XML Fundamentals

DigitalThink

THE WEB-BASED TRAINING LEADER℠

www.digitalthink.com

- Module 1:
  - Lesson 1 - Course introduction
  - Lesson 2 - Prerequisites
  - Lesson 3 - What you need
  - Lesson 4 - What to expect
    - Sidebar: Copying and pasting on Unix platforms
  - Lesson 5 - The course tutors
  - Lesson 6 - Who Wrote This Course?
  - Lesson 7 - Course exercises
- Module 2:
  - Lesson 1 - What is XML?
  - Lesson 2 - The vision of HTML
    - Sidebar: The birth of HTML
    - Sidebar: The vision of HTML (MouseOver transcript)
  - Lesson 3 - HTML hijacked
    - Sidebar: Separating format from structure
    - Sidebar: HTML extensions
    - Sidebar: HTML hijacked (MouseOver transcript)
  - Lesson 4 - HTML limitations
    - Sidebar: Searching and finding in HTML documents
    - Quiz: The development of HTML
  - Lesson 5 - XML defined
    - Sidebar: The history of SGML
    - Sidebar: What markup languages do
    - Sidebar: SGML to XML
    - Sidebar: HTML and XML contrasted
    - Sidebar: Describing content in intelligent terms (Audio transcript)
  - Lesson 6 - The goals of XML
  - Lesson 7 - Will XML replace HTML?
    - Sidebar: Targeted searches
    - Sidebar: XML uses beyond the Web
  - Lesson 8 - Module wrap-up
    - Quiz: Defining XML
- Module 3:
  - Lesson 1 - Creating XML documents
  - Lesson 2 - A simple, well-formed XML document
    - Sidebar: XML, the lazy developer's nightmare
    - Sidebar: What makes an XML document? (Audio transcript)
  - Lesson 3 - Rules for a well-formed XML document
    - Sidebar: The importance of starting and ending tags
    - Sidebar: New rules for empty tags

        - Sidebar: Attribute values
        - Sidebar: Nesting
        - Sidebar: Matching case between tags
        - Exercise: Correcting a poorly-formed HTML document
    - Lesson 4 - Discerning structure
        - Sidebar: Advantages to creating letters as XML documents
        - Sidebar: Structure of sample documents (FlipBook transcript)
        - Quiz: Rules for XML documents
    - Lesson 5 - Working with mixed content
        - Exercise: Creating a well-formed document from text
    - Lesson 6 - Adding information to XML documents
        - Sidebar: CDATA section example
        - Sidebar: Balkanization of the Web?
        - Sidebar: Encoding information (MouseOver transcript)
    - Lesson 7 - Module wrap-up
        - Quiz: XML: Power and problems
- Module 4:
    - Lesson 1 - DTD basics
    - Lesson 2 - Beyond well-formedness
        - Sidebar: The role of the DTD (FlipBook transcript)
    - Lesson 3 - DTD defined
        - Sidebar: Determining the root element (FlipBook transcript)
    - Lesson 4 - Creating a DTD
    - Lesson 5 - Element type declarations
        - Sidebar: Specifying information in the element declaration (FlipBook transcript)
        - Quiz: Defining the DTD
    - Lesson 6 - Mixed content and empty elements
        - Sidebar: Working with mixed content (FlipBook transcript)
        - Exercise: Declaring child elements and mixed content
    - Lesson 7 - Referencing DTD declarations
    - Lesson 8 - The DTD file and XML file as separate files
        - Sidebar: Separate DTD and XML (MouseOver transcript)
        - Quiz: Constructing the DTD
    - Lesson 9 - Declarations within the XML file
        - Sidebar: Embedding the DTD in an XML file (MouseOver transcript)
    - Lesson 10 - A checklist for creating DTDs
        - Exercise: Creating a simple DTD
        - Exercise: Creating a DTD for mixed content
    - Lesson 11 - Module wrap-up
        - Quiz: Developing the DTD
- Module 5:

- Lesson 1 - Building the DTD
- Lesson 2 - Declaring attributes in a DTD
    - Sidebar: Using attributes (FlipBook transcript)
- Lesson 3 - Attribute-list declaration syntax
    - Sidebar: Attribute syntax (MouseOver transcript)
    - Sidebar: Example of attribute usage: (MouseOver transcript)
    - Exercise: Creating an attribute-list declaration
    - Exercise: Creating an XML document from a DTD
- Lesson 4 - Defining entities
    - Sidebar: Defining multiple characters
- Lesson 5 - Parameter entities
    - Sidebar: Notation declarations
- Lesson 6 - Using parsers
    - Exercise: Parsing documents in a browser
- Lesson 7 - Module wrap-up
    - Quiz: DTD syntax

- Module 6:
    - Lesson 1 - XLink, XPointer, namespaces, and metadata
    - Lesson 2 - Linking in XML
        - Sidebar: An example of simple links (FlipBook transcript)
        - Sidebar: A theoretical example of using extended links
        - Sidebar: Out-of-line links
    - Lesson 3 - Link behavior
    - Lesson 4 - Links and the DTD
        - Sidebar: Creating Links attributes in a DTD (Audio transcript)
        - Quiz: Link Basics
    - Lesson 5 - XPointer
        - Exercise: XLink and XPointer
    - Lesson 6 - Namespaces
        - Sidebar: Examples of using namespaces (FlipBook transcript)
    - Lesson 7 - Metadata
        - Sidebar: Element usage for the Dublin core scheme
        - Sidebar: Dublin Core as the namespace
        - Exercise: Using namespaces and metadata
    - Lesson 8 - Module wrap-up
        - Quiz: XLink, XPointer, namespaces and metadata
- Module 7:
    - Lesson 1 - Styling XML files
    - Lesson 2 - Creating a style sheet
        - Sidebar: The CSS1 property list
    - Lesson 3 - CSS and XML

    - Sidebar: An example of linked files (FlipBook transcript)
  - Lesson 4 - XSL
    - Quiz: Defining XSL
  - Lesson 5 - XSL construction rules
    - Sidebar: An example of using XSL (FlipBook transcript)
    - Exercise: Creating style sheets
  - Lesson 6 - Defining a pattern
  - Lesson 7 - Using wildcards
    - Sidebar: An example of using wildcards (FlipBook transcript)
  - Lesson 8 - Qualifier attributes
    - Sidebar: An example of using qualifier attributes (FlipBook transcript)
    - Quiz: Using XSL
  - Lesson 9 - Attributes
    - Sidebar: Referencing an attribute value (MouseOver transcript)
    - Sidebar: Selecting other attributes (MouseOver transcripts)
  - Lesson 10 - Rule arbitration
  - Lesson 11 - Defining the action with flow objects
    - Sidebar: Elements in HTML
    - Sidebar: HTML flow objects available to XSL
    - Sidebar: An example of using flow objects (FlipBook transcript)
  - Lesson 12 - The root rule and literal text
    - Exercise: Using XSL
  - Lesson 13 - XSL and Internet Explorer 5.0
    - Sidebar: An XML and XSL file combination for IE 5.0 (MouseOver transcript)
  - Lesson 14 - Module wrap-up
    - Quiz: Styling XML files
- Module 8:
  - Lesson 1 - XML today and in the future
  - Lesson 2 - Supporting XML
    - Sidebar: Comparing IE browser versions 4.0 and 5.0
    - Sidebar: Using an XSL style sheet with XML in IE 4.x
  - Lesson 3 - Databinding XML to HTML
    - Exercise: Databinding XML to HTML in IE 4.x
    - Exercise: Databinding XML to HTML in IE 5.0
  - Lesson 4 - Using Dynamic HTML with XML
    - Sidebar: A catalog built using DHTML and XML (Simulation transcript)
  - Lesson 5 - An interactive demo
    - Sidebar: An XML demonstration file (Simulation transcript)
  - Lesson 6 - XML adoption scenario
    - Sidebar: Phase 1: XML is separate from HTML
    - Sidebar: Phase 2: XML begins to replace HTML

        - Sidebar: Phase 3: Write once, output everywhere
        - Sidebar: XML analyzed (Audio transcript)
    - Lesson 7 - The future of HTML
    - Lesson 8 - XML and e-commerce
    - Lesson 9 - XML and databases
        - Sidebar: Existing XML solutions
    - Lesson 10 - Module wrap-up
        - Quiz: XML today and in the future
- Module 9:
    - Lesson 1 - Course conclusion
    - Lesson 2 - We'd love your feedback
    - Lesson 3 - Help us help you learn!

Glossary- XML Fundamentals

Glossary

Each time you click a glossary term, you'll see a window like this one displaying the term and its definition. To see the entire glossary, click "Show All Terms."

#PCDATA

Stands for parsed character data.

attribute-list declaration

A declaration in the DTD to identify the attribute value of an element.

CBL

Common Business Language. An XML application currently under development to provide a vocabulary for facilitating e-commerce.

CDF

Channel Definition Format. An application developed by Microsoft for creating channels to which Web users can subscribe to receive push content.

CML

Chemical Markup Language. An XML application.

construction rule

The basic rule of XSL files that determines formatting for XML elements.

DHTML

Dynamic HTML. A term used to refer to a combination of HTML, scripts, styles, and the Document Object Model (DOM).

Document Object Model (DOM)

A model that specifies how objects are manipulated through script.

document type declaration

A declaration of DTD information that is embedded directly within the document that references it such as an SGML or XML document; denoted by the tag &lt;!DOCTYPE&gt;.

DSSSL

Document Style Semantics and Specification Language. DSSSL provides formatting information for SGML documents.

DTD

Document Type Definition. A set of rules contained in a simple text file that defines the structure, syntax and vocabulary as it relates to the tags and attributes for corresponding documents.

Dublin Core initiative

A resource description scheme created by experts from library, museum, research, and networking communities for the purposes of labeling information relating to intellectual property items.

EDI

Electronic Data Interchange. The common method used by entities to exchange business transaction information.

EDIFACT

United Nations Standard Messages Directory for Electronic Data Interchange For Administration, Commerce and Transport. The standard currently used for business-to-business communications.

element content

An element that contains only other elements is said to have element content.

element type declaration

A declaration required in the DTD to identify an element and what it can contain.

empty element

An element that contains no text but carries relevant information within its corresponding tag.

extended link

A type of link that points to multiple resources simultaneously and can create virtual links from other documents.

flow objects

Objects from either DSSSL or HTML that help define the visual and behavioral characteristics of the element being styled. In XSL, flow objects are elements that have predefined meanings in terms of display characteristics and specific usage.

HTML

Hypertext Markup Language. A markup language used for presenting Web page content.

inline link

A link that appears within text.

markup

Formatting instructions for publishing that use tags to indicate font, alignment and so forth.

MathML

Mathematical Markup Language. An XML application.

metadata

Data about data.

meta-language

A language for defining other languages.

mixed content

An element that contains both text and additional child elements is said to have mixed content.

namespace

A declaration that associates an element name with a particular set of definitions.

notation declaration

A declaration which references a non-XML resource and specifies instructions for that resource in a non-XML context

out-of-line link

A link that does not appear to the user on the page.

parser

A program that breaks tagged text down into its individual elements.

RDF

Resource Description Framework. The proposed specification for defining metadata about XML.

Recommendation

A new specification that is voted and agreed upon by the W3C. A standard, by contrast, is a specification that is widely implemented and accepted. Today's Recommendation is usually tomorrow's standard.

root element

The all-encompassing element in an XML file. All other elements must be contained within the root element.

SGML

Standard Generalized Markup Language. A specification used to create other languages, including HTML. XML is a subset of SGML.

simple link

A unidirectional link between a hypertext reference and a target page.

style sheet

A text file containing one or more rules or definitions for the style characteristics of page elements.

tags

Markup instructions embedded within the document.

valid

Describes XML documents that conform to a DTD.

WIDL

Web Interface Definition Language. An XML application for parsing data from Web-based applications.

W3C

W3C is the World Wide Web Consortium, an international association that develops standards for the Web.

WYSIWYG

What You See Is What You Get.

XLink

XML Linking Language. A proposed standard for defining the creation and behavior of link elements in XML pages.

XPointer

A proposed standard for addressing links into specific points within documents.

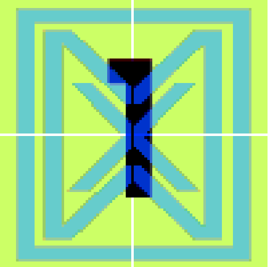
XSL

Extensible Style Language. A proposed style language for XML documents.

XML Fundamentals

Module 1

Lesson 1



Course introduction

Module introduction

XML *Fundamentals* shows you how to apply XML to your advantage. You'll create well-formed XML documents and DTDs. In addition, you'll be introduced to the related technologies of XLink, XSL and CSS.

Course goals

After completing the course, you will have the knowledge and skills necessary to:

- Explain the shortcomings of HTML and the origins of XML
- Clarify how XML is different from HTML
- Define what makes a "well-formed" XML document
- Create a well-formed XML document
- Create a DTD
- Use XML related technologies such as CSS, XSL and XLink
- Apply the concepts of namespaces and metadata
- Keep up with developments in XML

First-time students: To get the most out of this course, take the DigitalThink Orientation tour.

< !-- XML Fundamentals >

Module 1

Lesson 2
Objective

Prerequisites

Make sure you have the necessary background for this course.

To gain the greatest benefit from this course, prior experience coding Hypertext Markup Language (HTML) is required. A basic knowledge of HTML tags and syntax is necessary to complete the exercises.

A general understanding of Cascading Style Sheets (CSS), Dynamic HTML (DHTML) and Java applet technology provides a helpful introduction for some material, but is not necessary to complete the course.

Tip

If you don't have these skills, you might consider taking the HTML Programming Series. See the Catalog page for details.

ORIENTATION

If this is your first DigitalThink course, we recommend you complete *Setting Up Your Computer for a DigitalThink Course* before continuing. These setup procedures walk you through downloading the audio player, the chat software, and compression utilities you need in order to get the most from your DigitalThink courses.

After completing the setup procedures, take about ten minutes to complete our interactive orientation minicourse, *Orientation 101*. You might also want to read *The DigitalThink Training Method,* an article which introduces DigitalThink's general course structure and learning tools.

Setting Up Your Computer for a DigitalThink Course

Orientation 101: An Interactive Guide to a DigitalThink Course

The DigitalThink Training Method

DigitalThink

Locker Login
Course Catalog
Company
Corporate Training
Newsroom
Events
Support Services
Training Resources
Site Map
Home

Legal Info:

- Course License

If you have any questions about these terms, or if you would like information about licensing materials from DigitalThink, please contact us via email: sales@digitalthink.com.

Module 1

Lesson 3

Objective

What you need

Find out what you need to take this course.

To complete the required course exercises, you will need a text editor such as WordPad. You will also need a 4.x or higher browser.

In one optional course exercise, certain HTML files you'll modify will not render in a Netscape browser. In this exercise, we provide screen shots so Netscape users can see how the files would render.

You will also need a compression utility such as WinZip to unpack the course download. The course download contains the text files you'll edit, and is available from the Resources page.

Resources page

Visit the course Resources page by clicking the Resources button on the toolbar.

Course PDF

You can download a compressed file from the Resources page that contains a PDF file of the entire course. This PDF file provides the lesson, quiz, and exercise content so you can read and review the course material offline at your convenience. The PDF file is **not** required to complete this course.

You will need Adobe Acrobat Reader 3.0 or higher to view a PDF file. If you don't have Acrobat Reader, you can download it from the Adobe web site.

Internet resources

You can find a list of links to web sites with XML information on the Resources page. Remember, you can reach the Resources page at any time throughout the course by clicking the Resources button on the toolbar.

Course bookstore

Purchase the optional text for this course online by visiting the course Bookstore page. This book is **not** required to take this course, but does contain helpful, additional information. You can reach the Bookstore page at any time by clicking the Resources button on the toolbar.

DigitalThink

Course Catalog

Buy a Course | How We Teach | New Courses | Curriculum Paths | Academic Credit

Locker Login

Course Catalog

Company

Corporate Training

Newsroom

Events

Support Services

Training Resources

Site Map

Home

Browse complete list of courses

Search Our Site

Find

DigitalThink's online catalog provides the most current list of our IT training courses. View our catalog by topic area by selecting from the drop-down menu above, or view a complete list of all courses.

Students like our courses because:

- DigitalThink Tutors provide expert help. Take one of our courses, and you won't go back to old-fashioned classrooms again.
- DigitalThink courses are true Web-based training. We don't sell "Web-delivered" CD-ROM training and call it Web-based training. Try our free sample course to see how great Web-based training can be.
- Everything you need is included. All lessons, quizzes, and exercises are online. You won't be asked to study from a book or video. You learn in a complete online environment, with support from tutors and other students.
- DigitalThink courses work in your browser. No plug-ins, file downloads, or new software are required. Students can work on any computer platform, on any Web-connected computer.

Home | Courses | Corporate Training | Support | Events | Newsroom | Training Resources | Company | Site Map

XML Fundamentals

Module 1

Lesson 4

Objective

What to expect

Find out how the course is structured.

Throughout this course, you'll have the opportunity to use several instructional DigitalThink Java applets.

MouseOver applet

In this course, MouseOver applets explain or dissect XML or HTML code. The MouseOver will appear as a graphic.

To display pop-up explanations of the code or its results, move your mouse cursor over any section enclosed in a red box. If you don't see red boxes, simply move your mouse over each line of code.

FlipBook applet

FlipBook applets present a series of images you can flip through, forward or backward. In this course, we'll be using FlipBooks to walk through the process of coding in XML.

MouseOver and FlipBook applets used in this course are always followed by a link to a non-applet version. So even if your browser doesn't support Java, you won't miss any information.

Simulation applet

This course includes Simulation applets which allow you to view demonstrations of XML projects. These simulations are designed to give you a firsthand view of how XML might be used.

Glossary

Throughout this course, you'll have the opportunity to review definitions of key terms. Many terms will link to their definition, which will appear in a small window.

In addition, glossary links for key terms appear in each module wrap-up lesson. To get a feel for how it works, click the word *glossary* now.

Sidebars

Throughout this course, you will notice "sidebar" links to pages that contain examples or additional information. These sidebars contain important information that is required for successful completion of quizzes or exercises. Please take the time to read all sidebars in this course.

Exercise submissions

After completing some exercises, you will be required to cut and paste your code into a text submission box. Copying and pasting is generally easy to accomplish, but sometimes on Unix platforms, this may require a workaround.

View Image

Occasionally in this course, you'll need to examine screenshots or other images that are larger than our standard lesson pages. When this is necessary, you'll see a View Image button in the left margin. Click this button to open a new Web browser window and view the image. Try it now by clicking the View Image button to the left.

Due to browser compatibility issues, subsequent openings of new browser windows may not bring the secondary window to the front and it may be hidden behind your main browser window. You can avoid any confusion by either closing the new window after you have viewed the contents, or moving it to the side so that it is visible along with the main window.

Tip

Click the Syllabus button in the toolbar to easily navigate between lessons and modules. Clicking it will open a smaller browser window that displays a list with hyperlinks to this course's lessons, exercises, and quizzes. The red arrow indicates where you are in the course.

Resources

Optional book | Downloads | Internet resources

Course

XML Fundamentals

Instructor

Jeff Brown

Tutors

To send a message to one of the course tutors, click the Tutor button on the toolbar.

Return to course

To return to the course, click the Syllabus button on the toolbar and look for the red arrow. This is the page you last visited in the course.

Optional book

This book is *not* required to complete this course.

Back to top

Xml : Extensible Markup Language
by Elliotte Rusty Harold

This optional book is available for purchase through our online Bookstore.

Downloads

Please be sure to download the appropriate files for your platform.
Back to top

Resource downloads

Macintosh: in510_resources.sea.hqx (50KB)
Windows: in510_resources.zip (15KB)
UNIX: in510_resources.tar.gz (9KB)

Explanation: The download contains the starter files for the course exercises, as well as some XML demonstration files.

Course PDF

Macintosh: in510.pdf.sea.hqx
Windows: in510pdf.zip
UNIX: in510.pdf.gz

Explanation: These compressed files contain a PDF file of the course so you can read and review the lesson, exercise, and quiz content offline at your convenience. You will need Adobe Acrobat Reader version 3.0 or higher to read this file. If you don't have Acrobat Reader, you can download it from the Adobe Web site.

Internet resources

Note: these links will launch a new browser window.

Back to top

XML Resources Online

The following sites provide information on XML. Some provide demonstrations and instruction as well. Check these sites for updates on this evolving Internet language. Due to the constantly changing nature of the Internet, some addresses may no longer be valid.

The World Wide Web Consortium's XML Recommendation

You can track the progress of XML and find the entire XML specification at the World Wide Web Consortium (W3C) under the category of Architecture.

XML.com

Sponsored by O'Reilly & Associates, Songline Studios and Seybold Publications, this site offers a host of valuable resources on XML. Of special interest is Tim Bray's annotated copy of the W3C's XML 1.0 Recommendation.

Summer Institute of Linguistics

The Summer Institute of Linguistics maintains a wealth of information on both SGML and XML. The XML resources are maintained by Robin Cover.

The Microsoft XML Site

You can find a host of valuable resources here, both in terms of downloadable components, whitepapers and demonstrations. This site has a wealth of real-world, practical information that should give you good ideas for how to use XML in your enterprise.

DataChannel

DataChannel is a relatively new company, founded by Dave Pool (one of the original founders of SPRY), that has already developed a powerful XML application called DataChannel Rio. Rio allows corporate users to publish documents directly to the Web using a behind-the-scenes XML channel mechanism. DataChannel provides a variety of articles on XML at this site.

Finetuning

Lisa Rein, author and SGML/XML developer, has put together an ever-growing site of links relating to every part of XML and its companion technologies of namespaces, metadata, XSL, and more.

Adam Rifkin and Rohit Khare's XML Resources

Rifkin and Khare have collected a wealth of useful resources.

xmlinfo.com
xmlsoftware.com
schema.net

Three sister sites which aim to provide well-organized information and resources on XML.

Microsoft XML Public Newsgroup
News server: msnews.microsoft.com
Newsgroup: microsoft.public.xml

The Microsoft news server hosts an XML group. Access the news server and subscribe to the group if you want to monitor ongoing developments in this arena. Many of the authors of specifications you encountered in this course are present in this newsgroup.

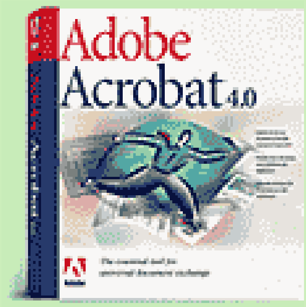
Adobe Acrobat Reader

View and print millions of documents on the Web with the free Adobe(R) Acrobat(R) Reader.

But Wait

There's more to Acrobat than the Reader!

Adobe Acrobat 4.0 will transform the way you communicate. With Acrobat, you can:



- ✓ **Easily share your documents, images, and graphics with anyone, anywhere via e-mail**
- ✓ **Capture entire Web sites in a single portable file** to easily view, navigate, annotate, print, and e-mail**
- ✓ **Add comments and annotations to documents and share them electronically**

...And much, much more. Now you can work smarter, not harder. U.S. and Canadian customers can order Acrobat 4.0 after **March 29** for just US$249, or upgrade from an earlier version for US$99.

Get more for $249*

*Volume pricing is also available for Acrobat 4.0.

**Initially available in Windows versions only.

Join the millions of people who have enhanced their communications with Adobe Acrobat!

Get the free Acrobat Reader

The Adobe Acrobat Reader lets you view and print PDF files on all major computer platforms.

Note: You will receive the most recent version of Acrobat Reader that is available for your language and platform. This means you will receive version 3 if Acrobat Reader 4.0 has not yet been released for your language or platform. Step 3 displays the version and file size of the downloaded file based on the selections you make in Step 1.

Important Information

- System Requirements
- What is Acrobat Reader?
- What's New in Acrobat Reader 4.0?
- Asian Font Packs for Acrobat Reader 4.0
- Acrobat Reader and Acrobat Reader + Search 4.0 CD-ROM
- How to Distribute Acrobat Reader
- Troubleshooting
- No Virus in Acrobat Reader Pre-Release
- PDF Access for the Visually Impaired
- Acrobat Viewer (Java) Beta

1

Select an Acrobat Reader version:

Language

English

Platform

Win 95

Location nearest you:

USA

□ Include option for searching PDF files (currently available only for 3.x versions; longer download)

Mac .bin or .hqx?

2

Which area are you most interested in?

○ Web Publishing
○ Business Documents
○ Graphics & Design
○ Dynamic Media
○ Personal Graphics

E-mail address:

This will be used once to send registration information; it will not be retained.

3

Now **download** and **install** - it's just that easy!

Downloaded file size will be:

Downloaded Reader version will be:

Download

If you have trouble downloading, try this alternate source.

| Home | Acrobat Reader | System Requirements | What's New in Reader 4.0 |
| Asian Font Packs | How to Distribute Reader | Acrobat Reader 4.0 CD-ROM |

Send Adobe your feedback!

Legal notices and trademark attributions.
Online Privacy Policy.

Resources

Book Store

The DigitalThink Bookstore lets you purchase course texts directly over the Internet. Working with Amazon.com, the most respected name in online booksellers, we supply you with everything you need to purchase your books right now. If you prefer, you can also use this information to find the books at your local bookstore.

Course

XML Fundamentals

Optional book

This book provides useful information, but it is **not** required to successfully complete the course.

Xml : Extensible Markup Language
by Elliotte Rusty Harold
Paperback, 426 pages
Published by IDG Books Worldwide
Publication date: September 1998
ISBN: 0764531999

XML Fundamentals

Module 1

Lesson 5

Objective

The course tutors

Send email to the course tutors.

The course tutors are the people who will review all your exercise submissions and send you personal email in response. You can also email the course tutors whenever you have a question or get stuck in a lesson or exercise.

Send email to the tutors

To introduce yourself to the tutors, write an introductory email that briefly describes your programming experience. You can also explain what you hope to gain from this course.

Let's walk through the process of sending email to the course tutors:

1. Click the Tutor button on the toolbar to the left. The WebMail page appears.
2. Type **`Student introduction`** in the Subject field.
3. Type your email message in the Message field.
4. Click the Send Message button. The WebMail response page appears.
5. Click the browser Back button twice to return to this page.

Module 1

What to expect

Copying and pasting on Unix platforms

All of the text files you need to get started on course exercises are included in the download file available on the Resources page. Open the text file in your text editor and make any modifications necessary to complete the exercise. Save the file under a different name.

From text editor to browser

Once you have the exercise solution in a text file, you're ready to submit your solution to the course tutors.

Some configurations of Unix operating systems make it difficult or impossible to copy and paste between applications. To work around this, you'll need to open your exercise solution text file in your Web browser, select and copy the text, then paste it into the exercise submission text area of the DigitalThink learning environment.

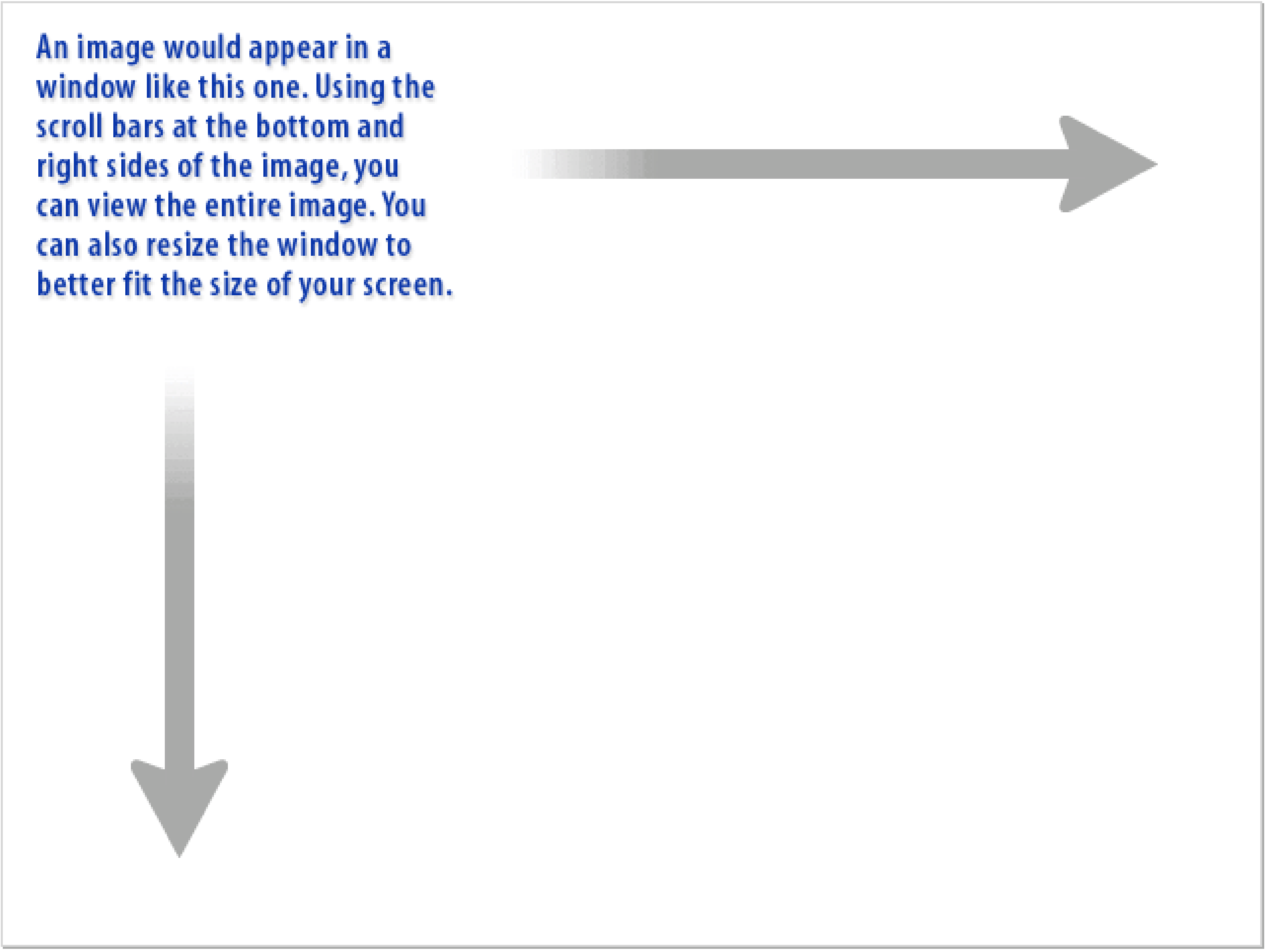
How to...

If you find you cannot cut and paste text from one application to another, follow the steps listed below. Please note that DigitalThink supports only Netscape Navigator 3.x or above on Unix platforms. The instructions refer to Navigator 4.x, but instructions for doing this procedure in Navigator 3.x are very similar:

1. When you are ready to submit an exercise solution, navigate to the appropriate exercise page in the course using the course Syllabus. You'll notice a text box in which you'll want to paste your exercise solution.
2. From the File menu in Navigator, choose the Open Page option. The Open Page dialog box appears.
3. Click the Choose File button. The Choose File dialog box appears.

4. Navigate to the appropriate directory and select your exercise solution text file. The name of your text file will now appear in the Open Page dialog box.
5. Click the Open in Navigator button. The text file will now appear in the browser window.
6. Select the text of your exercise solution and choose the Copy option from the Edit menu.
7. Click the browser's Back button to return to the course Exercise page.

8. Position your mouse cursor in the exercise submission text box, click the mouse, and then select Paste from the Edit menu. Your solution text is now ready to be submitted to the tutors.



An image would appear in a window like this one. Using the scroll bars at the bottom and right sides of the image, you can view the entire image. You can also resize the window to better fit the size of your screen.

Lesson 6

Who Wrote This Course?

Author

Lisa Pease

As an instructional designer and instructor for ProsoftTraining.com, Lisa Pease developed and delivered courses in subjects ranging from DHTML, JavaScript and XML, to Netscape Communicator and Interactive Web development. Lisa's Web site, "Real History Archive," was one of the first 600 Sites on the Web in 1993 and recently was rated in the Top 5 percent by Lycos.

Contributors

Jeff Brown and Susan M. Lane

Jeff Brown is a senior instructor for ProsoftTraining.com. He teaches Internet topics ranging from internetworking fundamentals to dynamic Web site design and intranet applications.

Susan M. Lane is a senior editor for ProsoftTraining.com and has extensive experience translating Internet topics such as XML and Java programming into clear, straightforward language.

DigitalThink Team

Amy Cowen, Tutor/Developer
Yingzhao Liu, Designer
Joel McKinnon, Producer
Amy Melnicsak, Developer/Editor

XML Fundamentals

Module 1

Lesson 7
Objective

Course exercises

Find out about the course exercises.

