip

The DataChannel XML Generator takes a character-delimited file (such as an exported spreadsheet or database) and an XML template as input and produces XML output based on the template and input.

30.2.5 *Electronic delivery*

30.2.5.1 XML browsers

Mozilla, 8.Apr.98 release

The Mozilla team

Information on web:

http://www.mozilla.org/
Software on web:
http://www.uwasa.fi/~e75644/mozilla/MozFAQ.html
Software included on CD-ROM (Win32 sources):
./freesw/mozilla/win_19980408.zip
Software included on CD-ROM (Mac sources):
./freesw/mozilla/mac_19980408.sit.bin
Software included on CD-ROM (Unix sources):
./freesw/mozilla/unix_19980408.tar.gz

This is version 5 of Netscape Navigator, which can display XML documents with CSS stylesheets. Please note that this is a very preliminary test version and is known to have bugs.

Jumbo, 9801a1

Peter Murray-Rust

Information on web:
http://ala.vsms.nottingham.ac.uk/vsms/java/jumbo/
Software on web:
http://ala.vsms.nottingham.ac.uk/vsms/java/jumbo/jan9801/jumbo9801a/
Software included on CD-ROM (Java):
./freesw/others/jumbo9801a1.zip

Jumbo was the first XML browser to appear, but does not support stylesheets, so documents are currently shown in a tree view. (There is built-in support for some DTDs, notably CML.) Jumbo is not delivered with a parser, but installing a parser with a SAX driver in your class path will automatically enable Jumbo to parse XML documents.

DataChannel XML Viewer, 28.Jul.97 release

DataChannel

Information on web:
http://xml.datachannel.com/XMLTreeViewer/demo.html
Software on web:
http://www.datachannel.com/products/xml/xmlviewerappletkit.zip
Software included on CD-ROM (Java):

./freesw/others/xmlviewerappletkit.zip

This is an applet (and an application) that shows a tree-based view of XML documents.

Prototype, I

Pierre Morel

Information on web:
http://www.pierlou.com/prototype/
Software on web:
http://www.pierlou.com/prototype/download.htm
Software included on CD-ROM (Java):
./freesw/others/proto-full.zip

Prototype is a program that reads a structured description of an application, in the form of an XML document corresponding to a particular DTD. It produces the look and feel of the application.

30.2.6 *Resources*

30.2.6.1 Useful programs

This is a collection of software that is needed to run some of the packages included, but which does not have XML capabilities natively. We've linked to them here to help you find these packages if you don't have them already.

Emacs

- *Web page*:
  http://www.geek-girl.com/emacs/emacs.html
- *Win32 version*:
  http://www.cs.washington.edu/homes/voelker/ntemacs.html
- *Software on CD-ROM (Win32)*:
  ./freesw/utils/emacs-19.34.zip

- *Software on CD-ROM (Linux)*: ./freesw/utils/emacs-MBSK-X11-20.2-4.i386.rpm
- *Software on CD-ROM (Source)*: ./freesw/utils/emacs-20.2.tar.gz
- Sadly, Emacs has not been available for the Mac since version 18.59.

Python

- *Web page*: http://www.python.org/
- *Software on CD-ROM (Win32)*: ./freesw/utils/pyth151.exe
- *Software on CD-ROM (Mac)*: ./freesw/utils/macpython15b3.bin
- *Software on CD-ROM (Linux)*: ./freesw/utils/python-1.5.1-1.i386.rpm
- *Software on CD-ROM (Source)*: ./freesw/utils/pyth151.tgz

Perl

- Web page: http://www.perl.com/.
- *Software on CD-ROM (Win32)*: ./freesw/utils/Pw32i316.exe
- *Software on CD-ROM (Mac)*: ./freesw/utils/Mac_Perl_519r4_appl.bin
- *Software on CD-ROM (Linux)*: ./freesw/utils/perl-5.004_03-1.i386.rpm
- *Software on CD-ROM (Source)*: ./freesw/utils/perl5.004_04.tar.gz

Other tools

- Java Development Kit: http://java.sun.com/products/jdk/1.1/
- The Tcl interpreter: http://sunscript.sun.com/TclTkCore/

30.2.6.2 Archiving software

Most of the software is distributed as a collection of files combined (or "archived") into a single file for easier network delivery. The utilities used to create these archives often originated on the UNIX operating system as standard UNIX tools, or as free software from the Free Software Foundation's GNU collection, but all are available in Windows and DOS versions as well:

