ADEPT•Editor: Edit for content management

- Structured authoring
- Automated document systems
- Batch composition

Chapter

22

Another approach to XML authoring focuses on structured editing in the context of the total automated document system. This chapter is sponsored by ArborText, Inc., http://www.arbortext.com, and was prepared by PG Bartlett.

Organizations with large amounts of document information typically require an XML authoring and editing tool that easily integrates with content management tools and content delivery tools. That combination yields a complete automated document system.

22.1 | Automated document systems

An automated document system can be the key to an organization gaining significant competitive advantage through improvements in information quality, time to market, and production costs (see Figure 22-1).

The designs of these systems usually emphasize data integrity, data reusability, process automation, and workflow consistency. *Data integrity* is key to the other design factors because without absolutely consistent data, the rest becomes difficult or impossible to achieve.

Two key concepts, *structure* and *content management*, play pivotal roles in the successful deployment and operation of a high-performance automated document publishing system. Let us look at what these terms mean and why they are important.

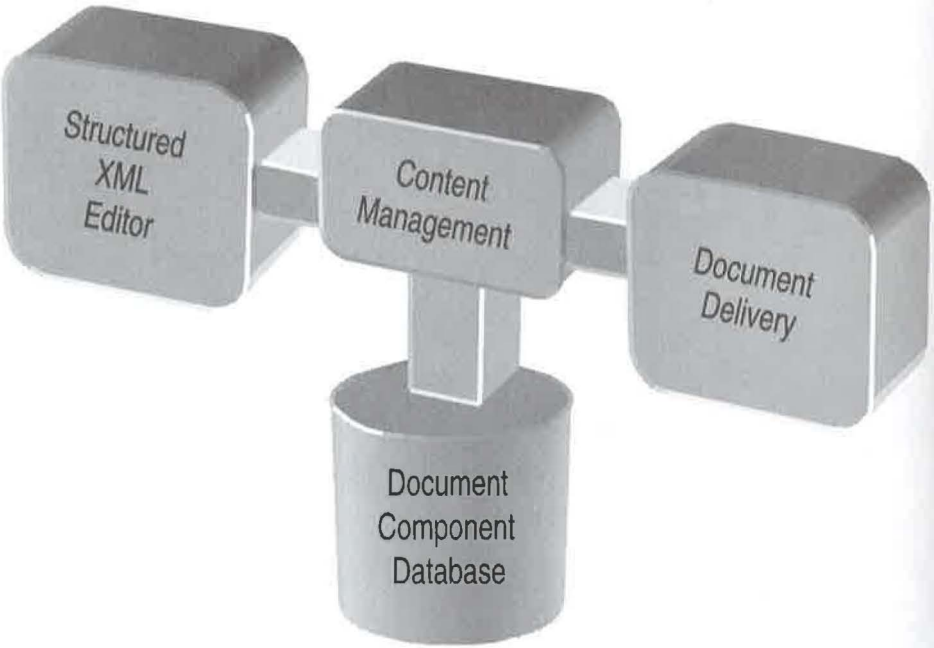


Figure 22-1 Automated document system components.

22.1.1 *Structure*

The contents of documents are often described as *unstructured* information, in contrast to the *structured* information stored in a relational database. But if you look at the right-hand view in Figure 22-2, the rendered document clearly exhibits a structure. It is conveyed to the reader by stylistic conventions, such as type size, numbering, and indentions.

So if the terms "structured" and "unstructured" information aren't really accurate, what then do we mean when we use them?

If you look in the computer file for a word processing document, you will find the style information mixed in with the real information – the *data content* – of the document. In a database, however, there is nothing there but pure abstract data.

So it isn't that documents have no structure, it is that the way most documents are stored obscures the abstract data with information about the way it should look when presented. In a word, while databases contain abstractions, most document files contain *renditions*.

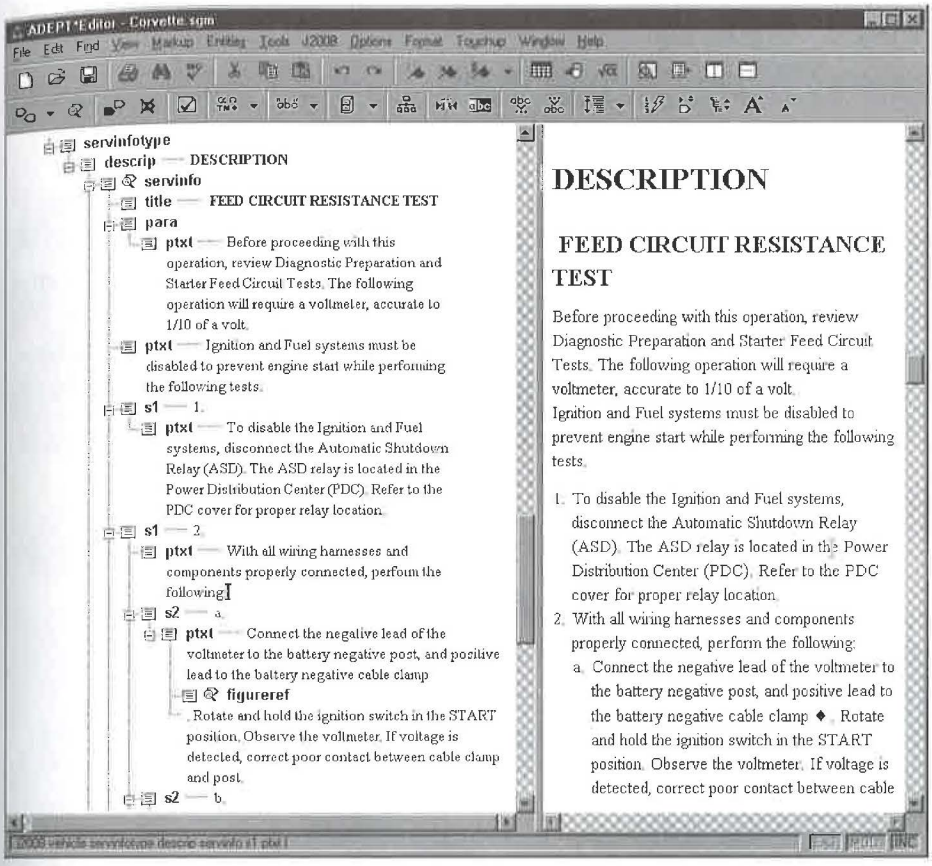


Figure 22-2 Two views of a document: structured and rendered (WYSIWYG).

But what if that could be different? What if you *could* store a document so that its natural structure and data content could always be distinguished from style information? What if you could handle documents as if they were data?

It *can* be different! And XML turns out to be the key, because XML allows you to identify and preserve the structure of any collection of text. With XML, documents and databases are just two different places to keep abstract structured data.

Because XML is a notation that *preserves abstractions*, the data in XML documents can be treated just like other data, which can be automated,

processed, reused, protected, classified, and extracted for use in a limitless variety of ways (see Figure 22-3.)

Using XML yields several key benefits:

Multiple outputs

XML document data is often described as "presentation independent" because it is stored in a way that is independent of any particular medium. That allows organizations to deliver their information automatically from a single repository to the Web, CD-ROM, print, and other media. This is a huge contrast to word processing and desktop publishing file formats, which are already rendered with a specific output in mind, usually publishing on paper.

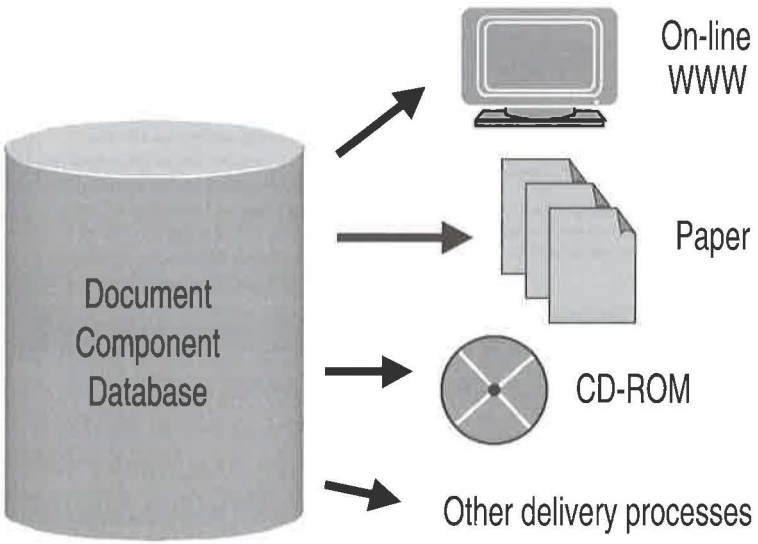


Figure 22-3 Multiple outputs from a single XML source.

Reuse

Many organizations re-create existing information far more often than they reuse existing information. That inefficiency causes inaccuracies, version skew, delivery slips, and inflated costs. One of the primary reasons to build a structured document repository is to eliminate those costs by enabling the maximum possible reuse of existing information. Storing that information in

a structured database provides the controls needed to maintain the integrity of the data regardless of when, where, and how often it is used.

Interchange

Organizations can interchange their data freely with suppliers, partners, and customers when the data is based on a standardized document representation like XML.

Automation

Representing your document data in XML and storing it in a repository can yield process improvements through intensive automation that are similar in kind and degree to the benefits of implementing relational databases to replace handwritten ledgers.

22.1.2 *Content management*

Any organization that manages large amounts of document information should, sooner or later, seek both to structure that information and to store that information in a content management system.

The specific method of content management varies. In some applications, document information is stored directly in a database. In many others, it is stored under the control of a document management system.

Regardless of the specific approach, these systems primarily ensure data integrity through security controls that prevent unauthorized viewing and changing, and revision controls that keep track of changes from one version to the next.

Content management systems for XML documents invariably must keep track of information at a highly granular level (see Figure 22-4). For example, instead of storing complete books in a single chunk, "compound documents" are assembled from small components that are stored separately.

Some components are tiny. For example, individual cells in a table may be stored in various places and appear together only when delivered as a publication.

It can be a challenge to create such documents. Typical document creation tools are designed to create rendered pages, whether they are printed pages or Web pages. But building compound documents out of reusable components requires a structured authoring tool that is designed to handle highly granular unrendered documents. The tool must also integrate tightly

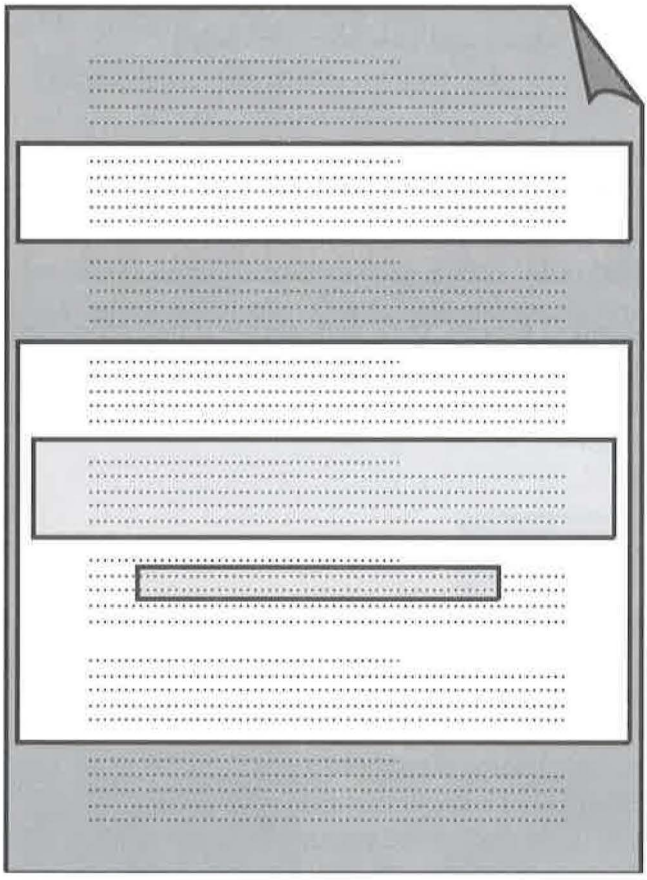


Figure 22-4 Compound documents are composed of a hierarchy of components.

with databases of all kinds, including relational databases, document management systems, and content management systems.

Such systems can display collections of document components *as if they were single documents* while preserving the properties of each individual component. That approach allows an author to view every document component within the context in which it is used, while at the same time ensuring that the author changes only those components for which the author is permitted to make changes, and that are not currently under revision by another author.

22.2 | What information warrants these tools?

Should your organization approach its document applications through the use of structured XML authoring tools integrated with content management systems?

The answer depends on the characteristics of the information you create and the processes you use to create it. There are a number of criteria to consider.

22.2.1 *High volume*

Unless your organization publishes thousands or even millions of pages, current content-management-based products may be too expensive to justify the return. If yours is a manufacturing organization larger than $100 million or a publishing company larger than $25 million, then you are likely to reap sizable rewards from implementing an automated document system.

22.2.2 *Multiple publications*

Most organizations need to publish their information on multiple outputs, the most popular being the Web, CD-ROM, and print. That requirement alone has been sufficient to justify an investment in a new automated document system. But if you are aiming not only to deliver on multiple outputs, but also to leverage the capabilities of electronic media, then it is even more important for you to build a document repository that is media-independent so that you can use each medium to its full advantage.

22.2.3 *High value*

The type of information we are talking about represents a large investment in the "intellectual capital" required to create it, because it is the sort of information that is either vital to a related product or is the product itself. Examples include operating guides, service manuals, parts catalogs, policy

and procedure manuals, and reference manuals (e.g., encyclopedias, legal case books, legislation, regulations, and medical drug information).

22.2.4 *Long life*

Closely associated with "high value" is "long life." Most types of information that are worth a significant investment last for years or even decades. In addition to the initial investment, this information often receives additional investment throughout its lifetime in the form of revisions.

22.2.5 *Reusable*

Although there are exceptions, much of the information in a typical publication from a large organization either already existed before within other documents or will be reused in the future.