The exercises in this course are designed to take you step-by-step through the process of XML coding. In order to learn proper coding syntax, you'll edit text files which present HTML and XML code. You'll also create your own text files to practice manual coding skills and logical markup. Throughout the course, you'll create tags based on content rather than visual formatting.

You'll also create Document Type Definitions (DTDs) to define these elements for XML files. In addition, you'll design style sheets for an XML document using the Extensible Style Language (XSL).

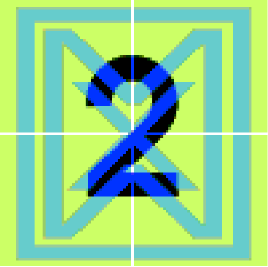
If you're using IE 5.0, one optional exercise shows you how to use your browser to check your code for well-formedness and validity. Additional optional exercises allow you to use a Java applet to perform databinding in an XML file.

At the end of the course, you'll be able to create your own projects with XML and its related technologies.

XML Fundamentals

Module 2

Lesson 1



What is XML?

Module introduction

Imagine that while driving through unknown territory, you could ask an onboard car computer for directions to the nearest gas station. For that to be possible, markup languages must be specific not just in terms of document structure. They must be specific about the actual content contained within the documents.

XML was created expressly to provide a manner of defining both structure and content. This module discusses the evolution of markup languages and the origins of XML.

Module learning objectives

By the end of the module, you will have the skills and knowledge necessary to:

- Explain the origins of HTML
- Describe why HTML strayed from its original specification
- Examine some of the limitations of HTML
- Define XML
- Cite the goals that XML creators had in mind
- Consider whether XML will replace the use of HTML
- Give examples of advantageous uses of XML

Discussion

You may find helpful information about XML in the Discussion area for this course. Click the Discuss button and browse through the folders to view questions and comments from your classmates and course tutor. If you have a question that someone else hasn't already asked, you can post it yourself.

Lesson 2

Objective

The vision of HTML

Explain the origins of HTML.

HTML was originally created to define structure, not formatting. The original specification had a bare-bones set of *markup* elements that designated text as a header, a paragraph, a list item, and other simple units. In its earliest form, HTML provided an incredibly simple and effective way for people to generate clear, readable documents.

When you remember that the Internet was originally made popular by academics who wanted to share research across great distances, you can understand why the first version of HTML was mostly focused on creating clear, simple pages rather than flashy, interactive home pages for movie studios or other corporations. The following code represents a fragment of an HTML file. Notice that the markup is straightforward:

MouseOver

Transcript

This original language focused on the structure of the data. It presented a simple set of tags to represent very simply structured documents. HTML made it possible for anyone to easily create Web pages. The sheer amount of data that was suddenly available in a short time quickly made the Web an integral part of society. The Web blossomed beyond even the most optimistic projections in terms of widespread acceptance and popularity.

Module 2

Lesson 3
Objective

HTML hijacked

Describe why HTML strayed from its original specification.

With the commercialization of the Internet, the number of Web page creators grew exponentially, and not all users cared about the original vision. Raised in the *WYSIWYG* environment of the current word-processor generation, users wanted the ability to make type bold, change the font size or face, and add color.

Netscape, producer of the first widely successful browser, came to the rescue of users frustrated with the limitations of the HTML language by creating extensions to the language. As these extensions became widely used, they were ultimately folded into later versions of the HTML language.

The following code fragment demonstrates not just structural markup, but procedural markup indicating formatting characteristics. Looking at this document, you can see that while it may be becoming more visually attractive, it is losing structure. Tags that do not define structure appear in bold:

MouseOver

Transcript

Because it doesn't offer very useful information to a machine, this document is not as "intelligent" as it could be.

XML Fundamentals

Module 2

The vision of HTML

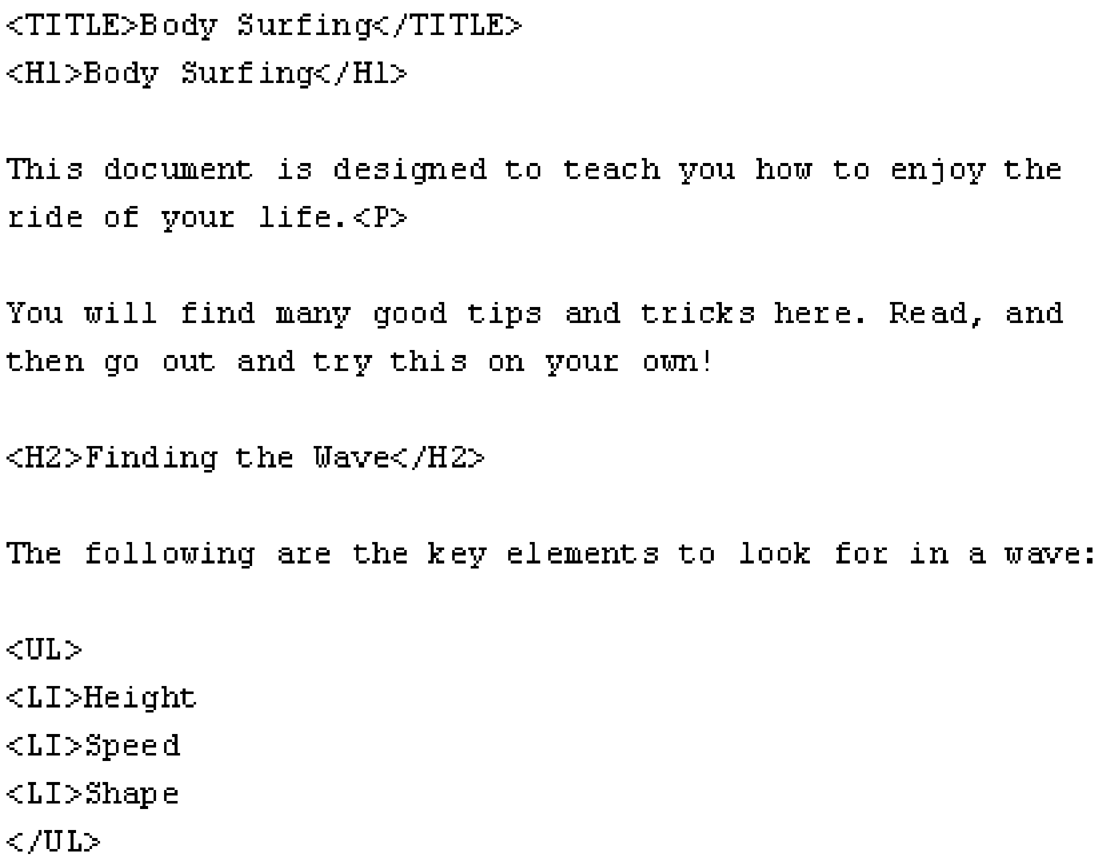
The birth of HTML

In 1989, Tim Berners-Lee wanted to find a language that would allow people from all over the world to create documents that could be read by a universal client, a product more commonly called the Web browser. From that noble idea, HTML was born.

Berners-Lee thought that *SGML* provided the most promising model, but knew the difficulty of mastering the language would hamper the development and sharing of information. He created a very simple, efficient language for marking up text. HTML is an application created in SGML. HTML is not extensible. There is a finite set of HTML elements, which are entered into pages as tags.

The vision of HTML

The vision of HTML (MouseOver transcript)



```
<TITLE>Body Surfing</TITLE>
<H1>Body Surfing</H1>

This document is designed to teach you how to enjoy the
ride of your life.<P>

You will find many good tips and tricks here. Read, and
then go out and try this on your own!

<H2>Finding the Wave</H2>

The following are the key elements to look for in a wave:

<UL>
<LI>Height
<LI>Speed
<LI>Shape
</UL>
```

<TITLE> A title for the document is identified by beginning and ending <TITLE> tags.

<H1> The main heading is marked as a heading level 1 by the <H1> tag.

<H2> <H2> marks the beginning of a subheading.

Plain text Plain text is not marked up.

<P>	When a paragraph break is needed, the <P> tag is used to start a new paragraph.
<UL>	<UL> begins an unordered or bulleted list.
<LI>	<LI> indicates a list item.
</UL>	</UL> ends an unordered or bulleted list.

XML Fundamentals

Module 2

Lesson 4

Objective

HTML limitations

Examine some of the limitations of HTML.

HTML will be used for a long time because it allows complex pages to be presented simply. However, continued devotion to the language causes other problems that may not be immediately apparent.

Server-load and Internet traffic jams

You have probably noticed that the Web gets bogged down at certain times of day. Traffic is up. With all the page requests hitting servers, traffic jams are to be expected. Given the current state of HTML, much data is processed on the server.

If you wish to view a data report and then view the same data sorted in a different order, you probably have to query the server twice. This action creates a load on the server, increases traffic across telephone lines, and adds to the congestion on the Internet.

One of the goals of those who are developing new specifications for the Internet is finding new ways of shifting processing to the client's computer and away from the server. *Dynamic HTML* (DHTML) offers potential for reducing some of this load. Without intelligently coded data, however, its potential is still limited.

Search engine overload

Anyone who has tried to search for a document on the Internet has inevitably entered a search word or phrase that returned literally thousands of matches, called "hits." Too much data is almost as useless as no data. By the time you read through all the documents on the Internet returned from your search, you probably could have found the information you needed by some other means.

Quiz

Click the Quiz button to check what you've learned about HTML's development.

HTML hijacked

Separating format from structure

The WYSIWYG wave made Tim Berners-Lee's vision of well-structured documents readable with a universal client more remote. In an effort to regain the original vision, a proposal was made and accepted by the World Wide Web Consortium (*W3C*) for separating formatting elements from HTML. The first version of this effort came to fruition in the Cascading Style Sheets, Level 1 *Recommendation* (CSS1). CSS2 is now the recommendation that supercedes and extends CSS1.

CSS states that all formatting should be defined either in a separate document called a style sheet, in a `STYLE` section within an HTML page, or as values for a `STYLE` attribute within tags. In other words, the use of tags such as `<CENTER>` and `<FONT>` are now discouraged by the W3C, and these elements are said to be "deprecated" in favor of style sheets. Authors are strongly discouraged from using deprecated elements when creating code for their Web pages.

HTML hijacked

HTML extensions

While users of new extensions enjoyed the ability to choose from a wide variety of font options and colors, most were unaware that Pandora's box had been opened. Netscape added non-standard formatting elements to the HTML language. Microsoft joined the fray, adding its own propriety language extensions.

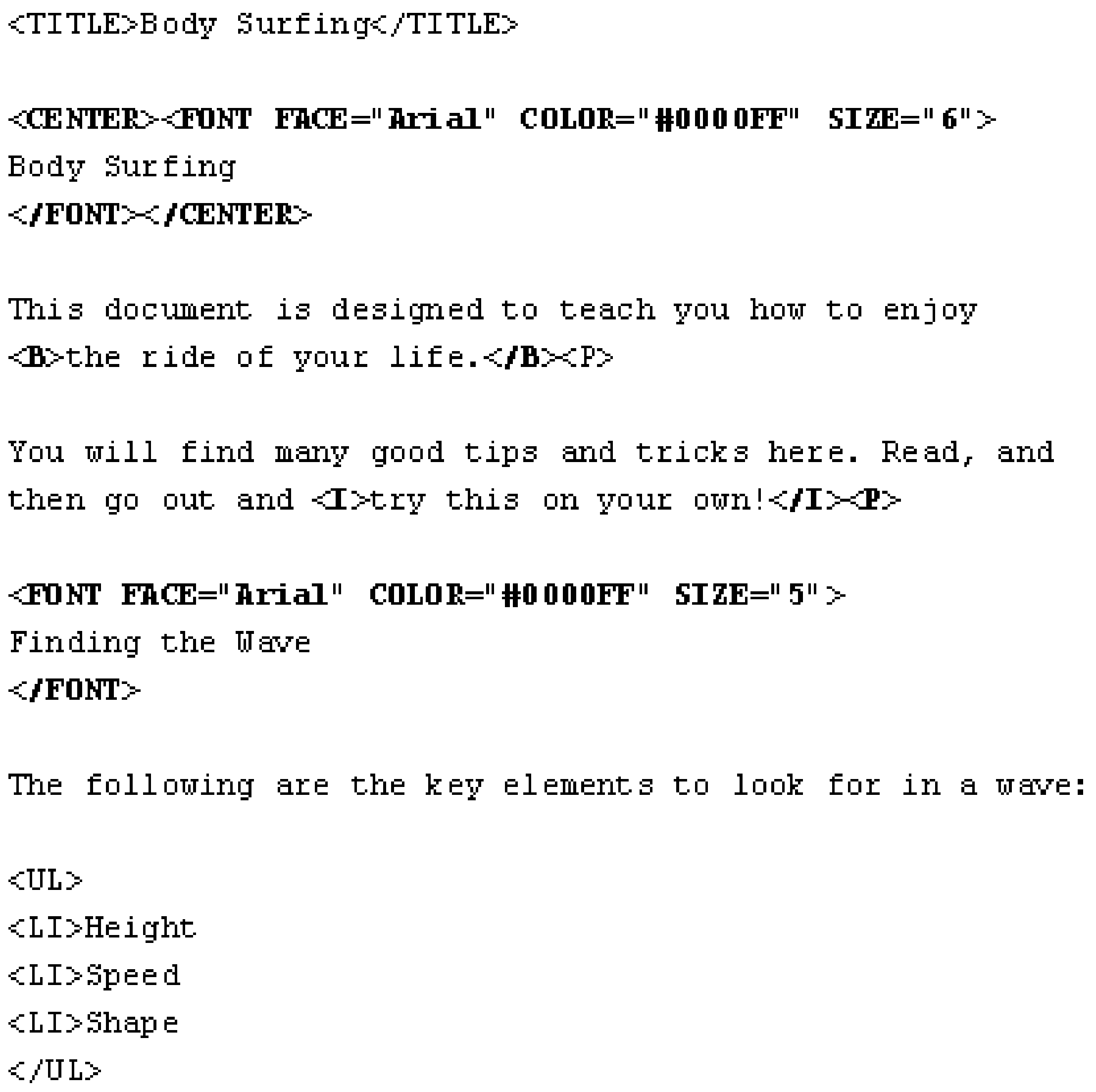
Soon, page developers found themselves frustrated. The tantalizing extensions were not part of any standard and were usually recognized only in the browser made by the company that created the extension. It takes a long time for a new markup element to be proposed, discussed, and accepted into a new version of HTML. Netscape and Microsoft opted to present first and propose later.

XML Fundamentals

Module 2

HTML hijacked

HTML hijacked (MouseOver transcript)



```
<TITLE>Body Surfing</TITLE>

<CENTER><FONT FACE="Arial" COLOR="#0000FF" SIZE="6">
Body Surfing
</FONT></CENTER>

This document is designed to teach you how to enjoy
<B>the ride of your life.</B><P>

You will find many good tips and tricks here. Read, and
then go out and <I>try this on your own!</I><P>

<FONT FACE="Arial" COLOR="#0000FF" SIZE="5">
Finding the Wave
</FONT>

The following are the key elements to look for in a wave:

<UL>
<LI>Height
<LI>Speed
<LI>Shape
</UL>
```

<CENTER>

Nothing in this document indicates that "Body Surfing" is a heading. A computer only knows that it should be displayed differently. The text's relative role in the document cannot be determined with the existing tags.

<FONT>

In essence, the <FONT> tags have been used to fake structure. Visually, the headings are clearly defined. To a machine, it is all just text.

XML Fundamentals

Module 2

Lesson 5
Objective

XML defined

Define XML.

XML stands for Extensible Markup Language. This new language will enable users to create documents that contain more specific information about content than ever before, adding a certain level of "intelligence."

HTML has devolved from its original intent of defining structure. But even defining structure is not enough. To fully exploit the potential of evolving technologies, we must define not only the document structure, but the actual content as well.

XML is derived from the Standard Generalized Markup Language (SGML). SGML is a meta-language, which is a language for creating other languages. To understand XML, you should first have a basic understanding of what a markup language is, what a *meta-language* is, how SGML spawned XML, and how XML differs from HTML.

Audio
Transcript

Click the Audio button to hear how Dan Connolly, Rohit Khare and Adam Rifkin illuminate the potential of XML.

XML Fundamentals

Module 2

HTML limitations

Searching and finding in HTML documents

HTML does not allow you to make very refined searches across data. For example, if you want to search for information relating to the box office performance of the 1997 film *Titanic*, you might be deluged with articles about the ship *Titanic*, the numerous books about its fatal voyage, or pages of sales promotions touting "titanic" discounts.

XML Fundamentals

Module 2

Quiz

The development of HTML

Each question is worth one point. Some questions ask you to select **the best answer**, others ask you to select **all the correct answers**. To receive credit for questions asking for **all the correct answers**, you must select all the correct answers and **only** the correct answers.

1.The original HTML specification:

Please select **the best answer**.

○ A. Provided a means to create "intelligent" documents
○ B. Was difficult to master, and hampered the development and sharing of information
○ C. Was a bare-bones set of markup tags created to define structure, not formatting
○ D. Was focused on creating flashy, interactive home pages for large corporations

2.Which of the following technologies was proposed to regain HTML's original vision by separating format from structure?

Please select **the best answer**.

○ A. DHTML
○ B. CSS
○ C. XML
○ D. W3C

3.What are some of the limitations of HTML?

Please select **all the correct answers**.

□ A. It contributes to Internet congestion.
□ B. It cannot generate clear, readable documents for the user.
□ C. It cannot accommodate highly refined searches across data.
□ D. It shifts processing to the client's computer and away from the server.
□ E. The syntax is complex to learn and use.

4.Which of the following is an HTML tag that complies with Tim Berners-Lee's original vision for the language?

Please select **the best answer**.

○ A. <P>
○ B. <STYLE>

○ C. <CENTER>
○ D. <B>
○ E. <FONT>

OK, I'm Done

Module 2

Lesson 6 Objective

The goals of XML

Cite the goals that XML creators had in mind.

XML was created by the XML Working Group and the XML Special Interest Group at the W3C, under chairman Jon Bosak of Sun Microsystems. The W3C's recommendation for XML 1.0 was published in February 1998.

The XML *Recommendation* delineates the 10 goals that XML creators had in mind:

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs that process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance.

XML Fundamentals

Module 2

XML defined

The history of SGML

In the late 1960s, as computers started to be widely used in certain arenas, a group called the Graphic Communications Association (GCA) created a layout language called GenCode. GenCode was designed to provide a standard language for specifying formatting information so that printed documents would come out looking the same, regardless of the hardware used.

In 1969, Charles Goldfarb led a group of people at IBM who built upon the GenCode idea and created what became known as the General Markup Language (GML). Where GenCode was primarily a procedural markup language, GML strove to define not only the appearance, but to some degree, the structure of the data.

Nearly 10 years after GML emerged, the American National Standards Institute (ANSI) established a working committee to build upon GML to create a broader standard. Goldfarb was asked to join this effort, and has since become known as the "father of SGML," which was the end product of ANSI's efforts. The first draft of SGML was made public in 1980. The final version of the standard emerged in 1986.

Since that time, the language has been enhanced as needed. For example, in 1988, a version of SGML was created that was designed specifically for military applications (MIL-M-28001). Other additions to the language have been incorporated over the years, and now some people feel that SGML suffers from complexity bloat.

XML defined

What markup languages do

Markup languages are designed to tell machines, particularly computers, how to process data. The term "markup" derives from early print publishers, who would "mark up" text by hand to indicate to the printer which font size to use where, in which weight, using what form of alignment, and so forth. In other words, the earliest markup languages were dedicated to passing formatting instructions.

Markup instructions are generally referred to as tags, and the process of marking up a document is sometimes called "tagging." Early word-processing programs required the user to perform manual tagging. Today, most tagging in programs happens behind the scenes, and usually takes place using a proprietary system. The divergence of methods for tagging text made it hard for people to exchange data with each other.

With the advent of the Internet, it became more valuable and more imperative for authors to be able to interchange documents in a format that was easy to use, yet powerful and aesthetically acceptable.

Markup specifically designed to affect the appearance of a document is commonly called "procedural" markup because it instructs the computer how to render the text. But organizations that process huge numbers of documents, such as government and bureaucratic entities, quickly found that it was more important to know what the data represented rather than how it looked.

Markup was created that described the *content* of the page. This type of markup is called "descriptive" or "logical" markup. The following is HTML procedural markup for tagging the word "Summary" to appear in bold print:

```
<B>Summary</B>
```

In logical or descriptive markup, it would likely make more sense to mark up the word "Summary" as a section header. The following is HTML logical markup for tagging the word "Summary" as a subheading:

```
<H4>Summary</H4>
```

XML defined

SGML to XML

SGML provides a means for every word, every number, every phrase or paragraph to have a contextual significance. For example, "1989" might be a year or a number. *Gone with the Wind* is the name of a book and a film, and it is also an expression. SGML provides a way to give the phrase the appropriate context through markup.

SGML is a *meta-language*. HTML is a language created by SGML.

SGML has many tools for defining structure, context and content. Consider some analogies to describe each of these. In English, sentences have a basic structure:

```
Subject verb object.
```

For example:

```
He ate grapes.
```

The rules of the language discourage playing with the structure. "Grapes he ate" is awkward. "Grapes ate he" is wrong both syntactically and semantically.

In HTML, you can use the `<H1>` tag to denote context, specifically a heading. But you cannot put a heading in the `HEAD` element of the HTML page. That use of structure is considered grammatically incorrect. The content of an `H1` element is primarily text, but a `SPAN` element could be used within an `H1` element to define some temporary formatting, such as an italic word within the heading. SGML gave HTML its structure, defined the context, or grammar, for the usage of elements, and provided rudimentary considerations as to content.

XML Fundamentals

Module 2

XML defined

HTML and XML contrasted

Many people think XML is just a way to add tags to HTML. **This assumption is incorrect.**

Several mantras, listed below, were repeated often at the XML conference held in Seattle in the Spring of 1998.

- HTML is display.
- XML is content.
- XML is not HTML.

Unlike HTML, XML is not an application, but a subset of *SGML*. XML is SGML made simpler and more accessible. It is sometimes referred to as "SGML Lite." As such, XML qualifies as a *meta-language* and can be used to write other languages. XML provides users with the ability to define their own set of markup tags-to write their own version of HTML, so to speak.

XML Fundamentals

Module 2

XML defined

Describing content in intelligent terms (Audio transcript)

"The Web itself is becoming a kind of cyborg intelligence: human and machine, harnessed together to generate and manipulate information. If automatability is to be a human right, then machine assistance must eliminate the drudge work involved in exchanging and manipulating knowledge...."

This quote is from the paper "The Evolution of Web Documents: The Ascent of XML," in which the authors illuminate the potential of XML as a markup language that describes content in intelligent terms. XML was created to function behind the scenes, providing machines with information about documents that will facilitate decision making at the machine level.

Module 2

Lesson 7
Objective

Will XML replace HTML?

Consider whether XML will replace the use of HTML.

Will XML replace HTML? The short answer is no, but over time, the answer may well be yes. HTML presents severe limitations to page authors. It will continue to be the appropriate language for documents not intended to have a long shelf life. But for data that needs persistence over time, XML will better serve the needs of those looking to retrieve and reuse that data.

XML will redefine the way we code HTML. In the short run, XML will be used in conjunction with HTML. As browsers develop and style technologies progress, XML may be used without HTML. But HTML and XML are apples and oranges, and one technology is not necessarily better or worse than the other. Both are tools that have appropriate and inappropriate uses. The more you know about each, the better equipped you are to choose the best tools to meet your enterprise's needs.

What is XML?

Module wrap-up

As you learned in this module, XML promises to meet needs which HTML can never fulfill, and for which SGML may be too complex. As a meta-language, XML is not a set of tags, but a specification for creating customized tags. By using XML, machines will begin to understand Web pages as well as humans do.

Because it is a budding technology, you can expect to see many changes in XML, just as we have seen with HTML.

This module discusses the following terms in relation to XML:

- *markup*
- *SGML*
- *WYSIWYG*
- *Recommendation*
- *W3C*
- *DHTML*
- *meta-language*

Discussion

Do you have any questions or comments about this module's material? If so, don't forget about the Discussion section, which you can reach by clicking the Discuss button on the toolbar. The folder on markup languages may contain the information you seek.

Quiz

Quiz

Click the Quiz button to test what you've learned in this module.

XML Fundamentals

Module 2

Will XML replace HTML?

Targeted searches

With the advent of XML, search engines will be able to perform more directly targeted searches, enhancing the precision of the results.

Consider the value of being able to define content by comparing the following code examples:

```
The best picture award in 1998 went
to the film <I>Titanic</I>.
```

In this example, a search engine cannot determine whether you are referencing a ship, a play, a film or an adjective. The second markup example is more useful:

```
The best picture award in 1998 went
to the film <FILM>Titanic</FILM>.
```

Now you know that this reference to Titanic is a reference to a film by that name. The last example takes XML a step further:

```
The

<ACADEMY-AWARD-CATEGORY>best
picture</ACADEMY-AWARD-CATEGORY>
award in <YEAR>1998</YEAR> went to
the film

<TITLE
MEDIA="Film">Titanic</TITLE>.
```

The last example makes the most "intelligent" document because the markup is explicit as to what the document contains. If you were to ask an intelligent search engine the question "Which film won the Academy Award for Best Picture in the year 1998?" this document would have a high degree of precision in matching your search query.

Will XML replace HTML?

XML uses beyond the Web

Although the focus of this course will be on how to prepare XML documents for deployment across the Internet, a substantial benefit of XML is the ability to define a language that allows different types of presentation.

Using XML, we may soon be able to create a single set of data that can be delivered on the printed page, on a CD, in HTML format, in streaming audio, or to a pager's LCD display panel. The format could be controlled by settings in the style sheet, but no recoding of XML pages need necessarily take place. Imagine the power of outputting anything you create to any of these formats without additional modification.

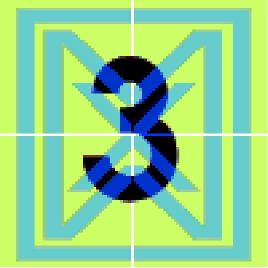
Module 3

Lesson 1

Creating XML documents

Module introduction

How difficult is it to meet the requirements of XML? Surprisingly, there's little to memorize. You must only adhere to a small and simple set of rules. This module explains the set of rules and shows you how to create useable, well-formed XML pages.



Module learning objectives

By the end of the module, you will have the skills and knowledge necessary to:

- Construct XML documents
- Clarify the differences between tags in XML and HTML
- Specify the five rules of creating a well-formed XML document
- Create an HTML document that follows the well-formed XML rules
- Create a well-formed XML document
- Add clarity and information to XML documents using comments, CDATA sections, and encoding

<! -- XML Fundamentals >

Module 2

Quiz

Defining XML

Each question is worth one point. Some questions ask you to select **the best answer**, others ask you to select **all the correct answers**. To receive credit for questions asking for **all the correct answers**, you must select all the correct answers and **only** the correct answers.

1.Markup that describes the content of a document is called:

Please select **all the correct answers**.

□ A. Descriptive markup
□ B. Hypertext markup
□ C. Procedural markup
□ D. Logical markup
□ E. Extensible markup

2.SGML was used to create HTML's structure. As such, SGML qualifies as a:

Please select **the best answer**.

○ A. Programming language
○ B. Markup language
○ C. Meta-language
○ D. Formatting language
○ E. Web language

3.Which of the following statements accurately describe the relationship between HTML and XML?

Please select **all the correct answers**.

□ A. XML is not HTML.
□ B. XML adds tags to HTML.
□ C. XML is content.
□ D. XML is a subset of HTML.

4.Which of the following is a characteristic of XML?

Please select **the best answer**.

○ A. It defines format in Web documents.
○ B. It enables search engines to perform more directly targeted searches.
○ C. It allows a single type of presentation.

OK, I'm Done

XML Fundamentals

Module 3

Lesson 2

Objective

A simple, well-formed XML document

Examine a simple, well-formed XML document.

The following text demonstrates a complete, well-formed XML file:

```
<greeting>Hello, World!</greeting>
```

The above is a well-formed XML document. It is not a large document, but it is error free.

For reasons of forward compatibility, one additional element is requested, but not required, at the start of XML documents: the XML declaration that specifies to which version of XML the document conforms. The following example begins with an XML declaration providing version information.

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

<greeting>Hello, World!</greeting>
```

Audio

Transcript

What exactly is an XML document? Consider the definition from the W3C's Recommendation for XML 1.0.

Tip

Although the opening `<?xml?>` declaration statement is considered optional, it is highly recommended for forward compatibility. As the language evolves, it will be helpful to future browsers to know which version of the XML specification was used to create the document. If used, this declaration must be all lowercase.

XML Fundamentals

Module 3

Lesson 3

Rules for a well-formed XML document

Objective

Memorize the rules for constructing well-formed XML documents.

Well-formedness is essential in XML. Documents that are not well-formed will not load in the browser, according to the XML Recommendation. The W3C instructs us that "violations of well-formedness constraints are fatal errors." This statement means that the page will not appear in the browser at all unless this minimum requirement is met.

Five basic rules will help you construct well-formed XML documents. You should commit these rules to memory:

1. Tags cannot be inferred; they must be explicit.
   All start tags must have corresponding ending tags. All ending tags must have corresponding opening tags.
2. Empty tags need a / character before the closing angle bracket.
3. All attribute values must be enclosed in single or double quotation marks.
4. Tags must nest correctly.
5. Tags are case-sensitive and must match each other in every implementation.

Exercise

Exercise

Once you're familiar with these rules, click the Exercise button to correct a poorly-formed HTML document.

XML Fundamentals

Module 3

A simple, well-formed XML document

XML, the lazy developer's nightmare

HTML rendering programs have been very forgiving with developers. If you forget to close a tag, the closing tag is usually inferred. If you use the wrong tag, the browser disregards it and renders the document, no matter how ill-constructed it is. For that reason, HTML is great for the lazy developer. Half an effort can still produce attractive pages.

XML, on the other hand, is far stricter. Any single error will prevent your page from rendering. XML is not for those who dislike detail or want to construct pages haphazardly. XML requires attention and discipline.

Module 3

A simple, well-formed XML document

What makes an XML document? (Audio transcript)

“ Before you create an XML document, consider what exactly constitutes an XML document, according to the W3C's Recommendation for XML 1.0:

"A data object is an XML document if it is well-formed, as defined in this specification. A well-formed XML document may in addition be valid if it meets certain further constraints."

This definition indicates that the *minimum* requirement for a document to be considered an XML document is that it be well-formed. ”

Module 3

Lesson 4

Objective

Discerning structure

Determine the inherent structure of information within documents.

The first step in creating any XML document should be to determine the inherent structure of the information within the document. Structure depends on individual preferences. You can structure a simple document in many ways.

Flip through the following frames to consider the structure of sample documents. You will examine a business letter and a product catalog:

Transcript

Quiz

Quiz

Click the Quiz button to see what you've learned about XML documents.

Rules for a well-formed XML document

The importance of starting and ending tags

In HTML, you became familiar with the fact that you could use the following syntax to create a bulleted list:

```
<ul>
<p><li>Bullet list item one
<p><li>Bullet list item two
<p><li>Bullet list item three
</ul>
```

Note that the `<UL>` and `</UL>` tags denote the beginning and ending of an unordered list. `<LI>` denotes the beginning of the current list item, but also can be inferred to denote the end of the previous list item in its second and third appearance. In other words, the end of the element has been inferred. If you were to code like this in an XML document, however, the browser would be instructed not to render this page because it is not well-formed.

In XML, tags will never be inferred and must always be made explicit. In other words, if the above example were to be included in an XML document, it would have to be written like this:

```
<ul>
<li>Bullet list item one</li>
<li>Bullet list item two</li>
<li>Bullet list item three</li>
</ul>
```

In the second instance of this list, you see that each list item is enclosed between beginning and ending tags, meeting one criterion for a well-formed XML document.

XML Fundamentals

Module 3

Rules for a well-formed XML document

New rules for empty tags

Tags that contain the relevant information within the tag boundaries but enclose no text are called empty tags. Examples of empty tags in HTML are `<BR>` and `<IMG>`, neither of which needs an ending tag because the tag itself carries all the relevant information. `<BR>` instructs the rendering program to display a line break at that point. `<IMG>` is used to include an image in the text, and the name of the image file is specified as a value of the required `SRC` attribute within the tag.

HTML is a language with a set number of recognized tags. It is important to bear in mind that XML is *not* a language and has *no* recognized tags. Rather, XML is a *meta-language* that allows you to create a language of your own for your pages. For this reason, XML cannot be expected to recognize that `<BR>` and `<IMG>` are empty tags. To indicate to XML that a tag is in fact empty, you must include the forward slash character (`/`) at the end of the tag. If you had `<BR>` and `<IMG>` tags in your XML document, you would code them as follows:

```
<BR/>
```

```
<IMG SRC="image.gif"/>
```

Failing to do this will prevent your XML document from being well-formed. It will also prevent it from rendering, according to the W3C's guidelines.

XML Fundamentals

Module 3

Rules for a well-formed XML document

Attribute values

In HTML, either of the following would be considered correct:

```
<TD WIDTH=25%>
```

```
<TD WIDTH="25%">
```

XML, however, requires that all attribute values be enclosed with quotes. In other words, of the two examples above, only the second would be valid for XML, and then only presuming the ending `</TD>` tag was also present.

Some users say XML documents should have no attribute/value pairs, or very few, and others wish to convey a great deal of data in attribute/value pairs. The concept of attributes and attribute values will be explored in the module *Building the DTD*.

XML Fundamentals

Module 3

Rules for a well-formed XML document

Nesting

In XML documents, tags must follow the strictest nesting rules. Consider an onion. You will not find an onion with a layer that starts on the outside and ends on the inside. An onion's layers nest one within the other.

In XML terms, you must simply close tags from the inside out.

Incorrect:

```
<NAME>Judy<SURNAME>Wilson</NAME></SURNAME>
```

Correct:

```
<NAME>Judy<SURNAME>Wilson</SURNAME></NAME>
```

XML Fundamentals

Module 3

Rules for a well-formed XML document

Matching case between tags

In XML, case specificity is required.