- Win95/NT gunzip, unzip, tar: ftp://ftp.cs.washington.edu/pub/ntemacs/latest/i386/utilities/i386/
- DOS unzip and gzip (gzip also unzips *.gz files): ftp://ftp.uu.net/pub/archiving/zip/MSDOS/
- DOS untar: ftp://wuarchive.wustl.edu/systems/ibmpc/garbo.uwasa.fi/unix/
- NT/Win 95 unzip and gzip: ftp://ftp.uu.net/pub/archiving/zip/WIN32/
- NT/Win 95 tar (tar program also untars): ftp://sunsite.doc.ic.ac.uk/pub/packages/simtel/win95/archiver/

XML basics

- Syntactic details
- The prolog and the document instance
- XML declaration
- Elements and attributes

Chapter

31

XML's central concepts are quite simple, and this chapter outlines the most important of them. Essentially, it gives you what you need to know to actually create XML documents. In subequent chapters you will learn how to combine them, share text between them, format them, and validate their structure.

Before looking at actual XML markup (don't worry, we'll get there soon!) we should consider some syntactic constructs that will recur throughout our discussion of XML documents. By *syntax* we mean the combination of characters that make up an XML document. This is analogous to the distinction between sounds of words and the things that they mean. Essentially, we are talking about where you can put angle brackets, quote marks, ampersands, and other characters and where you cannot! Later we will talk about what they mean when you put them together.

After that, we will discusses the components that make up an XML document instance[1]. We will look at the distinction between the prolog (information XML processors need to know about your document) and the instance (the representation of the real document itself).

1. Roughly, what the XML spec calls the "root element".

31.1 | Syntactic details

XML documents are composed of characters from the *Unicode* character set. Any such sequence of characters is called a *string*. The characters in this book can be thought of as one long (but interesting) string of text. Each chapter is also a string. So is each word. XML documents are similarly made up of strings within strings.

Natural languages such as English have a particular syntax. The syntax allows you to combine words into grammatical sentences. XML also has syntax. It describes how you combine strings into well-formed XML documents. We will describe the basics of XML's syntax in this section.

31.1.1 *Case-sensitivity*

XML is *case-sensitive*. That means that if the XML specification says to insert the word "ELEMENT", it means that you should insert "ELEMENT" and not "element" or "Element" or "ElEmEnT".

For many people, particularly English speaking people, case-insensitive matching is easier than remembering the case of particular constructs. For instance, if a document type has an element type named `img` English speakers will often forget and insert `IMG`. They confuse the two because they are not accustomed to considering case to be significant. This is also why some people new to the Internet tend to TYPE IN ALL UPPER CASE. Most applications of SGML, including HTML, are designed to be case-insensitive. They argue that this eliminates case as a source of errors.

Others argue that the whole concept of case-insensitivity is a throwback to keypunches and other early text-entry devices. They also point out that case-sensitivity is a very complicated concept in an international character set like Unicode for a variety of reasons.

For instance, the rules for case conversion of certain accented characters are different in Quebec from what they are in France. There are also some languages for which the concept of upper-case and lower-case does not exist at all. There is no simple, universal rule for case-insensitive matching. In the end, internationalization won out in XML's design.

So mind your "p's" and "q's" and "P's" and "Q's". Our authoritative laboratory testing by people in white coats indicates that exactly 74.5% of all XML errors are related to case-sensitivity mistakes. Of course XML is also

spelling-sensitive and typo-sensitive, so watch out for these and other byproducts of human fallibility.

Note that although XML is case-sensitive it is not case-prejudiced. Anywhere that you have the freedom to create your own names or text, you can choose to use upper- or lower-case text, as you prefer. So although you must type XML's keywords exactly as they are described, your own strings can mix and match upper- and lower-case characters however you like.

For instance, when you create your own document types you will be able to choose element type names. A particular name could be all upper-case (SECTION), all lower-case (section) or mixed-case (SeCtION). But because XML is case-sensitive, all references to a particular element type would have to use the same case. It is good practice to create a simple convention such as all lower-case or all upper-case so that you do not have to depend on your memory.

31.1.2 *Markup and data*

The constructs such as tags, entity references, and declarations are called *markup*. These are the parts of your document that are supposed to be understood by the XML processor. The parts that are between the markup are typically supposed to be understood only by other human beings. That is the *character data*. Here is what the XML specification says on this issue:

Spec. Reference 31-1. Markup

Markup takes the form of start-tags, end-tags, empty-element tags, entity references, character references, comments, CDATA section delimiters, document type declarations, and processing instructions.

We haven't explained what all of those things are yet, but they are easy to recognize. All of them start with less-than ("<") or ampersand ("&") characters. Everything else is character data.

31.1.3 *White space*

There is a set of characters called *white space* characters that XML processors treat differently in XML markup. They are the "invisible" characters: space (Unicode/ASCII 32), tab (Unicode/ASCII 9), carriage return (Unicode/ASCII 13) and line feed (Unicode/ASCII 10). These correspond roughly to the `spacebar`, `tab`, and `Enter` keys on your keyboard.