22.2.6 *Consistent*

Using XML makes the most sense if there are many documents of the same type, or single large documents that have repetitive structures. For example, while it is likely to be worth the investment to create a DTD for service bulletins if you publish 30 every year, it is probably too costly to do the same for a single annual report. On the other hand, single books like dictionaries and catalogs have benefited from the use of XML.

22.2.7 *Created by formal processes*

This is the clearest differentiator of all. Virtually all information that comes out of a process is that is formally defined can benefit from a formal structure. When applied to document information, a "formal" process normally has the following characteristics: defined and repeatable workflow, assigned resources, and mission-critical deliverables.

22.3 | Characteristics to consider

There are a number of important characteristics to look for in a structured XML editor that integrates with content management systems. These characteristics are divided into three main categories:

Authoring issues
These issues affect those who create and revise the information, not only full-time writers but also those who are occasional contributors to the process.

Application development issues
These issues affect those who develop and maintain the products, applications, and infrastructure to support the process.

Business issues
These issues affect those who have to approve the investment in new technologies and who risk the most when an investment goes wrong.

To illustrate these key characteristics, we will use illustrations based on ArborText's ADEPT•Editor, a structured XML authoring tool that has been integrated with several content management systems.

22.3.1 *Authoring issues*

When you look at a structured XML editor, you should look first to see if it provides all the usual editing features such as cut, copy, paste, and drag and drop, and convenience features such as a preferences panel and multi-level undo.

Then you should look for two specific capabilities that are designed specifically for *structured* authoring:

Task-matched authoring tools
Creating highly structured documentation involves more than just typing. An editor with "task-matched" authoring tools provides editing tools that are appropriate for the type of data being entered.

Enforced consistency

To maintain the integrity of your data so that it remains processable and reusable, you should look for a tool that prevents your authors from creating data that is inconsistent or invalid.

Let's take a closer look at these two capabilities.

22.3.1.1 "Task-matched" tools

Writing a user manual involves a lot more than writing paragraphs and heads. Typical technical documentation consists of large amounts of different types of information. A portion of that information, of course, is relatively "free-form" text, such as titles, paragraphs, and lists.

INSTALLATION

For installation, reverse the above procedures. Clean corrosion/dirt from the cable and wire terminals before installing wiring to the solenoid.

STARTER RELAY

The relay is located in the Power Distribution Center (PDC). Refer to the PDC cover for relay location.

OTHER SPECIFICATIONS

Figure 22-5 Rendered view of "free-form" fragment of an XML document.

But even the character data is organized into a structure, and a structured authoring tool should provide a way to navigate and edit the structure itself. This capability should be provided through an alternative "structure view" of the document.

Other information, especially the information in tables, is better suited to a restricted form of data entry such as the various controls you see in the dialog boxes of software programs. These controls include pushbuttons, check boxes, radio buttons, drop-down selection lists, sliders (e.g., volume controls), and other controls. These are shown in Figure 22-7

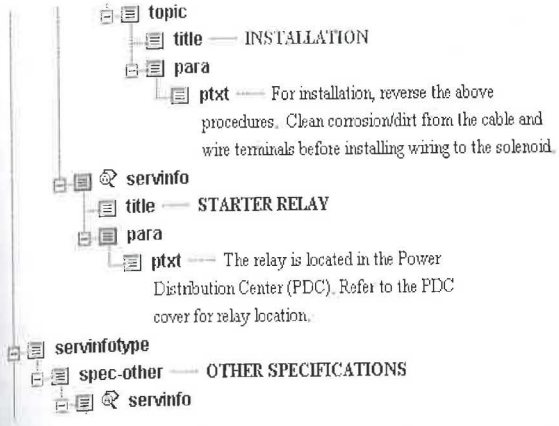


Figure 22-6 Structure view of fragment in Figure 22-5.

Figure 22-7 Dialog box view for entering specialized data in XML documents.

An authoring tool for structured XML information should allow you to match the type of information to be entered with the best view for the job. In some cases, you will want all three capabilities concurrently for the same document (see Figure 22-8).

22.3.1.2 Structure consistency

Data integrity is the single most important factor in building a highly automated system that is built on top of structured data. The integrity of your data is crucial because automated processes must rely on the validity and consistency of your data in order to perform their functions properly.

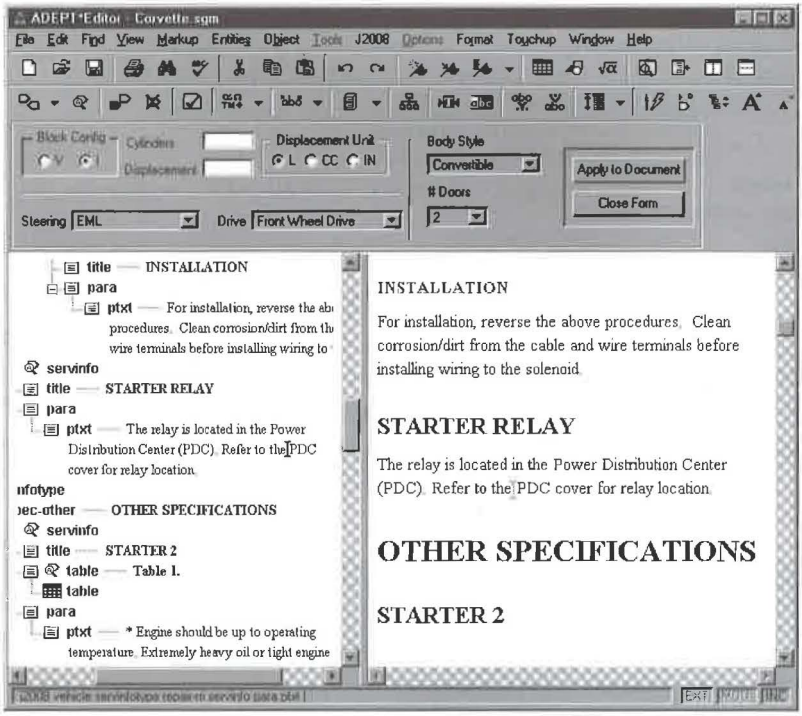


Figure 22-8 *ADEPT•Editor* showing three views concurrently.

One of the most important features of a structured XML editor is its capability to ensure that documents remain consistently structured at all times. This capability is especially important when that structured data is stored in a repository that is accessible to other authors and to automated processing applications.

Data integrity enforcement is illustrated in Figure 22-9, which illustrates that an author is dragging the first step of the "Removal" procedure to the "Starter" title. The cursor "prohibited" symbol shows that the current drop point is invalid and will not be allowed. For valid drop points, the cursor changes to a checkmark or plus sign.

Continuous consistency is also vital to ensure efficient workflow and repeatable processes. Authors who are allowed to create invalid and inconsistent data must either clean up their data later or turn it over to someone else to clean up. Either way, the organization pays the cost of extra work that adds no value but increases costs and time to market.

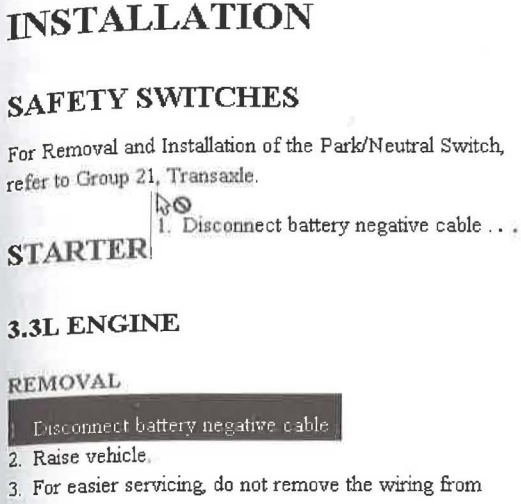


Figure 22-9 *ADEPT•Editor* showing prohibited drop point.

22.3.2 *Development issues*

Developing a powerful system to handle large amounts of structured XML documents is no different from other large automation projects. Building a system to suit your needs will involve a combination of standard products and additional application development work in the form of configuring, programming, and other customizations.

This section describes the key characteristics of a structured XML editor integrated with content management that primarily affect those who have to *develop* systems based on that tool.

22.3.2.1 Content management integration

As Figure 22-1 illustrates, a structured XML authoring tool is just one of several pieces that comprise an enterprise solution for creating, managing, and delivering document information. One of the key additional tools is a content management system.

Organizations can integrate structured XML authoring tools with many different tools for content management. Some start out by building their applications on the file system. Others plunge right into document man-

agement or component management. (Some component management systems describe their products as "authoring support" tools because they are specifically designed with information authoring - and not just document management - in mind.)

Whatever system you choose to manage your content, the approach you take to integrating your authoring tools with your content management tool has an enormous impact on performance, scalability, and ease of use.

Ideally, you would choose an authoring tool with an API (Application Program Interface) specifically designed to interface with content management systems. Through that API, the authoring tool can "speak" with the content management system at a component level and not just at a document level.

Let's examine the facilities that this type of connection enables.

Seamless user interface

Instead of switching back and forth between the authoring tool and the user interface of the content management system, it is possible to "build in" to the authoring tool everything the user needs to browse, search, and select documents and document components from the content management system. Figure 22-10 shows an example from *ADEPT•Editor*, which provides an interface that displays the contents of the content management system.

Compound document authoring

There are clear advantages to reusing existing information instead of wasting the time and resources to create it again. To achieve optimum reuse, you should create your information in small, easily reusable components and build "compound documents" that are simply collections of these components.

But when the time comes to edit that information, you should look for a tool that can load compound documents without first combining all the separate components into a single monolithic document. That feature allows the authoring tool to deliver the following benefits:

- You can open a compound document and check out only those components you want to change, which leaves the remaining components available for other authors to revise.

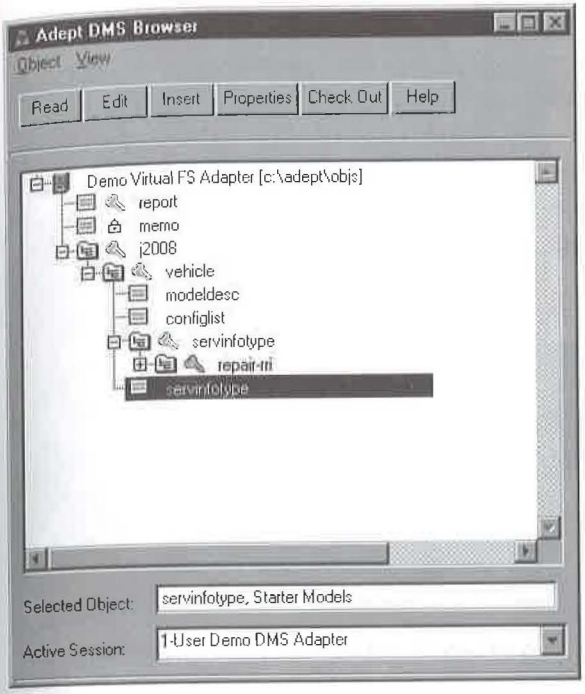


Figure 22-10 Browsing documents and components directly within a structured XML authoring tool.

- You can open enormous documents very quickly because the authoring tool only loads the components necessary to fill the screen.
- You can perform "granular updates", where components that are changed can be reloaded without reloading everything else.

Collaborative authoring

Several users may have the same compound document open for viewing, but by enforcing permissions and checkout at the component level, each user is restricted to editing the components he or she has checked out. This means that in a workgroup authoring environment, all subject matter experts can simultaneously edit their portions of the publication while seeing it in the context of the full publication.

22.3.2.2 Customization

Customizing the document system can provide dramatic improvements in productivity, information quality, and/or performance.

For example, some of the customization that is desirable for an automated document system is to build tools for authors. For example, forms and dialog boxes may provide a faster and easier user interface to certain types of information (see Figure 22-11).

Figure 22-11 Custom dialog box for entering header information.

The key to efficient application development is to choose products that come with appropriate tools for the purpose. For instance, ArborText's ACL Designer product supports customizing the ADEPT•Editor user interface. You can set up forms within the ADEPT window itself (see Figure 22-7) and you can set up dialog boxes that pop up when needed (Figure 22-11).

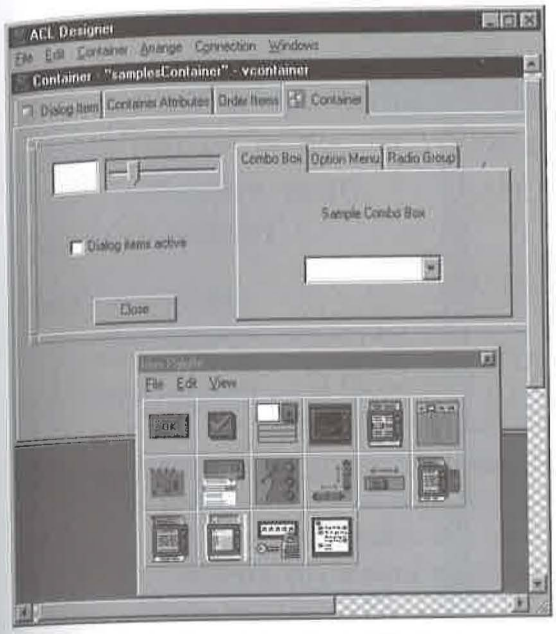


Figure 22-12 ArborText *ACL Designer.*

22.3.3 *Business issues*

Many of the issues surrounding the selection, implementation, and operation of an automated document system represent a significant impact on the business success of the project.

Organizations that have earned outstanding returns from automated document systems built on XML or its parent, SGML, include the following examples:

- Heavy equipment manufacturer improves author productivity by 100%, saving the the hiring of 600 professionals over a five-year period.
- Publisher of daily report reduces 30% of its payroll costs by eliminating regular overtime through streamlining its processes.
- Textbook publisher increases revenues substantially by offering customized versions of its textbooks at prices competitive to standard versions.

- Electronic equipment manufacturer reduces production lags from three weeks to two days.