```
<P>The quick brown fox jumped over
the lazy dog.</p>
```

If you were to write the previous code in XML, you would get a fatal error. The `</p>` tag cannot be used to close the `<P>` tag because they were written in different cases. All tags must be in a matching case to be considered part of a matching set.

Rules for a well-formed xml document

Correcting a poorly-formed HTML document

Objective

Correct a poorly-formed HTML document to meet the well-formedness requirements for XML.

Exercise scoring

Full credit for this exercise is 10 points. You'll submit your code to the course tutors.

Overview

As you can imagine, most HTML documents do not currently meet the simple but strict requirements to rate as XML documents. However, a properly-coded HTML document could become a well-formed XML document without much trouble. In this exercise, you will correct a poorly formed HTML document so that it meets the well-formedness requirements of XML.

With the exception of how today's browsers treat some "empty" tags, the changes you make to this document will not alter the visual display of the page. XML is not about display. It is about machine-readability to facilitate more intelligent processing. So, if you are testing the finished file in your browser, you may not notice anything different after you've made the document well-formed, but to a machine, the page will make more sense.

Starter files

We've supplied the starter files in the download file available from the Resources page. The specific file for this exercise can be found in the 03-03 directory.

Instructions

1. Open the file named `poor.htm`.

```
<HTML>
<TITLE>A Poorly Formed HTML Page</HEAD>
</TITLE>

<BODY>
<H1>Preparing for Tomorrow Today</H1>

Those who take the time to code their pages properly in HTML today will have little difficulty
when it comes to converting those pages into a fully XML-compliant set of tags.
<P>

```

```
<B><FONT SIZE="5">XML Benefits</B></FONT>
<P>

<DL>
<DT>The benefits for using XML include the following:
<OL TYPE=a>
<LI> Data tagged using the XML specification will be more intelligent to machines that read the information
<BR>
<LI> Bringing XML data sets to the client will reduce the number of roundtrips needed to the server
<BR>
<LI> Reducing the number of roundtrips to the server will lessen overall Internet traffic
<BR>
<LI> XML data will be more readily sent between unlike systems, allowing for an explosion of e-commerce activities
</ol>

<h3>XML is the future!</H3>
```

2. Review the five rules of well-formedness. What violations do you see? Consider them one at a time. Which tags are inferred? Do all empty tags have a / before the ending angle bracket? Are all attribute values enclosed in quotation marks? Are all opening and closing tags nested correctly? Do all tags match in terms of case? Correct all violations of the five rules of well-formedness.

Warning

While you have to use <BR/> instead of <BR> to create well-formed XML documents, some browsers, like Netscape 4, will ignore the <BR> tag completely if it is written as an empty tag: <BR/>.

3. Recall also that a true XML document defines only structure and content, not display. Formatting will happen outside the XML document. What tags could be changed to reflect structure instead of display? Change these to more appropriate sets of tags.

3. Save your document as `well.htm`.

What to submit to the course tutors

In the text box below, cut and paste the contents of `well.htm`. Click OK, I'm Done to submit your code to the course tutors.

OK, I'm Done

XML Fundamentals

Module 3

Lesson 5 Objective

Working with mixed content

Create a well-formed XML document from text.

In the book catalog example, the XML file was structured such that every individual element was tagged.

View Image

Examine what happens when you mix more text with the information from the book catalog.

This XML document is considerably different in terms of the look and the intermingling of text with tags and text inside the `BOOK` element. Yet this still meets the rules for well-formedness and would be considered an XML document.

Exercise

Exercise

Click the Exercise button to create a well-formed XML document from text.

Module 3

Discerning structure

Advantages to creating letters as XML documents

Consider a simple business letter sent to a law office. Attorneys, among other business professionals, need to be able to track all correspondence. Some law offices have developed complex systems that store letters for retrieval using complex, coded directory structures. Others put letters into a database. Some perform full-text searches of the data by looking for names until they find the needed document.

But if attorneys start creating letters as XML documents, they could search using more specific criteria to find what they need more quickly. This system would reduce the amount of time required to service one client and free up time to pursue additional revenue.

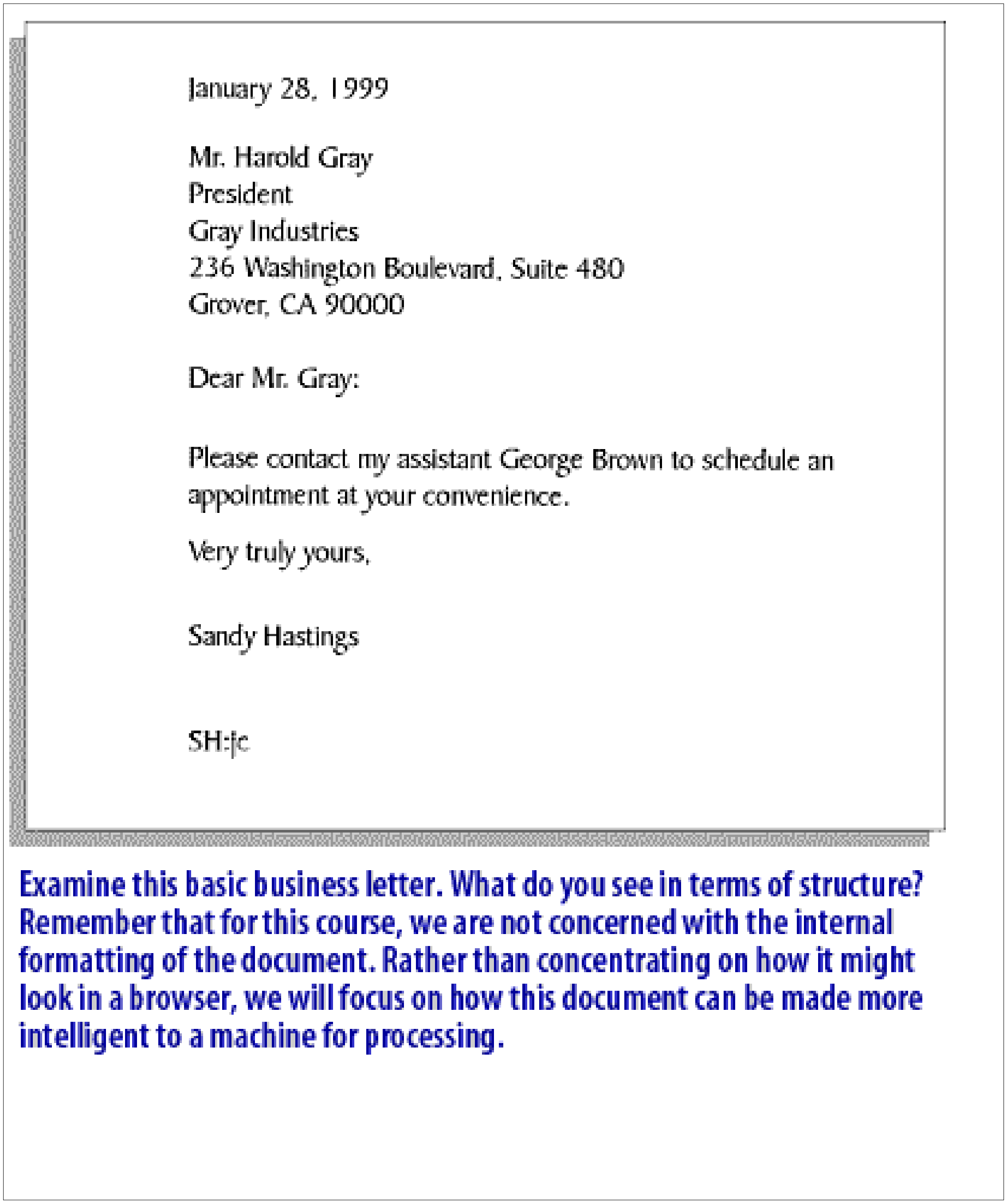
As you can imagine, XML is not appropriate for every task. As in carpentry, users are encouraged to use the most appropriate tool to complete the job at hand. If this letter is never going to be searched for, referenced or read again, there is no point in making it an XML document. But if this document will be kept and viewed in the future, then thought should be given to structuring the information for maximum future usability.

Module 3

Discerning structure

Structure of sample documents (FlipBook transcript)

The following frames explore the structure of sample documents. You will examine a business letter and a product catalog:



January 28, 1999

Mr. Harold Gray
President
Gray Industries
236 Washington Boulevard, Suite 480
Grover, CA 90000

Dear Mr. Gray:

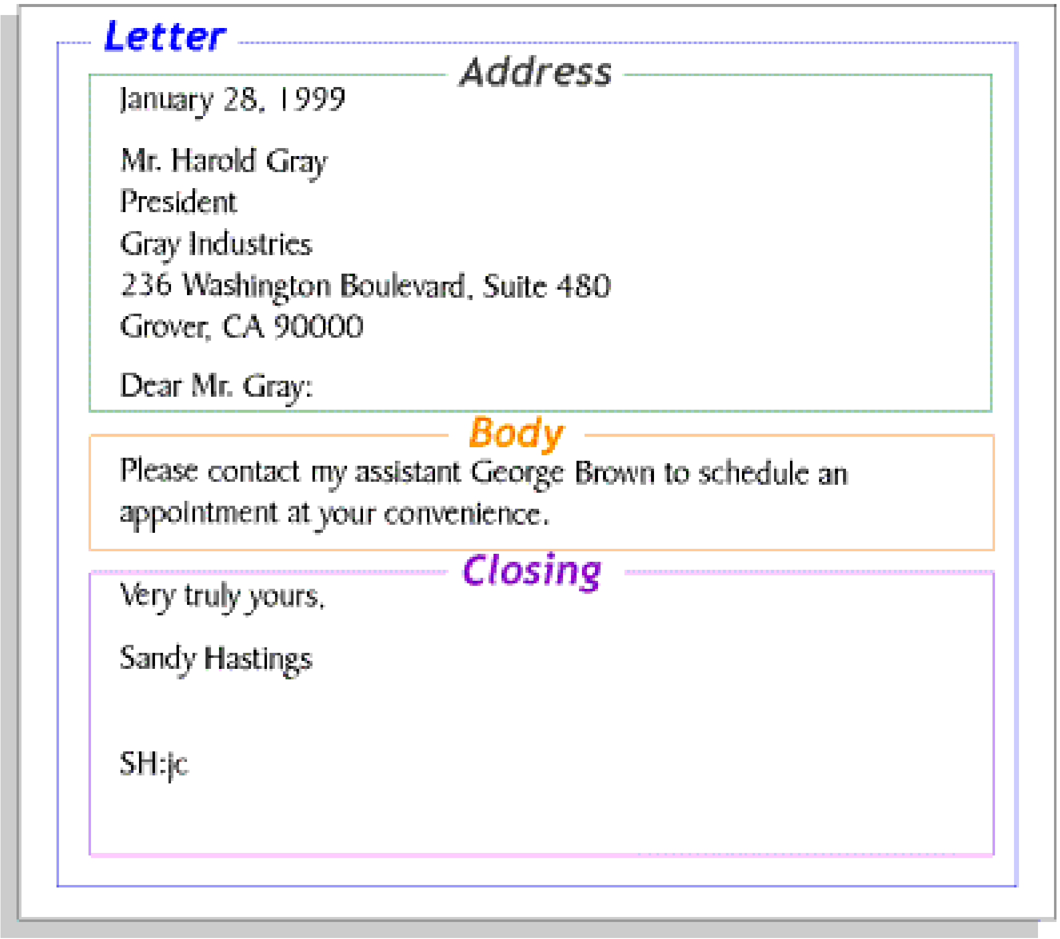
Please contact my assistant George Brown to schedule an appointment at your convenience.

Very truly yours,

Sandy Hastings

SH:jc

Examine this basic business letter. What do you see in terms of structure? Remember that for this course, we are not concerned with the internal formatting of the document. Rather than concentrating on how it might look in a browser, we will focus on how this document can be made more intelligent to a machine for processing.



January 28, 1999

Mr. Harold Gray
President
Gray Industries
236 Washington Boulevard, Suite 480
Grover, CA 90000

Dear Mr. Gray:

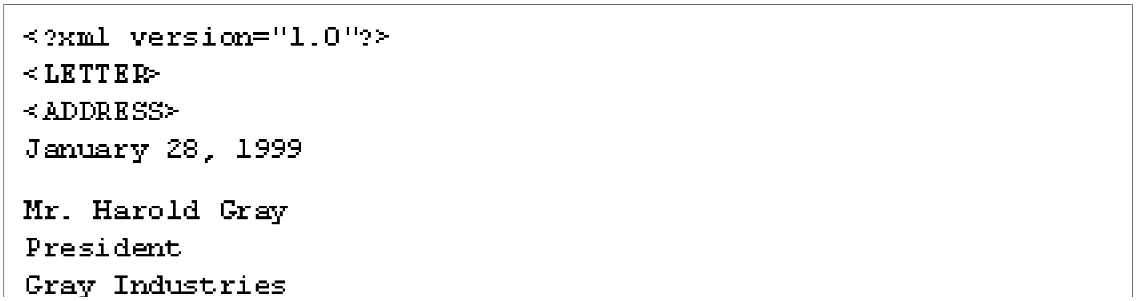
Please contact my assistant George Brown to schedule an appointment at your convenience.

Very truly yours,

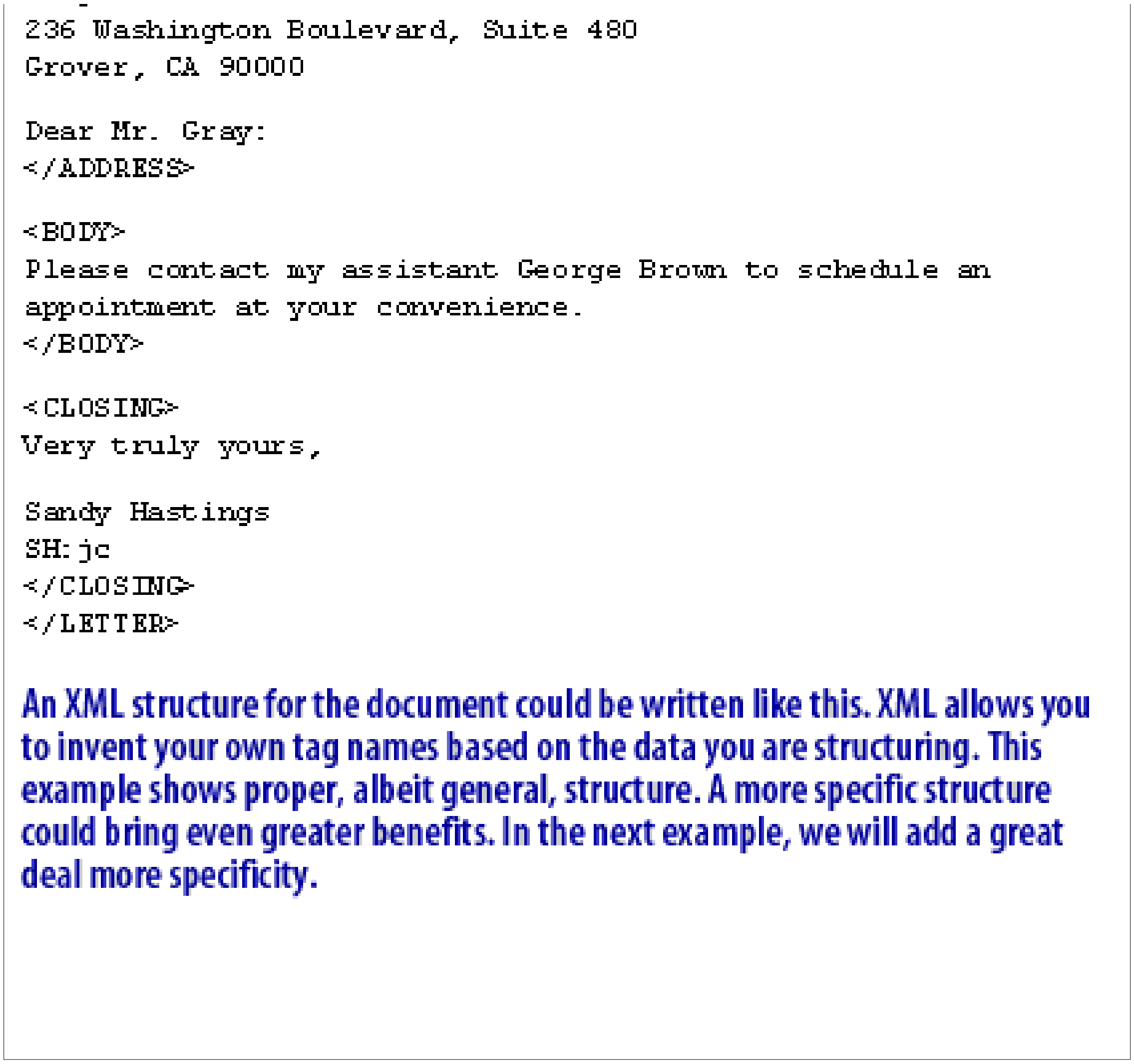
Sandy Hastings

SH:jc

Here is one very basic way of structuring the data in the business letter.

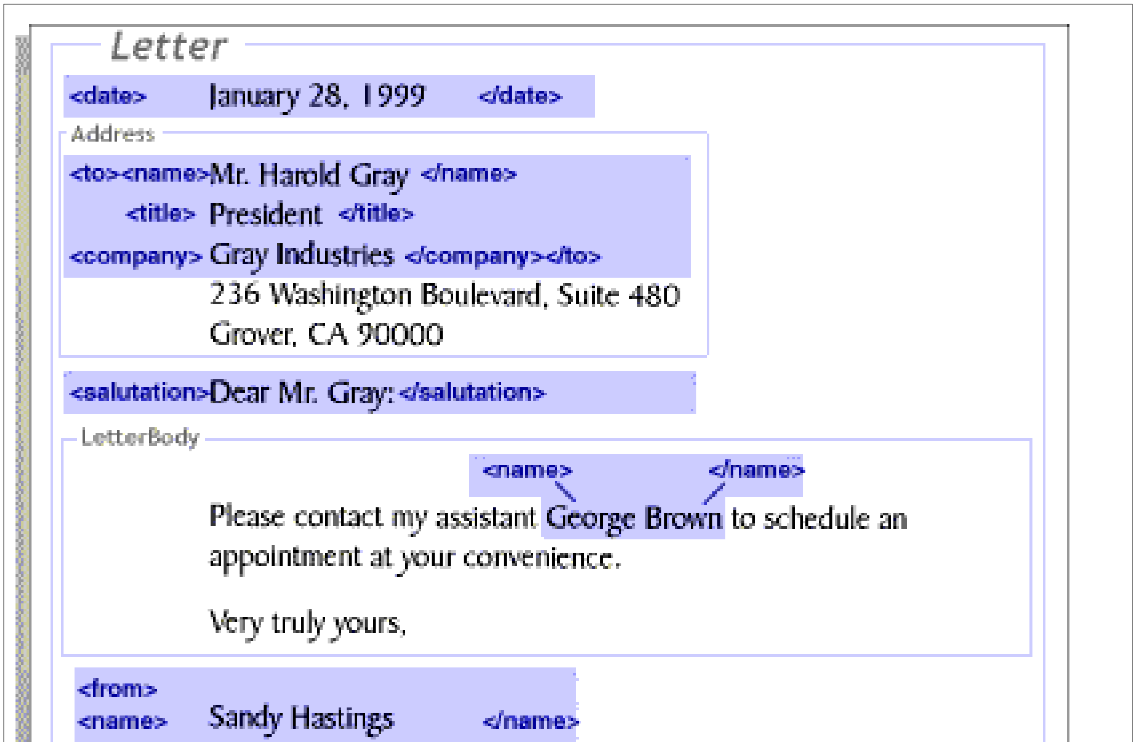


```
<?xml version="1.0"?>
<LETTER>
<ADDRESS>
January 28, 1999

Mr. Harold Gray
President
Gray Industries
```



```
236 Washington Boulevard, Suite 480
Grover, CA 90000

Dear Mr. Gray:
</ADDRESS>

<BODY>
Please contact my assistant George Brown to schedule an
appointment at your convenience.
</BODY>

<CLOSING>
Very truly yours,

Sandy Hastings
SH:jc
</CLOSING>
</LETTER>
```

An XML structure for the document could be written like this. XML allows you to invent your own tag names based on the data you are structuring. This example shows proper, albeit general, structure. A more specific structure could bring even greater benefits. In the next example, we will add a great deal more specificity.



Letter

<date> January 28, 1999 </date>

Address

<to><name>Mr. Harold Gray </name>
<title> President </title>
<company> Gray Industries </company></to>
236 Washington Boulevard, Suite 480
Grover, CA 90000

<salutation>Dear Mr. Gray: </salutation>

LetterBody

<name> </name>
Please contact my assistant George Brown to schedule an appointment at your convenience.

Very truly yours,

<from>
<name> Sandy Hastings </name>

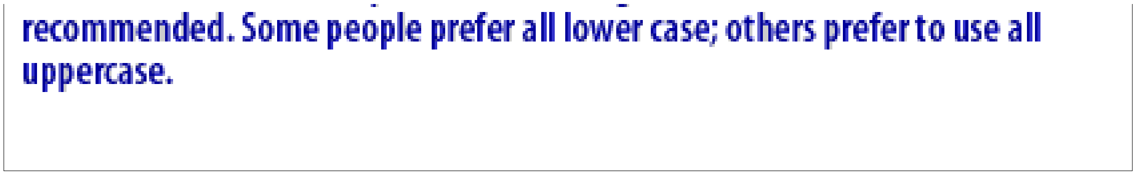
</from>

<typed by> SH:jc </typed by>

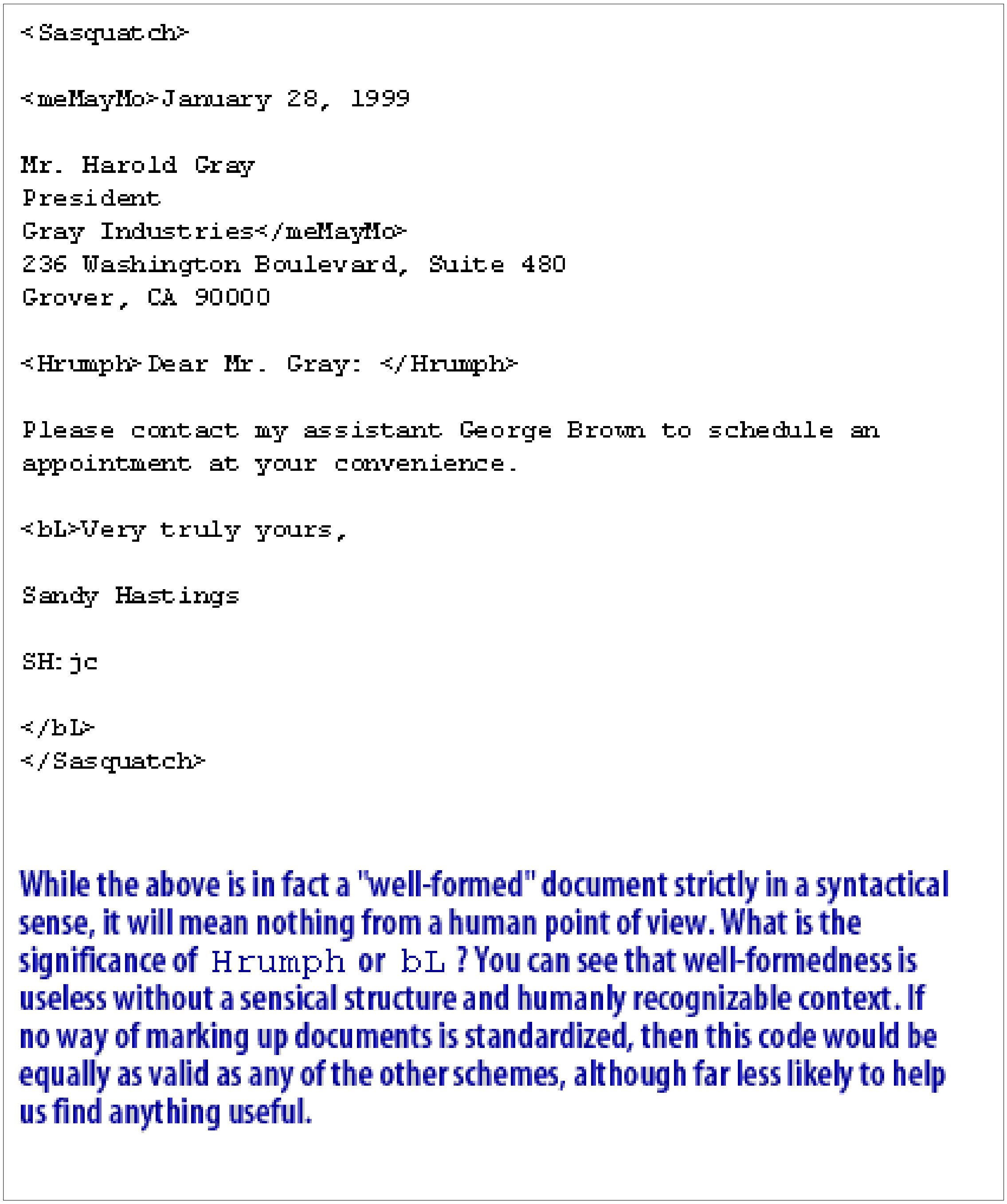
Here is the same letter structured into more specific pieces of information. In this example, a lot more specificity has been added. However, this specificity requires much more work for the user, who will have to encode the letter.

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

<date>January 28, 1999</date>

<to><Address>
<name>Mr. Harold Gray</name>
<title>President</title>
<company>Gray Industries</company>
236 Washington Boulevard, Suite 480
Grover, CA 90000
</Address></to>

<salutation>Dear Mr. Gray:</salutation>

<LetterBody>
Please contact my assistant <name>George Brown</name> to
schedule an appointment at your convenience.

Very truly yours,
</LetterBody>

<from>
<name>Sandy Hastings</name>
</from>

<typed by>SH:jc</typed by>
</Letter>
```

The XML code for the version of the business letter in the previous frame could be written like this. Remember that XML is case-specific. You can choose to use any capitalization scheme you desire, but you must adhere to your choice strictly. This example uses a combination of cases, which makes it difficult to remember whether to capitalize certain tag names. This method is not

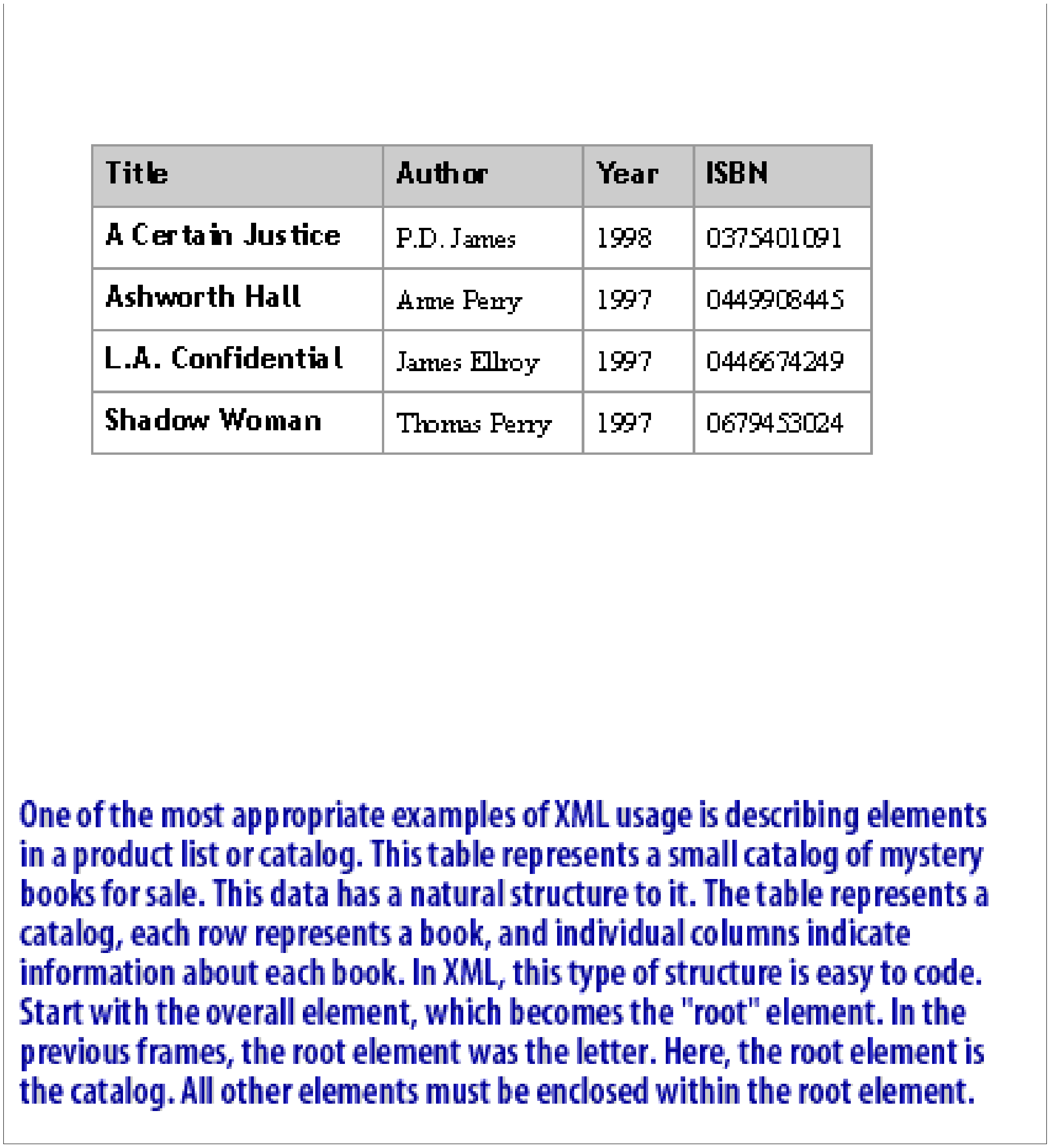


recommended. Some people prefer all lower case; others prefer to use all uppercase.