When the XML specification says that white space is allowed at a particular point, you may put as many of these characters as you want in any combination. Just as you might put two lines between paragraphs in a word processor to make a printed document readable, you may put two carriage returns in certain places in an XML document to make your source file more readable and maintainable. When the document is processed, those characters will be ignored.

In other places, white space will be significant. For instance you would not want the processor to strip out the spaces between the words in your document! Thatwouldmakeithardtoread. So white space outside of markup is always preserved in XML and white space within markup may be preserved, ignored, and sometimes combined in weird, and wonderful ways. We will describe the combination rules as we go along.

31.1.4 *Names and name tokens*

When you use XML you will often have to give things names. You will name logical structures with element type names, reusable data with entity names, particular elements with IDs, and so forth. XML names have certain common features. They are not nearly as flexible as character data:

Spec. Reference 31-2. Names

A Name [begins] with a letter or one of a few punctuation characters, and [continues] with letters, digits, hyphens, underscores, colons, or full stops, together known as name characters. Names beginning with the string "xml", [matched case-insensitively] are reserved for standardization in this or future versions of this specification.

In other words, you cannot make names that begin with the string "xml" or somecase-insensitive variant like "XML" or "XmL". Letters or underscores can be used anywhere in a name. You may include digits, hyphens and full-stop (".") characters in a name, but you may not start the name with one of them. Other characters, like various symbols and white space, cannot be part of a name.

There is another related syntactic construct called a *name token*. Name tokens are just like names except that they *may* start with digits, hyphens, full-stop characters, and the string XML.

Spec. Reference 31-3. Name tokens

An Nmtoken (name token) is any mixture of name characters.

In other words every valid name is also a valid name token, but here are some name tokens that are not valid names:

Example 31-1. Name tokens

```
.1.a.name.token.but.not.a.name
2-a-name-token.but-not.a-name
XML-valid-name-token
```

Like almost everything else in XML, names, and name tokens are matched case-sensitively. Names and name tokens do not allow white space, most punctuation or other "funny" characters. The remaining "ordinary" characters are called *name characters*.

31.1.5 *Literal strings*

The data (text other than markup) can contain almost any characters. Obviously, in the main text of your document you need to be able to use punctuation and white space characters! But sometimes you also need these characters *within* markup. For instance an element might represent a hyperlink and need to contain a URL. The URL would have to go in markup, where characters other than the name characters are not usually allowed.

Literal strings allow users to use funny (non-name) characters within markup, but only in contexts in which it makes sense to specify values that

might require those characters. For instance, to specify the URL in the hyperlink, we would need the slash character. Here is an example of such an element:

```
<REFERENCE URL="http://www.documents.com/document.xml">
```

The string that defines the URL is the literal string. This one starts and ends with double quote characters. Literal strings are always surrounded by either single or double quotes. The quotes are not part of the string. Here is what the XML spec says:

Spec. Reference 31-4. Literal data

Literal data is any quoted string not containing the quotation mark used as a delimiter for that string. Literals are used for specifying the content of internal entities, the values of attributes, and external identifiers.

You may use either single (“'”) or double (“"”) quotes to mark (*delimit*) the beginning and end of these strings in your XML document. Whichever type of quote the string starts with, it must end with. The other type may be used within the literal and has no special meaning there. Typically you will use double quotes when you want to put an actual single-quote character in the literal and single quotes when you want to embed an actual double quote. When you do not need to embed either, you can take your pick. Here are some examples:

```
"This is a double quoted literal."
'This is a single quoted literal.'
"'tis another double quoted literal."
'"And this is single quoted" said the self-referential example.'
```

The ability to have quotes within quotes is quite useful when dealing with human speech or programming language text:

```
"To be or not to be"
'"To be or not to be", quoth Hamlet.'
"'BE!', said Jean-Louis Gassee."
'B = "TRUE";'
```

Note that there *are* ways of including a double quote character inside of a double-quoted literal. This is important because a single literal might (rarely) need both types of quotes.

31.1.6 *Grammars*

Natural language syntax is described with a grammar. XML's syntax is also. Some readers will want to dig in and learn the complete, intricate details of XML's syntax. We will provide grammar rules for them as we go along. These come right out of the XML specification. If you want to learn how to read them, you should skip ahead to Chapter 37, "Reading the XML specification", on page 546. After you have read it, you can come back and understand the rules as we present them. Another strategy is to read the chapters without worrying about the grammar rules, and then only use them when you need to answer a particular question about XML syntax.

You can recognize grammar rules taken from the specification by their form. They will look like this:

Spec. Reference 31-5. An example of a grammar rule

```
xhb ::= 'a' 'good' 'read'
```

We will not specifically introduce these rules, because we do not want to interrupt the flow of the text. They will just pop up in the appropriate place to describe the syntax of something.