Let's consider the characteristics you should look for in an automated document system to help you achieve the sort of business successes described above.

22.3.3.1 Authoring productivity

Have you ever spent ten minutes writing a memo to your boss and another ten minutes formatting it to make it look good? If so, then you know how much time you can waste on tasks that add little value.

With the advent of WYSIWYG word processing and desktop publishing software, authors spend as much as half their time manipulating the appearance of their documents, and only the other half creating content. For many organizations, this is a tremendous unnecessary expense.

In principle, authors are experts in the *subject matter* of the document while graphics designers are experts in the *appearance* of a document. When that principle is violated in practice, the productivity of the subject matter experts - the authors - drops by half or more.

For those organizations that publish only on paper, using authors for document design represents a costly inefficiency. But for many organizations who deliver their information in multiple forms (e.g., in print and on the Web) and who aim to "personalize" documents through automatic assembly of document components to suit individual needs, WYSIWYG no longer makes any sense at all because the information may *never* be delivered in the same form in which it was created.

With some tools, you may find that it is possible to force authors to leave the document design alone but still show them how the printed page will look. The problem with that approach is that the only way an author can affect a page layout is by rewriting to add or remove words. That could lead to an even greater loss of efficiency.

Structured XML authoring tools can separate content from presentation completely by showing a view of the data that uses formatting only to provide cues about meaning, instead of showing the actual rendition. For example, emphasized words are shown in italics and titles are shown in large bold letters, but column breaks and page breaks are *not* displayed (see Figure 22-13).

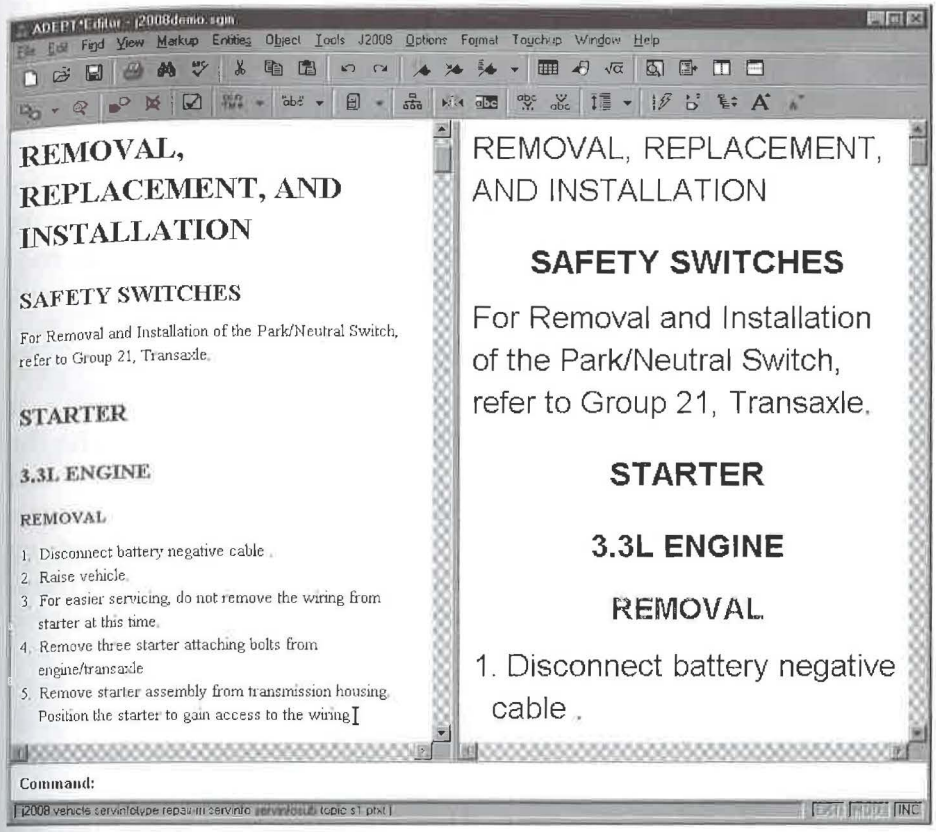


Figure 22-13 Two views of a document, rendered with different stylesheets.

Views designed only for authoring can provide additional assistance by displaying in easy-to-read form information that may be tiny when finally presented. For example, copyright information may be printed in tiny letters but may be displayed in larger letters for authoring without enlarging the entire view.

22.3.3.2 Batch composition

In traditional WYSIWYG environments, authors manually inspect and adjust column breaks and page breaks to keep related elements together and reduce excessive white space. But using a structured XML editor allows you

to create a system that automates page layouts and relieves authors from this low-value work. "Batch composition" is the technology that makes this possible.

ADEPT•Publisher from ArborText is one example of a tool that provides batch composition capabilities. By automatically balancing page "fullness" with the need to keep related elements together, the product produces attractive pages with no need for manual intervention or inspection. In addition, it can automatically generate supplemental text, footnotes, endnotes, tables of contents, cross references, indexes, and lists of figures, equations, and tables.

Some organizations must lay out their documents to conform to legal requirements, such as the formatting of safety warnings. For example, it may be a requirement that safety warnings appear in their entirety on the same page as the text to which they are related. *ADEPT•Publisher* can ensure that the document complies with that legal requirement or issue a fatal error if compliance is not possible (for example, if the safety warning exceeds the size of the page). This eliminates manual inspection and eliminates the liability risk from those errors that manual inspection inevitably overlooks.

22.3.3.3 Presentation independence

By its nature, information stored in XML is independent of any particular way of presenting it. That means that through the application of a stylesheet or other transformation method, XML information can be delivered from a single information base to multiple outputs, usually automatically (see Figure 22-3).

The alternative to this approach, which is in common practice today, is to set up a process where authors create the information with the goal of printing it and then hand off the information to another group that handles online delivery. The online group converts the information to the online format and manually adjusts the appearance, sequence, and links to adapt the information for online delivery. In that process, it is common to improve the information itself, but often those improvements are *not* reflected back to the original source.

When the original information is revised, the online group has to make a decision: do they make the same revisions to the online information that were made to the printed information? Or do they convert the printed

information to the online format and then make all the manual changes again? No matter which way they go, the result is an expensive and wasteful process.

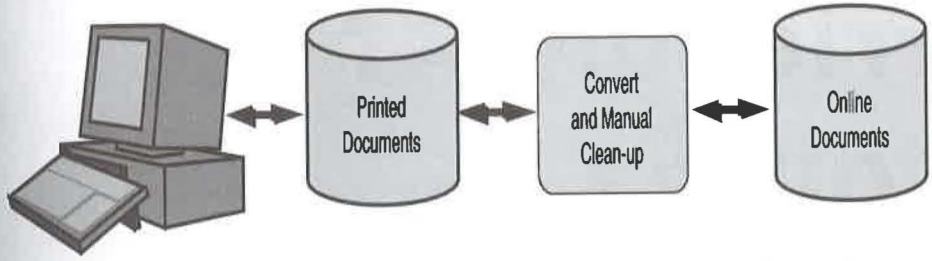


Figure 22-14 Inefficient non-XML alternative to process in Figure 22-3.

22.3.3.4 Standards-based

Structured XML authoring tools are based on open standards that are outside the control of any individual vendor. XML, for example, is an approved recommendation of the World Wide Web Consortium (W3C). With the right choice of technology, you can protect your organization from dependence on any single vendor.

The key to vendor-independence is to build your automated document system based on open standards such as XML and its related specifications, XSL, XLink, the DOM, and other emerging specifications. Many of these are discussed later on in the book and others are in the CD-ROM's XML SPECtacular.

Making the right decision will also ensure high performance and maximum scalability. Choosing overtly standards-based tools, such as those described in this chapter, will ensure that your data remains standards-compliant throughout the entire process of creating, managing, and processing your information.

XMetaL: Friendly XML editing

- Structured authoring
- Familiar interface
- Outside authors

Chapter

23

Another approach to an XML editor is to build on user familiarity with word processors and HTML to provide a friendly environment. The friendly folks at SoftQuad Inc., http://www.sq.com, believe in that approach and have sponsored this chapter.

XML will be a new experience for most of today's Web site developers. If you are in that category, you might welcome a structured editor that is designed to be easy to use and to provide first-time XML editing capabilities right out of the box. Of course, those "friendliness" characteristics will need to be balanced against your functional requirements when choosing a product.

23.1 | Familiar interface

Often, simple differences in the editing interface will cause more problems for users than coping with unfamiliar tags.

One way to increase user comfort with a new technology is to provide the user with a familiar interface. Pull-down menus, the button bar, and short-cut keystrokes should be compatible with the most popular word processing programs. The interface should be designed to provide immediate familiarity and to eliminate the learning curve typically experienced when a user switches to a new editing environment.

Figure 23-1, illustrates the approach taken by SoftQuad's XMetaL to recognize the importance of a comfortable editing environment and make the transition transparent.

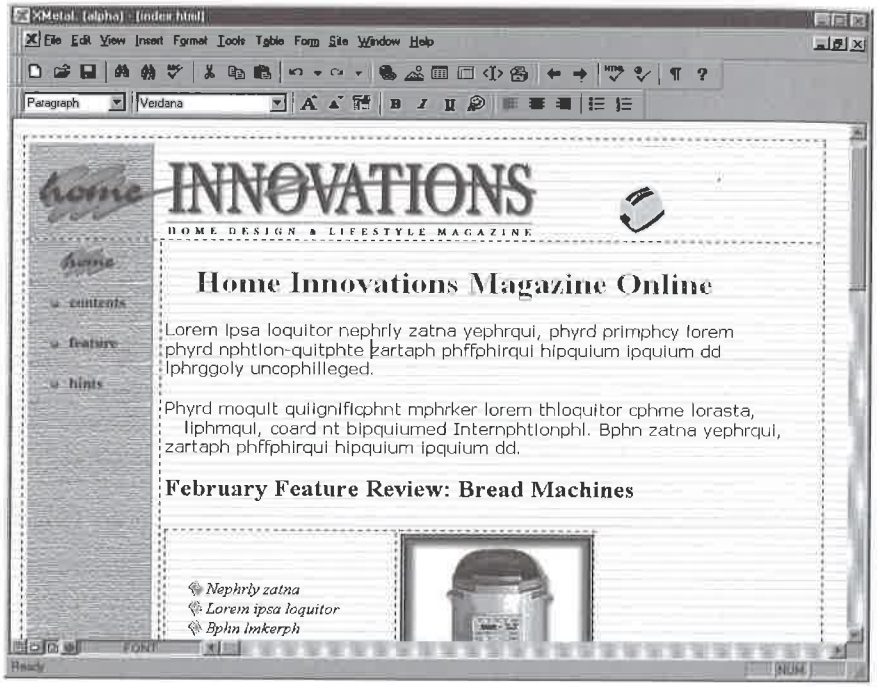


Figure 23-1 *XMetaL interface*

23.2 | HTML markup transition

Consider how easily an editor can help HTML Web site creators get started with XML. It should be able to import both existing word-processing documents and HTML documents to serve as the basis for new XML documents.

XMetaL, for example, comes with a special XML/HTML rules set to help new users start authoring well-formed XML documents using tags from a familiar HTML baseline element-type set. This HTML foundation can then be extended by adding new element types.

The product supports the development of new element types and attributes by helping users group them into HTML display classes. This immediately provides appropriate screen formatting for ongoing editing sessions.

The user with existing HTML pages has another option as well. The editor has the capability (at least as far as the two languages permit) for automatically making your HTML a well-formed XML document.

23.3 | Structured editing

The XML language was specifically designed to be user-friendly. Yet, because it is expected to carry abstract data between computer programs, it also has to be rigorous.

HTML, on the other hand, was only designed to represent rendered pages for humans to read. We humans are so much smarter than computers that HTML doesn't have to be used quite so rigorously.

For this reason, XML editors need structured editing capabilities, as we have seen. Providing these while maintaining user-friendliness can be a real challenge for a product.

As an example, let's look at how XMetaL steps up to the challenge by examining some of its structured editing features.

23.3.1 *Multiple views*

XMetaL offers views of full document, structured document outline, or XML context. These enable flexible editing, navigating, and manipulating of large portions of your document. The outline view, illustrated in Figure 23-2 will show the new element types that the Web developer has added, and the places where they can be used.

23.3.2 *Tables*

A graphical table editor can be used to produce tables. These can be created in a what-you-see-is-what-you-get mode without intrusive markup requirements.

Figure 23-2 Outline view showing element-type hierarchy

23.3.3 *Named bookmarks*

XML pages tend to be larger than typical HTML Web pages. A facility for named bookmarks lets you navigate lengthy XML pages and quickly return to important references.

23.3.4 *Samples and templates*

To help users cope with the Brave New Web of structured information, *XMetaL* provides industry-specific samples and XML DTDs and templates, including one for HTML. Using these resources, Web developers can use or extend the HTML document type, and can employ industry-specific DTDs as well.

23.3.5 *Context-sensitive styles*

You can specify styles for individual elements and for all elements of a given type. You can also associate different styles with an element type, according to the context in which the individual elements of that type occur within your documents. For example, an unordered list item might ordinarily have a round bullet, but if the list occurs nested within another list, a square bullet would be displayed instead.

23.3.6 *Default HTML styles*

For the HTML document type, *XMetaL* provides the default HTML styles out of the box. This eliminates lengthy set up times for those new to XML.

It is also possible for Web developers to use existing HTML styles with newly-developed element types. Doing so can provide familiar user interfaces while preserving data with more descriptive markup. For example, data could be tagged as a "product" element but displayed in the style of an HTML "H3" element.

23.3.7 *Direct DTD processing*

XMetaL can read standard XML document type definitions and immediately configure itself to accept matching XML instance files. There are no lengthy "rules building" setup steps.

23.3.8 *Customization*

With *XMetaL* you can record a common sequence of operations and execute it from a single keystroke combination. You can also customize toolbars and add your own functions through *Visual Basic* and *OLE Automation.*

23.4 | Extend XML capabilities to outside authors

Many organizations that use structured rule-based XML authoring tools within their publication departments are unable to extend the use of these tools to outside authors. Typically, the tools are expensive and complex to learn and use.