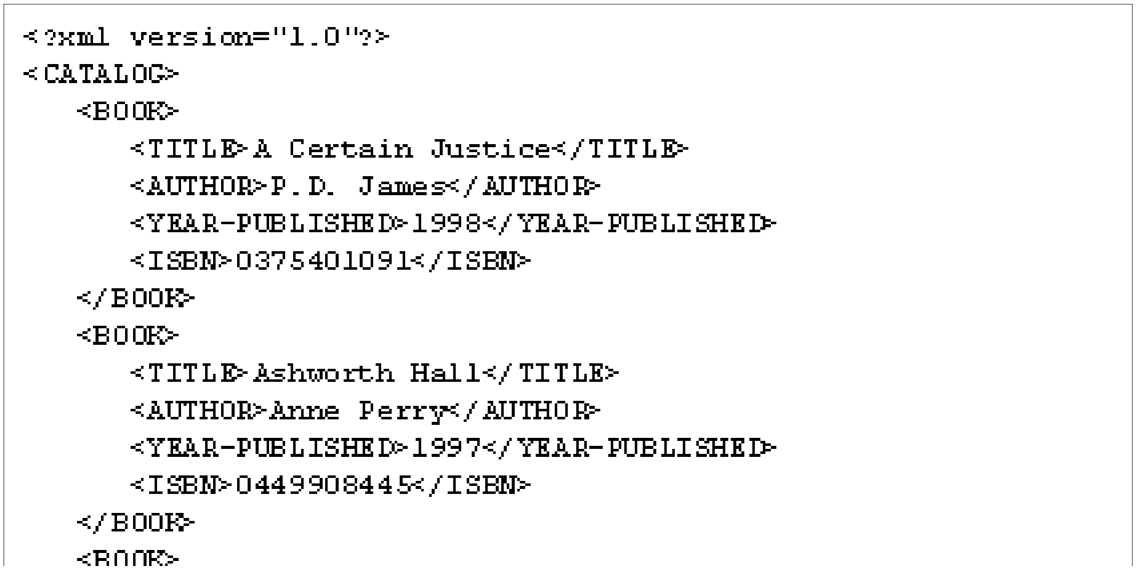
```
<Sasquatch>

<meMayMo>January 28, 1999

Mr. Harold Gray
President
Gray Industries</meMayMo>
236 Washington Boulevard, Suite 480
Grover, CA 90000

<Hrumph>Dear Mr. Gray: </Hrumph>

Please contact my assistant George Brown to schedule an
appointment at your convenience.

<bL>Very truly yours,

Sandy Hastings

SH:jc

</bL>
</Sasquatch>
```

While the above is in fact a "well-formed" document strictly in a syntactical sense, it will mean nothing from a human point of view. What is the significance of `Hrumph` or `bL` ? You can see that well-formedness is useless without a sensical structure and humanly recognizable context. If no way of marking up documents is standardized, then this code would be equally as valid as any of the other schemes, although far less likely to help us find anything useful.

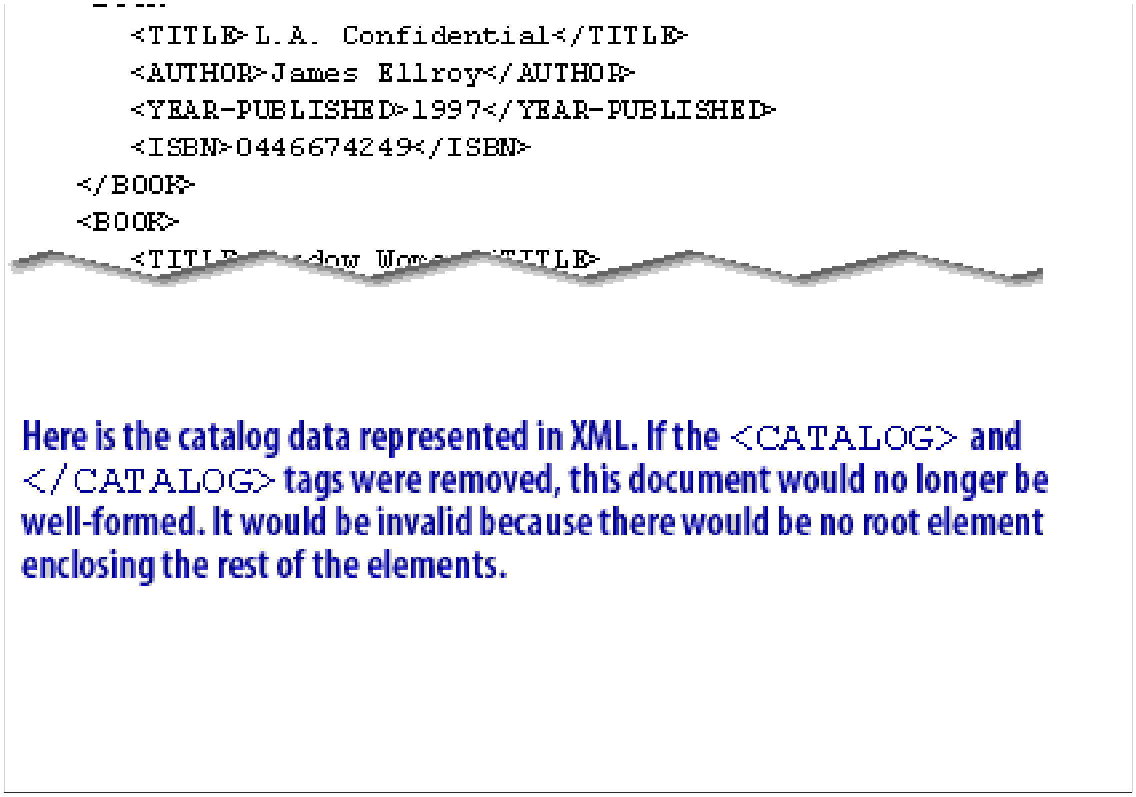


Title	Author	Year	ISBN
A Certain Justice	P.D. James	1998	0375401091
Ashworth Hall	Anne Perry	1997	0449908445
L.A. Confidential	James Ellroy	1997	0446674249
Shadow Woman	Thomas Perry	1997	0679453024

One of the most appropriate examples of XML usage is describing elements in a product list or catalog. This table represents a small catalog of mystery books for sale. This data has a natural structure to it. The table represents a catalog, each row represents a book, and individual columns indicate information about each book. In XML, this type of structure is easy to code. Start with the overall element, which becomes the "root" element. In the previous frames, the root element was the letter. Here, the root element is the catalog. All other elements must be enclosed within the root element.

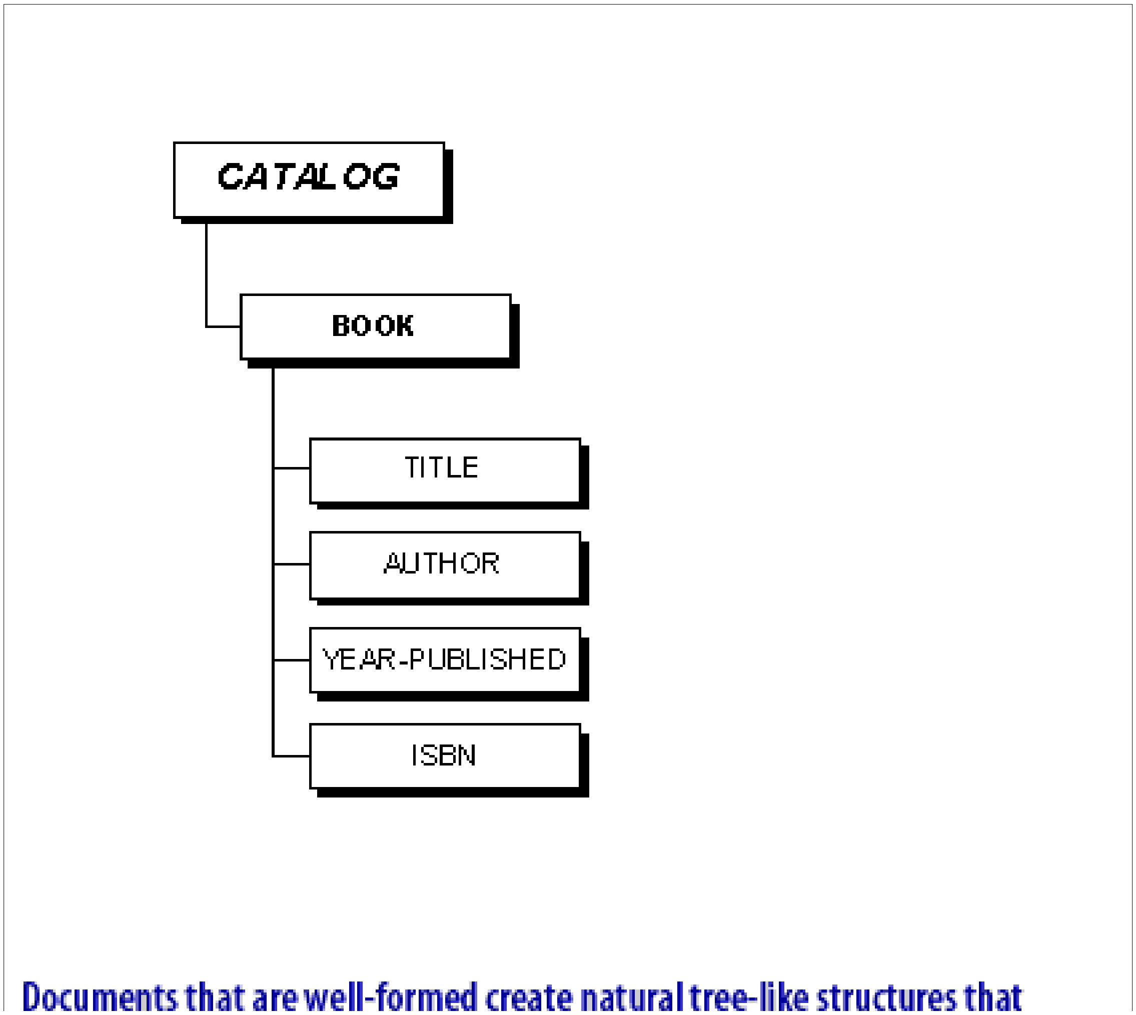


```
<?xml version="1.0"?>
<CATALOG>
   <BOOK>
      <TITLE>A Certain Justice</TITLE>
      <AUTHOR>P.D. James</AUTHOR>
      <YEAR-PUBLISHED>1998</YEAR-PUBLISHED>
      <ISBN>0375401091</ISBN>
   </BOOK>
   <BOOK>
      <TITLE>Ashworth Hall</TITLE>
      <AUTHOR>Anne Perry</AUTHOR>
      <YEAR-PUBLISHED>1997</YEAR-PUBLISHED>
      <ISBN>0449908445</ISBN>
   </BOOK>
   <BOOK>
```

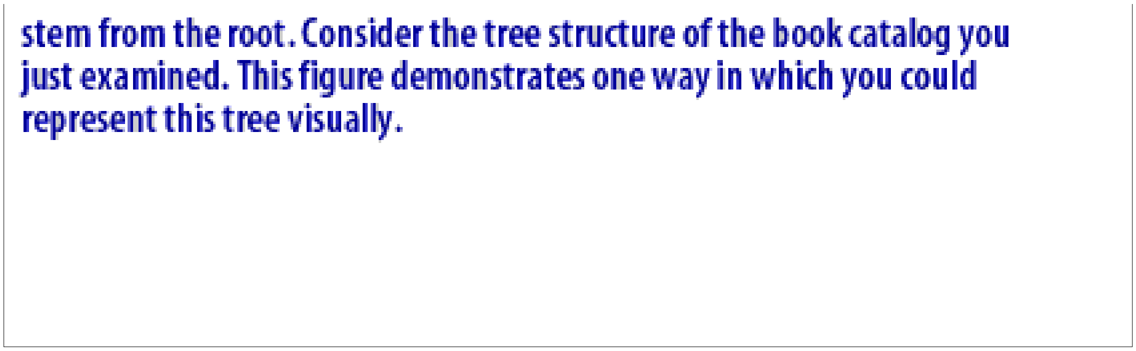


```
        <TITLE>L.A. Confidential</TITLE>
        <AUTHOR>James Ellroy</AUTHOR>
        <YEAR-PUBLISHED>1997</YEAR-PUBLISHED>
        <ISBN>0446674249</ISBN>
    </BOOK>
    <BOOK>
        <TITL    dow Wom    TITLE>
```

Here is the catalog data represented in XML. If the <CATALOG> and </CATALOG> tags were removed, this document would no longer be well-formed. It would be invalid because there would be no root element enclosing the rest of the elements.



Documents that are well-formed create natural tree-like structures that



stem from the root. Consider the tree structure of the book catalog you just examined. This figure demonstrates one way in which you could represent this tree visually.

XML Fundamentals

Module 3

Quiz

Rules for XML documents

Each question is worth one point. Some questions ask you to select **the best answer**, others ask you to select **all the correct answers**. To receive credit for questions asking for **all the correct answers**, you must select all the correct answers and **only** the correct answers.

1. According to the W3C Recommendation for XML 1.0, the minimum requirement for a document to be considered an XML document is that:

   Please select **the best answer**.

   ○ A. It includes the XML version declaration statement <?xml?>.
   ○ B. It is well-formed.
   ○ C. It uses all lowercase HTML tags.
   ○ D. It guarantees forward compatibility.

2. XML is different from HTML in that:

   Please select **the best answer**.

   ○ A. XML will infer closing tags.
   ○ B. Browsers will still render ill-constructed XML documents.
   ○ C. HTML requires attention and discipline.
   ○ D. Any single error will prevent the XML page from rendering in the browser.
   ○ E. HTML documents must be error-free.

3. Which of the following rules apply to well-formed XML documents?

   Please select **all the correct answers**.

   □ A. Empty tags need a / character before the closing bracket.
   □ B. All tags must be uppercase.
   □ C. It is optional to enclose attribute values in quotation marks.
   □ D. Starting and ending tags must be explicit.
   □ E. Tags must nest correctly.

4. What is the first step you should take when transforming an HTML document into an XML document?

   Please select **the best answer**.

   ○ A. Make sure your document will render in a browser.
   ○ B. Close your document with the <?xml?> declaration.
   ○ C. Determine the structure of information in the document.
   ○ D. Eliminate any HTML tags which define formatting.

OK, I'm Done

Module 3

Lesson 6

Adding information to XML documents

Objective

Use comments, CDATA sections, and encoding to add clarity and information to XML documents.

Comments

To help make your documents as understandable to humans as they will be to machines, you should consider adding comment tags within your XML document. Use the same syntax you would for an HTML comment:

```
<!-- comment text here -->
```

CDATA sections

Sooner or later, you will want a simple way to include text that carries the forbidden < character. In both XML and HTML, you could use **<** to indicate the "less-than" sign. What if you wanted to include extensive use of such characters in your file, however?

For this and other reasons, XML provides a way to include any kind of data you want, without much effort on the part of the author. You can use the following to include any combination of characters in a portion of your page:

```
<![CDATA[ data to include ]]>
```

Tip

To embed script (ECMAScript or any script) within an XML document, presently you must enclose the script in a CDATA section as follows:

```
<SCRIPT><![CDATA[Script statements here]]></SCRIPT>
```

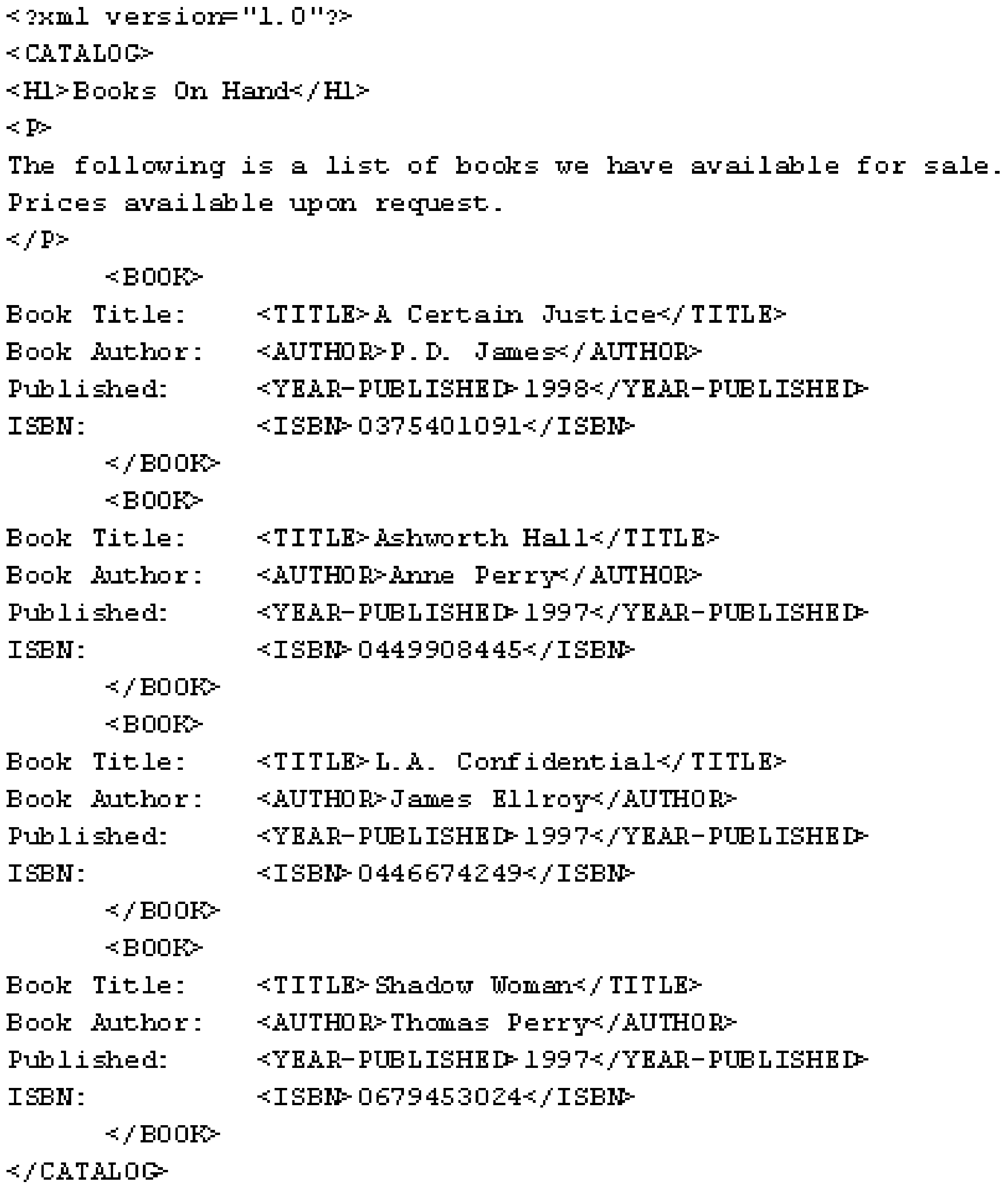
Encoding

Bearing in mind that the Internet is a global technology, it behooves us to include information about what encoding scheme (i.e., standard character set for a language) was used to create an XML file.

The following XML declaration includes encoding information:

MouseOver

Transcript



```
<?xml version="1.0"?>
<CATALOG>
<H1>Books On Hand</H1>
<P>
The following is a list of books we have available for sale.
Prices available upon request.
</P>
      <BOOK>
Book Title:    <TITLE>A Certain Justice</TITLE>
Book Author:   <AUTHOR>P.D. James</AUTHOR>
Published:     <YEAR-PUBLISHED>1998</YEAR-PUBLISHED>
ISBN:          <ISBN>0375401091</ISBN>
      </BOOK>
      <BOOK>
Book Title:    <TITLE>Ashworth Hall</TITLE>
Book Author:   <AUTHOR>Anne Perry</AUTHOR>
Published:     <YEAR-PUBLISHED>1997</YEAR-PUBLISHED>
ISBN:          <ISBN>0449908445</ISBN>
      </BOOK>
      <BOOK>
Book Title:    <TITLE>L.A. Confidential</TITLE>
Book Author:   <AUTHOR>James Ellroy</AUTHOR>
Published:     <YEAR-PUBLISHED>1997</YEAR-PUBLISHED>
ISBN:          <ISBN>0446674249</ISBN>
      </BOOK>
      <BOOK>
Book Title:    <TITLE>Shadow Woman</TITLE>
Book Author:   <AUTHOR>Thomas Perry</AUTHOR>
Published:     <YEAR-PUBLISHED>1997</YEAR-PUBLISHED>
ISBN:          <ISBN>0679453024</ISBN>
      </BOOK>
</CATALOG>
```

Exercise

Working with mixed content

Creating a well-formed document from text

Objective

Create a well-formed XML document from text.

Exercise scoring

Full credit for this exercise is 10 points. You'll submit your source code to the course tutors.

Overview

In this exercise, you will take an existing set of information and create your own well-formed document.

Instructions

Examine the following data. You will use this data to create a well-formed XML document.

```
Academy Awards 1998

Best Picture Nominees

As Good As It Gets

Producer: James L. Brooks, Bridget Johnson, Kristi Zea
Director: James L. Brooks
Screenwriter: Mark Andrus, James L. Brooks
Distributor: Sony
Box Office: $101,614,491

The Full Monty

Producer: Uberto Pasolini
Director: Peter Cattaneo
Screenwriter: Simon Beaufoy
Distributor: Fox Searchlight
Box Office: $39,840,641
```

```
Good Will Hunting

Producer: Lawrence Bender
Director: Gus Van Sant
Screenwriter: Ben Affleck, Matt Damon
Distributor: Miramax
Box Office: $79,671,173

L.A. Confidential

Producer: Curtis Hanson, Arnon Milchan, Michael Nathanson
Director: Curtis Hanson
Screenwriter: Curtis Hanson, Brian Helgeland
Distributor: Warner Bros.
Box Office: $45,477,469

Titanic

Producer: James Cameron, Jon Landau
Director: James Cameron
Screenwriter: James Cameron
Distributor: Paramount
Box Office: $376,270,721
```

Create your own XML tags and use them to mark up this file.

- Do not delete or add any text. Add only markup tags.
- You can create any tags you think will help you accomplish your task (such as), and place them wherever makes sense to you.
- Be sure to follow the rules for creating a well-formed XML document.
- Remember that you are tagging data, not formatting instructions using HTML tags.
- Remember also that XML allows you to invent your own tag names based on the data you are structuring.
- Name tags in such a way that the tags reflect actual contents.

Starter files

We've supplied the text shown above in a file called `bestpic.txt`. You'll find this file in the 03-05 folder of the course download available from the Resources page.

What to submit to the course tutors

In the text box below, cut and paste the contents of your XML file. Click OK, I'm Done to submit your code to the course tutors.

OK, I'm Done

Module 3

Lesson 7

Creating XML documents

Module wrap-up

In this module, you learned how to recognize structure, and how to write custom tags to denote this structure in an XML document. You learned the five basic rules for well-formedness, and you know that well-formedness is only the minimum requirement for a document to be considered an XML document.

You have seen that tag names can help or hinder the human readability factor in XML documents. Finally, you learned that we may well miss the potential for interoperability unless individuals adhere to common vocabularies when creating XML documents.

Discussion

Do you have any questions or comments about the material in this module? If so, look in the folder Creating XML documents in the Discussion section, which you can reach by clicking the Discuss button in the toolbar.

Quiz

Quiz

Click the Quiz button to test what you've learned in this module.

Module 3

Adding information to XML documents

CDATA section example

Here's an example of using a CDATA section. If you wanted the user to see the text sequence of `<HATSTYLE>Derby</HATSTYLE>`, you could use the following to include that literal expression in your document:

```
<![CDATA[
   <HATSTYLE>Derby</HATSTYLE>
]]>
```

XML Fundamentals

Module 3

Adding information to XML documents

Balkanization of the Web?

In a now-famous article for Feed Magazine, Mark Pesce sounded a warning bell about XML. As you have seen, there are many ways to decide how to tag content. If every user does her own thing, we will lose the potential for interoperability between documents.

Pesce referred to this situation as the "balkanization" of the Web, and said that we would be in a sort of tag gumbo, with no way to discern valuable content. Others have argued that we are already in tag gumbo, and that without a new standard for defining content, the Web will continue to present billions of words but little intelligent data. The freedom to create tags is also the freedom to create chaos.

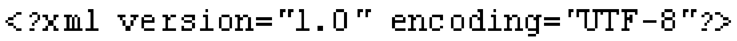
Fortunately, standardized industry vocabularies are being created, which will define the sets of tags we may ultimately use. Rather than continuing to reinvent the wheel, you are encouraged to seek out and pursue emerging industry languages created from XML.

XML Fundamentals

Module 3

Adding information to XML documents

Encoding information (MouseOver transcript)



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

encoding

You can include encoding information within the XML declaration by adding the ENCODING attribute along with the appropriate value.

UTF-8

When creating an XML document, it is useful for people in other countries or using other encoding schemes to know that our data is in the standard common English character set, which is a subset of UTF-8. UTF stands for UCS (Universal Character Set) Transformation Format. UTF-8 represents an 7-bit character set or the characters 0 through 127.

Complete & Unabridged: The FEED Books Issue

INTRO | 7.14.99

FEED Books Issue

Alex Abramovich introduces this special issue.

DIALOG | 7.14.99

Sudden Diction

A FEED Dialog on the state of the American sentence with Jenny Offill, Ben Marcus, Ethan Nosowsky, Samantha Gillison, and Gilbert Sorrentino.

- Question One: What makes an American sentence?
- Question Two: How does it work?
- Question Three: What's in store for it?

ESSAY | 7.14.99

Coming of Age

Why are today's young writers such old fogeys? An essay by Alissa Quart.

ESSAY | 7.14.99

Conclusive Evidence

What can the effluvia of Nabokov's life tell us about his art? An essay by Gloria Fisk

BOTTOMFEEDER | 7.14.99

Heavy Vetting

What goes on behind the closed doors of a prurient publishing house? By William O'Shea.

RE | 7.14.99

RE: Jean Stein

Adam Lehner talks to the Editor of *Grand Street*

This Special Issue is brought to you by
Quality Paperback Book Club and The New School.

FEED DAILY

JOSH GLENN 07.20.99

HEARST MAGAZINES has announced that next year they'll begin publishing an uplifting women's monthly, one devoted not just to fashion and beauty but also to spirituality, community, and books. Its premiere issue will have a tremendous print run of 850,000, thanks to the star power of Oprah Winfrey, who has agreed to allow Hearst to "translate Winfrey's message of encouragement and inspiration to the magazine page." Uplift is in, it seems, since Hearst's *Talk* magazine -- co-published with Miramax's Talk Media -- has, in these past several months leading up to August's launch, transformed itself from the provocative, upscale gossip rag that it was initially touted to be into an enlightening, democratic "dialogue of the American culture." Ugh. Do people really want *more* middlebrow? Why can't Tina and Oprah stick to the lowbrow *divertissement* they do best?

continue

Click here to share your thoughts in the Daily Loop.

For discussion of other recent FEED Dailies, check out last month's Daily Loop.

PREVIOUS FEED DAILIES

Matthew DeBord on John F.Kennedy, Jr.
Alex Ross on *Eyes Wide Shut*
Clay Shirky on the UN's internet ignorance
Nicholas Joseph on *Shadow* and Watergate's non-lesson lesson

Illustrations by Marcellus Hall.

MATERIALIST | 07.13.99

Life in the 'Burbs

Stefanie Syman looks at snapshots of suburbia.

ESSAY | 07.09.99

TV Nationalism

Newly discovered footage reveals why the Nazi propaganda machine just didn't get television. David Hudson reports.

BOTTOMFEEDER | 07.08.99

The Manuscript Madame

Susan Katz Keating talks to Tanya Field about being behind the scenes at a literary escort service.

RE | 07.02.99

RE: Ira Glass

David Brown talks to Ira Glass about intimacy, edited stories, and being the hand of God.

ESSAY | 07.01.99

The Humor Quotient

Can collaborative filtering software help us understand what makes something funny? Rachel Chalmers investigates.

PREVIOUS SPECIAL ISSUES:

The New Brain

(June 1999)

- The Elaborate Machine: FEED editors introduce FEED's special issue on the science of the mind
- My Favorite Lobe: FEED invites eight of the world's leading experts to describe the brain's most fascinating regions.
- Plus: Peter Kramer reconsiders Prozac; David Shenk on memory drugs; Erik Davis on Zen and the brain.

Brave New Worlds: The Video Game Issue

(May 1999)

- Trigger Finger: Game designer Theresa Duncan plays the world's most violent games

Sasha Smith on *The Dinner Game* and remaking French films

Matthew DeBord on *Surfer* magazine's 40th anniversary

Gavin McNett on Sonic Youth's stolen gear

Ana Marie Cox on Spike Lee's *Summer of Sam*

Clay Shirky on how 19th-century politics determined the 21st-century

Matthew DeBord on the un-private architecture at the MOMA

Josh Glenn on Christoph Schlingensief's money-throwing performance

Steven Johnson on *Home Page* and the web's loss of innocence

OTHER MEDIA&CULTURE

ESSAY | 05.25.99

Loony Boon

Time-Warner is pimping its cultural legacy... and ours. Josh Ozersky explains.

ESSAY | 05.21.99

Death and the Maiden

Gloria Fisk on what happens when guns become a girl's best friend.

RE | 05.14.99

RE: The Residents

Tom D'Antoni talks to The World's Most Famous Unknown Band about anonymity, the Bible, and corporate patrons.

ESSAY | 05.18.99

Rock Slide

Gavin McNett makes sense of the latest rock-crit crisis.

MATERIALIST | 05.17.99

Abject of My Affections

Stefanie Syman talks about the art of Tracey Emin and Patty Chang

- Sim Theory: Steven Johnson plays SimCity 3000 with three urban scholars.
- Plus: Game creators debate gaming's next generation; games go Hollywood; FEED interviews Mario's creator.

**Liberation Technology:
The Open Source Software Issue**
(March 1999)

- What Is To Be Done?: Open source visionaries Eric Raymond, Eric Allman, and Richard Stallman debate the future of the movement
- Plus: The politics Of Open Source; Perl creator Larry Wall; profiles of OS programmers.

RE | 06.15.99

Howard Zinn

Amanda Griscom talks to the legendary historian about authority, emotionalism, and Matt Damon.

ESSAY | 06.11.99

The Organization Plan

Jared Diamond, author of *Guns, Germs, and Steel*, looks at the natural history of getting rich.

ESSAY | 06.09.99

Victorian Secret

Will traditional gender differences serve us well in the 21st century? Emily Jenkins profiles anthropologist Helen Fisher, champion of the "natural woman."

BOTTOMFEEDER | 06.08.99

The Music Man

What it takes to be a TV jingle writer.

ESSAY | 06.07.99

Battlespaces

The Kosovo conflict signals a new kind of war -- and a new kind of risk. Matthew DeBord reports.

INTERFACE | 06.04.99

The Third Wave

Steven Johnson on the latest revolution in web community software.

RE | 06.03.99

Tom Perrotta

Alissa Quart talks to Tom Perrotta about his *Election* screenplay, protest politics, and good

OTHER TECHNOLOGY

ESSAY | 05.19.99

Up From Celluloid

Saul Anton explains how digital technology will liberate indie film.

ESSAY | 04.20.99

The Hit Parade

The World Wide Web is now as big as TV. Will it ever be the same? Clay Shirky reports.

INTERFACE: | 03.18.99

Maps and Legends

What can the new online geographers teach us about the state of the web? Steven Johnson reports.

FEATURE: | 02.28.99

Special Issue on Open Source

A collection of articles on the Open Source/ Free Software movement.

OUR PARTNERS

THE LOOP

Books -- where are they going, where have they been?
Share your thoughts in the Loop.

Can photography reveal the poignancy of a cliched landscape?
Talk about the art of suburbia.

sex.

DEEPREAD | 06.01.99

Beyond the Pleasure Principle

Do advertisers really pull our purse strings? Ana Marie Cox defends our inalienable right to shop.

RE | 05.28.99

Yaphet Kotto

Sarah Miller talks to "Homicide" hero Yaphet Kotto about lying down and getting down.

ESSAY | 05.27.99

Marine Biology

Today's soldier fears the anthrax vaccine more than the disease itself. Susan Katz Keating explains.

FEED Newsflash:

PLEASE HELP US TO CONTINUE TO BRING YOU THE BEST FEED POSSIBLE

The information you provided us in last November's Reader Survey was enormously helpful to us as we worked our hardest to bring you the best possible FEED at no cost to you. In order to continue to bring you the best product possible, we need your help once again. Could you please take a moment to fill out our 1999 Reader Survey? The information you provide is anonymous and will only be used to help us to sustain and expand the FEED community. (In order to make sure that you feel perfectly comfortable providing us with this information, please feel free to read FEED's Privacy Policy Statement.) Thank you so much!

Steven Johnson,
Editor-in-Chief

Click here to sign up for our free weekly e-mail newsletter.

Did Nazi television capture the banality of evil?

Discuss the history of television as a tool for propaganda.

What will the future of neuroscience be?

Join the discussion on the New Brain.

Should there be more impressionistic vignettes in the media?

Discuss the storytelling technique of Ira Glass.

Can technology help us understand what makes things funny?

Share your thoughts on computing humor.