31.2 | Prolog vs. instance

Most document representations start with a header that contains information about the real document and how to interpret its representation. This is followed by the representation of the real document.

For instance, HTML has a HEAD element that can contain the TITLE and META elements. After the HEAD element comes the BODY. This is where the representation of the real document resides. Similarly, email messages have "header lines" that describe who the message came from, to whom it is addressed, how it is encoded, and other things.

An XML document is similarly broken up into two main parts: a *prolog* and a *document instance*. The prolog provides information about the interpretation of the document instance, such as the version of XML and the document type to which it conforms. The document instance follows the

prolog. It contains the actual document data organized as a hierarchy of elements.

Spec. Reference 31-6. Document production

```
document ::=  prolog element Misc*
```

31.3 | The logical structure

The actual content of an XML document goes in the document instance. It is called this because if it has a DTD, it is an instance of a class of documents defined by the DTD. Just as a particular person is an instance of the class of "people", a particular memo is an instance of the class of "memo documents". The formal definition of "memo document" is in the memo DTD.

Here is an example of a small XML document.

Example 31-2. Small XML Document

```
<?xml version="1.0"?>
<!DOCTYPE MEMO SYSTEM "memo.dtd">
<memo>
<from>
   <name>Paul Prescod</name>
   <email>papresco@prescod.com</email>
</from>
<to>
   <name>Charles Goldfarb</name>
   <email>charles@sgmlsource.com</email>
</to>
<subject>Another Memo Example</subject>
<body>
<paragraph> Charles, I wanted to suggest that we
<emphasis>not</emphasis> use the  typical memo example in
our book. Memos tend to be used anywhere a small, simple
document type is needed, but they are just
<emphasis>so</emphasis> boring!
</paragraph>
</body>
</memo>
```

Because a computer cannot understand the data of the document, it looks primarily at the *tags*, the markup between the less-than and greater-than symbols. The tags delimit the beginning and end of various elements. The computer thinks of the elements as a sort of tree.

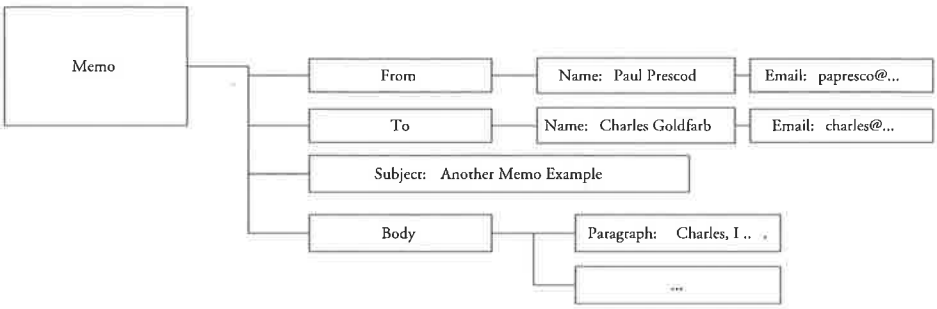

Figure 31-1 The memo XML document viewed as a tree.

Figure 31-1 shows a graphical view of the logical structure of the document. The MEMO element is called either the *document element* or the *root element.*

The document element (`memo`) represents the document as a whole. Every other element represents a component of the document. The `from` and `to` elements are meant to indicate the source and target of the memo. The `name` elements represent people's names. Continuing in this way, the logical structure of the document is apparent from the element-type names.

Experts refer to an element's real-world meaning as its semantics. In a particular DTD, the semantics of a `P` element might be "paragraph" and in another it might mean `pence`. If you find yourself reading or writing markup and asking: "But what does that *mean*?" or "What does that look like?" then you are asking about semantics.

Computers do not know anything about semantics. They do not know an HTTP protocol from a supermodel. Document type designers must describe semantics to authors some other way. For instance they could send email, write a book or make a major motion picture (well, maybe some day). What the computer does care about is how an element is supposed to look when it is formatted, or how it is to behave if it is interactive, or what to do with the data once it is extracted. These are specified in *stylesheets* and computer programs.

31.4 | Elements

XML elements break down into two categories. Most have content, which is to say they contain characters, elements or both, and some do not. Those that do not are called *empty elements.*

Here is an example of an element with content:

Example 31-3. Simple element

```
<title>This is the title</title>
```

Most elements have content. Elements with content begin with a start-tag and finish with an end-tag. The "stuff" between the two is the element's content. In Example 31-3, "This is the title" is the content.