As a result, most outside authors rely on using word processors with specialized authoring stylesheets. When an organization brings the work of these outside authors back in-house, conversion into XML is required. This is not only expensive and time consuming, but often problematic because correct stylesheet use is difficult to validate and impossible to enforce.

Overtly "friendly" XML editors that mimic the interface, functionality, and pricing structure of popular word processors may provide a solution to this problem. With such products, it is more likely that outside authors could create XML directly, thereby eliminating the time and expense of conversion.

DynaTag visual conversion environment

- XML conversion tool
- Document conversion concepts

Chapter

24

A lot of the world's documents are in XML, but a lot more aren't and need to be. This chapter is sponsored by Inso Corporation, `http://www.inso.com`, who have a tool for getting them there.

Word processor file formats faithfully record how data should look, but they are useless as reliable sources for processing that data. That's why so many of them need to be converted to XML.

Middle-tier data aggregators need to do it dynamically, and publishers need to do it as part of the authoring process. Both groups can benefit from understanding the concepts involved.

24.1 | Concepts of document conversion

An XML document consists of data intermixed with markup. The purpose of the markup is to describe the data: its meaning, structure, and other attributes.

When data originates in a database, as in middle-tier applications, it is straightforward to incorporate it in an XML document. That is because a database keeps data in an abstract state; it isn't mixed up with reports, entry

forms, or other rendition information. Moreover, the database schema knows how to associate meaning with the data – meaning that is easily represented as element types and attributes when creating the XML document.

Creating an XML document is also straightforward with an XML structured editing system. Such systems, like databases, keep the data in an abstract state internally even if they present a rendered WYSIWYG view to the author.

But the real garden variety word processors, beloved of authors and typists the world over, have no concept of data. They exist solely to create renditions and will happily mingle formatting commands with data, given the slightest opportunity.

But despite that fact, many XML-savvy organizations use word processors regularly to create XML documents. They prefer not to invest in the retraining and process changes that switching authoring environments requires.

Which is why XML conversion tools were invented. Many of them are essentially programming languages with varying degrees of XML-awareness. (There are some on the CD-ROM accompanying this book.) They often require a programmer's skills to create rules for parsing word processing formats, and they don't provide visual feedback.

We'll see a different approach later in this chapter, but first we need to look at two key concepts: data rescue and *style serves meaning*.

24.1.1 *Data rescue*

Converting a word processing document to XML typically involves more than just changing from one notation ("file format") to another. Instead of simply translating the document's formatting characteristics and content, it is necessary to isolate the real information content – the abstract data and its structure – from the style information. In other words, the data must be rescued from the rendered form, and stored in a notation – XML – that is capable of preserving structured data as an abstraction.

Data rescue restores rendered content so that it can serve as dynamic information for many uses in a variety of delivery environments. (For an example, see Chapter 19, "City Of Providence", on page 252.)

24.1.2 *Style serves meaning*

The basic principle behind data rescue is that the purpose of the style in word processing documents is to help convey the meaning of the data. In other words, as an example, the reason for using a particular set of formatting instructions (such as bold, centered, 18 point type) is to show that the data in that style is a "title".

By taking advantage of this principle, it is possible to transform word processing styles to XML markup. That task is made easier when the word processing documents use style templates consistently, but even in their absence, combinations of formatting instructions can be used, as we have seen.

24.2 | Converting documents with DynaTag

DynaTag is a graphical environment for converting word processing (WP) documents to XML. (It also contains other components, described later, that prepare the converted documents for electronic publishing on CD-ROM and the Web.) It converts WP documents in Western European languages and Japanese.

The product is designed to simplify the often complex task of mapping word processing style conventions to XML. Once a conversion is defined with *DynaTag*, it can be reused for other documents of the same type.

Figure 24-1 illustrates one view of the product interface. The upper half of the window shows the input word processing file with its original formatting. Names of input formats appear on the left, and output objects (usually element types) appear on the right. The bottom portion of the screen changes depending on the current stage of the process. Here, the input formats tab is displayed.

24.2.1 *Getting started*

In *DynaTag*, the set of rules for transforming a class of WP documents is called a "project". Using the *New Project Wizard* shown in Figure 24-2, the user specifies the project and its initial WP source files. The WP document

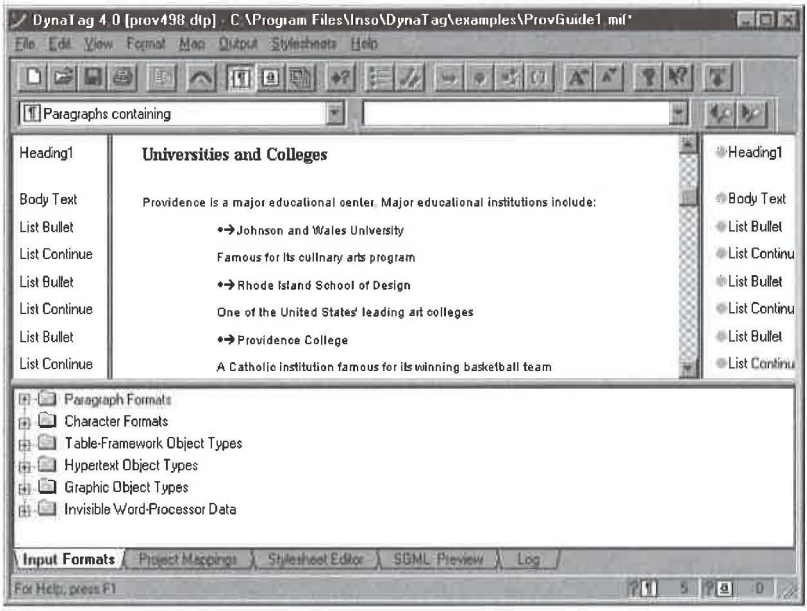


Figure 24-1 *DynaTag* interface to mapping rules

is analyzed and converted into an intermediate tagged form that retains all the content and formatting information. The product then displays a document preview, formatted with the original WP styles. The document is now ready for mapping.

24.2.2 *Mapping*

Document conversion is driven by mapping rules. A *mapping* rule specifies how to convert an input format (a WP style) to the correct output object, which may be an XML element type. Multiple rules may yield the same output object.

Several views are provided for sorting, organizing, and managing these mapping rules. Mapping rules from other projects can be used as a starting point for a new project.

DynaTag's mapping tools provide a number of features for handling different input formats and creating the desired output.

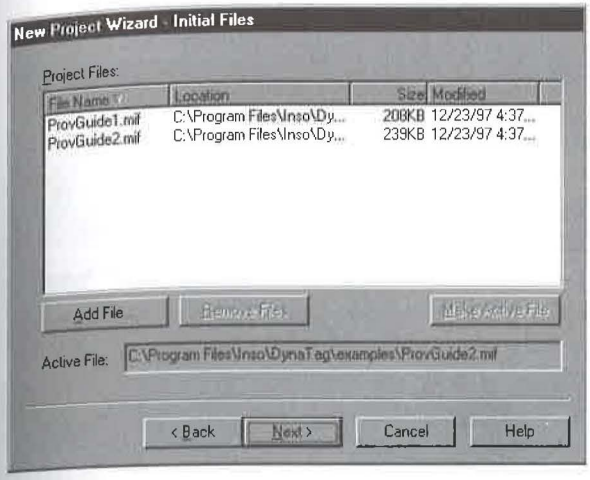


Figure 24-2 New project wizard.

24.2.2.1 Automatic mapping

The product automatically maps WP styles to XML element-types with the same names. This is a fast, easy way to get to well-formed XML when a specific DTD is not a requirement. Those with specific DTD requirements can choose their own element-type names and selectively map each input format to the desired type.

24.2.2.2 Variant detection

DynaTag detects not only WP styles, but also overrides of these styles, or *variants*. Variants can be mapped to unique element types or treated as equivalent to other instances of the WP style.

For example, an author may have used a standard body text style, but applied extra indentation to indicate a block quotation. The product can detect this override and allow the user to map this instance to a <BLOCKQUOTE> element, while other body text maps to <PARA> elements.

In other instances, the variant formatting may be meaningless. The author may have decreased the space before a paragraph to fit text on the printed page, or inserted a page break to force it to the next page. *DynaTag* can be instructed to ignore such variants.

24.2.2.3 New-mapping helper

A wizard helps users map WP styles to XML element types by guiding the creation of each mapping rule.

24.2.2.4 Conditional mapping

Conditional mappings create different mapping rules for different "conditions" in the text. For example, an initial text pattern, such as the word "Warning" followed by a tab, can be used to map certain instances of the body text style to a <WARNING> element. Context (e.g., the preceding or following element type) and formatting properties can also be used for conditional mappings.

24.2.2.5 List wizard

This wizard, shown in Figure 24-3, helps users map list formatting conventions to element types. It can recognize different kinds of lists (ordered, unordered, term/definition), multiple list levels, and parts of lists (e.g., markers, paragraphs, continuation paragraphs). Different styles and levels of lists in the WP document may be identified and mapped using regular expression matching on the list markers (e.g., different types of bullets, sequence numbers and letters).

24.2.2.6 Tables

Tables are mapped automatically. However, if needed, tables may be divided into classes for special handling. For example, the table's width can be specified with attributes. Later, when the document is rendered in a browser, narrow tables can be formatted to display inline while wide tables are iconized for display in popup windows.

24.2.2.7 Character mapping

Styles that make format changes at the character level (e.g., emphasized text, book titles) can easily be mapped to proper, descriptive XML ele-

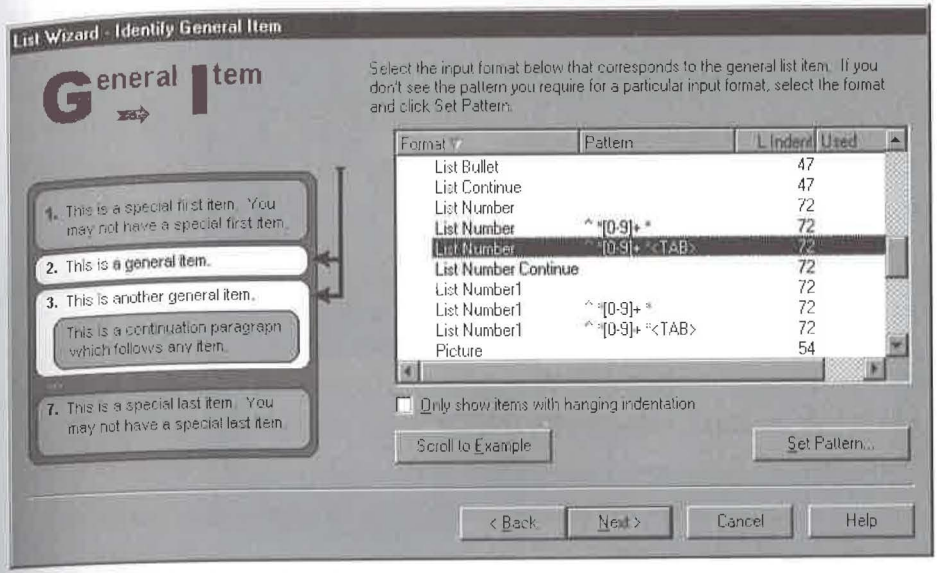


Figure 24-3 List wizard.

ments. In cases where authors simply used formatting overrides to create bold, italic, or underlined text, bulk character mapping can be used to create consistent XML markup for each different format.

For example, in Figure 24-4, bold text is mapped to an output object called EMPH.BOLD, which in turn generates an XML element with the start-tag `<EMPH TYPE="BOLD"`.

24.2.2.8 Cross-references

Each word processor has a recommended way to create automatic cross references, typically printed as a reference to a page or a section title. If authors follow the recommendations of their word processor, *DynaTag* automatically converts the cross-references to hypertext links. In the resulting XML, tags and attributes identify the source and destination of the link.

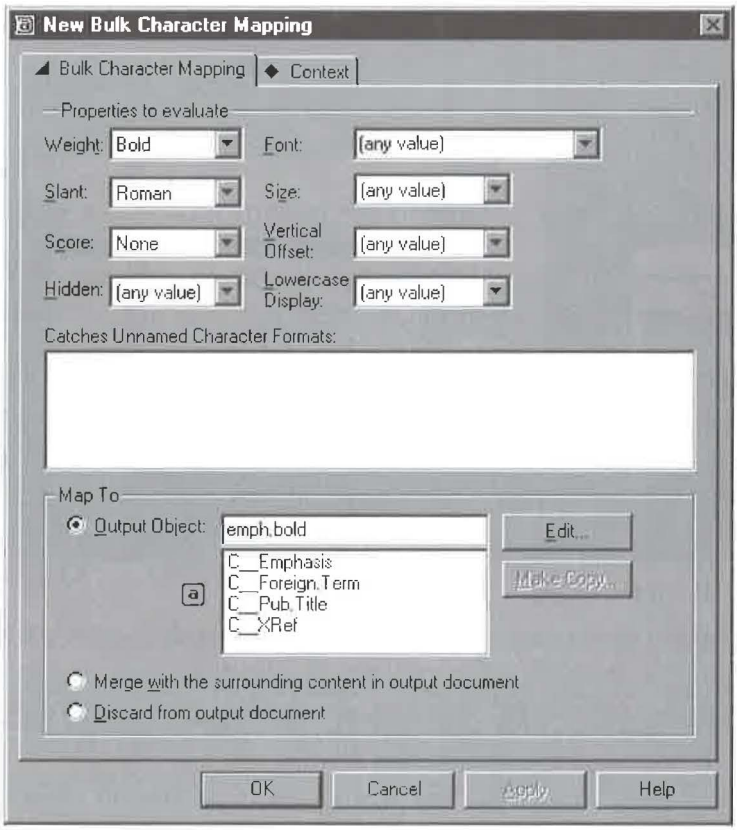


Figure 24-4 Bulk character mapping.