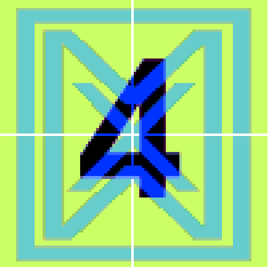
And drop by the Daily Loop where topics of conversation range from French films to surf writing.

About FEED | Masthead | Contributors

Module 4

Lesson 1

DTD basics



Module introduction

The best way to understand the role of a Document Type Definition (DTD) is to create your own. In this module, you will create DTDs for existing XML documents. You will also use a DTD as a guideline for creating new XML documents that conform to the rules laid out in the DTD.

Module learning objectives

After completing this module, you will have the skills and knowledge necessary to:

- Understand the value of using a DTD with XML documents
- Consider the concept of a valid XML document
- Examine a simple DTD
- Declare basic elements in a DTD
- Write element declarations for mixed content and declare empty elements
- Reference DTD declarations in XML
- Create a DTD file which works with a separate XML file
- Create a DTD from an existing set of tags
- Specify the considerations involved in creating a DTD
- Create a simple DTD for a simple XML file
- Create a DTD describing elements containing mixed content

Module 3

Quiz

XML: Power and problems

Each question is worth one point. Some questions ask you to select **the best answer**, others ask you to select **all the correct answers**. To receive credit for questions asking for **all the correct answers**, you must select all the correct answers and **only** the correct answers.

1.When creating your own well-formed XML document from existing text, you must:

Please select **the best answer**.

○ A. Use the recognized set of XML tags
○ B. Use a set number of tags specified by the W3C
○ C. Use the tags specified by a particular industry's scheme
○ D. Use HTML tags if you are not sure which XML tags apply to your content
○ E. Use any set of tags you think will best accomplish your purpose

2.You can add information to XML documents by:

Please select **all the correct answers**.

□ A. Using comment tags to embed script
□ B. Using CDATA sections to include forbidden characters
□ C. Including the encoding information used to create the XML file
□ D. Using comment tags to provide information to human readers
□ E. Using CDATA sections to specify the standard character set used for a language

3.XML's flexibility in defining content also introduces some potential problems, such as:

Please select **all the correct answers**.

□ A. A lack of interoperability between documents
□ B. The "balkanization" of the Web
□ C. A halt to emerging industry languages.
□ D. Difficulty discerning valuable content

OK, I'm Done

XML Fundamentals

Module 4

Lesson 2
Objective

Beyond well-formedness

Consider the concept of a valid XML document.

A document is only an XML document if it is well-formed. But as Veo Systems director Robert Glushko has stated, "Well-formed XML is a black hole." Anyone can create structure. The challenge is to create a meaningful structure and apply it consistently.

To share information across documents, you should find or create a *Document Type Definition* or *DTD* to define which elements will be used and how.

Transcript

When creating XML documents, you should strive to create documents that are both well-formed and valid. These characteristics will give your documents the greatest possible range of options for interoperability with other applications.

Module 4

Lesson 3

Objective

DTD defined

Examine a simple DTD.

A DTD defines the syntax, structure and vocabulary that can be used within a corresponding XML file. XML files that conform to the specified DTD are "valid." The basic unit in a DTD is an element. In *SGML*, HTML and XML, each tag represents an element in the DTD.

A DTD is a simple text file that contains the instructions for the elements contained within the corresponding XML document. Instructions include which attributes can be used with those elements, and how those elements relate within the document's tree structure.

The root element

XML documents are required to have a *root element*. Every other element must appear between the beginning and ending tags for the root element.

Transcript

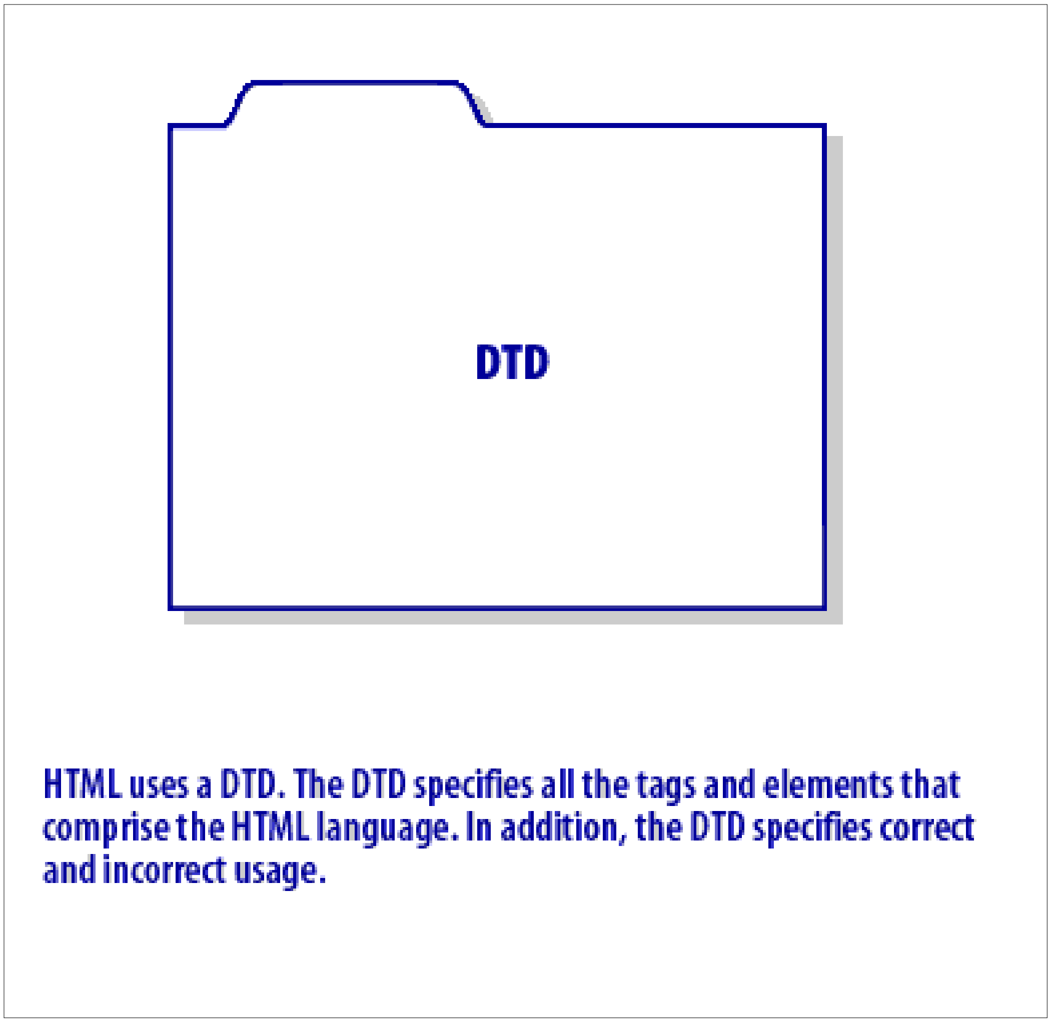
XML Fundamentals

Module 4

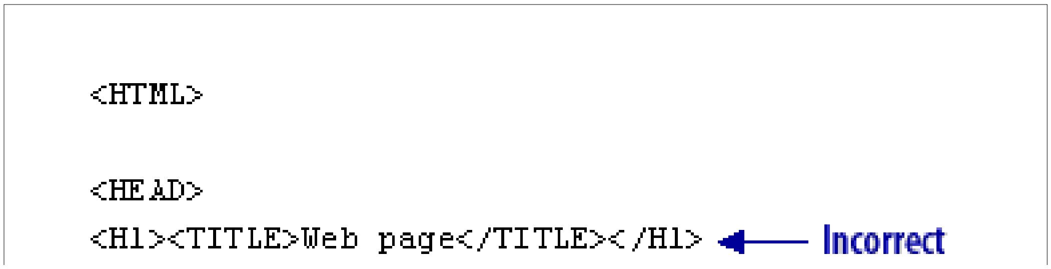
Beyond well-formedness

The role of the DTD (FlipBook transcript)

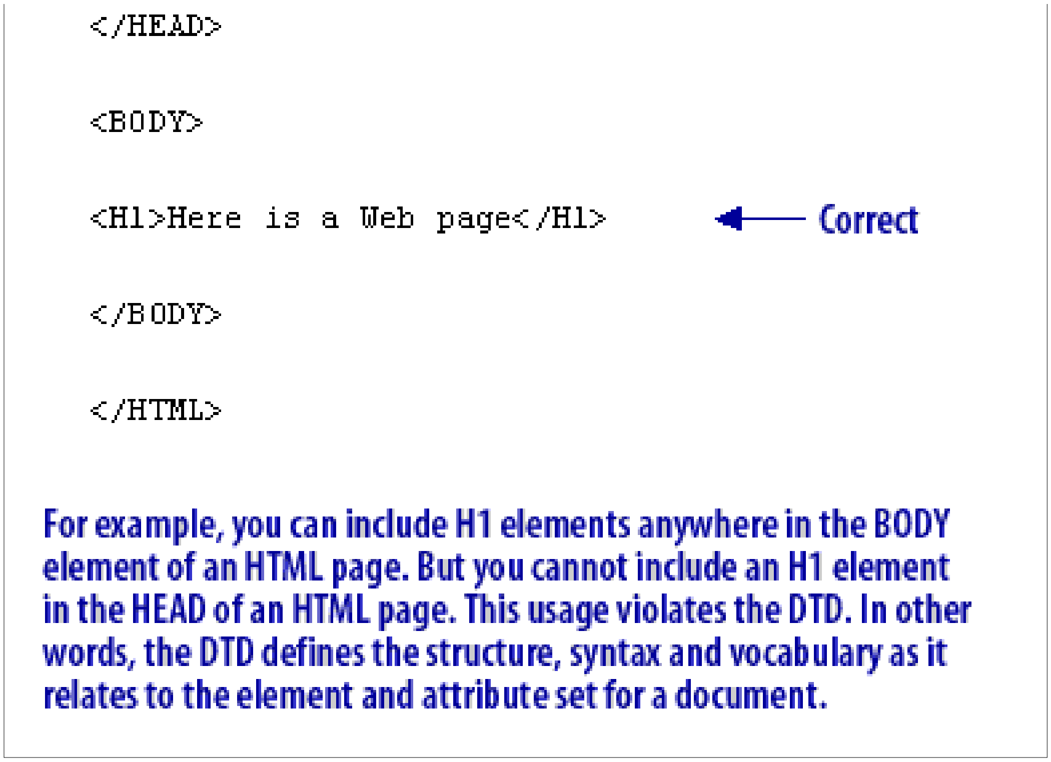
The DTD will define which elements will be used and how:



HTML uses a DTD. The DTD specifies all the tags and elements that comprise the HTML language. In addition, the DTD specifies correct and incorrect usage.



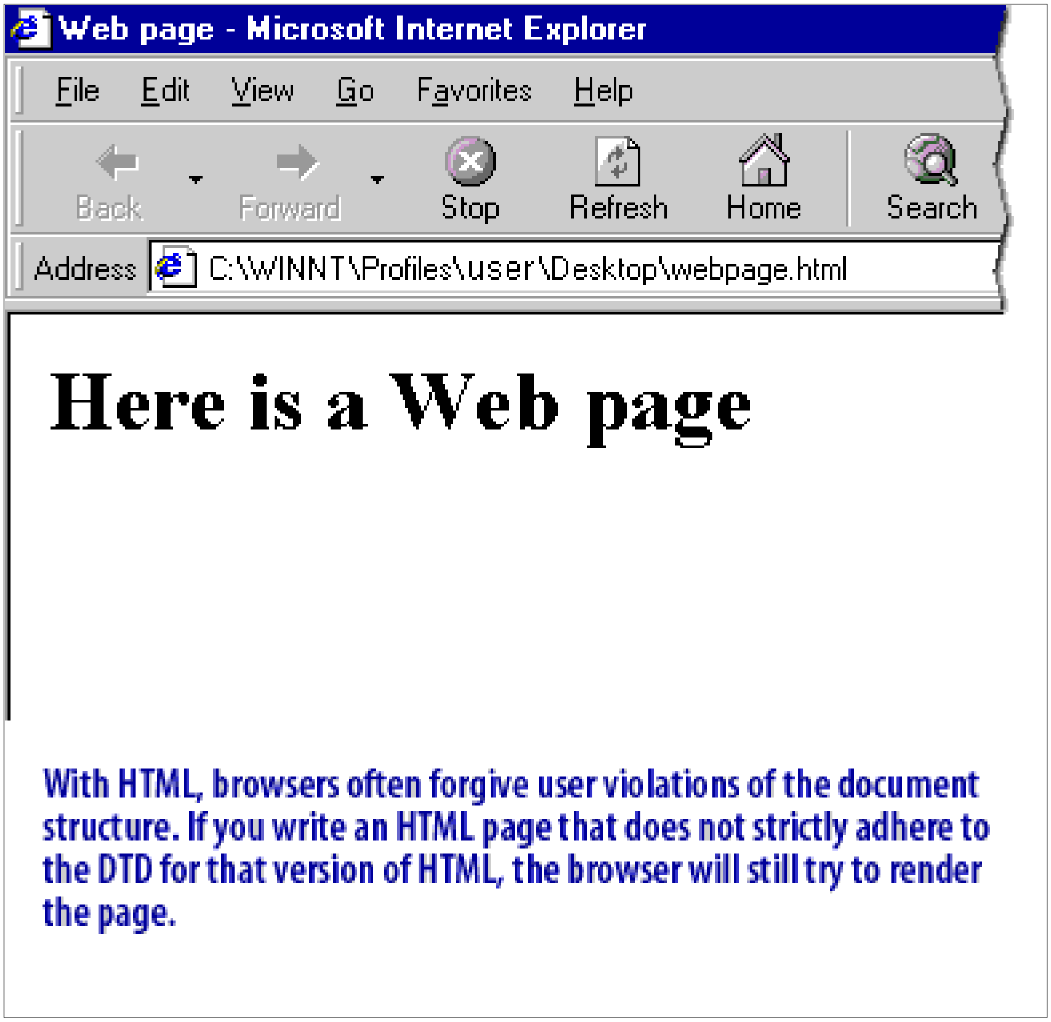
```
<HTML>

<HEAD>
<H1><TITLE>Web page</TITLE></H1>
```



```
</HEAD>

<BODY>

<H1>Here is a Web page</H1>

</BODY>

</HTML>
```

Correct

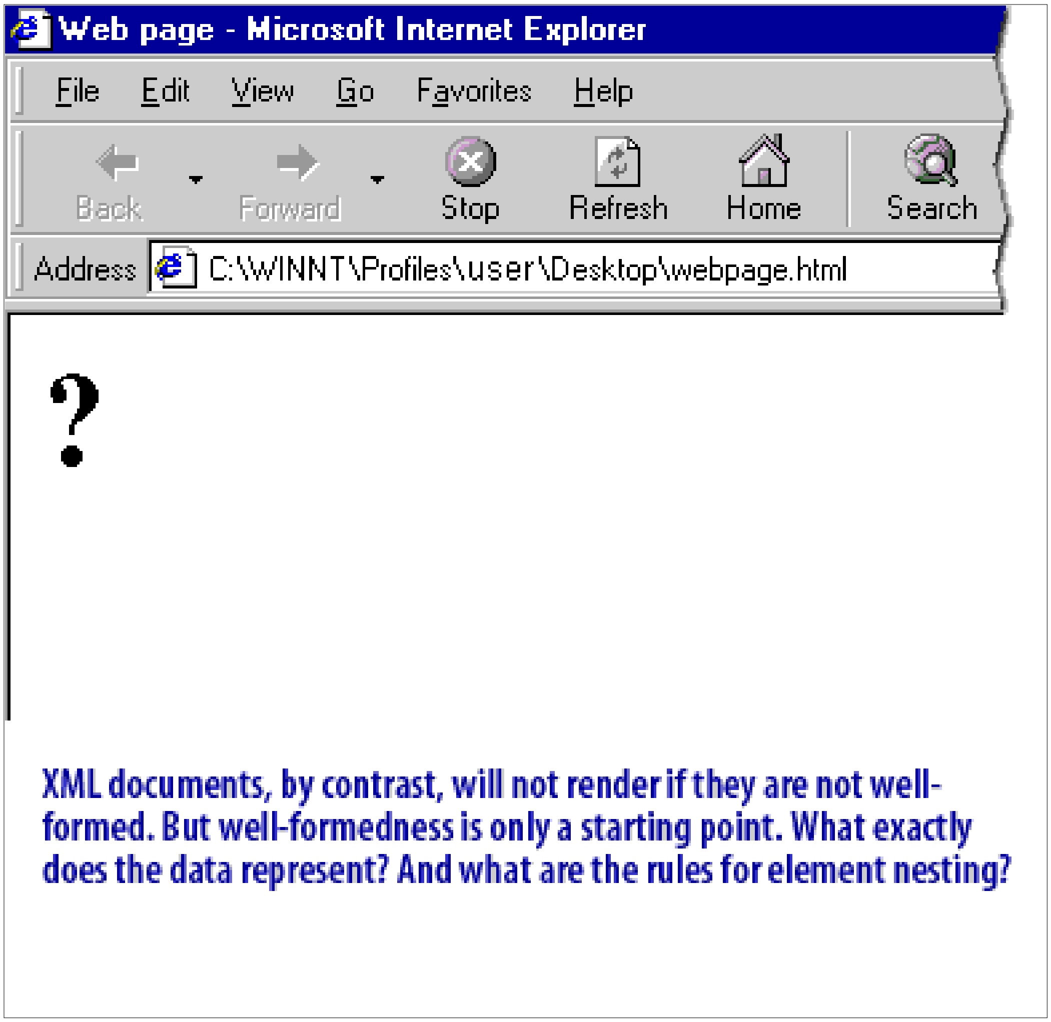
For example, you can include H1 elements anywhere in the BODY element of an HTML page. But you cannot include an H1 element in the HEAD of an HTML page. This usage violates the DTD. In other words, the DTD defines the structure, syntax and vocabulary as it relates to the element and attribute set for a document.



C:\WINNT\Profiles\user\Desktop\webpage.html

Here is a Web page

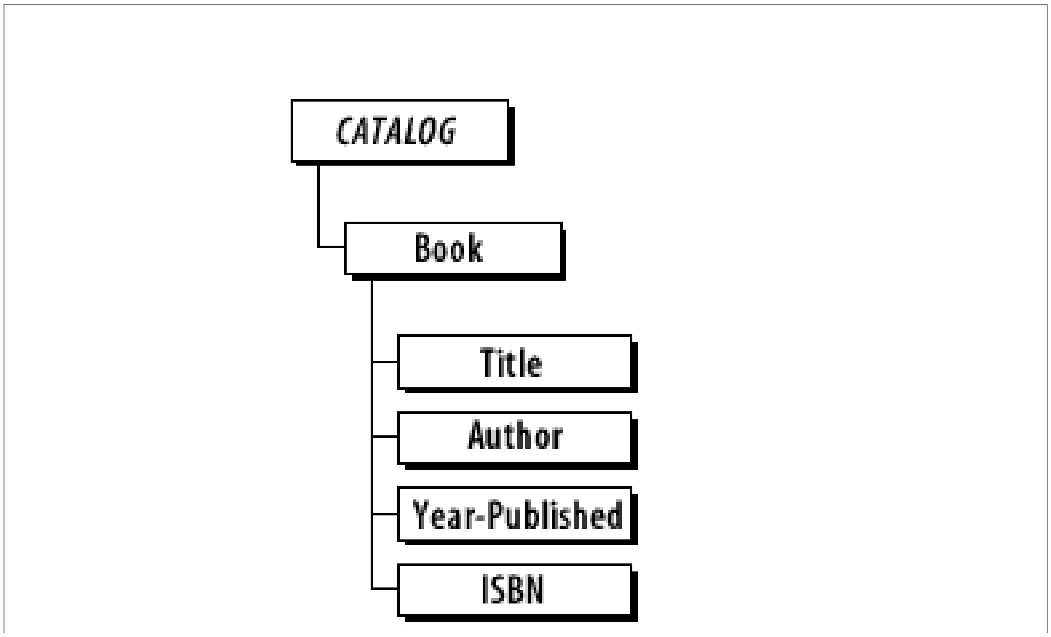
With HTML, browsers often forgive user violations of the document structure. If you write an HTML page that does not strictly adhere to the DTD for that version of HTML, the browser will still try to render the page.



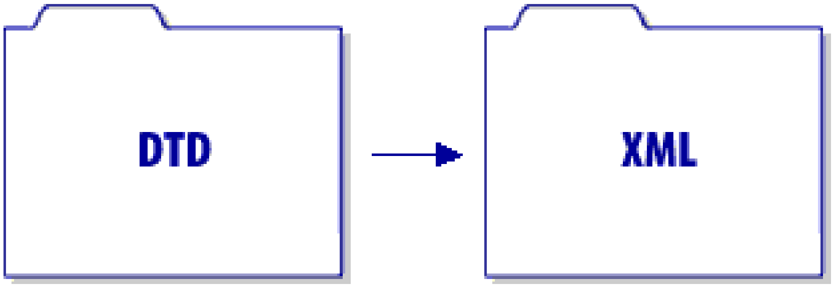
Address C:\WINNT\Profiles\user\Desktop\webpage.html

?

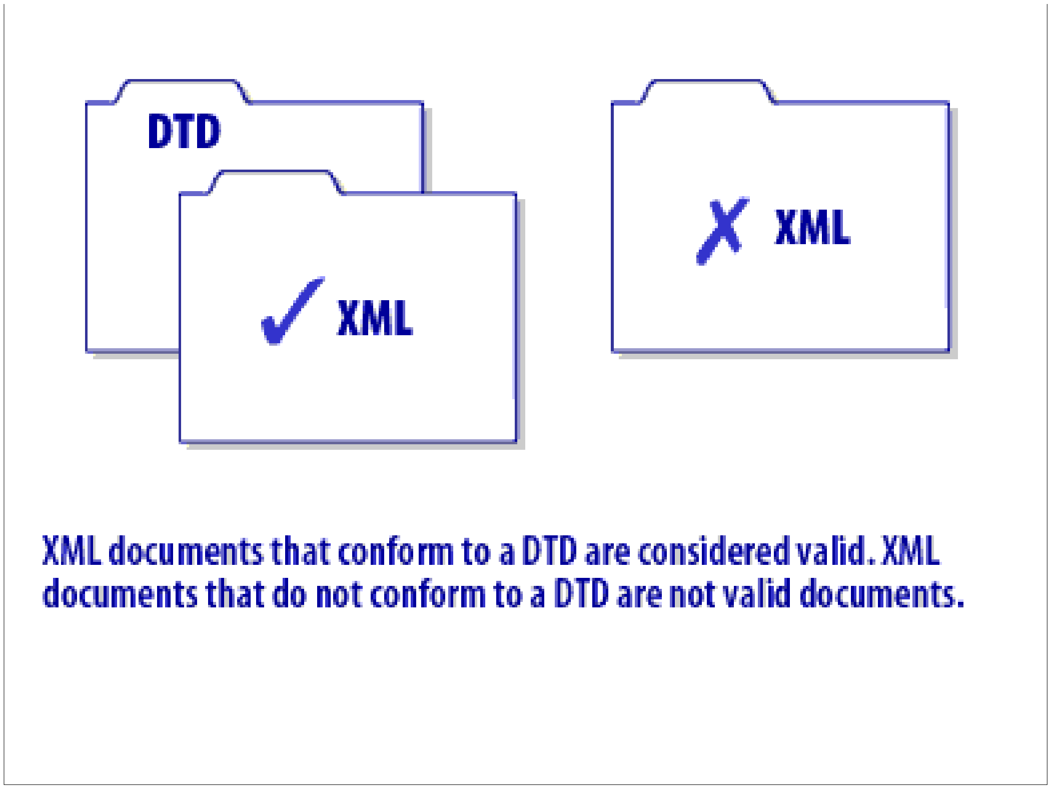
XML documents, by contrast, will not render if they are not well-formed. But well-formedness is only a starting point. What exactly does the data represent? And what are the rules for element nesting?



Is it appropriate for the BOOK element to contain the AUTHOR element, or should the AUTHOR element contain the BOOK element? Or should they be separate, unnested elements?



Some of these decisions will be yours, especially if you are creating standards within an organization or industry. But increasingly, these decisions will be made ahead of time, and the user need only be sure that the XML document he or she creates conforms to an existing DTD.



XML documents that conform to a DTD are considered valid. XML documents that do not conform to a DTD are not valid documents.

XML Fundamentals

Module 4

Lesson 4 Objective

Creating a DTD

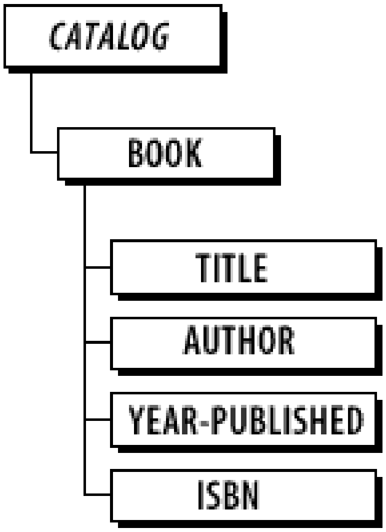
Specify the considerations involved in creating a DTD.

This lesson will walk through the process of creating a DTD for the following XML file:

```
<?xml version="1.0"?>
<CATALOG>
   <BOOK>
      <TITLE>A Certain Justice</TITLE>
      <AUTHOR>P.D. James</AUTHOR>
      <YEAR-PUBLISHED>1998</YEAR-PUBLISHED>
      <ISBN>0375401091</ISBN>
   </BOOK>
   <BOOK>
      <TITLE>Ashworth Hall</TITLE>
      <AUTHOR>Anne Perry</AUTHOR>
      <YEAR-PUBLISHED>1997</YEAR-PUBLISHED>
      <ISBN>0449908445</ISBN>
    </BOOK>
    <BOOK>
      <TITLE>L.A. Confidential</TITLE>
      <AUTHOR>James Ellroy</AUTHOR>
      <YEAR-PUBLISHED>1997</YEAR-PUBLISHED>
      <ISBN>0446674249</ISBN>
   </BOOK>
   <BOOK>
      <TITLE>Shadow Woman</TITLE>
      <AUTHOR>Thomas Perry</AUTHOR>
      <YEAR-PUBLISHED>1997</YEAR-PUBLISHED>
      <ISBN>0679453024</ISBN>
   </BOOK>
</CATALOG>
```

First, you'll want to determine the root element of the document. The root element in this example is `CATALOG`. Next, you'll list all the other elements used in the document. The other elements used in the catalog are: `BOOK, TITLE, AUTHOR, YEAR-PUBLISHED` and `ISBN`.

After listing all the elements, create a document tree structure for them. This figure represents the document tree for the elements in the book catalog:



Note whether elements need to appear in a specific order. The elements `TITLE, AUTHOR, YEAR-PUBLISHED` and `ISBN` must appear in that order.

Also note whether certain elements have content that must be present, can be present, or may be present multiple times. In this example, it is clear that with the exception `ISBN` (which may not be present for books published more than a few years ago), you need only and exactly one `TITLE` and `YEAR-PUBLISHED` element per `BOOK` element. For now, you will use only one `AUTHOR` element as well, although that could change. Any number of `BOOK` elements can exist, though there should be at least one book present to consider this a catalog.

Finally, you'll note which elements will contain text, and which will only contain other elements. The elements that will only contain other elements in this example are the root element `CATALOG` and the `BOOK` element. All other elements will contain character data.

From the information gathered, you can now create a comprehensive DTD that can be used to validate this catalog and any others expected to have the same format.

```
<!DOCTYPE CATALOG [
   <!ELEMENT CATALOG (BOOK)+>
      <!ELEMENT BOOK (TITLE,AUTHOR,YEAR-PUBLISHED,ISBN?)>
      <!ELEMENT TITLE (#PCDATA)>
      <!ELEMENT AUTHOR (#PCDATA)>
      <!ELEMENT YEAR-PUBLISHED (#PCDATA)>
      <!ELEMENT ISBN (#PCDATA)>
]>
```

This DTD indicates the following:

1. The root element is `CATALOG`.
2. The `CATALOG` element is followed by one or more `BOOK` elements.
3. The next elements must be `TITLE, AUTHOR, YEAR-PUBLISHED,` and `ISBN`, in this order. Each of these can contain only character data.
4. `ISBN` need not be present, but if it is present, no more than one `ISBN` element can be used per `BOOK` element, and it will be the last element in the `BOOK` element set. `ISBN` can only contain character data.

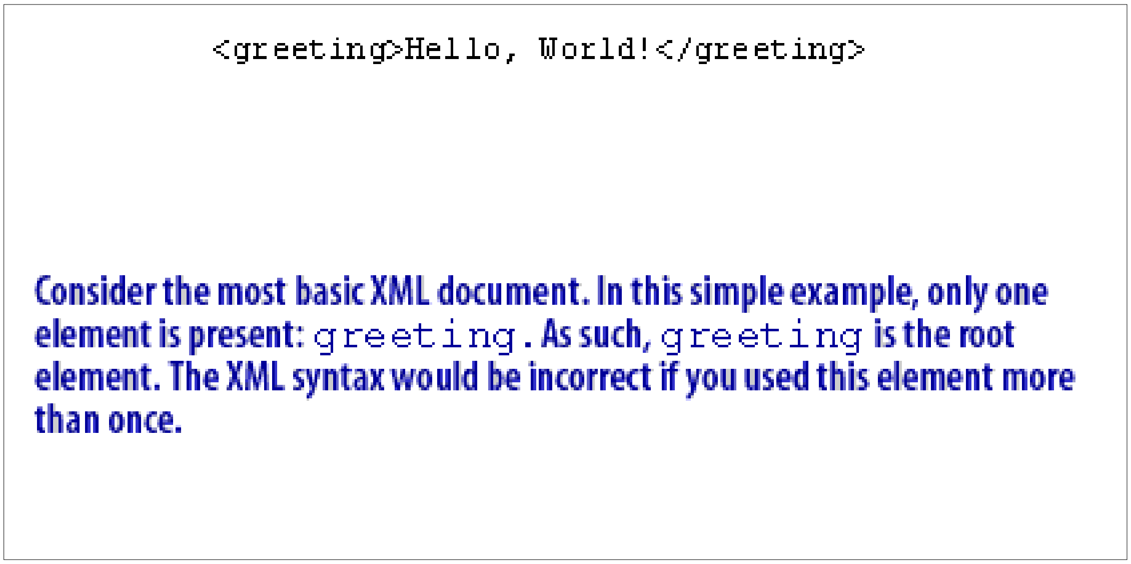
You'll learn about DTD syntax characters throughout this module. In most of the examples we'll use, the vertical white space in the DTD is optional. Indentation has been used to make the code more readable, but you need not indent your DTDs to use them.