Spec. Reference 31-7. Element with content

```
[39] element ::=  start-tag content End-tag
```

XML start-tags consist of the less-than (<) symbol ("left angle bracket"), the name of the element's type, termed a generic identifier (gi), and a greater-than (>) symbol ("right angle bracket"). Start-tags can also include attributes. We will look at those later in the chapter. The start-tag in Example 31-3 is `<TITLE>` and its generic identifier is "TITLE".

Spec. Reference 31-8. Start-tag

```
[40] STag ::=  '<' Name (S Attribute)* S? '>'
```

XML end-tags consist of the string "</", the same generic identifier (or *GI*) as in the start-tag, and a greater-than (>) symbol. The end-tag in Example 31-3 is `</TITLE>`.

You must always repeat the GI in the end-tag. This helps you to keep track of which end-tags line up with which start-tags. If you ever forget one or the other, the processor will know immediately, and will alert you that the document is not well-formed. The downside of this redundancy is that it requires more typing. Some people like belts and some prefer suspenders. The XML Working Group likes belts *and* suspenders.

Spec. Reference 31-9. End-tag

```
[42] ETag ::=  '</' Name S? '>'
```

Note that less-than symbols in content are always interpreted as beginning a tag. If the characters following them would not constitute a valid tag, then the document is not well-formed.

Caution *Use the word "tag" precisely*

Many people use the word "tag" imprecisely. Sometimes they mean "generic identifier", sometimes "element-type name", sometimes "element type" and sometimes they actually mean "tag". This leads to confusion. In XML, tags always start with less-than symbols and end with greater-than symbols. Nothing else is a tag. Tags are not defined in DTDs; element types are defined in DTDs.

It is possible for an element to have no content at all. Such an element is called an *empty element*. One way to denote an empty element is to merely leave out the content. But as a shortcut, empty elements may also have a different syntax. Because there is no content to delimit, they may consist of a single empty-element tag. That looks like this: `<EmptyTag/>`.

The slash at the end indicates that this is an empty-element tag, so there is no content or end-tag coming up. The slash is meant to be reminiscent of the slash in the end-tag of an element with both tags.

Spec. Reference 31-10. Empty-element tag

```
[44]  EmptyElemTag ::=  '<' Name (S Attribute)* S? '/>'
```

Usually empty elements have *attributes*. Occasionally an empty element without attributes will be used to flag a particular location in a document. Here is an example of an empty element with an attribute:

```
<EMPTY-ELEMENT ATTR="ATTVAL"/>
```

In summary, elements are either empty or have content. Elements with content are represented by a start-tag, the content, and an end-tag. Empty elements can either have a start-tag and end-tag with nothing in between,

or a single empty-element tag. An element's type is always identified by the generic identifiers in its tags.

The reason we distinguish element types from generic identifiers is because the term "generic identifier" refers to the syntax of the XML document – the characters that represent the real document. The term "element type" refers to a property of a component of the real document.

31.5 | Attributes

In addition to content, elements may have *attributes*. Attributes are a way of attaching characteristics or properties to elements of a document. Attributes have *names*, just as real-world properties do. They also have values. For instance, two possible attributes of people are their "shoe size" and "IQ" (the attributes' names), and two possible values are "12" and "12" (respectively).

In a DTD, each attribute is defined for a specific element type and is allowed to exhibit a certain type of value. Multiple element types could provide attributes with the same name and it is sometimes convenient to think of them as the "same attribute" even though they technically are not.

Attributes have semantics also. They always *mean* something. For example, an attribute named `height` might be provided for `person` elements (allowed occurrence), exhibit values that are numbers (allowed values), and represent the person's height in centimeters (semantics).

Here is how attributes of `person` elements might look.

Example 31-4. Elements with attributes

```
<person height="165cm">Dale Wick</person>
<person height="165cm" weight="165lb">Bill Bunn</person>
```

As you can see, the attribute name does not go in quotes, but the attribute value does.

Spec. Reference 31-11. Attributes

```
[41]  Attribute ::=  Name Eq AttValue
[25]  Eq ::=  S? '=' S?
```

Like other literals (see page 429), attributes can be surrounded by either single (') or double (") quotes. When you use one type of quote, the other can be used within that attribute value. As we discussed earlier, this makes it convenient to create attribute values that have the quote characters within them:

Example 31-5. Attribute values can have quotes in them

```
<PERSON HEIGHT='80"'>
<PERSON QUOTE="'To be or not to be'">
```

There are other ways of getting special characters into attribute values and we will discuss them in 36.2, "Character references", on page 535.

A DTD constrains an attribute's allowed occurrence and values. One possibility is to require an attribute to be specified for all elements. For example, a military document might require `section` elements to have a `security` attribute with the value `unclassified`, `classified`, or `secret`.

```
<!ATTLIST SECTION
          SECURITY (unclassified | classified | secret) #REQUIRED >
```

The attribute would need to be specified for each `section` element:

```
<SECTION SECURITY="unclassified">...</SECTION>
```

It would be a validity error to create a `section` element without a `security` attribute.