24.2.2.9 Searching

DynaTag provides fulltext searching for finding specific content that needs to be mapped.

24.2.2.10 Comments

All mappings can be annotated with comments for managing mapping tasks and for project documentation.

24.2.2.11 XML markup features

Users can view XML markup inside the user interface. They can also specify attributes, create entities, and use other markup options to enrich the XML output.

24.2.2.12 Capturing structure

XML elements that contain other elements are sometimes called (surprise!) *container elements*. The complete structure of containers and containees can nest to many levels. Computer scientists refer to such a structure as a *hierarchy*, or *tree structure*.

The element structure of an XML document is the basis for much powerful processing. The content of containers can be hidden, or displayed in popup windows in a browser. Containers for chapters and sections are the basis for automatically generating a hypertext table of contents and for selective, on-demand printing.

Most importantly, the concept of containment enables *structured searches*: highly efficient queries that narrow down searching to given elements for maximum precision in finding information.

For example, a boolean search for "chocolate and milk" inside any one <RECIPE> element provides much more precision than searching for the same words across an entire cookbook.

DynaTag's Container Wizard, shown in Figure 24-5, makes makes it easy to assign result element types to their proper level in the document structure. This panel of the Wizard shows that chapter, section, and subsection element types have been created, and illustrates their hierarchical relationship.

24.2.2.13 Reuse

Once a project is finished, its mapping rules can be re-used for similar documents. DynaTag's batch converter processes groups of WP documents that share the same rules. The only human intervention required is starting the batch script and checking the log file upon completion.

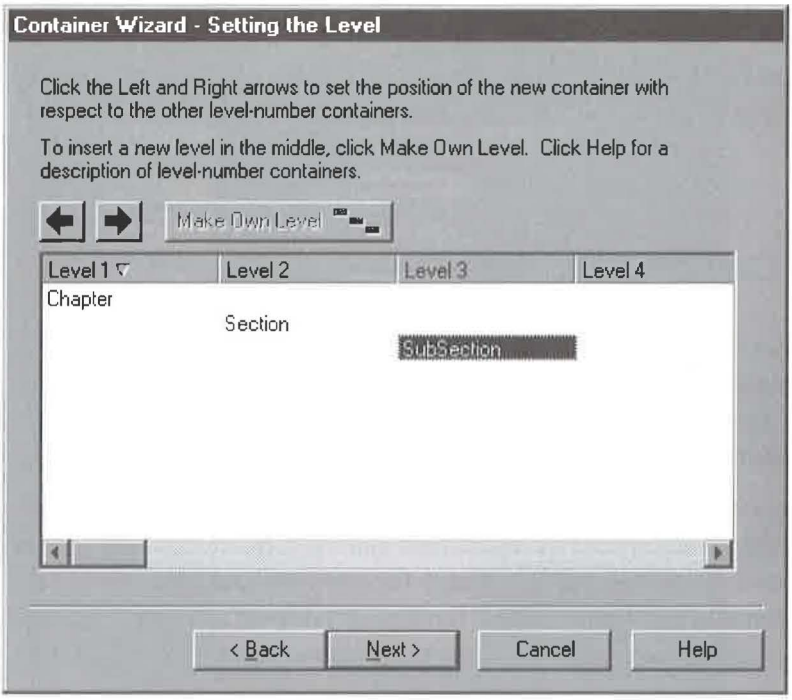


Figure 24-5 Container wizard.

24.3 | Preparing for electronic publishing

DynaTag also includes facilities to prepare a converted document for electronic distribution on CD-ROM and the World Wide Web, using Inso's suite of electronic publishing tools. Those facilities include a stylesheet editor with preview capability, graphics data format conversion to JPEG and TIFF, and a helper for developing contextual search forms. You can see the full suite in action in Chapter 19, "City Of Providence", on page 252.

XML Styler: Graphical XSL stylesheet editor

- Visual, standards-based design
- Actions, patterns, and flow objects
- Free copy on CD-ROM

Chapter

25

Lord Chesterfield said that "style is the dress of thought". With *XML Styler* you can dress up your XML documents in stylesheets without having to weave the cloth yourself. This chapter is sponsored by ArborText, Inc., `http://www.arbortext.com`, and was prepared by Norman Walsh, `http://nwalsh.com/~ndw/`.

The *XML Styler* is a freeware graphical XSL stylesheet editor from ArborText, Inc. The product simplifies the creation and modification of stylesheets for XML documents. It is implemented as a *Java* application that is designed to run on any Java Virtual Machine (JVM). It has been tested on Microsoft's JVM for Windows and Sun's JVM for Solaris.

25.1 | Introduction to XSL

XSL stylesheets separate form from content so that authors can present media-independent XML information. XSL offers powerful features for generating and suppressing content, reordering content, and associating style information with elements in different contexts.

Although *XML Styler* hides most of the complexity of XSL from the stylesheet author, it is useful to have a general understanding of how XSL works. Broadly speaking, an XSL stylesheet consists of a collection of rules. Each rule associates a formatting behavior with an element type.

A rule has two parts: a pattern and an action. The pattern identifies the element types that the rule applies to; for example, all EMPH elements, or all PARA elements occurring within an ABSTRACT. The action defines how the selected elements will be formatted.

25.2 | Creating a stylesheet with *XML Styler*

Before looking at XML Styler and XSL in more detail, let's use XML Styler to create a stylesheet for a simple document type. We'll do this by formatting a particular XML document, `mydoc.xml` in Example 25-1.

Example 25-1. A simple XML document.

```
<?xml version='1.0'?>
<doc><title>A Document</title>
<para>This is a paragraph of text.</para>
<para>Paragraphs can contain <emph>emphasized</emph> text.</para>
</doc>
```

Start *XML Styler* by running `xmlstyler` at the DOS or UNIX shell prompt or by double clicking on the *XML Styler* icon on your desktop. You can get *XML Styler* from the CD that accompanies this book.

XML Styler will start, as shown in Figure 25-1.

Choose "Create a new style sheet" and click "OK". This will start the new-stylesheet wizard.

In order to simplify the creation of a new stylesheet, XML Styler can load the element type names used in your documents from either a plain-text file or an XML document. In this case, we'll get them from `mydoc.xml`, see Figure 25-2.

The next panel in the-new stylesheet wizard lets you set the default font for your documents. Click "Next" to accept the defaults listed.

One of the most useful features in *XML Styler* is the ability to link directly to a test document from within the editor. On the next panel, shown in Figure 25-3, select the preview option and point to the XML document that you would like to preview while editing your stylesheet.

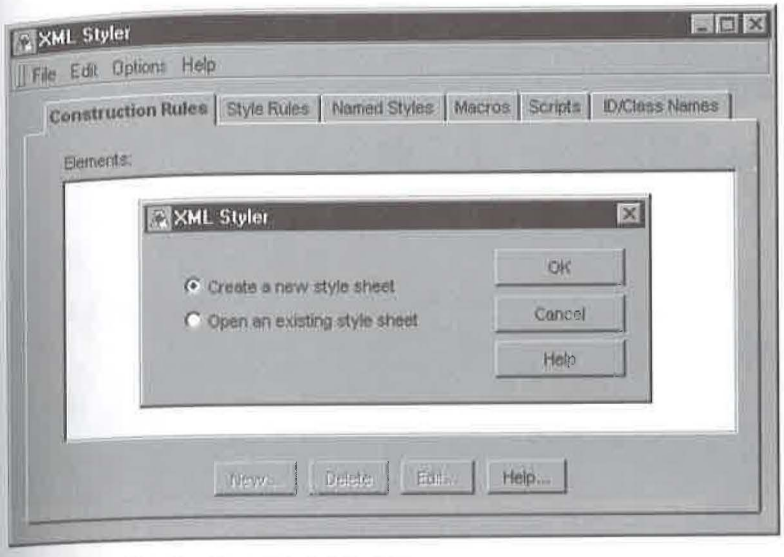


Figure 25-1 Starting *XML Styler*.

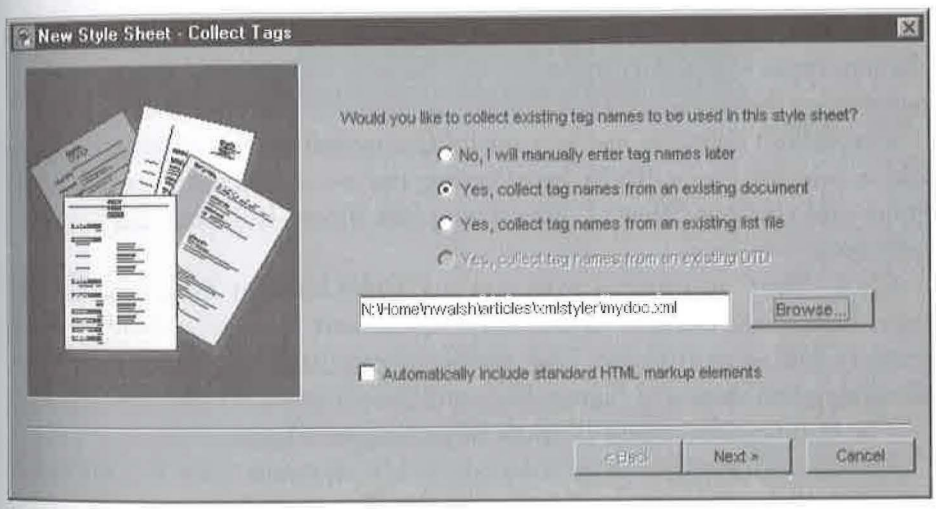


Figure 25-2 Collecting element-type names.

In order to link the stylesheet to the document instance, *XML Styler* has to build a little HTML "glue document". On the next panel of the wizard,

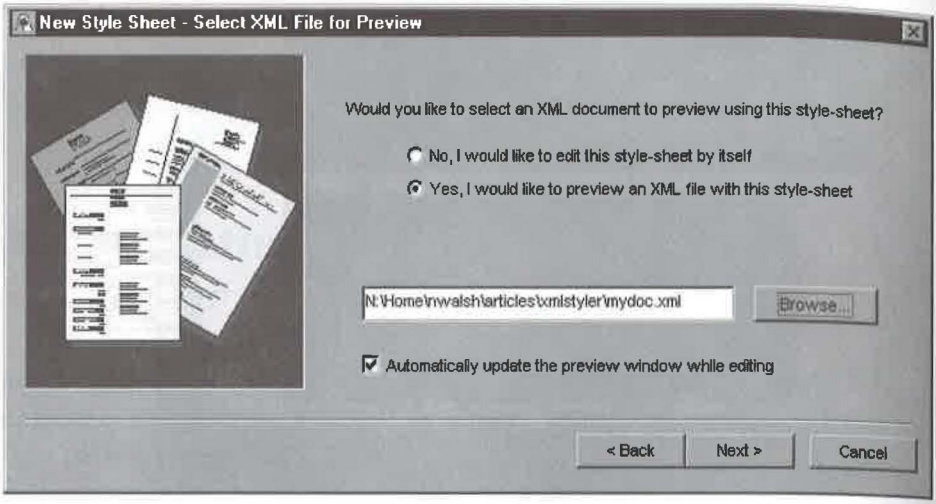


Figure 25-3 Select an XML document for preview.

you can choose the name of the glue document. The default is almost always a good choice.

When the wizard finishes, *XML Styler* will have default rules for all the element types in the document and the browser will display your test document using these default rules. See Figure 25-4.

The default rules are not very useful for presenting this document. Let's add a rule for `para`. Begin by selecting the `para` element type in XML Styler and clicking "New...". The dialog box shown in Figure 25-5 will be presented.

Click "Next" to proceed with creating a new rule for the `para` element type. Each `para` element in our XML document must be associated with some HTML element type. The obvious choice in this case is `P`; enter `P` in the dialog box shown in Figure 25-6 and click "Finish" to proceed.

The resulting document is much improved, see Figure 25-7.

Adding additional rules to associate `title` elements with `H1`, and `emph` elements with `EM` completes the picture. See Figure 25-8.

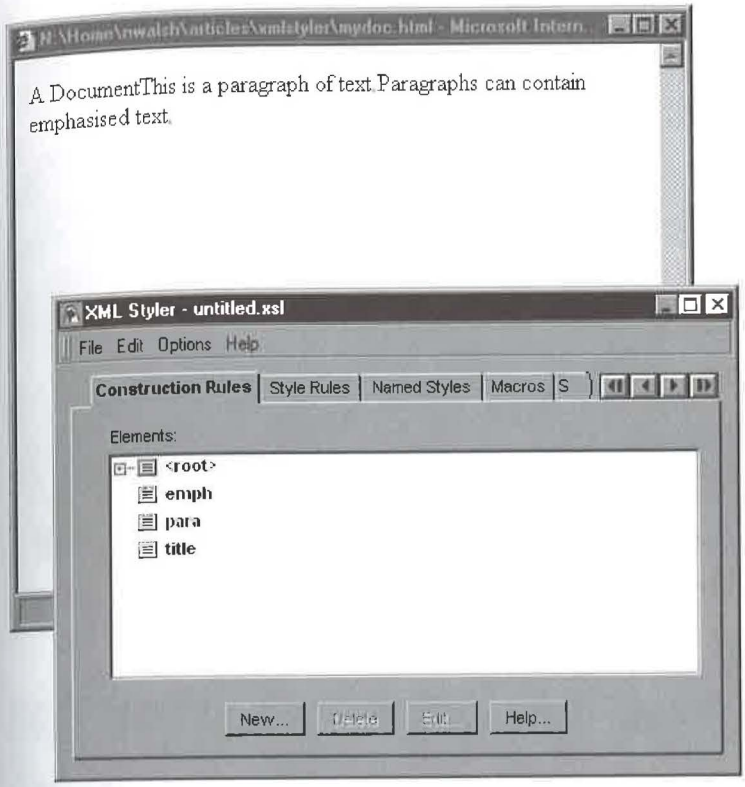


Figure 25-4 Preview using default rules.