Module 4

DTD defined

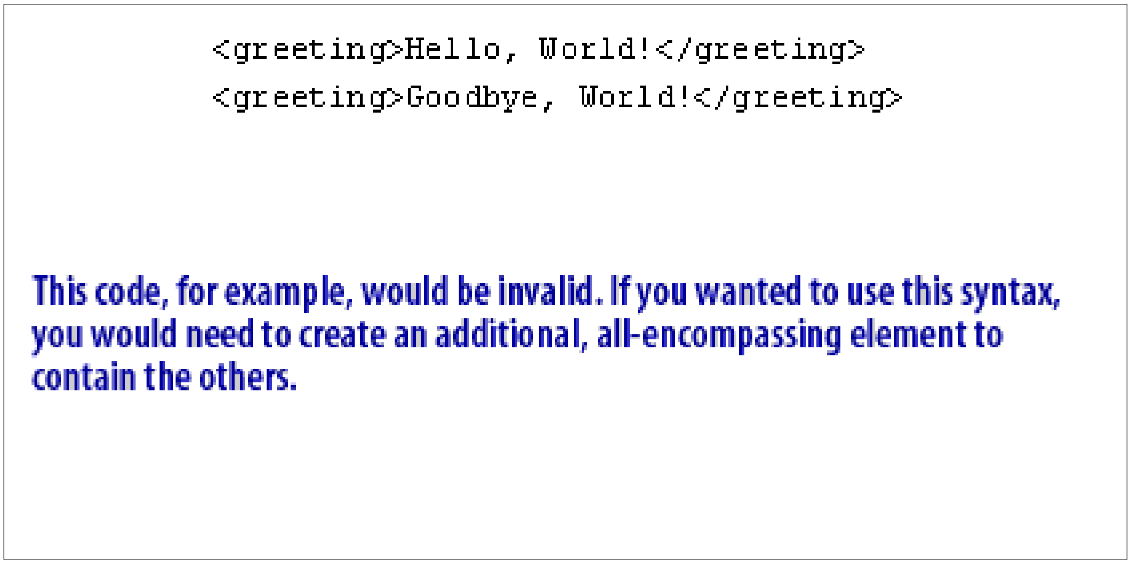
Determining the root element (FlipBook transcript)

XML documents are required to have a *root element*:



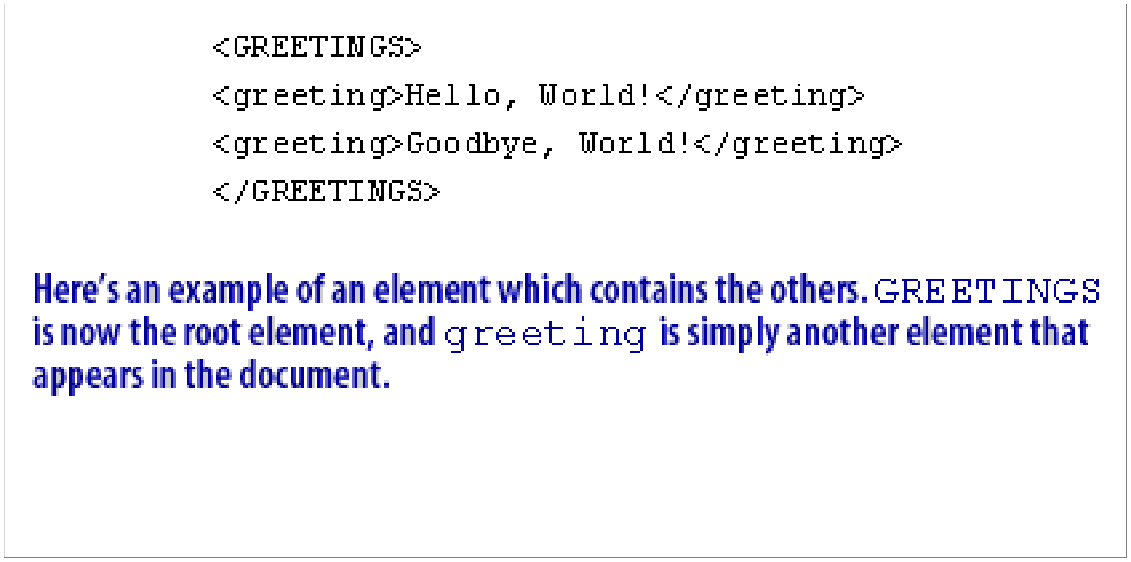
```
<greeting>Hello, World!</greeting>
```

Consider the most basic XML document. In this simple example, only one element is present: `greeting`. As such, `greeting` is the root element. The XML syntax would be incorrect if you used this element more than once.



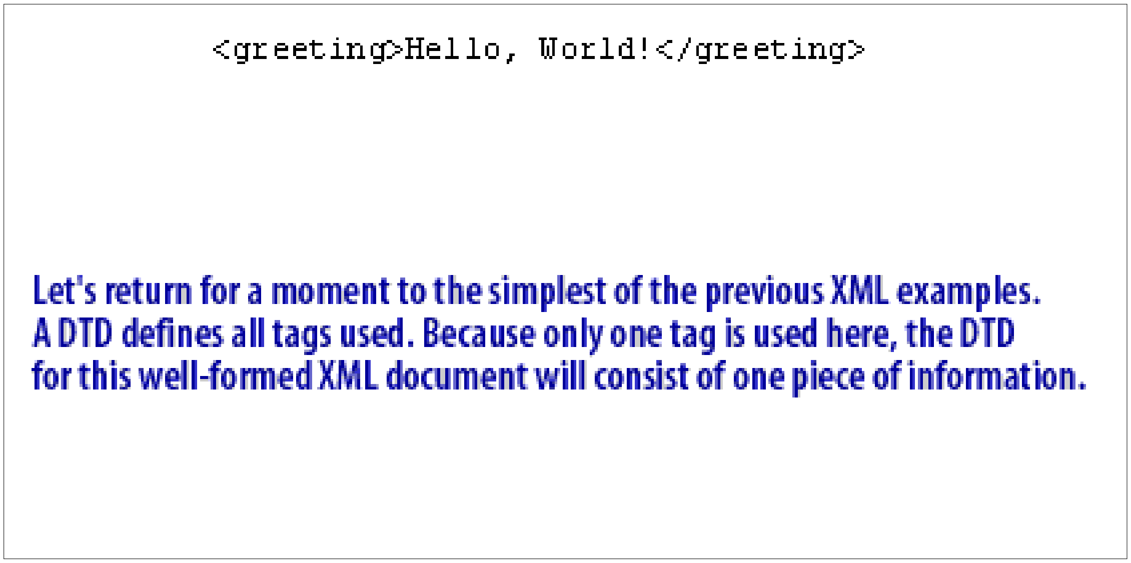
```
<greeting>Hello, World!</greeting>
<greeting>Goodbye, World!</greeting>
```

This code, for example, would be invalid. If you wanted to use this syntax, you would need to create an additional, all-encompassing element to contain the others.



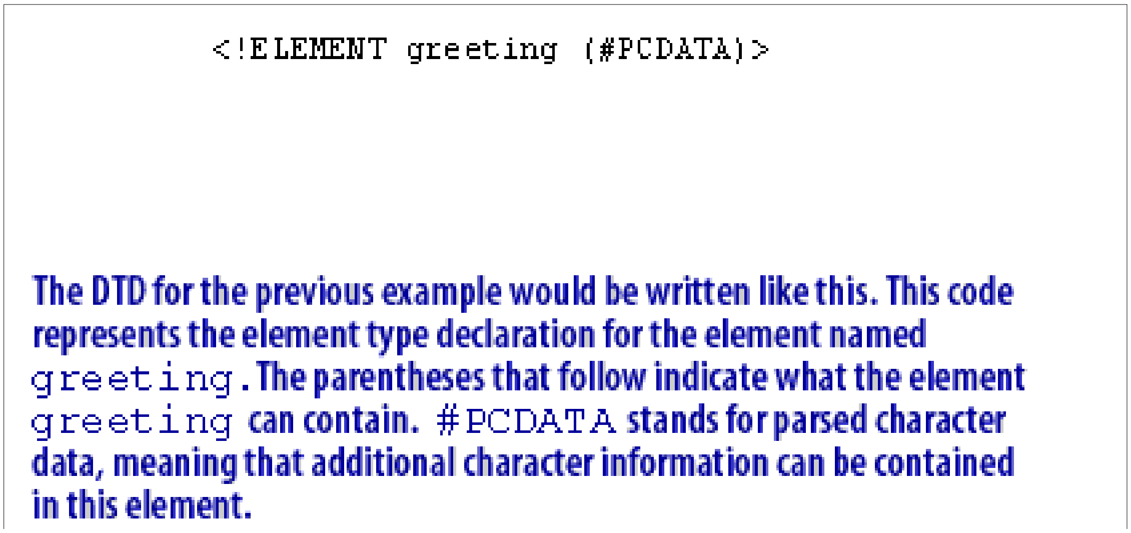
```
<GREETINGS>
<greeting>Hello, World!</greeting>
<greeting>Goodbye, World!</greeting>
</GREETINGS>
```

Here's an example of an element which contains the others. `GREETINGS` is now the root element, and `greeting` is simply another element that appears in the document.



```
<greeting>Hello, World!</greeting>
```

Let's return for a moment to the simplest of the previous XML examples. A DTD defines all tags used. Because only one tag is used here, the DTD for this well-formed XML document will consist of one piece of information.



```
<!ELEMENT greeting (#PCDATA)>
```

The DTD for the previous example would be written like this. This code represents the element type declaration for the element named `greeting`. The parentheses that follow indicate what the element `greeting` can contain. `#PCDATA` stands for parsed character data, meaning that additional character information can be contained in this element.

XML Fundamentals

Module 4

Lesson 5

Objective

Element type declarations

Declare basic elements in a DTD.

The syntax for declaring an element in a DTD is as follows:

```
<!ELEMENT elementName (allowed
element contents)>
```

Element names must begin with a letter but cannot begin with the letters "xml" in either upper or lower case. These three letters are reserved for the specification itself. In addition, you cannot use the colon character in an element name because it is reserved for other purposes.

Declaring elements that contain only text

When an element will contain only character data, declare the element in this manner:

```
<!ELEMENT elementName (#PCDATA)>
```

If only *#PCDATA* is specified, any text--but only text--can be included in the declared element. This requirement means that no child elements may be present in the element within which #PCDATA is specified.

Declaring child elements

When one element will contain other elements, this information must be specified in the element declaration.

Transcript

Tip

If an element will contain only child elements and no text other than those elements, it is said to have element content.

View Table

Click here to see a table of DTD symbols.

Quiz

Quiz

Click the Quiz button to see how well you understand DTD basics.

XML Fundamentals

Module 4

Lesson 6

Objective

Mixed content and empty elements

Write element declarations for mixed content and declare empty element.

If an element will contain a mix of text and additional child elements, it is said to have *mixed content*.

Transcript

Declaring an empty element

In HTML, the element BR is an *empty element*. There is no ending tag for the <BR> tag. To declare an empty element in XML, you must use the following syntax:

```
<!ELEMENT elementName EMPTY>
```

Note that there are no allowed element contents in this declaration. If there were, it would not be an empty element.

Remember that when adding empty elements to your XML page, you must add the / character before the closing angle bracket. If you have an element with this type declaration:

```
<!ELEMENT Marker EMPTY>
```

you would include the tag in an XML file as follows:

```
<Marker/>
```

Tip

Use the same method to include comments in the DTD as you would within an HTML document. At any point in the DTD, you can use the following syntax to include comments: <!-- comment text here -->

Exercise

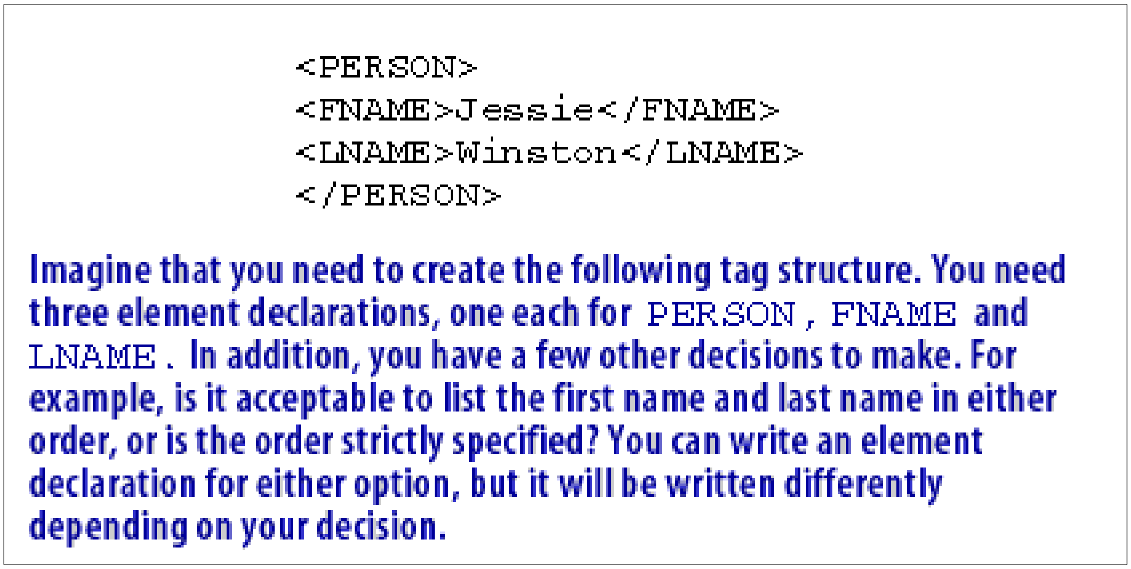
In this exercise, you'll declare child elements and mixed content in a DTD.

Module 4

Element type declarations

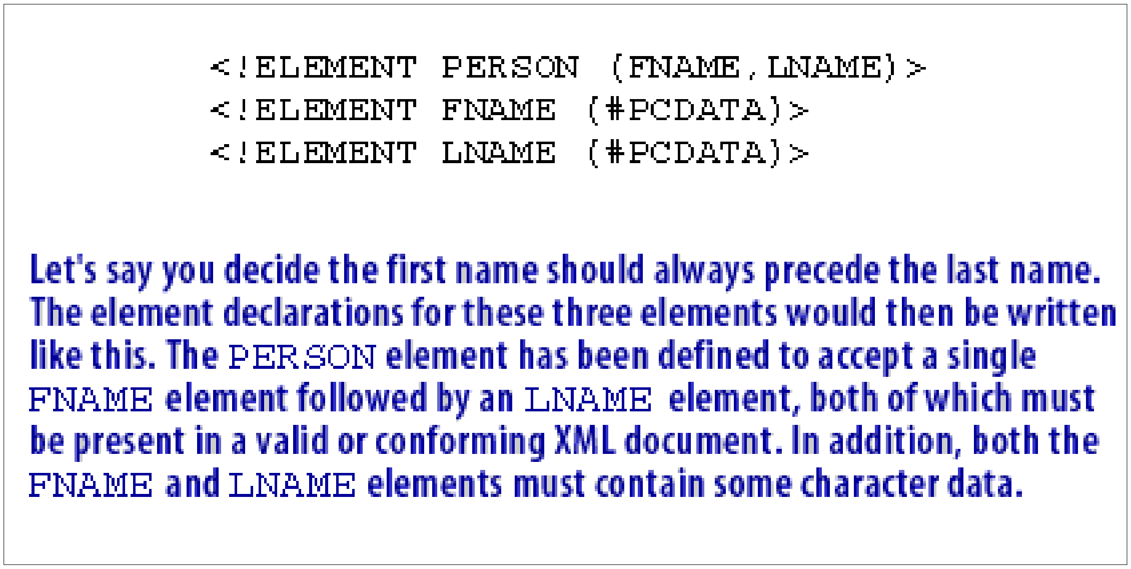
Specifying information in the element declaration (FlipBook transcript)

Here's an example of how you specify information in the element declaration:



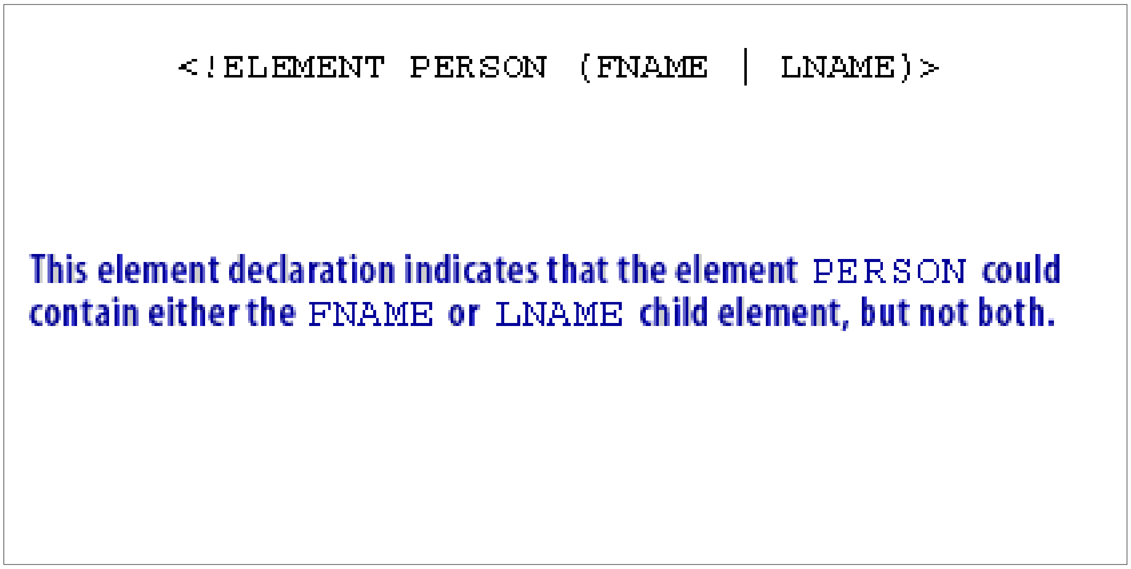
```
<PERSON>
<FNAME>Jessie</FNAME>
<LNAME>Winston</LNAME>
</PERSON>
```

Imagine that you need to create the following tag structure. You need three element declarations, one each for PERSON, FNAME and LNAME. In addition, you have a few other decisions to make. For example, is it acceptable to list the first name and last name in either order, or is the order strictly specified? You can write an element declaration for either option, but it will be written differently depending on your decision.



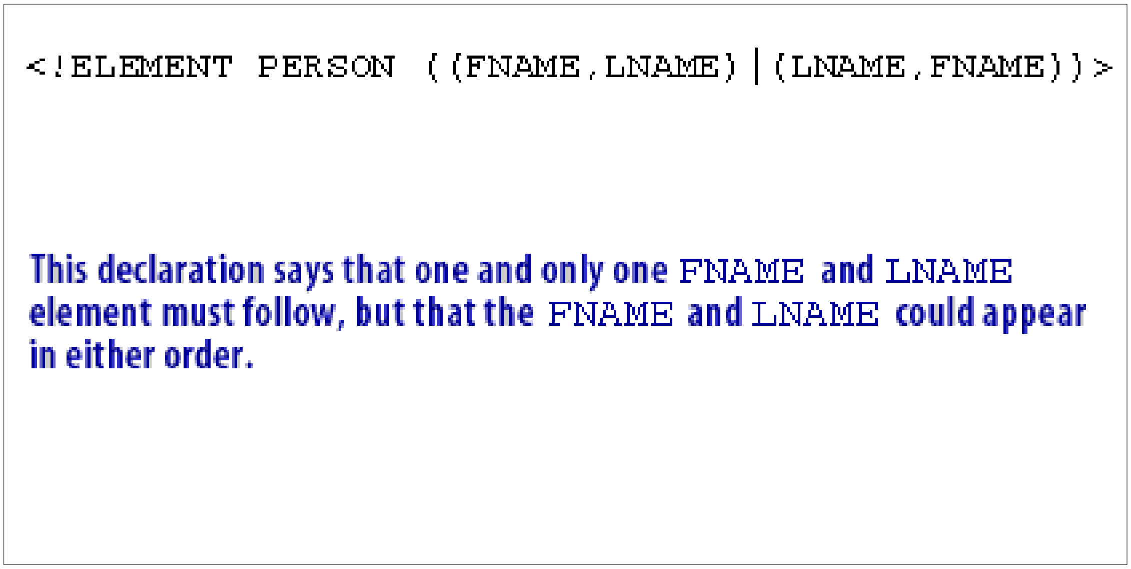
```
<!ELEMENT PERSON (FNAME,LNAME)>
<!ELEMENT FNAME (#PCDATA)>
<!ELEMENT LNAME (#PCDATA)>
```

Let's say you decide the first name should always precede the last name. The element declarations for these three elements would then be written like this. The PERSON element has been defined to accept a single FNAME element followed by an LNAME element, both of which must be present in a valid or conforming XML document. In addition, both the FNAME and LNAME elements must contain some character data.



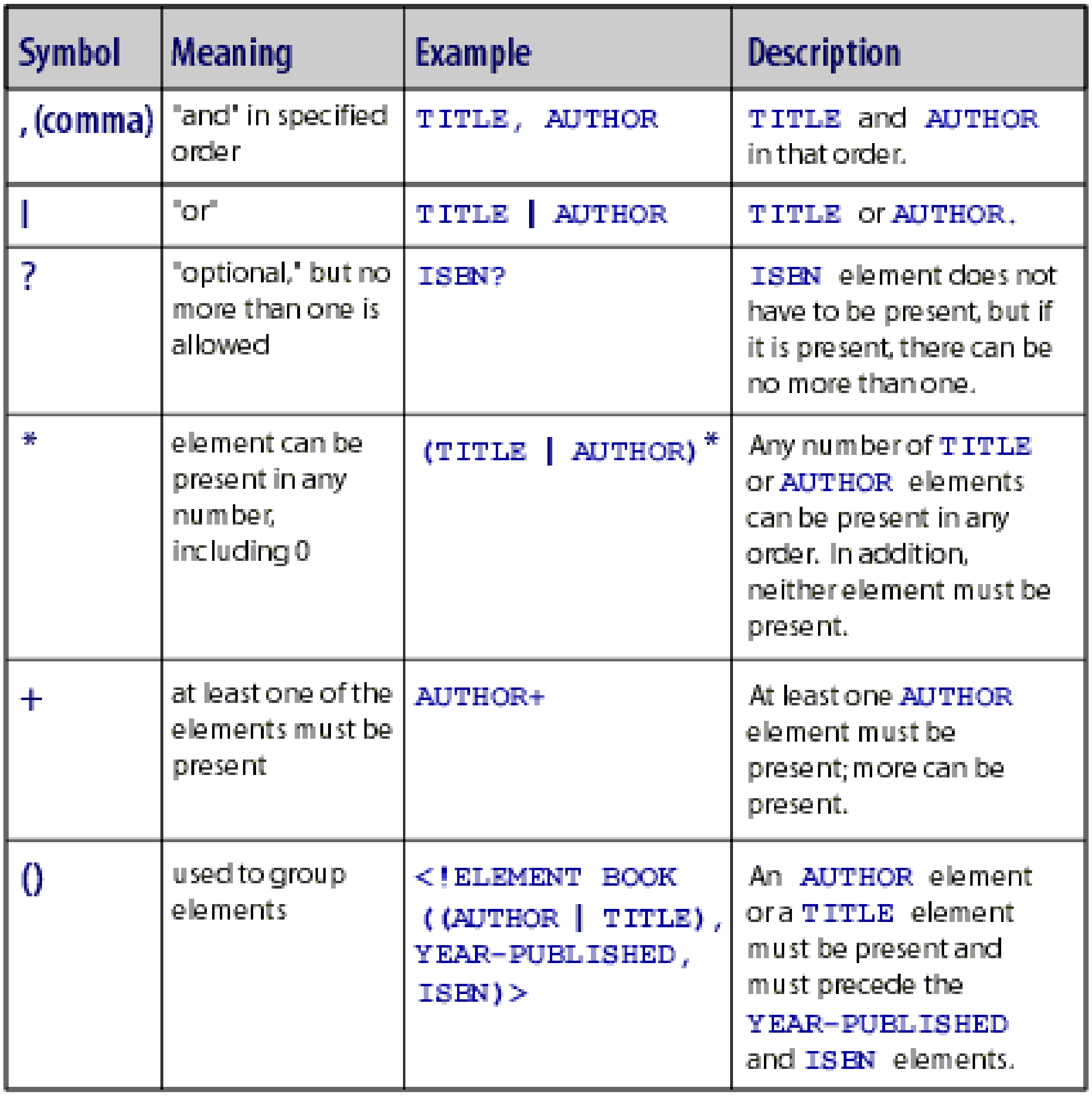
```
<!ELEMENT PERSON (FNAME | LNAME)>
```

This element declaration indicates that the element `PERSON` could contain either the `FNAME` or `LNAME` child element, but not both.



```
<!ELEMENT PERSON ((FNAME,LNAME)|(LNAME,FNAME))>
```

This declaration says that one and only one `FNAME` and `LNAME` element must follow, but that the `FNAME` and `LNAME` could appear in either order.



Symbol	Meaning	Example	Description
, (comma)	"and" in specified order	TITLE, AUTHOR	TITLE and AUTHOR in that order.
|	"or"	TITLE | AUTHOR	TITLE or AUTHOR.
?	"optional," but no more than one is allowed	ISBN?	ISBN element does not have to be present, but if it is present, there can be no more than one.
*	element can be present in any number, including 0	(TITLE | AUTHOR)*	Any number of TITLE or AUTHOR elements can be present in any order. In addition, neither element must be present.
+	at least one of the elements must be present	AUTHOR+	At least one AUTHOR element must be present; more can be present.
()	used to group elements	<!ELEMENT BOOK ((AUTHOR | TITLE), YEAR-PUBLISHED, ISBN)>	An AUTHOR element or a TITLE element must be present and must precede the YEAR-PUBLISHED and ISBN elements.

Module 4

Quiz

Defining the DTD

Each question is worth one point. Some questions ask you to select **the best answer**, others ask you to select **all the correct answers**. To receive credit for questions asking for **all the correct answers**, you must select all the correct answers and **only** the correct answers.

1.An XML Document Type Definition (DTD) allows you to:

Please select **all the correct answers**.

- □ A. Create meaningful document structure and apply it consistently
- □ B. Define which elements will be used and how
- □ C. Share information across documents
- □ D. Create well-formed XML documents

2.An XML file is "valid" if it:

Please select **the best answer**.

- ○ A. Conforms to the specified DTD
- ○ B. Is well-formed
- ○ C. Includes a root element at the end of the file
- ○ D. Defines the syntax, structure and vocabulary used within the DTD

3.A DTD consists of a simple text file in which:

Please select **all the correct answers**.

- □ A. The basic unit is the root element
- □ B. The basic unit is an element
- □ C. You specify the basic unit
- □ D. You specify which attributes can be used with which elements
- □ E. You specify instructions for elements in the corresponding XML file

4.In the DTD, a child element:

Please select **the best answer**.

- ○ A. Is an element that contains other elements
- ○ B. Is said to have element content
- ○ C. Must be specified in the parent element declaration
- ○ D. Includes the `#PCDATA` statement

5.To declare an element in a DTD, you must:

Please select **all the correct answers**.

□ A. Begin the element name with the letters "xml"

□ B. Not begin the element name with the letters "xml"

□ C. Use all lowercase letters

□ D. Use the colon character

□ E. Not use the colon character

OK, I'm Done

XML Fundamentals

Module 4

Lesson 7
Objective

Referencing DTD declarations

Reference DTD declarations in XML.

Every XML file that uses a *DTD* must have a *document type declaration* immediately after the XML declaration.

The `<!DOCTYPE>` declaration can point to internal or external DTD information. A special `standalone` attribute can be added to the `<?xml?>` declaration to indicate whether the `<!DOCTYPE>` declaration should point to internal or external DTD declarations.

If all the DTD information will be enclosed within the XML document, you can specify that the `standalone` attribute's value is `"yes"`. If the XML document should be referencing an external DTD, the value of this attribute should be `"no"`. The following tag indicates that the DTD information should appear in the XML document itself:

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

If the DTD will exist in a separate file, you would use instead the following XML declaration:

```
<?xml version="1.0"
encoding="UTF-8" standalone="no"?>
```

Both the internal and external DTD examples require a `<!DOCTYPE>` declaration in some form.

Tip

The `<!DOCTYPE>` declaration should be on the line just below the <?xml?> declaration to avoid errors when parsed.

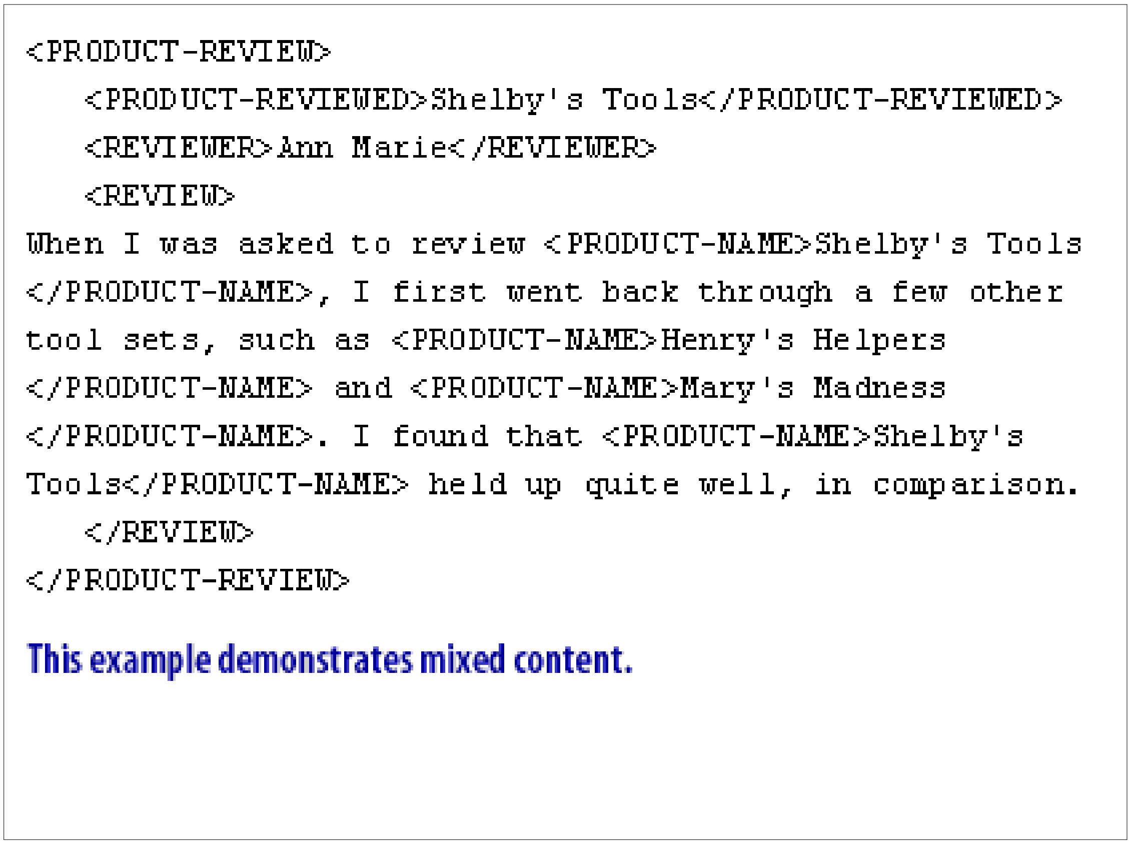
XML Fundamentals

Module 4

Mixed content and empty elements

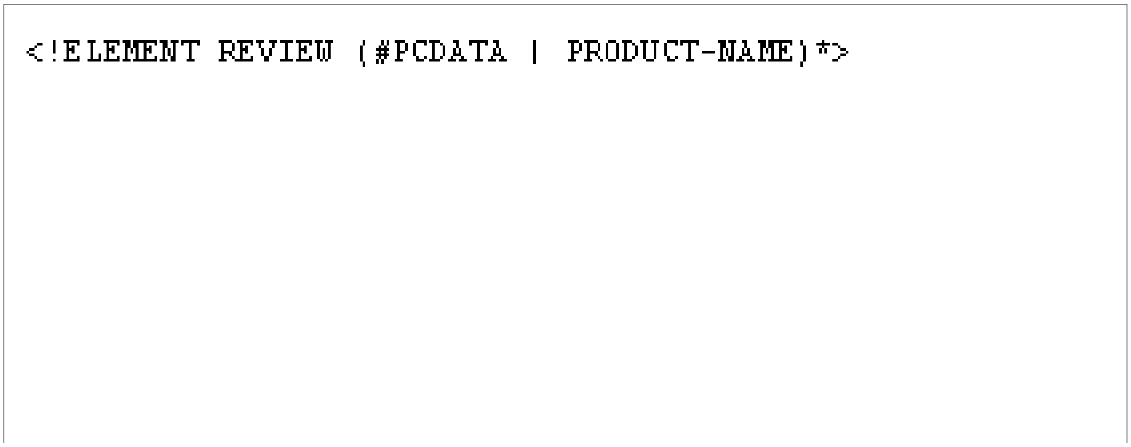
Working with mixed content (FlipBook transcript)

An example of working with mixed content:

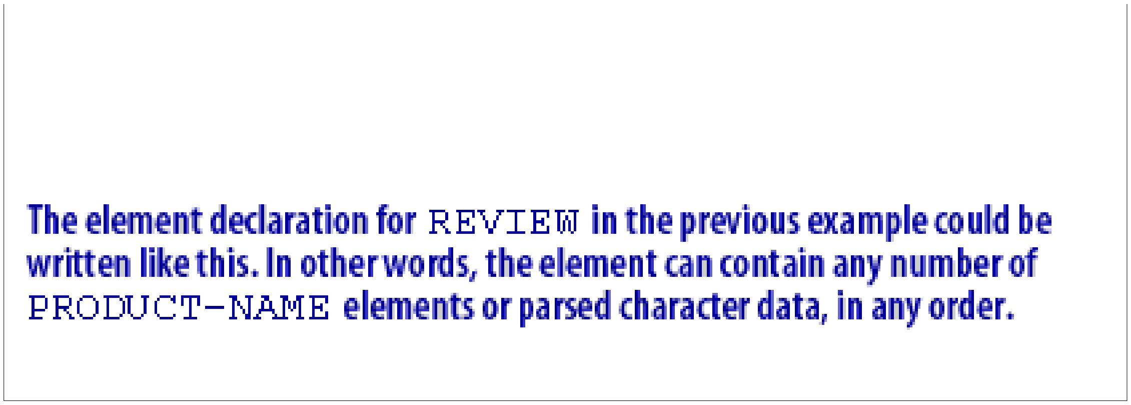


```
<PRODUCT-REVIEW>
   <PRODUCT-REVIEWED>Shelby's Tools</PRODUCT-REVIEWED>
   <REVIEWER>Ann Marie</REVIEWER>
   <REVIEW>
When I was asked to review <PRODUCT-NAME>Shelby's Tools
</PRODUCT-NAME>, I first went back through a few other
tool sets, such as <PRODUCT-NAME>Henry's Helpers
</PRODUCT-NAME> and <PRODUCT-NAME>Mary's Madness
</PRODUCT-NAME>. I found that <PRODUCT-NAME>Shelby's
Tools</PRODUCT-NAME> held up quite well, in comparison.
   </REVIEW>
</PRODUCT-REVIEW>
```

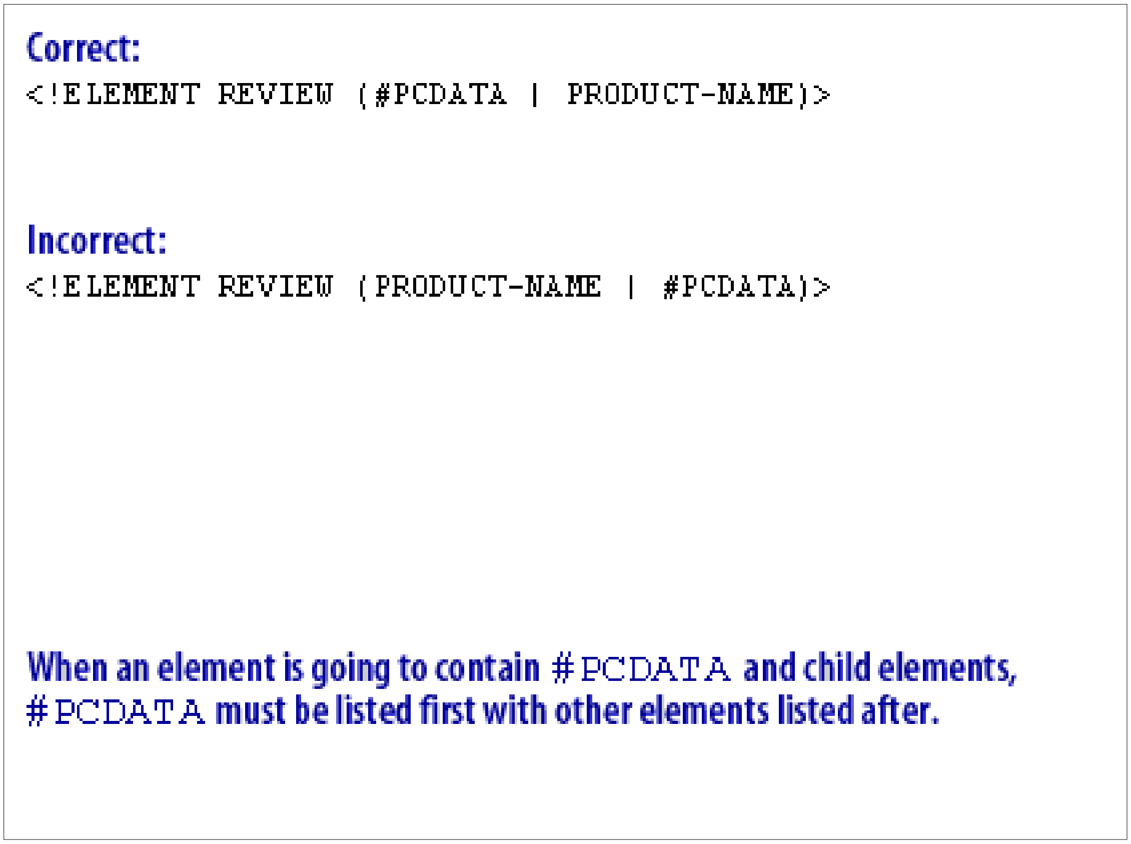
This example demonstrates mixed content.



```
<!ELEMENT REVIEW (#PCDATA | PRODUCT-NAME)*>
```



The element declaration for REVIEW in the previous example could be written like this. In other words, the element can contain any number of PRODUCT-NAME elements or parsed character data, in any order.