Usually empty elements have attributes. Sometimes an element with subelements can be modeled just as well with an empty element and attributes. Here are two ways of modeling a `person` element in an email message:

Example 31-6. Alternative `person` element

```
<FROM><NAME>Paul Prescod</NAME>
      <EMAIL>"papresco@prescod.com"</EMAIL>
</FROM>
vs.
<FROM NAME="Paul Prescod" EMAIL="papresco@prescod.com"/>
```

Yet another way to do it would be to let the person's name be data content:

As you can see, there can be many different ways to represent the same construct. There is no one right way to do so. In the case of `person`, the last

Example 31-7. Another alternative `person` element

```
<FROM><PERSON EMAIL="papresco@prescod.com">Paul Prescod
      </PERSON>
</FROM>
```

version shown is the most typical because the character data of a document generally represents what you would expect to see in a "print-out".

But that is not a hard and fast rule (after all, renditions vary widely). Because there are so many ways to represent the same thing, it is advisable to use a DTD. The constraints in a DTD can maintain consistency across a range of documents, or even within a single large document. There may be many ways to represent a particular concept, but once you choose one, let the DTD help you stick to it.

31.6 | The prolog

XML documents should start with a prolog that describes the XML version ("1.0", for now), document type, and other characteristics of the document.

The prolog is made up of an *XML declaration* and a *document type declaration*, both optional. Though an author may include either, neither, or both, it is best to try to maximize the amount of prolog information provided. This will make later processing more reliable.

The XML declaration must precede the document type declaration if both are provided. Also, comments, processing instructions, and white space can be mixed in among the two declarations. The prolog ends when the first start-tag begins.

Here is a sample prolog as a warm-up:

Example 31-8. A simple prolog

```
<?xml version="1.0"?>
<!DOCTYPE DOCBOOK SYSTEM "http://www.davenport.org/docbook">
```

This DTD says that the document conforms to XML version 1.0 and declares adherence to a particular document type, `DOCBOOK`.

Here are the grammar rules for the prolog:

Spec. Reference 31-12. Prolog

```
[22]  prolog ::=  XMLDecl? Misc* (doctypedecl Misc*)?
[27]  Misc ::=  Comment | PI |  S
```

31.6.1 *XML declaration*

The XML declaration is fairly simple. It has several parts and they fit together one after another.

Spec. Reference 31-13. XML declaration

```
[23]  XMLDecl ::=  '<?xml' VersionInfo EncodingDecl? SDDecl? S? '?>'
```

A minimal XML declaration looks like this:

Example 31-9. Minimal XML declaration

```
<?xml version="1.0"?>
```

Here is a more expansive one, using all of its parts:

Example 31-10. More expansive XML declaration

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

There is one important thing to note in the last example. It looks like a start-tag with attributes, but it is *not*. The different parts of the XML declaration just happen to look like attributes. Well, not quite "just happen": it could have had a completely different syntax, but that would have been harder to memorize. So the parts were chosen to look like attributes to reduce the complexity of the language. One important difference between XML declaration parts and attributes is that the parts are strictly ordered whereas attributes can be specified in any order.

31.6.1.1 Version info

The *version info* part of the XML declaration declares the version of XML that is in use. It is required in all XML declarations. At the time of writing, the only valid version string is "1.0". But if you always use the version string, you can be confident that future XML processors will not think that your document was meant to conform to XML version 2.0 or 3.0 when and if those languages become available. Since they do not exist yet, you cannot know if your documents will be compatible with them.

In fact, the only reason that the XML declaration is optional is so that some HTML and SGML documents can be used as XML documents without confusing the software that they usually work with. You can imagine that an older browser would not react nicely to an HTML document with an XML declaration. But this "backwards compatibility" consideration is only temporary. Future versions of XML may require the XML declaration.

The XML version information is part of a general trend towards information representations that are *self-identifying*. This means that you can look at an XML document and (if it has the declaration) know immediately both that it is XML and what version of XML it uses. As more and more document representations become self-identifying, we will be able to stop relying on error-prone identification schemes like file extensions.

31.6.1.2 Encoding declaration

An XML declaration may also include an *encoding declaration*. It describes what character encoding is used. This is another aspect of being self-identifying. If your documents are encoded in the traditional 7-bit-ASCII used on most operating systems and with most text editors, then you do not need to worry about the encoding-declaration. 7-bit-ASCII is a subset of a Unicode encoding called *UTF-8* which XML processors can automatically detect and use. If you use 7-bit ASCII and need to encode a character outside of 7-bit-ASCII, such as the trademark sign or a non-English character, you can do so most easily by using a numeric character reference, as described in 36.2, "Character references", on page 535.