25.3 | XSL patterns

In the simple document described above, every element type was used in a unique context. In more complex documents, this is not likely to be the case. Consider the recipe fragment shown in Example 25-2.

Here we see the element type name used in three different contexts, as:

- The recipe name
- The name of a person, and
- The name of a book.

It is likely that each of these will be formatted in a different way.

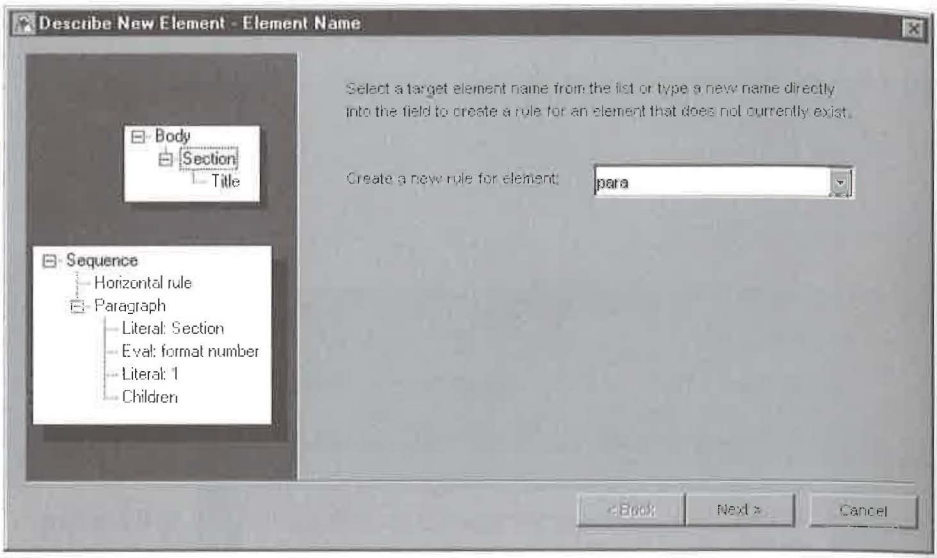


Figure 25-5 Element-type name dialog box.

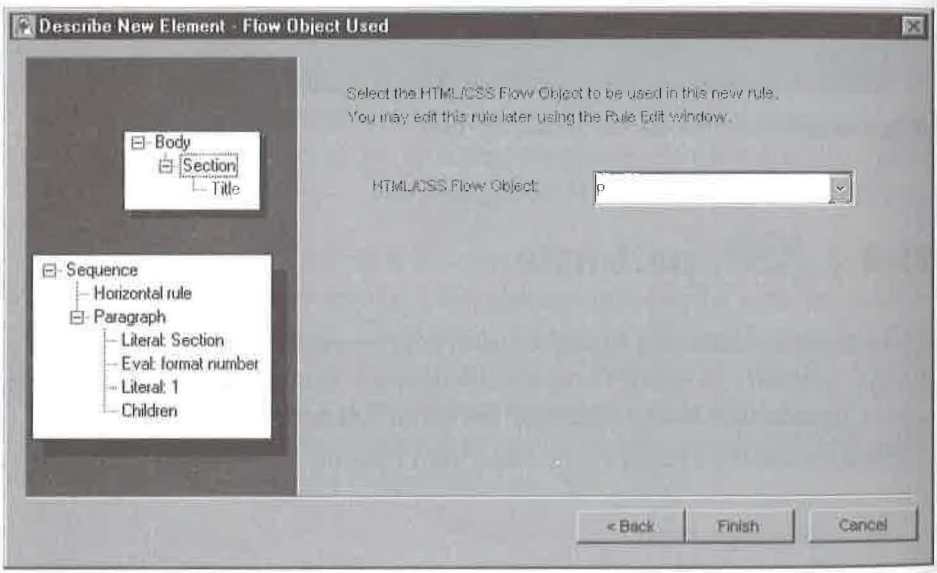


Figure 25-6 Flow object selection dialog box.

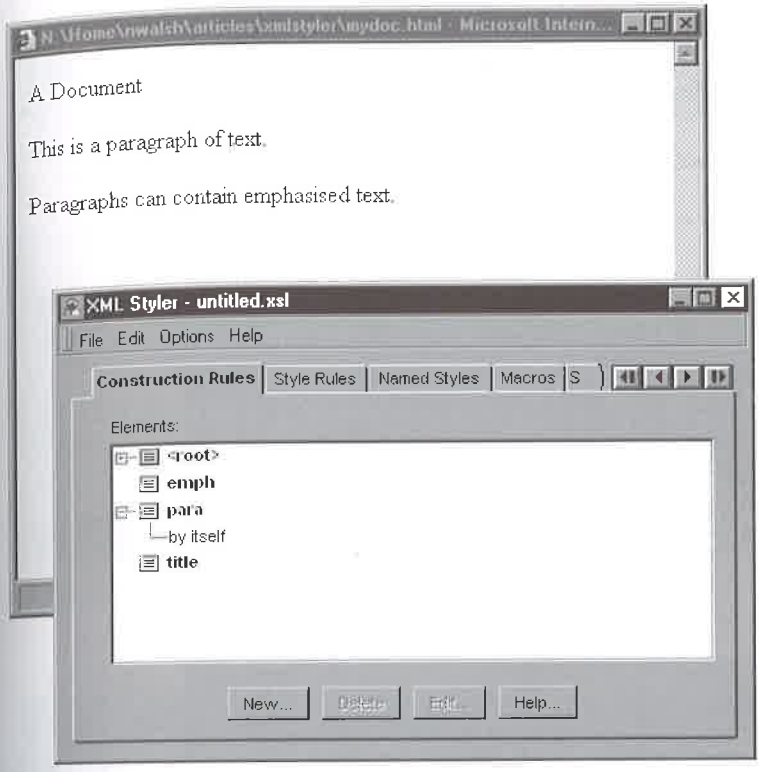


Figure 25-7 Preview using the new rule for `para` elements.

XSL patterns can identify element types in different contexts in two ways: with attribute tests, and by position.

Figure 25-9 shows the rule that matches only the recipe name.

As you can see, the structure of the pattern on the left-hand side is roughly analogous to the structure of the document. In this case, we see that the rule applies only to `name` elements within (i.e., that are children of) `recipe` elements. The dot next to `name` indicates that it is the element type that is the target of this rule.

A slightly more complex pattern is required to format book names properly. In this case, we want to make the contents of `name` elements italic if they occur inside of `note` elements that have a `status` attribute who's value is "credit". See Figure 25-10.

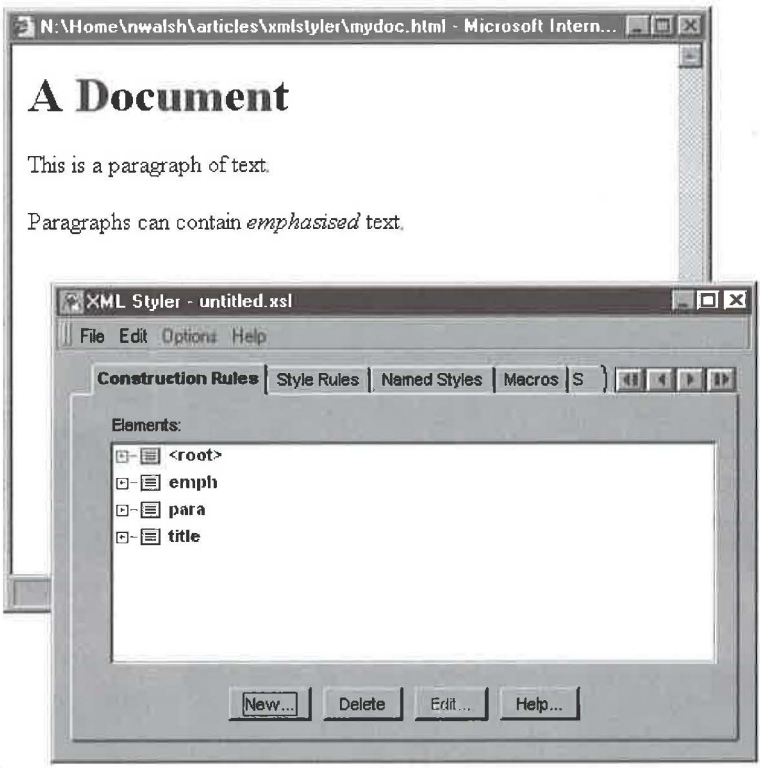


Figure 25-8 Preview using our rules.

Example 25-2. A fragment of a recipe in XML.

```
<recipe>
<name>Corned Beef and Cabbage</name>
<description>A classic New England boiled dinner. This is a
delectable dinner if composed only of beef, onions, and
cabbage. But for authenticity, additional vegetables
are included.</description>
<note>This dish is a Saint Patrick's Day favorite.</note>
<note status="credit">This recipe comes from
<name>The Joy of Cooking</name> by Rombauer and Becker.</note>
<note status="personal"><name>Grandma Luhmen</name> likes it
better without the <ingredient>beets</ingredient>.</note>
<ingredient-list yields="...">
...
</recipe>
```

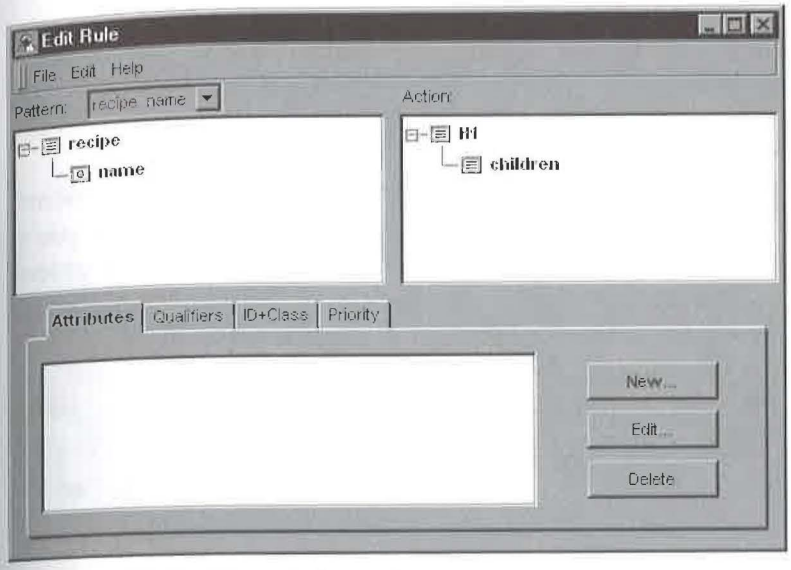


Figure 25-9 The rule for recipe names.

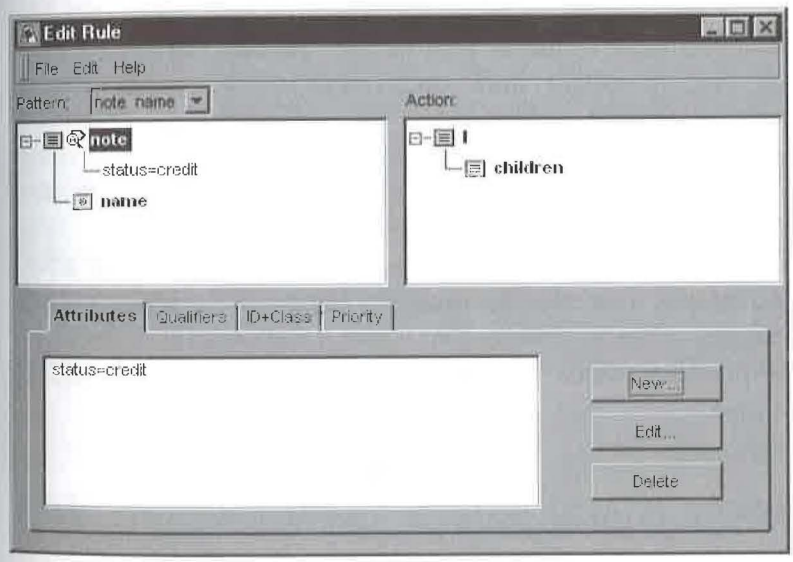


Figure 25-10 The rule for book names.

25.4 | XSL actions

The action part of an XSL rule describes how to format the element type selected by the pattern. XSL describes two sets of formatting objects, "HTML/CSS flow objects" and "DSSSL flow objects".

In addition to the flow objects, there are several "processing elements" that control the behavior of the XSL processor. In these examples, you've already seen the children processing element. This element tells the processor to recursively format the children of the current element (by finding the appropriate rule for each child), and insert the formatted results of that process at the location where *children* occurs.

Other examples of processing elements are *select-elements*, which provides a mechanism for reordering the content, and *eval*, which allows the stylesheet to insert the result of evaluating an expression (calculating the child number of a list item element, for example) into the output.

25.4.1 *HTML/CSS flow objects*

The HTML/CSS flow objects are really nothing more than the HTML element types that you want to use in your output. *Internet Explorer 4.0* and the Microsoft XSL processor both understand the HTML/CSS flow objects.

When you are using XSL to generate HTML from your XML documents, you can use the CSS properties as attributes of the HTML element types. The XSL processor will automatically translate them into the appropriate CSS style attributes. For example, the action shown in Figure 25-11 will format paragraphs in an abstract using the HTML P element type with smaller, italic text.

XML Styler provides simple "tab pages" for easy access the properties of each element type.

25.4.2 *DSSSL flow objects*

The DSSSL flow objects are an abstract representation of formatted output. Each DSSSL flow object has a set of properties that control the details of the formatting for that object. The complete catalog of flow objects and their properties will have well-defined semantics in the final XSL standard.