Correct:

```
<!ELEMENT REVIEW (#PCDATA | PRODUCT-NAME)>
```

Incorrect:

```
<!ELEMENT REVIEW (PRODUCT-NAME | #PCDATA)>
```

When an element is going to contain #PCDATA and child elements, #PCDATA must be listed first with other elements listed after.

XML Fundamentals

Module 4

Exercise

Mixed content and empty elements

Declaring child elements and mixed content

Objective

Declaring child elements and mixed content in a DTD.

Exercise scoring

Full credit for this exercise is 6 points (3 points for each step). You'll submit your code to the course tutors.

Instructions

Consider the following form letter, which alerts the recipient that he or she has won a prize in a contest. The code needed to transcribe this letter into a well-formed XML document has been added:

```
<?xml version="1.0"?>
<WINNERALERT>
Dear <WINNER> <FNAME>Samson</FNAME>
<LNAME>Lane</LNAME> </WINNER>,
Congratulations! You are a prize winner in the Editors
Clearinghouse Sweepstakes.
Your entry was chosen from thousands in the drawing for
<PRIZE> <PRIZE1>a new car</PRIZE1> or <PRIZE2> $25,000
</PRIZE2> </PRIZE>.
Simply return this claim form by
<CLAIMBY> <DATE> November 30 </DATE> <YEAR> 1999 </YEAR>
</CLAIMBY> to claim your prize.
</WINNERALERT>
```

1. Write element type declarations for each of your elements. Consider the following points as you code:
   - Use proper syntax for declaring DTD elements, and consider the order in which you declare child elements.
   - Use DTD symbols to specify the contents allowed for each element.

   - Each of the three main elements in the form letter should have child elements. Do any child elements have children of their own?
   - The `PRIZE` element should have mixed content. Review the text in the form letter carefully if you are unsure of the mixed content.
   - Be sure to declare the root element.

   In the text box below, cut and paste your DTD code, and continue on to the next step.

2. Suppose you want the `PRIZE` element to include different possible pairs of prizes from which the winner can choose. For example, `PRIZEA` offers a choice between a new car and $25,000, `PRIZEB` offers a choice between a home entertainment system and $5,000, and so forth. Write the element type declarations for this type of `PRIZE` element. Consider the following points as you code:
   - Each winner is offered only one prize pair from which to choose, so only one pair can appear on any `WINNERALERT`.
   - The word "`or`" between prizes provided mixed content for the `PRIZE` element in Step 2. Where does the mixed content occur in this scenario?

   In the text box below, cut and paste your `PRIZE` element code. When you're ready to submit all your answers to the course tutor, click the OK, I'm Done button.

Starter files

We've supplied the form letter shown above in a file called `formletter.xml`. You'll find this file in the 04-06 folder of the course download available from the Resources page.

OK, I'm Done

XML Fundamentals

Module 4

Lesson 8

The DTD file and XML file as separate files

Objective

Create a DTD file to work with a separate XML file.

Let's look at an example in which the DTD exists as a separate file. The name of the XML document is `hello.xml`, and the name of the DTD file is `greeting.dtd`:

MouseOver

Transcript

Note that in this example, the word `SYSTEM` is used to specify that this DTD is located on the same system as the document itself. The DTD for an HTML document uses the word `PUBLIC` in this position to indicate that the DTD is a publicly available document, and is followed by a reference to where that document can be found. Of course, `greeting.dtd` refers to the name of the file in which the DTD information can be found.

```
Filename: greeting.dtd
```

```
<!ELEMENT greeting (#PCDATA)>
```

In this example, the XML file will be validated against the DTD. If the first file contained any other tags, this document would not be considered valid because only the `greeting` element has been defined by the DTD.

Quiz

Quiz

Click the Quiz button to check your knowledge of DTD construction.

<!!-- XML Fundamentals -->

Module 4

Lesson 9
Objective

Declarations within the XML file

Embed declarations within the XML file.

For a very simple document, it can make sense to embed the DTD information directly within the XML document:

MouseOver

Transcript

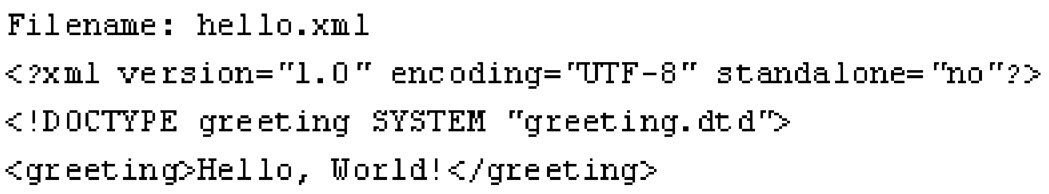
This example shows a *document type declaration* statement, indicated by `<!DOCTYPE>`. Following `DOCTYPE` is the root element or the containing element for the XML file, which is `greeting`. The element `greeting` is also the only element within the document, and it contains character data.

XML Fundamentals

Module 4

The DTD file and XML file as separate files

Separate DTD and XML (MouseOver transcript)



```
Filename: hello.xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE greeting SYSTEM "greeting.dtd">
<greeting>Hello, World!</greeting>
```

standalone

The `standalone` attribute has a value of "no" because the DTD information will not be included in the XML file.

<!DOCTYPE>

The <!DOCTYPE> declaration points to the external DTD file.

Module 4

Quiz

Constructing the DTD

Each question is worth one point. Some questions ask you to select **the best answer**, others ask you to select **all the correct answers**. To receive credit for questions asking for **all the correct answers**, you must select all the correct answers and **only** the correct answers.

1.A DTD element that contains both text and additional child elements is said to have:

Please select **the best answer**.

○ A. Complex declaration
○ B. Mixed content
○ C. Tree structure
○ D. Character elements

2.In the DTD, an empty element:

Please select **the best answer**.

○ A. Is similar to an empty tag in HTML
○ B. Allows element content in the declaration
○ C. Requires a / character before the closing bracket
○ D. Need not be specified in the element declaration

3.Every XML file that uses a DTD must have a:

Please select **the best answer**.

○ A. `<!DTD>` declaration
○ B. `<!DOCTYPE>` declaration
○ C. `<STANDALONE>` declaration
○ D. `<DOCTYPE>` declaration
○ E. `<DTD>` declaration

4.Which syntax should you use to declare that your XML file's DTD information exists in a separate file?

Please select **the best answer**.

○ A. `<?xml version="1.0" standalone="no"?>`
○ B. `<!doctype standalone="no">`
○ C. `<?xml version="1.0" standalone="yes"?>`
○ D. `<!doctype standalone="yes"!>`
○ E. `<?xml version="1.0" standalone="no">`

OK, I'm Done

Module 4

Lesson 10
Objective

A checklist for creating DTDs

Create a DTD from an existing set of tags.

The following checklist outlines the steps you can take to create a DTD from an existing set of tags:

1. Determine the *root element* for the document.
2. List all the other elements used in the document.
3. Create the document tree structure for these elements.
4. Note whether elements need to appear in a specific order.
5. Note whether certain elements have content that must be present, can be present, or may be present multiple times.
6. Note which elements will contain text, and which will only contain other elements.

Exercise

Exercise
In this exercise, you'll create a simple DTD

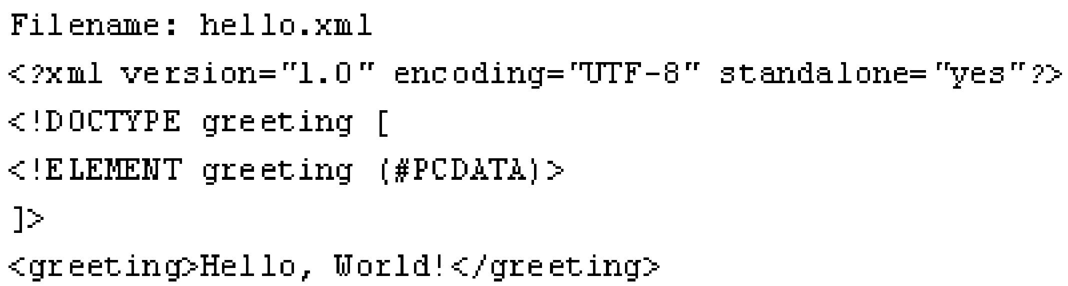
Exercise

Exercise
Once you've practiced creating a simple DTD, try creating a DTD for mixed content.

Module 4

Declarations within the XML file

Embedding the DTD in an XML file (MouseOver transcript)



```
Filename: hello.xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE greeting [
<!ELEMENT greeting (#PCDATA)>
]>
<greeting>Hello, World!</greeting>
```

<!DOCTYPE> Because the DTD declarations are included within the XML file, the standalone attribute will have the value "yes".

standalone The <!DOCTYPE> declaration is used to point to internal DTD declarations.

DTD basics
Module wrap-up

In this module, you learned how to create DTDs to define the syntax, grammar and vocabulary of XML documents. You also created both simple and complex DTDs.

This module discusses the following terms in relation to XML:

- *Document Type Definition (DTD)*
- *root element*
- *#PCDATA*
- *mixed content*
- *empty element*
- *document type declaration*

Discussion
Do you have any questions or comments about the material in this module? If so, look in the folder Creating and using DTDs in the Discussion section, which you can reach by clicking the Discuss button in the toolbar.

Quiz

Quiz
Click the Quiz button to test what you've learned in this module.

A checklist for creating DTDs

Creating a simple DTD

Objective

Create a simple DTD for a simple XML file.

Exercise scoring

This exercise is worth a total of 1 point. You don't have to submit your code to the course tutors to receive credit for this exercise.

Overview

In this exercise, you will create a simple DTD for a simple XML file.

Instructions

Start with the file `invent.xml`, which can be found in the 04-10 folder in the course download. This file will contain the following information:

```
<?xml version="1.0"?>
<INVENTORY>
   <PRODUCT>
      <NAME>
      Papa Joe's Salsa
      </NAME>
      <QTYAVAIL>
      10
      </QTYAVAIL>
      <UNITPRICE>
      $5.95
      </UNITPRICE>
   </PRODUCT>
   <PRODUCT>
      <NAME>
      Mama Louise's Spaghetti Sauce
      </NAME>
      <QTYAVAIL>
      25
      </QTYAVAIL>
      <UNITPRICE>
      $4.98
      </UNITPRICE>
   </PRODUCT>
</INVENTORY>
```

Follow these steps to create a DTD for this document:

1. Indicate that this document will contain the DTD within the document itself. Do this by adding `standalone="yes"` to the `xml` declaration.
2. Create the `DOCTYPE` declaration for the `INVENTORY` document immediately below the `xml` declaration.
3. Create the element declaration for the root element `INVENTORY`. Because your `INVENTORY` should be comprised of at least one `PRODUCT`, use the + symbol to indicate that at least one of the element should be present.
4. Create the element declaration for the `PRODUCT` element such that the order of elements in a `PRODUCT` element must conform to that already shown in the XML file. Each product must have one and only one entry for name, quantity available, and unit price, which must appear in that order.
5. Add the appropriate element declarations for the remaining elements.

Starter files

Remember, we've supplied the code shown above in a file called `invent.xml`. You'll find this file in the 04-10 folder of the course download available from the Resources page.

After completing the exercise, click the OK, I'm Done button and the system will award you full credit.

OK, I'm Done

XML Fundamentals

Module 4

Exercise

A checklist for creating DTDs

Creating a DTD for mixed content

Objective

Create a DTD describing elements containing mixed content.

Exercise scoring

Full credit for this exercise is 10 points. You'll submit your code to the course tutors.

Overview

In this exercise, you will create a DTD describing elements containing mixed content.

Here is the confirmation order for which you will be creating a DTD:

```
Thank you, Shelby Malone, for your recent order of 5 copies of

Where Wolves Cry, the latest novel from Percy Forbes.

Our records indicate the bill should be shipped to:

Shelby Malone
45 Main Street
Corning, NY 10508 USA
Thank you for your business.
```

Instructions

Start with file `confirm.xml`, which can be found in the 04-10 folder in the course download. This file will contain the following code:

```
<?xml version="1.0"?>
<ORDER-CONFIRMATION>
<PAR>
Thank you, <BUYER>Shelby Malone</BUYER>, for your recent order of
<ORDER><QTY>5</QTY> copies of <PRODUCT-NAME>Where Wolves Cry</PRODUCT-NAME>,
the latest <TYPE>novel</TYPE> from <AUTHOR>Percy Forbes</AUTHOR></ORDER>.
</PAR>
```

```
<PAR>
Our records indicate the bill should be shipped to:
</PAR>
<PAR>
<SHIP-TO-INFO>
   <SHIP-TO-NAME>Shelby Malone</SHIP-TO-NAME>
   <SHIP-TO-ADDRESS>45 Main Street</SHIP-TO-ADDRESS>
   <SHIP-TO-CITY>Corning, </SHIP-TO-CITY>
   <SHIP-TO-STATE>NY </SHIP-TO-STATE>
   <SHIP-TO-POST>10508</SHIP-TO-POST>
<SHIP-TO-COUNTRY>USA</SHIP-TO-COUNTRY>
</SHIP-TO-INFO>
</PAR>
<PAR>
Thank you for your business.
</PAR>
</ORDER-CONFIRMATION>
```

Here are the steps you'll take to create the DTD:

1. At the top of this file, create the DTD information to validate this document. The root element is `ORDER-CONFIRMATION`. Open a `<!DOCTYPE>` declaration just below the `xml` declaration.
2. Examine the confirmation letter components. What is the first element contained within the `ORDER-CONFIRMATION` element? Create a declaration for the `ORDER-CONFIRMATION` element.
3. The `PAR` (paragraph) element is tricky because it opens and closes repeatedly, and contains both plain text and child elements. To help denote the inherent structure, you might want to view the first `PAR` element by itself:

```
<PAR>
Thank you,
   <BUYER>
   Shelby Malone
   </BUYER>
, for your recent order of
   <ORDER>
      <QTY>
5
      </QTY>
copies of
      <PRODUCT-NAME>
Where Wolves Cry
      </PRODUCT-NAME>
, the latest
      <TYPE>
novel
      </TYPE>
from
```

```
      <AUTHOR>
Percy Forbes
      </AUTHOR>
   </ORDER>
.
</PAR>
```

When viewed in this form, you can see that the `PAR` element contains the following elements: `BUYER`, `ORDER` and `SHIP-TO-INFO`. The `ORDER` element in turn contains additional elements. Write a declaration for the `PAR` element such that the order of appearance of these elements does not matter.

4. Write the element declaration for the `ORDER` element such that the order of elements does not matter, and so that any number of them can be present.
4. Include the element declarations for those elements contained within `ORDER`.
5. The next element, `SHIP-TO-INFO`, contains several other elements. Add the appropriate declaration for the `SHIP-TO-INFO` element.
6. Add the remaining element declarations below the `SHIP-TO-INFO` element declaration.

Starter files

Remember, we've supplied the code shown above in a file called `confirm.xml`. You'll find this file in the 04-10 folder of the course download available from the Resources page.

What to submit to the course tutors

In the text box below, cut and paste the contents of your DTD. Click OK, I'm Done to submit your code to the course tutors.

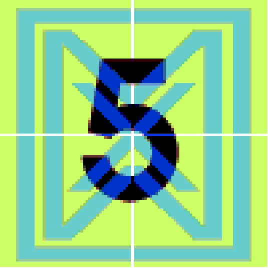
OK, I'm Done

Module 5

Lesson 1

Building the DTD

Module introduction



This module shows you how to build your DTD by creating element, attribute-list, and entity declarations.

Module learning objectives

After completing this module, you will have the skills and knowledge necessary to:

- Determine the best way to use attributes in a DTD
- Declare lists of attributes
- Define entities in a DTD
- Create entities to use within a DTD
- Create an XML file based on an existing DTD
- Explain how a parsing utility can help you verify the well-formedness and validity of XML documents

Module 4

Quiz

Developing the DTD

Each question is worth one point. Some questions ask you to select **the best answer**, others ask you to select **all the correct answers**. To receive credit for questions asking for **all the correct answers**, you must select all the correct answers and **only** the correct answers.

1.Proper DTD syntax requires that you:

Please select **the best answer**.

○ A. Use vertical white space
○ B. Use the file name extension `.dtd` on separate DTD files
○ C. Indent your DTD elements hierarchically
○ D. Embed DTD information directly within the XML document

2.When creating a DTD for an existing XML document, you should determine the root element:

Please select **all the correct answers**.

□ A. After you list all the elements used in the document
□ B. As the first step
□ C. Only if your document will not render the document in a browser
□ D. As the final step
□ E. Before you determine the document tree structure for the elements

3.The steps you should take when creating a DTD for an existing XML document include:

Please select **all the correct answers**.

□ A. Determining the DTD's root element
□ B. Deciding whether elements should appear in a specific order
□ C. Noting which elements will contain text, and which will contain only other elements
□ D. Listing all the document elements in alphabetical order
□ E. Determining whether certain elements' content must be present

4.How do you specify that child elements contained in an element must appear in the same order specified by the DTD file?

Please select **the best answer**.

○ A. Use vertical slashes (|) to separate ordered child elements in the element declaration

○ B. Add a plus sign (+) after the child elements in the element declaration

○ C. Add the `#PCDATA` statement before the child elements in the element declaration

○ D. Use commas (,) to separate ordered child elements in the element declaration

○ E. Add a forward slash (/) before the closing bracket of the child element declaration

OK, I'm Done

XML Fundamentals

Module 5

Lesson 2 Objective

Declaring attributes in a DTD

Determine the best way to use attributes in a DTD.

In HTML, you are familiar with using not only tags but attribute values to provide information about the tag. The `INPUT` element provides a great deal of variety in HTML, depending on the value of the `TYPE` attribute. A tag such as this:

```
<INPUT TYPE="text">
```

indicates a text box element. If the type is set to "`radio`", the `INPUT` element becomes a radio button. If the type is set to "`checkbox`", the `INPUT` element becomes a check box, and so forth.

In XML, you can use attributes. However, *SGML* purists tend to look down on the practice of embedding useful information *within* tags, as opposed to *between* tags.

These frames show various ways of using attributes:

Transcript

XML Fundamentals

Module 5

Lesson 3

Objective

Attribute-list declaration syntax

Declare lists of attributes.

You declare lists of attributes in a manner similar to how you declare elements-by means of an *attribute-list declaration*.

The generic syntax for declaring an attribute is as follows:

MouseOver

Transcript

View Table

Click the View Table button to see the most commonly-used attribute types.

The following example demonstrates an attribute-list declaration and an example of attribute usage:

MouseOver

Transcript

Exercise

Exercise
In this exercise, you'll create an attribute-list declaration.

Exercise

Exercise
Click the Exercise button to create an XML document from a DTD.

Tip

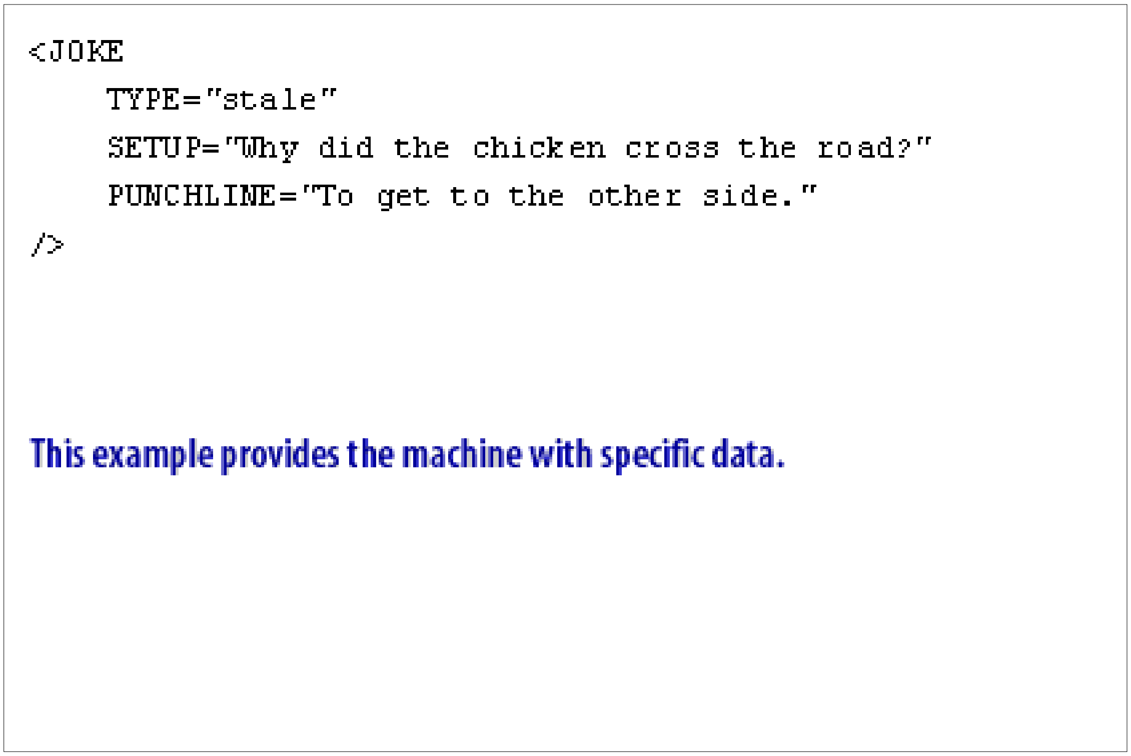
For any attribute-list declaration, only one attribute can be specified as type ID.

Module 5

Declaring attributes in a DTD

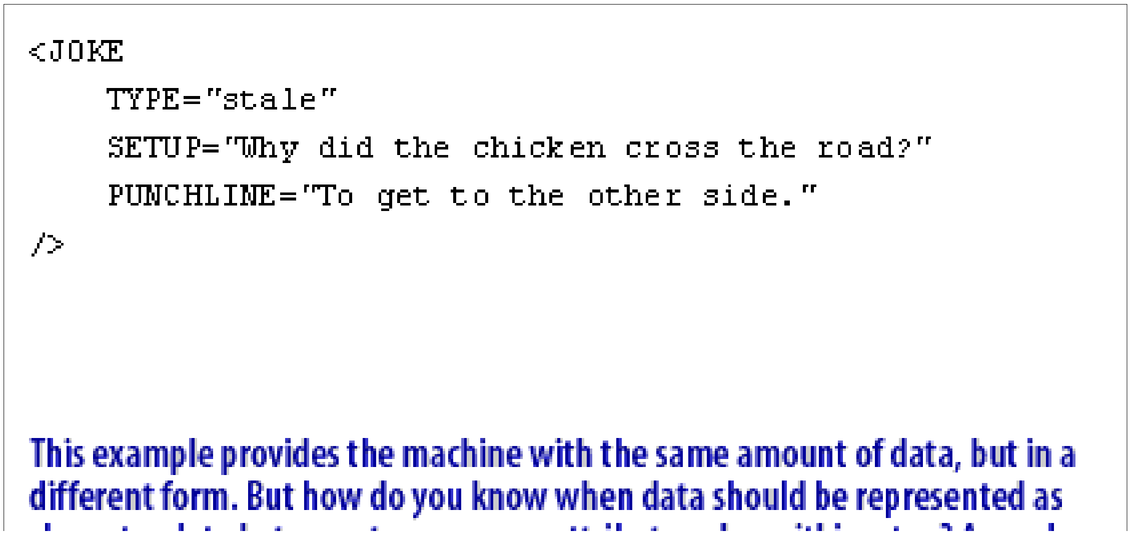
Using attributes (FlipBook transcript)

These frames show various ways of using attributes:



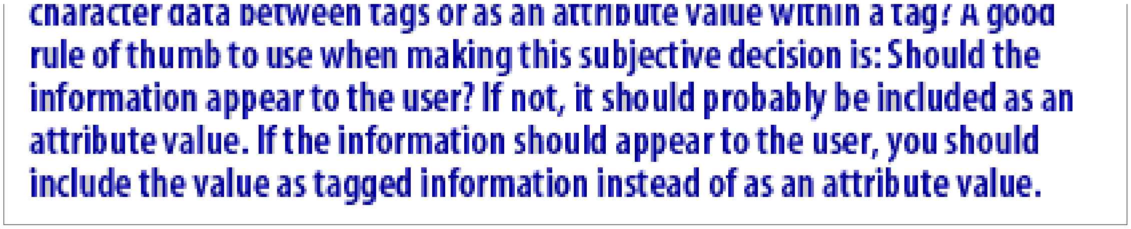
```
<JOKE
    TYPE="stale"
    SETUP="Why did the chicken cross the road?"
    PUNCHLINE="To get to the other side."
/>
```

This example provides the machine with specific data.

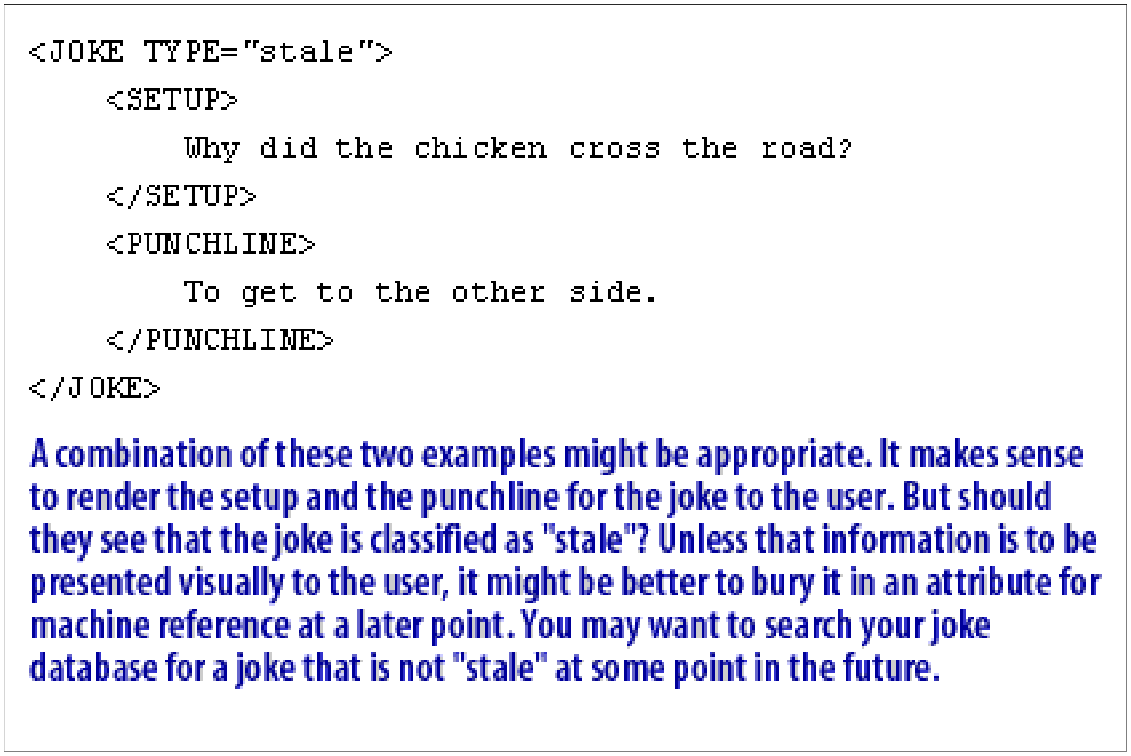


```
<JOKE
    TYPE="stale"
    SETUP="Why did the chicken cross the road?"
    PUNCHLINE="To get to the other side."
/>
```

This example provides the machine with the same amount of data, but in a different form. But how do you know when data should be represented as



character data between tags or as an attribute value within a tag? A good rule of thumb to use when making this subjective decision is: Should the information appear to the user? If not, it should probably be included as an attribute value. If the information should appear to the user, you should include the value as tagged information instead of as an attribute value.



```
<JOKE TYPE="stale">
    <SETUP>
        Why did the chicken cross the road?
    </SETUP>
    <PUNCHLINE>
        To get to the other side.
    </PUNCHLINE>
</JOKE>
```

A combination of these two examples might be appropriate. It makes sense to render the setup and the punchline for the joke to the user. But should they see that the joke is classified as "stale"? Unless that information is to be presented visually to the user, it might be better to bury it in an attribute for machine reference at a later point. You may want to search your joke database for a joke that is not "stale" at some point in the future.