Spec. Reference 31-14. Encoding declaration

```
[80]EncodingDecl::= S 'encoding'
                    Eq ('"' EncName '"' | "'" EncName "'" )
[81]EncName::= [A-Za-z] ([A-Za-z0-9._] |'-')*
```

31.6.1.3 Standalone document declaration

An XML declaration can include a *standalone document declaration*. It declares what components of the document type definition are necessary for complete processing of the document. This declaration is described in 36.4, "Standalone document declaration", on page 541.

31.6.2 *Document type declaration*

Somewhere after the XML declaration (if present) and before the first element, the *document type declaration* declares the document type that is in use in the document. A "book" document type, for example, might be made up of chapters, while a letter document type could be made up of element types such as ADDRESS, SALUTATION, SIGNATURE, and so forth.

The document type declaration is at the heart of the concept of *structural validity*, which makes applications based on XML robust and reliable. It includes the markup declarations that express the *document type definition (DTD)*.

The DTD is a formalization of the intuitive idea of a document type. The DTD lists the element types available and can put constraints on the occurrence and content of elements and other details of the document structure. This makes an information system more robust by forcing the documents that are part of it to be consistent.

31.7 | Markup miscellany

This section contains information on some more useful markup constructs. They are not as important or as widely used as elements, attributes and the XML declaration, but they are still vital parts of a markup expert's toolbox.

31.7.1 *Predefined entities*

Sometimes when you are creating an XML document, you want to protect certain characters from markup interpretation. Imagine, for example, that you are writing a user's guide to HTML. You would need a way to include an example of markup. Your first attempt might be to create an `example` element and do something like this:

Example 31-11. An invalid approach to HTML examples in XML

```
<p>HTML documents must start with a DOCTYPE, etc. etc. This
is an example of a small HTML document:
<example>

  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
  <HTML>
  A document's title
  <H1>A document's title</H1>
  </HTML>

</example>
```

This will not work, however, because the angle brackets that are supposed to represent HTML markup will be interpreted as if they belonged to the XML document you are creating, not the mythical HTML document in the example. Your XML processor will complain that it is not appropriate to have an HTML DOCTYPE declaration in the middle of an XML document! There are two solutions to this problem: predefined entities and CDATA sections.

Predefined entities are XML markup that authors use to represent characters that would otherwise be interpreted as having a special meaning, such as a start-tag or an entity reference. There are five *predefined* ("built-in") entities in XML. These were included precisely to deal with this problem. They are listed in Table 31-1.

Table 31-1 Predefined entities

Entity reference	Character
&amp;	&
&lt;	<
&gt;	>
&apos;	'
&quot;	"

Why these specific five characters?

Spec. Reference 31-15. Predefined entities

The ampersand character (&) and the left angle bracket (<) may appear in their literal form only when used as markup delimiters, or within a comment, a processing instruction, or a CDATA section. [...] If they are needed elsewhere, they must be escaped using either numeric character references or the strings "&" and "<" respectively.

Spec. Reference 31-16. Attribute values

To allow attribute values to contain both single and double quotes, the apostrophe or single-quote character (') may be represented as "'", and the double-quote character (") as """.

An entity for the right angle bracket is also provided because it is sometimes useful to avoid putting a special string called *CDEnd* (discussed later) into your document. But you do not have to use this entity in most cases.

We can use references to the predefined entities to insert these characters, instead of typing them directly. Then they will not be interpreted as markup:

When your XML processor parses the document, it will replace the entity references with actual characters. It will not interpret the characters it inserts as markup, but as "plain 'ol data characters" (character data).

Example 31-12. Writing about HTML in XML

```
<p>HTML documents must start with a DOCTYPE, etc. etc. This
is an example of a small HTML document:
<EXAMPLE>
   &lt;!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
   &lt;HTML>
   &lt;HEAD>
   &lt;TITLE>A document's title
   &lt;/TITLE>
   &lt;/HEAD>
   &lt;/HTML>
</EXAMPLE>
```

31.7.2 *CDATA sections*

While predefined entities are convenient, human beings are not as good at decoding them as computers are. Your readers will get the translated version, so they will be fine. But as the author, you will spend hours staring at character entity references while you are editing your XML document. You may also spend hours replacing special characters with character entity references. This can get annoying.[1]

Another construct, called a CDATA section, allows you to ask the processor not to interpret a chunk of text as containing markup: "Hands off! This isn't meant to be interpreted." CDATA stands for "character data". You can mark a section as being character data using this special syntax:

Example 31-13. CDATA section

```
'<![CDATA[' content  ']]>'
```

Here are some examples:

As you can see, it does not usually matter what you put in CDATA sections because their content is not scanned for markup. There is one obvious exception (and one not-so-obvious corollary). The string that ends the CDATA section, "]]>" (known as *CDEnd*) cannot be used inside the section:

```
<![CDATA[
 JavaScript code: if( a[c[5]]> 7 ) then...
]]>
```

1. This is especially nasty when you are writing an XML book, where examples tend to contain may angle brackets.

Example 31-14. Writing about HTML in a CDATA section

```
<![CDATA[
<HTML>
This is an example from HTML for Dumbbells!
<p>It may be a pain to write a book about HTML in HTML,
but it is easy in XML!
</HTML>
]]>
```

Example 31-15. Java code in a CDATA section

```
<![CDATA[
if( foo.getContentLength() < 0  && input = foo.getInputStream() )
     open = true;
]]>
```

The first occurrence of CDEnd in the middle of the JavaScript expression will terminate the section. You simply cannot use a CDATA section for content that includes CDEnd. You must end the section and insert the character:

```
<![CDATA[
 JavaScript code: if( a[c[5]]]]>><![CDATA[ 7 ) then...
]]>
```

This is quite painful and can cause a problem for embedding programming languages. But even in those languages, CDEnd is probably a fairly rare character string, so you should just keep an eye out for it.

The non-obvious corollary is:

Caution *CDEnd ("]]>") should only be used to close CDATA sections. It must not occur anywhere else in an XML document.*

This is an absolute requirement, not just a recommendation. Because of it you can easily check that you have closed CDATA sections correctly by comparing the number of CDEnd strings to the number of sections. If you do not close a CDATA section correctly, some of your document's markup may be interpreted as character data. Since ("]]>") is not something that typical documents contain, this restriction is rarely a problem.

With all of these warnings, CDATA sections may sound tricky to use, but they really aren't. This book, for example, has several hundred. Mistakes involving CDATA sections are usually quite blatant, because either markup will show up in your rendered document, or data characters will be interpreted as markup and probably trigger an error message.

Predefined entities and CDATA sections only relate to the interpretation of the markup, not to the properties of the real document that the markup represents.

31.7.3 *Comments*

Sometimes it is useful to embed information about a document or its markup in a manner that will be ignored by computer processes and renditions of the document. For example, you might insert a note to yourself to clean up the wording of a section, a note to a co-author explaining the reason for a particular section of the document, or a note in a DTD describing the semantics of a particular element. This information can be hidden from the application in a *comment*. Comments should never be displayed in a browser, indexed in a search engine, or otherwise processed as part of the data of the real document. They may, however, be treated as metadata.

Example 31-16. A comment

```
<!-- This section is really good! Let's not change it. -->
```

Comments consist of the characters "<!--" followed by almost anything and ended by "-->". The "almost anything" in the middle cannot contain the characters "--". This is a little bit inconvenient, because people often use those two characters as a sort of dash, to separate thoughts. This is another point to be careful of, lest you get bitten.

Spec. Reference 31-17. Comment

```
[15] Comment ::= '<!--'((Char - '-')|('-'(Char - '-')))*'-->'
```

Comments can go just about anywhere in the instance or the prolog. However, they cannot go within declarations, tags, or other comments. Here is a document using some comments in several correct places:

Example 31-17. Comments all over the place

```
<?xml version="1.0"?>
<!-- There is no other version yet! -->
<!-- Now on to the doctype ->
<!DOCTYPE EXAMPLE [
  <!-- This is a comment in the
  doctype declaration internal subset! -->
  <!ELEMENT EXAMPLE (#PCDATA)>
  <!-- This is a very simple DTD. -->
]> <!-- Here comes the "root" or "document" element. -->
<EXAMPLE>This is some character data.
<!-- That was some character data. -->
</EXAMPLE>
<!-- That's all folks -->
```

Markup is not recognized in comments. You can put less-than and ampersand symbols in them, but they will not be recognized as the start of elements or entity references.

Comments are a good place to describe the semantics of element types and attributes. So you might use a comment to tell other DTD maintainers and authors that an element type with a cryptic name like `p` is actually intended to model paragraphs and not (for example) British currency. Comments are not just about being helpful to other people. After all, even expert document type designers have a limited and imperfect memory. Some day even you will wonder exactly what it was you meant by a particular element-type name. The DTD comments will help. The job that you are saving might be your own!

31.8 | Summary

An XML document is composed of a prolog and a document instance. The prolog is optional, and provides information about how the document is structured both physically (where its parts are) and logically (how its elements fit together). Elements and attributes describe the logical structure. Entities describe the physical structure. To use a rough analogy, the entities are like a robot's body parts, the elements are his thoughts, and stylesheets and software provide his behavior.