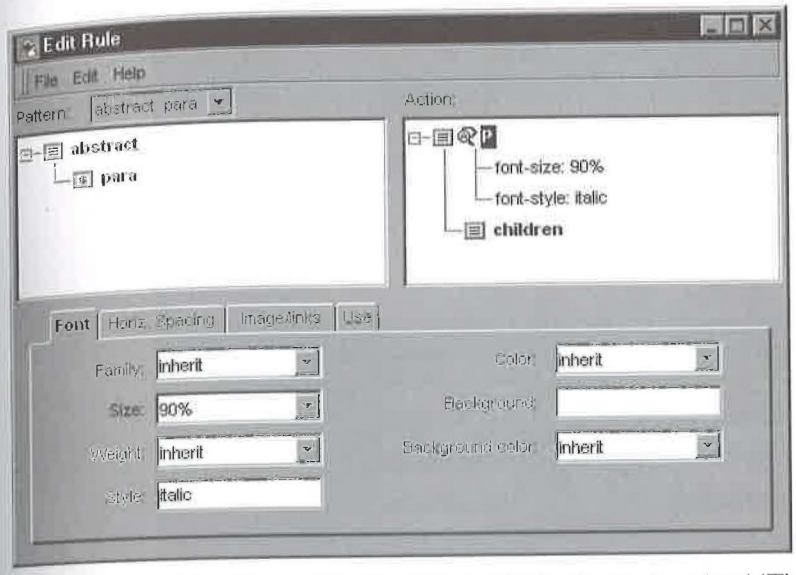


Figure 25-11 The action for paragraphs in abstracts, using HTML.

The XSL processor builds a tree of flow objects and then a rendering engine processes the flow objects and builds the output (on the appropriate media: online, print, aural, etc.). At the time of this writing, Henry Thompson's xslj processor, which works in conjunction with James Clark's Jade engine, offers the only way to process XSL stylesheets that use DSSSL flow objects. Both of these tools are available on the CD-ROM that accompanies this book.

Figure 25-12 shows the same rule for formatting paragraphs within an abstract using DSSSL flow objects.

As in the HTML/CSS case, *XML Styler* has tab pages for each of the DSSSL flow objects.

25.5 | Conclusion

In this chapter, we've seen a brief overview of some of the features of XML Styler and demonstrated how it can be used to quickly and easily create stylesheets without having to learn the syntactic details of XSL. By creating stylesheets for your XML document instances, you can begin to see how

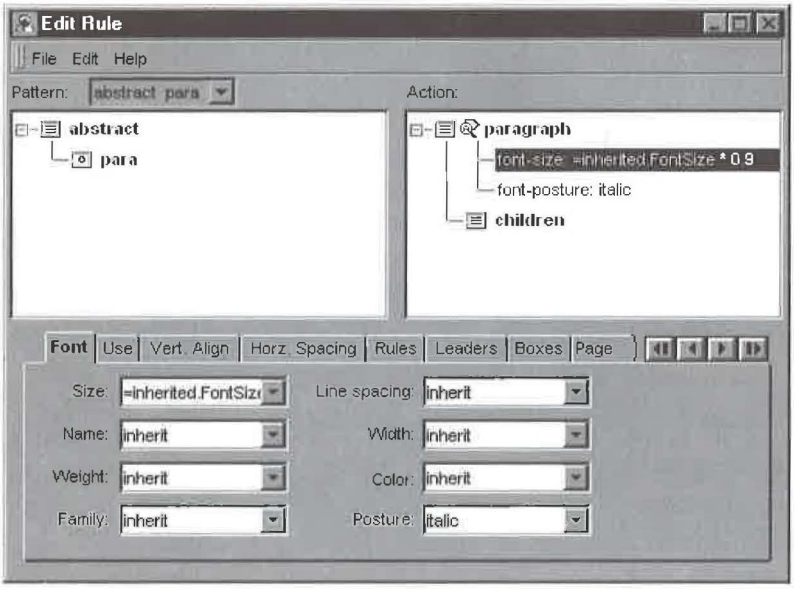
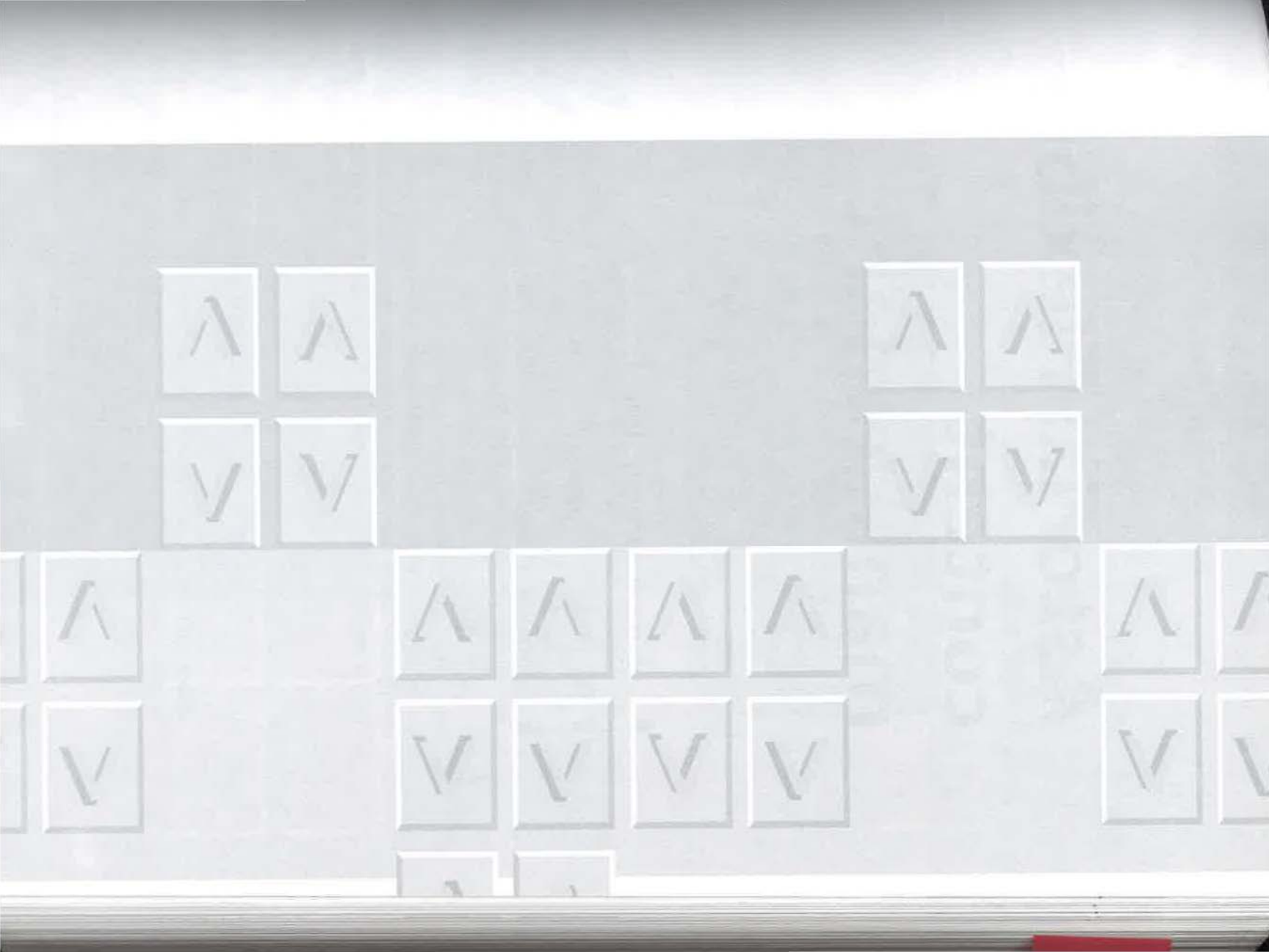


Figure 25-12 The action for paragraphs in abstracts, using DSSSL.

XML plus XSL will allow you to deliver your information the way you want it displayed.

You can find more about XSL in Chapter 35, "Extensible Style Language (XSL)", on page 516.



Astoria: Flexible content management

- Content management defined
- Document components
- Information reuse

Chapter

26

Document management is about managing documents as a whole, regardless of what is inside them. *Content* management, on the other hand, gets deep down inside and so is far more powerful. This perspective on content management focuses on the business problems it can solve. It is sponsored by Chrystal Software, a Xerox New Enterprise company, `http://www.chrystal.com`, and was prepared by Sean Baird, Robin Gellerman, and Kari Johnson.

Technical publications are critical in today's corporation. Behind these documents are the writers, artists, and editors who develop and maintain the massive amounts of documentation that keep a company running. Now, both publication managers and corporate directors are looking for better ways to leverage this wealth of data for higher returns throughout the enterprise.

Many enterprises have found leverage in managing document content as *components*, rather than as entire publications. This practice is called *content management*, in contrast to *document management*. Middle-tier Web applications, in particular, benefit from the abilty to assemble components with other data for delivery to the client.

26.1 | Components are everywhere

From new cars to software, components are the way we make things today.

In manufacturing industries as much as 80% of products now consist of components drawn from a company's part library or purchased from suppli-

ers. Product designers routinely tap into internal databases and online parts warehouse services in the course of drafting and specifying new models.

In software, most of the new code being written is as objects, self-contained bundles of information and operations with the ability to send and receive messages in standard ways. Programs can be created by assembling a bunch of these object components and making them exchange information and services with each other.

26.1.1 *Components in publishing*

And now components are becoming the trend in publishing as well. Why? Because in publishing, as in other endeavors, components simplify complexity and increase flexibility for adapting to change. Consider these general advantages of components and how they come into play in a content management publishing environment.

26.1.1.1 System simplification

Components make it possible to break down complex systems into pieces that are easier to understand and work with. For publishing groups this means that teams of writers and editors can work on components for the same document simultaneously. Users can more easily locate specific information since components can be explicitly searched for.

26.1.1.2 Easier revision

When something needs to be revised or customized, changes can be made to just the component(s) affected, without having to redesign the whole document. If a single paragraph in a document needs to be revised, the author can check out just that paragraph from the content management system rather than the whole document. Or, if it's important to see the change in context, the author can check out the section the paragraph appears in. After editing, when the section is checked back in, versioning information is applied only to the paragraph that has changed.

26.1.1.3 Efficient authoring

Studies show that at least 30% of the content created by technical publishing groups is reusable – or would be, if people could find the information. Typically, it's buried in documents, scattered here and there in file systems on various desktop systems and servers. Content management eliminates the need to redo work by providing a universal repository for managing published and in-progress documents. The ability to unlock content from structured documents so that individual components of information can be independently accessed, tracked, and versioned enables writers and editors to immediately focus on exactly what they're looking for.

26.1.1.4 Less routine editing

A huge amount of the editing process involves checking documents for consistency and correcting them for corporate style. Content management minimizes editing time and tedium by enabling editors to maintain glossaries as collections of components. This information can be added to or revised rapidly, every day if necessary, with the new material instantly available to all users.

26.1.1.5 Fast, easy customization

Component-level management means that documents can be customized by changing only what is unique about them. This approach makes it possible to rapidly provide markets and customers with tailored information.

26.1.1.6 Universal updates

Each information component exists in the repository as a single object. When authors want to reuse a component, instead of copying it, they simply create a pointer to the object. This approach eliminates the redundant work of having to try to find all places where the information appears and update them independently. Instead authors can revise the component in the repository once, and it will be automatically updated in all documents that contain it.

26.1.1.7 Streamlined translations

Translators typically work with a moving target, a source document that continues to change while translation is going on. Translated versions then have to be returned to translators for a laborious manual process of identifying, changing, and checking new material. Content management can speed this process by providing translators with only those document components that are new, along with information about what has changed and exactly where the revisions should be inserted in the document.

26.1.1.8 Flexible distribution

Content management makes it easy to repurpose content for different media. Users can assign custom attributes to a particular component. For example, an attribute of an element could tell the software whether or not the element should be included when exporting a document for the Web, as opposed to printing it. Users can automate document assembly, including adjustments for target media.

26.1.2 *XML makes components*

XML brings intelligence to data. It breaks up the information into smaller information components. The smaller and more specific the component is, the more addressable and reusable it is.

For example, the document in Figure 26-1 uses descriptive element type names to identify the components and structure of the document. A component is a piece of information that can be used independently, such as a paragraph, chapter, instructional procedure, warning note, part number, order quantity, graphic, side-bar story, video clip, or one of an infinite variety of additional information types.

When managed by a content management system (Figure 26-2), these pieces can be controlled, revised, reused, and assembled into new documents.

Another way XML adds value to information is through attributes, or "metadata" (Figure 26-3). By adding "information about information", users can further describe the information for repurposing. A user assigns attributes to a particular component, for example to specify whether or not

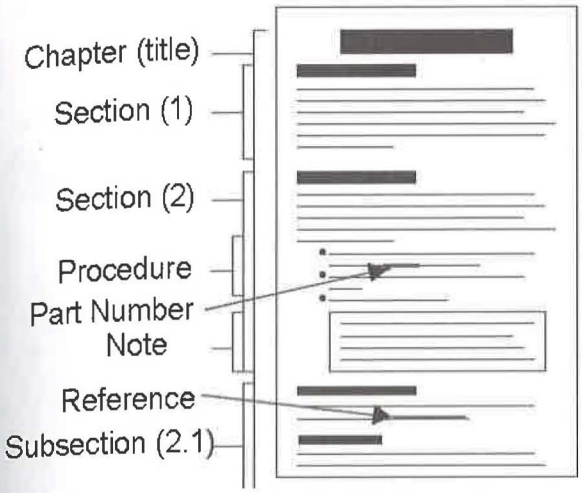


Figure 26-1 Document components described with XML.

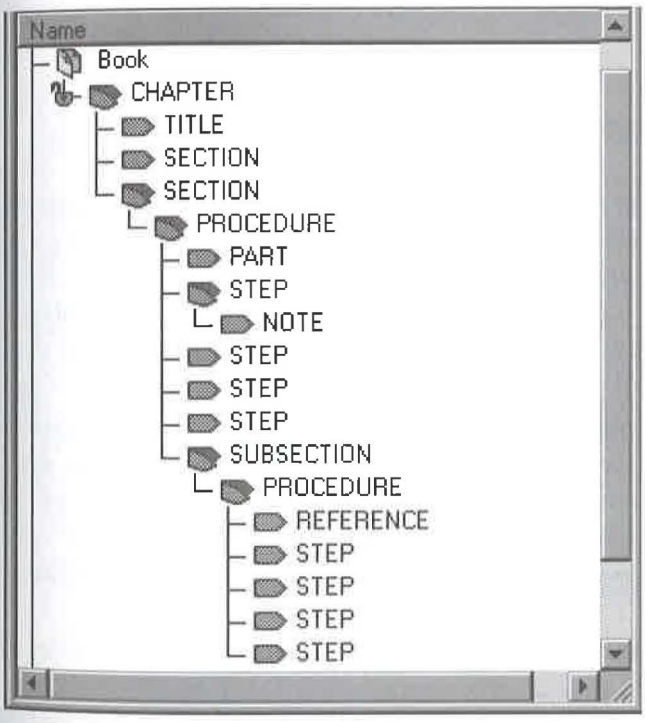


Figure 26-2 Hierarchical structure shown in content management system.

to include it when publishing the document for the Web, as opposed to printing it. When the document is published, the content management system will make the proper adjustments for the target medium.

Metadata can also be used to identify the intended audience for specific components. In this case, a "beginner" requires more information than an "expert". The content management system will assemble a document and publish the information that matches these criteria.

```
<step audience="beginner">Keep the
engine running and park car
on level ground.</step>

<step audience="expert">Keep the
engine running.</step>
```

Figure 26-3 Metadata can identify the intended audience.

26.1.3 *Applications for content reuse*

Reuse, the most compelling feature of content management, allows content within any document to be used elsewhere in the repository. Reuse means writing the information once and linking to it from other documents. This can be very useful when multiple documents contain standard "boilerplate" information. This repurposing of information saves users countless hours of rework and duplication of effort.

Applications for information reuse are everywhere. Reuse can be as simple as finding a description from one document and linking it into a new document. Common content creates an "information pool" of reusable pieces available to individuals or groups inside the company Figure 26-4. Linked reuse, instead of copying, makes updates more efficient and reduces redundant storage.

Organizations that maintain common glossaries of business terms can benefit from reuse. When glossary information stored in a content management system changes, the information is revised only one time. All of the documents containing that information are automatically updated.

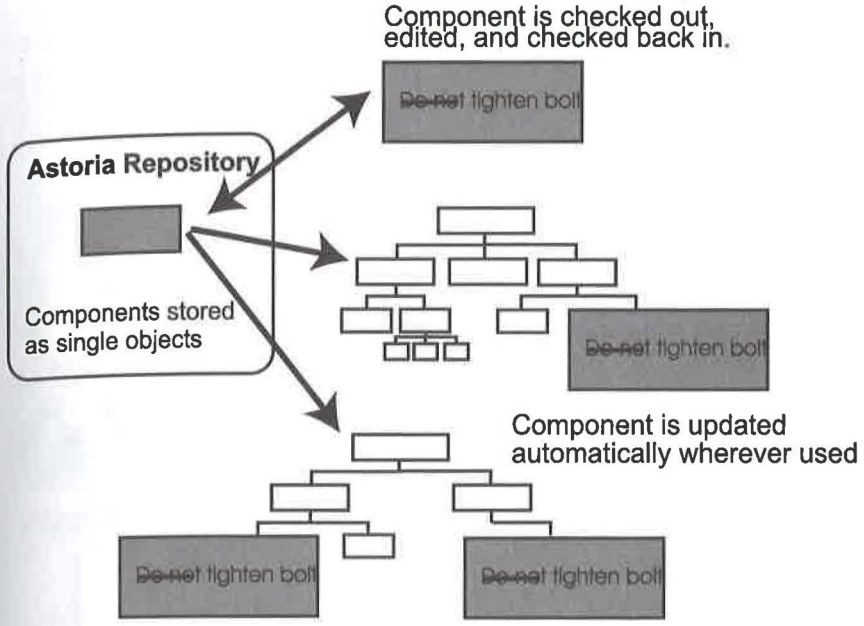


Figure 26-4 Component reuse.

Because warnings and cautions usually require careful wording, organizations strive for uniformity across all documents. Manually locating and changing dozens of these elements in hundreds of contexts can consume countless hours. Content management solves that problem by allowing XML documents to reuse content across documents.

For global business processes, linked reuse helps organizations get to market faster around the world. By identifying the newly revised information in a repair manual, only the new information will be translated into the target languages saving valuable time and money.

26.2 | A content management implementation

To better understand what content management systems provide, it is helpful to look at an actual product in action.

Chrystal Software's Astoria, like other component-based *content* management systems, attempts to provide value beyond that of generalized *document* management systems. It does so by managing the content of the document as a set of components (see Figure 26-5).

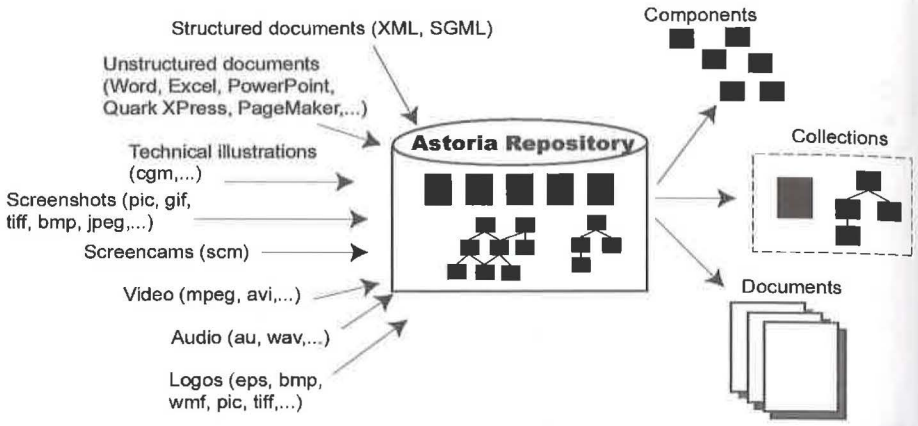


Figure 26-5 The *Astoria* repository.

Some of the product's "off-the-shelf" capabilities are described in the following sections. Customization for specialized requirements is possible through its software development kit, a public C++ application programming Interface.

26.2.1 *Revision tracking*

Astoria automatically collects revision information at each check-in, indicating time, date, author, revision number, and an optional comment. Past versions are available for republishing or to provide an audit trail (see Figure 26-6).

For XML documents, revision history is detected and maintained at the component level, not just at the document level. A sophisticated differencing engine is used to apply revision information to only the content that changes during an editing session.

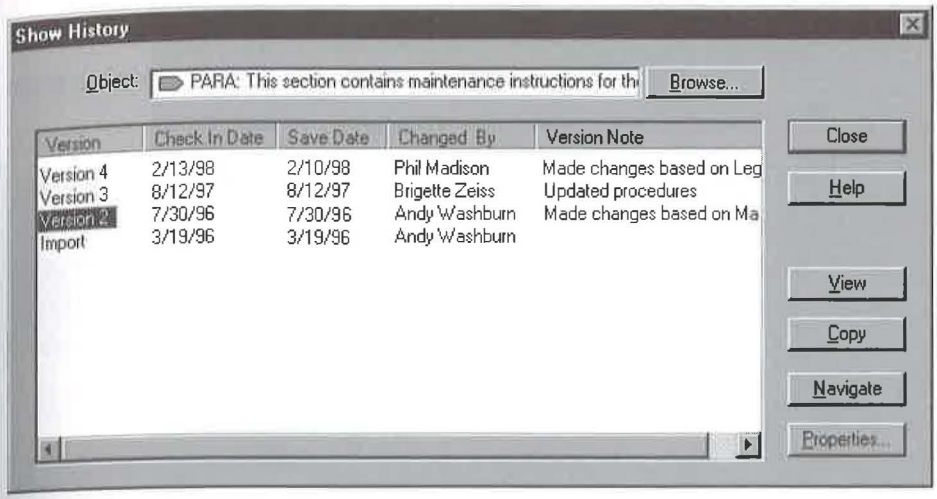


Figure 26-6 Revision history of a paragraph.

At important milestones such as release dates and the beginning of review cycles, users can formalize document versions into "editions". The document state can then be recreated for that point in time by opening the appropriate edition.

26.2.2 *Search*

Astoria's search options let users locate documents created in more than 50 common applications. Advanced indexing enhances search by looking for various forms of the word (e.g., plural, tenses, root). Matching documents can be selected for viewing and editing.

By applying "custom attributes" to documents, users add "information about information." Custom attributes can automatically be created from XML metadata. In addition to custom attributes, document structure, data content, and version information can be used in queries (see Figure 26-7).

Another form of search, "where-used queries", locates content that is reused in multiple XML documents. Users can determine whether changes to that content are appropriate in all contexts before committing to the change.

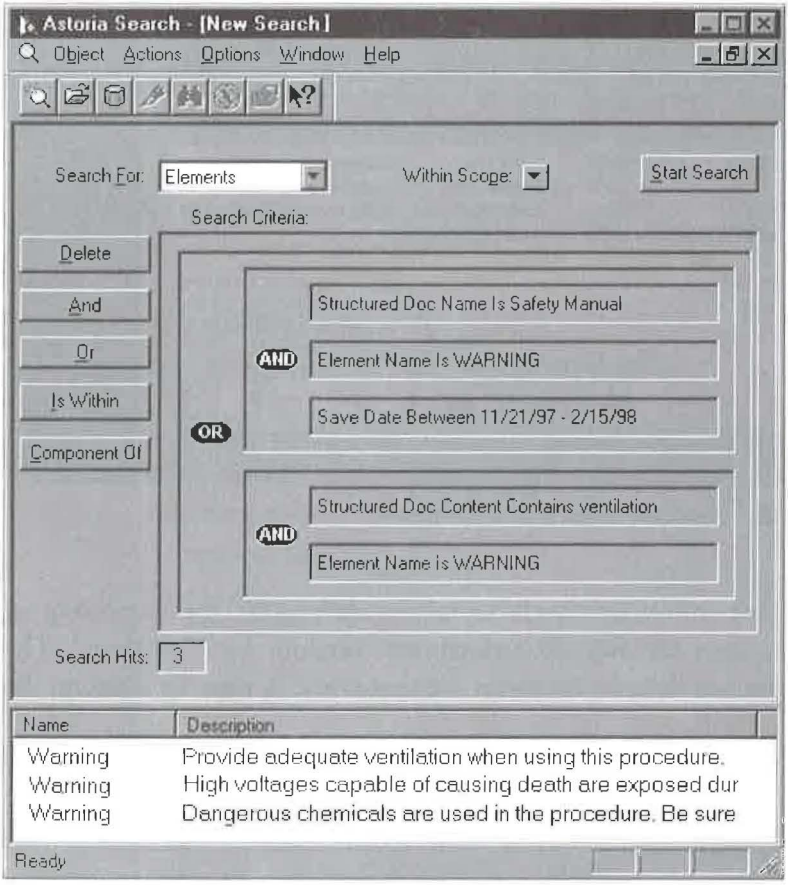


Figure 26-7 Search results refined with XML structure and metadata.

26.2.3 *Dynamic document assembly*

One of the compelling benefits of managing documents at the component level is that users can effectively create an information pool from which to draw. Figure 26-8 shows how users can search for information meeting unique criteria, organize it as they wish, and dynamically create a new deliverable.

For example, a financial portfolio manager could create a series of articles and recommendations which could then be organized dynamically into unique documents based on the profile of each investor.

Analysis *The management and production of information as components is a powerful idea. Not suitable for a novel, perhaps, but eminently appropriate for the mission-critical data that is part-and-parcel of creating and marketing today's complex products – on the Web and elsewhere.*

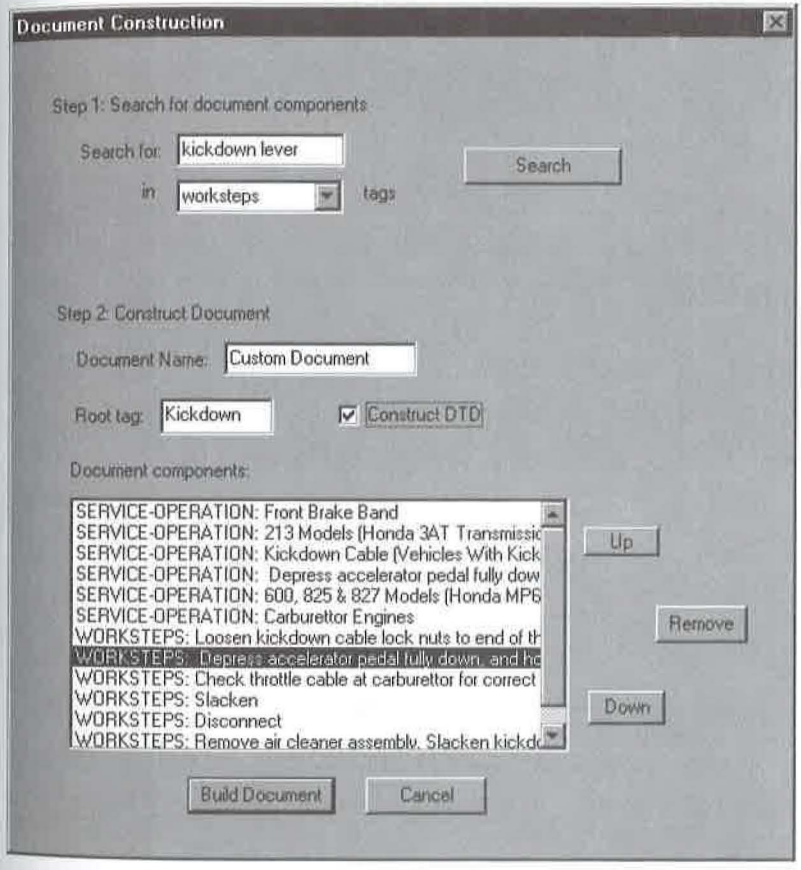


Figure 26-8 Assembling new documents by searching for relevant components.