Lesson 4
Objective

Defining entities

Define entities in a DTD.

In HTML, you can include special characters through the use of pre-defined entities such as these:

`&lt;`

`&quot;`

`&copy;`

These represent predefined characters that are not keyboard characters or that might be interpreted incorrectly. For example, `&lt;` represents the "less than" left-angle bracket character (<). Entities are defined in HTML to help define certain characters that might otherwise be interpreted differently.

In XML, there are five built-in entities that you need not define but can readily use. These are:

`&lt;`

`&gt;`

`&quot;`

`&apos;`

`&amp;`

In a DTD, you can define your own entities in addition to these five. You can also define multiple characters.

If you have ever performed a "mail merge," you are familiar with using replaceable fields to represent actual text. Think of an entity as a replaceable field representing some other text. The beauty of using entities is that you can define them once in a DTD and then use them throughout a range of documents.

Declare an entity using the following syntax:

```
<!ENTITY entityName "character
string represented">
```

If in your DTD you declared the following entity:

```
<!ENTITY prodname "ACME Calendar">
```

you could use the following in your XML file:

```
Thank you for choosing &prodname;
as your primary scheduling program.
```

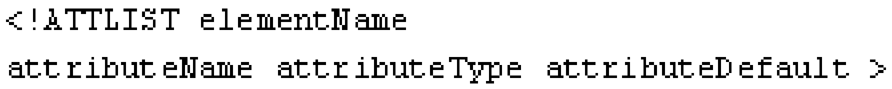
When rendered by a user agent, the actual text would then read as follows:

```
Thank you for choosing ACME
Calendar as your primary scheduling
program.
```

Module 5

Attribute-list declaration syntax

Attribute syntax (MouseOver transcript)



```
<!ATTLIST elementName
attributeName attributeType attributeDefault >
```

elementName

The `elementName` represents the element for which the attribute (or attribute list) is being defined.

attributeName

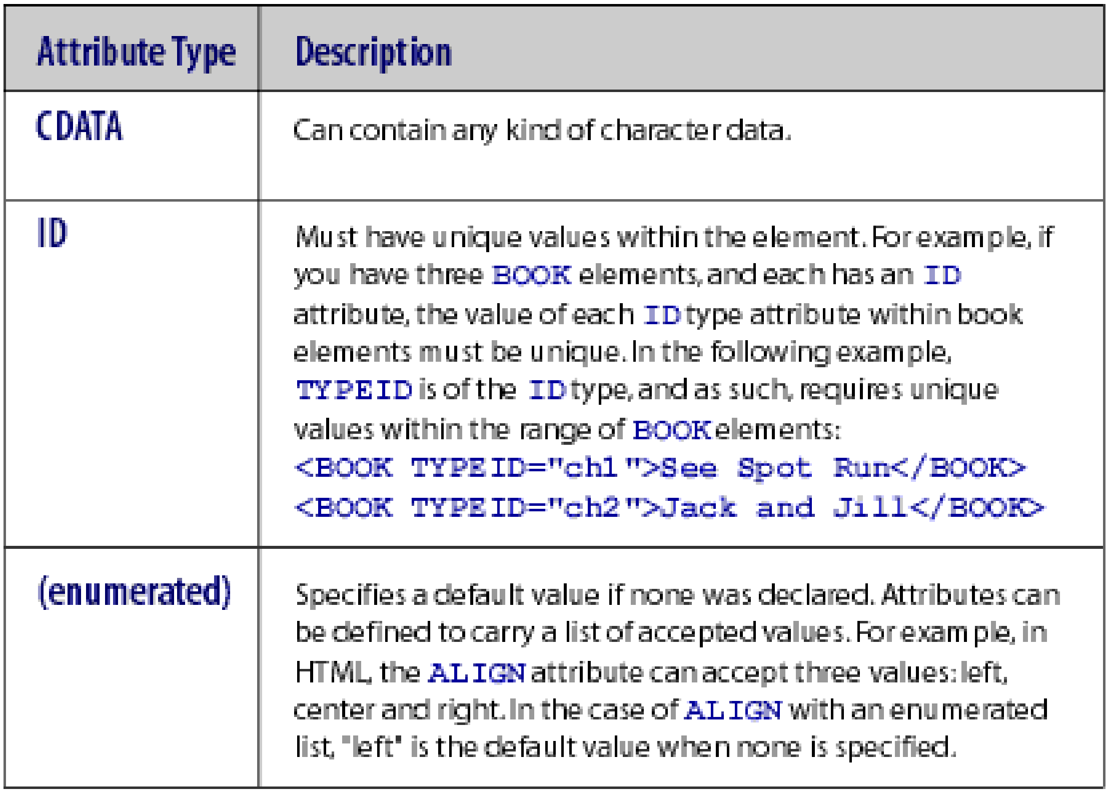
The `attributeName` represents the name of the attribute being declared. Attribute names can contain letters, numbers, dashes, underscores, periods and colons.

attributeType

The `attributeType` specifies the kind of attribute this will be. There are eight types you can use when defining attributes; three are commonly used.

attributeDefault

The `attributeDefault` represents whether the attribute is required (#REQUIRED), optional (#IMPLIED), and/or whether it has a fixed value (#FIXED `value`) or a specified default value from an enumerated list.



Attribute Type	Description
CDATA	Can contain any kind of character data.

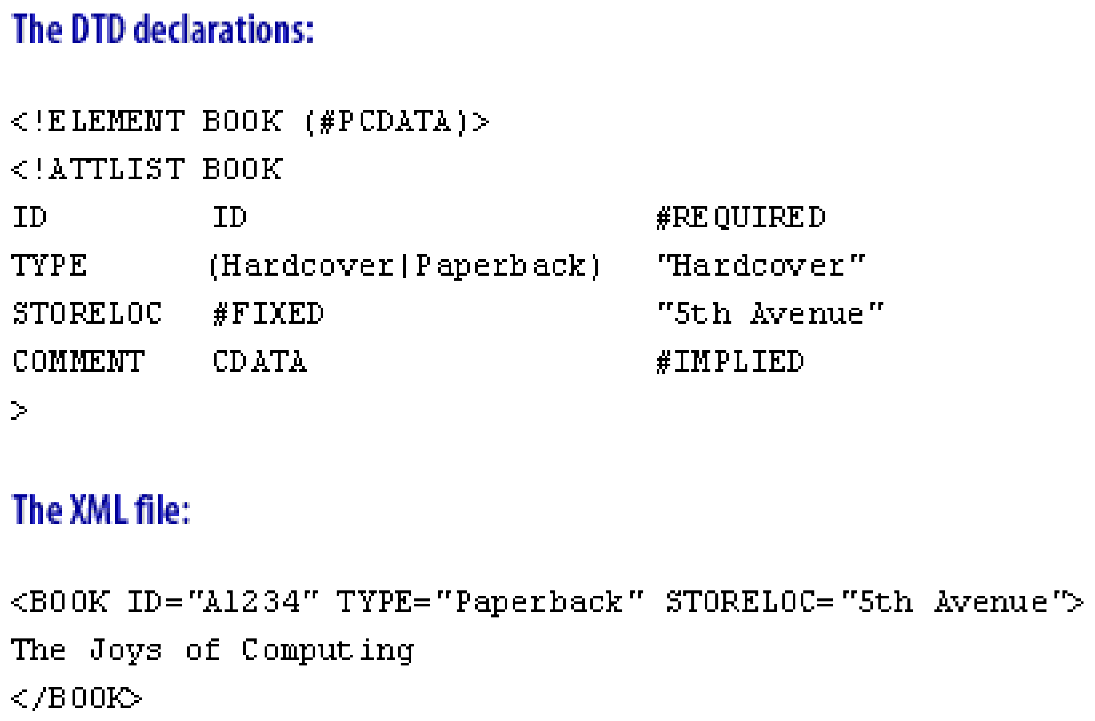
ID	Must have unique values within the element. For example, if you have three BOOK elements, and each has an ID attribute, the value of each ID type attribute within book elements must be unique. In the following example, TYPEID is of the ID type, and as such, requires unique values within the range of BOOK elements:
	`<BOOK TYPEID="ch1">See Spot Run</BOOK>`
	`<BOOK TYPEID="ch2">Jack and Jill</BOOK>`

(enumerated)	Specifies a default value if none was declared. Attributes can be defined to carry a list of accepted values. For example, in HTML, the ALIGN attribute can accept three values: left, center and right. In the case of ALIGN with an enumerated list, "left" is the default value when none is specified.

Module 5

Attribute-list declaration syntax

Example of attribute usage: (MouseOver transcript)



The DTD declarations:

```
<!ELEMENT BOOK (#PCDATA)>
<!ATTLIST BOOK
ID          ID                        #REQUIRED
TYPE        (Hardcover|Paperback)     "Hardcover"
STORELOC    #FIXED                    "5th Avenue"
COMMENT     CDATA                     #IMPLIED
>
```

The XML file:

```
<BOOK ID="A1234" TYPE="Paperback" STORELOC="5th Avenue">
The Joys of Computing
</BOOK>
```

ID The `ID` attribute had to be present and had to be of the `ID` type.

TYPE The `TYPE` attribute was needed because the book would otherwise have been identified internally as having a "Hardcover" when in fact it was a "Paperback" type of book.

STORELOC The `STORELOC` attribute was optional, and this file would have been valid with or without this attribute/value pair. However, had any value other than "5th Avenue" been specified as the `STORELOC` attribute's value, this file would have been invalid.

COMMENT

The `COMMENT` attribute was not required, and no comment was added to the BOOK element.

< !-- XML Fundamentals >

Module 5

Exercise

Attribute-list declaration syntax

Creating an attribute-list declaration

Objective

Create a declaration for an attribute name, type and usage.

Exercise scoring

Full credit for this exercise is 10 points. You'll submit your code to the course tutors.

Overview

In this exercise, you will practice creating a declaration for an attribute name, type and usage.

Instructions

1. Start with the file `employs.xml`. Add the `xml` and `!DOCTYPE` declarations at the top of the document. Include the version and encoding information, as well as the standalone information.
2. Add the appropriate element type declarations.
3. Create the attribute-list declaration for the `EMPLOYEE` element.
4. Create the attribute-list declaration for the `POSITION` element. Assume that the `TITLE` attributes can contain any character data, and that the possible `AREA` attributes include northern and western divisions. You can determine the attribute type and default information.

Starter files

The starter file for this exercise is called `employs.xml`. You'll find this file in the 05-03 folder of the course download available from the Resources page.

What to submit to the course tutors

In the text box below, cut and paste the modified contents of `employs.xml`. Click OK, I'm Done to submit your code to the course tutors.

OK, I'm Done

XML Fundamentals

Module 5

Exercise

Attribute-list declaration syntax

Creating an XML document from a DTD

Objective

Create an XML file based on an existing DTD.

Exercise scoring

Full credit for this exercise is 10 points. You'll submit your code to the course tutors.

Overview

In this exercise, you will create an XML file based on a provided DTD.

Instructions

1. Open the files `airline.dtd` and `flights.txt`. Note that the structure of the information in the `flights.txt` does not conform to the structure outlined in the file `airline.dtd`. Transform `flights.txt` into an XML document that contains only essential data. In other words, you would not want to state "There are two seats in business class left." Find a way to indicate the number of seats available in business class in the appropriate part of the structure by using only tags and the number 2. Delete text as you see fit. Pertinent information should remain visible on screen as well as be placed in attributes where appropriate.
2. Save flights.txt as `flights.xml`.
3. Create a `<!DOCTYPE>` reference in the XML file to the external file `airline.dtd`.
4. Move, add and delete text, tags and attributes values as needed to change the XML data file so that it fully conforms to the DTD.

Starter files

We've supplied the starter files for this exercise in the download file available from the Resources page. The specific files for this exercise can be found in the 05-03 folder.

What to submit to the course tutors

In the text box below, cut and paste the contents of `flights.xml`. Click OK, I'm Done to submit your code to the course tutors.

OK, I'm Done

Lesson 5
Objective

Parameter entities

Create entities to use within a DTD.

You can create entities to use within the DTD itself. To denote that an entity is to be used as a parameter entity, you must include the `%` character in the manner indicated.

The syntax for declaring a parameter entity is as follows:

```
<!ENTITY % parameterEntityName
parameterEntityDefinition >
```

The syntax for referencing a parameter entity is as follows:

```
%parameterEntityName
```

Suppose that you need to create an *attribute-list declaration* to be used for several different elements. Rather than define each element with the full listing of attributes, you can create an attribute-list entity and then reference it for each of the several elements. In the following example, three elements will have the same three attributes: `ID`, `MAKE` and `MODEL`. These three attributes are defined once in a parameter entity, and then used in the attribute-list declaration for each of the elements.

```
<!ENTITY % commonAtts
   "ID ID #REQUIRED
      MAKE CDATA #IMPLIED
      MODEL CDATA #IMPLIED">

<!ELEMENT CAR (#PCDATA)>
<!ATTLIST CAR %commonAtts>

<!ELEMENT COMPUTER (#PCDATA)>
<!ATTLIST COMPUTER %commonAtts>

<!ELEMENT MODEL (#PCDATA)>
<!ATTLIST MODEL %commonAtts>
```

In these declarations, each element of `CAR`, `COMPUTER` and `MODEL` shares the same common set of attributes defined by the parameter entity `commonAtts`.

Tip

As you may have noticed, the order in which you declare elements and attributes does not matter, so long as you accurately declare all that you will use before the XML document loads. Within an element declaration, the order of child elements matters, but the actual order of element declarations or attribute declarations does not matter.

Defining entities

Defining multiple characters

Imagine that you are working on a new software product code-named "Maui." You may need to create a host of marketing information for your Web site, and you would not want to manually search and replace the word "Maui" on each of 50 pages with the eventual product name.

So you could create an entity named `&NewProduct;` such that the entity could contain "Maui" until the name of the product is decided. You would then simply replace the string "Maui" in the entity declaration with the new product name. Once you do this, everywhere that `&NewProduct;` has been specified, the new product name will appear.

Module 5

Lesson 6

Objective

Using parsers

Explain how a parsing utility can help you verify the well-formedness and validity of XML documents.

A parsing tool helps you locate and correct errors in your XML and DTD files.

Parsing involves reducing that which is being parsed into its structural units. To parse English is to break down sentences into phrases and then nouns, verbs, pronouns and other elementary units of the language.

An XML parser is one that takes tagged text and breaks it down into its individual elements. XML parsers check for well-formedness constraints. Some parsers have the ability to return the parsed data in the form of a tree structure. Some parsers can also check the validity of an XML document by comparing the XML file to the associated internal or external DTD.

HTML browsers contain an HTML parser that breaks down and processes the tagged text, facilitating the display of HTML pages. Browsers that natively support XML will contain XML parsers within the source code.

Several parsers are available that provide simple and effective tools for evaluating both the well-formedness and validity of your XML documents. If you're using IE 5.x, the browser will parse documents for you if you load them into view.

Discussion

Would you like to hear what parsers other students are using? Do you have a parsing tool you'd recommend? If you have any questions or advice for other students, click the Discuss button on the toolbar and look in the folder Using XML parsers.

Exercise

If you're using IE 5.0, this optional exercise will allow you to parse an XML file in your browser.

Module 5

Parameter entities

Notation declarations

A *notation declaration* references a non-XML resource and specifies instructions for that resource in a non-XML context. For example, if you want to include a special media type, you would add a notation declaration that specifies the media type and perhaps an application on the system that could be used to render that media. The following code shows a fictitious notation declaration for the GIF image file type. In this example, the fictitious file name `gifview.exe` represents a file that enables the viewing of GIF images.

```
<!NOTATION GIF SYSTEM
"gifview.exe">
```

The following entity declaration references non-XML content:

```
<!ENTITY BulletGif SYSTEM
"bullet1.gif" NDATA GIF>
```

In this example, an entity referenced as `BulletGif`has been declared to reference the file `"bullet1.gif"` on the current system as non-character data `(NDATA)` and of the `GIF` type. If a notation for `GIF` appears in the file, the application can choose whether or not to use the notation reference for rendering that file.

Module 5

Lesson 7

Building the DTD

Module wrap-up

In this module, you learned how to create element, attribute-list, and entity declarations. With this understanding of DTD syntax, you wrote an XML document that conformed to a specified DTD.

This module discusses the following terms in relation to XML:

- *attribute-list declaration*
- *notation declaration*

Discussion

Do you have any questions or comments about the material in this module? If so, look in the folder Creating and using DTDs in the Discussion section, which you can reach by clicking the Discuss button in the toolbar. There's also a folder on using XML parsers.

Quiz

Click the Quiz button to test what you've learned in this module.

Using parsers

Parsing documents in a browser

Objective

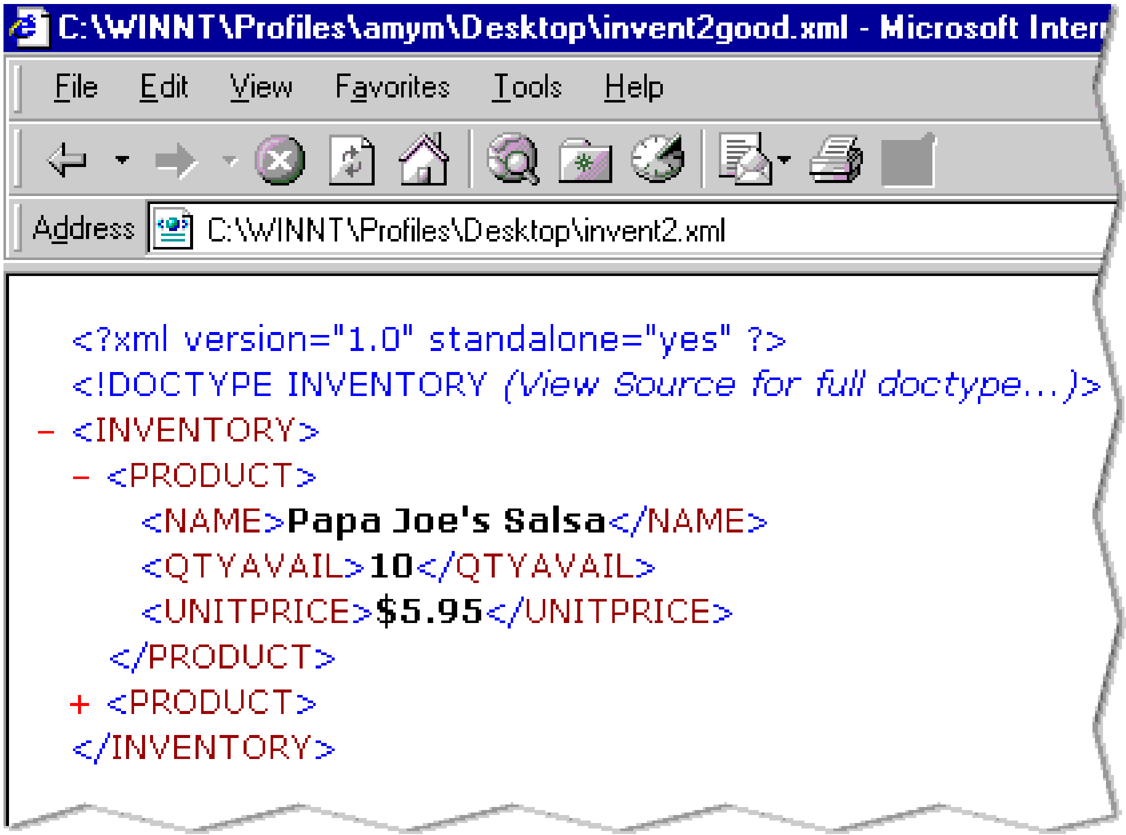
Parse an XML file using IE 5.0.

Exercise scoring

This exercise is not scored.

Instructions

Start with the file `invent2.xml`, which can be found in the 05-06 folder of the course download. Open this file in IE 5.0. If a file is both well-formed and valid, you will see the fully tagged document with clickable plus/minus buttons that allow you to expand and collapse the elements.


C:\WINNT\Profiles\amym\Desktop\invent2good.xml - Microsoft Inter
File
Edit
View
Favorites
Tools
Help
Address
C:\WINNT\Profiles\Desktop\invent2.xml

```
  <?xml version="1.0" standalone="yes" ?>
  <!DOCTYPE INVENTORY (View Source for full doctype...)>
- <INVENTORY>
  - <PRODUCT>
      <NAME>Papa Joe's Salsa</NAME>
      <QTYAVAIL>10</QTYAVAIL>
      <UNITPRICE>$5.95</UNITPRICE>
    </PRODUCT>
  + <PRODUCT>
  </INVENTORY>
```

If a document is not well-formed or not valid, IE 5.0 will give you information pertaining to the error and will not show the page until all the error have been corrected.

Make any necessary corrections to the source code. To practice your own XML skills, don't try and open this file in the browser until you have looked at it carefully and corrected all the bugs you can find.

When you believe your file is both valid and well-formed, open it in the browser. If you missed a bug, IE 5.0 will let you know!

Starter files

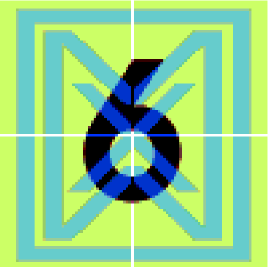
Remember, we've supplied the file for this exercise (`invent2.xml`) in the 05-06 folder of the course download available from the Resources page.

OK, I'm Done

Module 6

Lesson 1

XLink, XPointer, namespaces, and metadata



Module introduction

Many developers have been thrilled with the simplicity of creating a hyperlinked Web site using HTML. However, you have probably encountered one of the following situations:

- You wanted to link to just a portion of another page.
- You wanted to embed the full text from a link into the current page.
- You wanted to link to the third paragraph in the second section of the current page without having to create an anchor tag there.

XLink and XPointer are two specifications undergoing development at the W3C that address these issues and many more.

In addition, because tag names conflict between documents, there must be a way to give meaning to the tag names based on some other piece of information. To address this issue, a proposal for namespaces has come forward. Metadata proposals are also emerging as people try to find ways of codifying the data we give our computers about the data in our XML files.

In this module you'll learn about these related XML technologies.

Module learning objectives

After completing the module, you will have the skills and knowledge necessary to:

- Create simple links in XML
- Consider how extended links would work in XML
- Define link behavior with the show and actuate attributes
- Define the attribute list in the DTD
- Support links in XML with XPointer
- Use namespaces to establish clear naming conventions
- Specify the namespace from which elements derive meaning

XML Fundamentals

Module 5

Quiz

DTD syntax

Each question is worth one point. Some questions ask you to select **the best answer**, others ask you to select **all the correct answers**. To receive credit for questions asking for **all the correct answers**, you must select all the correct answers and **only** the correct answers.

1. According to SGML purists, it is better to embed useful information about elements between tags rather than within tags in XML. Such useful information includes classifications or descriptions that further define the specific behavior of the tag. This additional information about a given element is called:

   Please select **the best answer**.
   ○ A. An element type
   ○ B. An attribute
   ○ C. An input type
   ○ D. A tag value

2. You can declare multiple attributes in XML:

   Please select **all the correct answers**.
   □ A. By using the `ELEMENT` declaration
   □ B. By using the `ELEMENT-ATTRIBUTES` declaration
   □ C. By pointing the `ATTRIBUTE` reference to a separate file containing the attribute names
   □ D. By using the `ATTRIBUTE-LIST` declaration

3. Like HTML, XML allows you to include special characters not found on the keyboard or other characters that may be misinterpreted by using:

   Please select **the best answer**.
   ○ A. The `CHAR` keyword
   ○ B. Attribute values
   ○ C. Entities
   ○ D. Predefined character values

4. Entities are useful because they enable you to:

   Please select **all the correct answers**.
   □ A. Specify all your special characters within one declaration
   □ B. Replace one string of characters with another throughout a document

□ C. Define an entity once in a DTD then use it throughout a range of documents

□ D. Essentially use a replaceable field to represent actual text

5. You can create entities to use within the DTD itself by including the % character in the `ENTITY` declaration. This type of entity is called a:

Please select **the best answer**.

○ A. Special-character entity

○ B. Multiple-character entity

○ C. DTD entity

○ D. Document entity

○ E. Parameter entity

6. You can reference a non-XML resource and specify instructions for it in a non-XML context by using:

Please select **the best answer**.

○ A. A special media declaration

○ B. A resource declaration

○ C. A notation declaration

○ D. The `<?nxml>` statement

7. It is important to take note and carefully specify the order in which you declare:

Please select **the best answer**.

○ A. Elements and attributes in the DTD

○ B. Child elements within an element declaration

○ C. Child entities in the DTD

○ D. Attribute-list values

OK, I'm Done

XML Fundamentals

Module 6

Lesson 2

Objective

Linking in XML

Create simple links and consider extended links in XML.

XLink (formerly XLL) stands for *XML Linking Language*, a working draft now moving through the *W3C* approval process. XLink defines the creation and behavior of link elements in XML pages. The XLink draft defines a link as "an explicit relationship between two or more data objects or portions of data objects."

In HTML, a link is a simple means of expressing a unidirectional relationship between two items. A word might link to an image, or to another page, or to another word on the same page. The A element in HTML provides the function of determining the linking element (the text and/or image from which the link is launched) and the link target or destination.

In XLink, an `<A>` type link is called a *simple link*. The following code examples would accomplish the same function:

Transcript

What makes an element function as a link is the `xml:link` attribute. When the attribute value is `"simple"`, the element functions in the same manner as an HTML hyperlink.

Extended links

HTML has no equivalent for an extended link. The XLink draft proposes that this special link be made available. The extended link can point to several resources at once and can be used to create links from other documents to your own, even when you have no write privileges in that document. Because this is a new proposal, no examples exist to demonstrate the extended link.

Tip

A link that appears within the text is called an inline link. A link that does not appear to the user on the page is an out-of-line link. By default, all links are inline. An attribute named inline can have a value of false to indicate that the link is out-of-line.

XML Fundamentals

Module 6

Lesson 3

Objective

Link behavior

Define link behavior with the show and actuate attributes.

In the context of the HTML model, links are used to navigate from the link source to another page or file. XLink proposes new link behaviors. Two attributes have been proposed that could define certain link behaviors: `show` and `actuate`.

View Table

A `show` attribute has been proposed that would take one of three values: `embed`, `replace` or `new`. Click the View Table button to see a description of the behaviors represented by those values.

The `actuate` attribute is used to specify when the traversal of the link should take place. The two values proposed at this time are `auto` and `user`. If the value is specified as `auto`, the link will be followed, retrieved or embedded automatically. If the value is specified as `user`, the link will only be followed when the user activates the link.

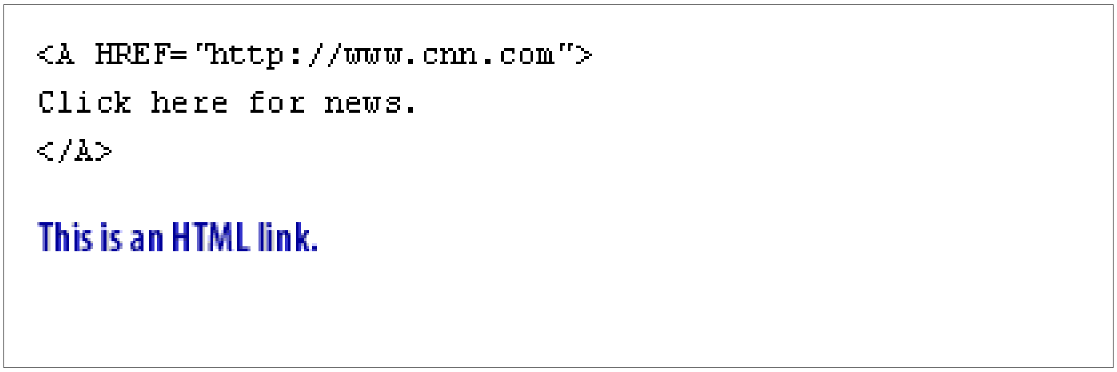
In addition to the individual and separate uses, the `show` and `actuate` attributes could be used together to provide a broad range of link behaviors.

Module 6

Linking in XML

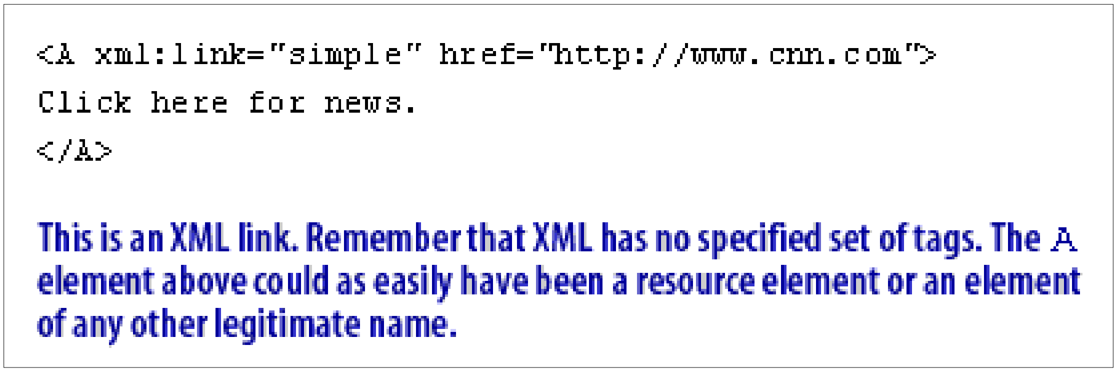
An example of simple links (FlipBook transcript)

The following code examples would accomplish the same function:



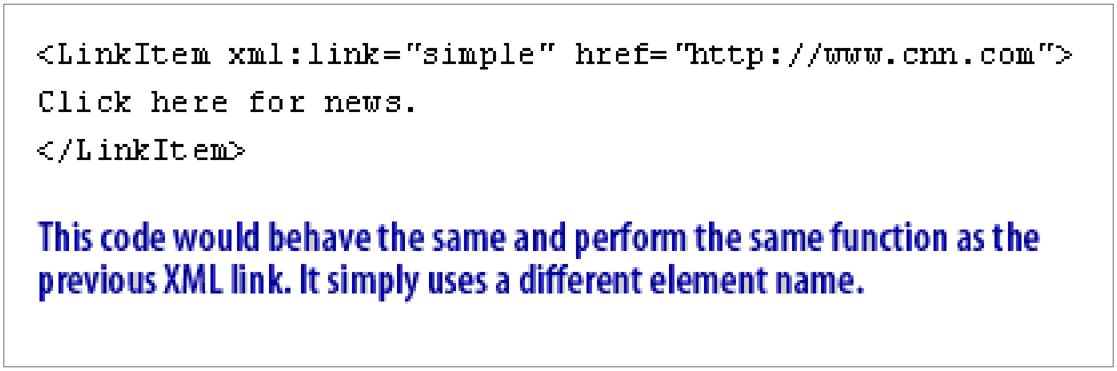
```
<A HREF="http://www.cnn.com">
Click here for news.
</A>
```

This is an HTML link.



```
<A xml:link="simple" href="http://www.cnn.com">
Click here for news.
</A>
```

This is an XML link. Remember that XML has no specified set of tags. The `A` element above could as easily have been a resource element or an element of any other legitimate name.



```
<LinkItem xml:link="simple" href="http://www.cnn.com">
Click here for news.
</LinkItem>
```

This code would behave the same and perform the same function as the previous XML link. It simply uses a different element name.

Module 6

Linking in XML

A theoretical example of using extended links

Here's a fictitious example to illustrate how extended links might work. Imagine that you want a CNN news story to point back to your Web page. Theoretically, if someone browsed your page, the extended links could be put in the browser application memory store. Then, if that person hit a particular story on the CNN site, the stored link to your original page might render in a separate window, in a link overlay, by means of a pop-up or some other method.

This arrangement would allow the user to return to the related story on your page. No specific behavior is currently defined.

Linking in XML

Out-of-line links

The following sample code represents an extended, out-of-line link:

```
<PROPOSAL xml:link="extended" inline="false">
   <locator href="item1" role="Essay"/>
   <locator href="item2" role="Rebuttal"/>
   <locator href="item3" role="Comparison"/>
</PROPOSAL>
```

XML Fundamentals

Module 6

Lesson 4
Objective

Links and the DTD

Define the attribute list in the DTD.

To create valid documents with links, you must define the attribute list in the DTD. Bearing in mind that this is subject to change, the following represents a valid way to define the simple link attributes:

```
<!ATTLIST elementName
   xml-link  CDATA #FIXED "simple"
   HREF   CDATA #REQUIRED
   TITLE  CDATA #IMPLIED
   INLINE  (TRUE|FALSE) "TRUE">
```

Audio

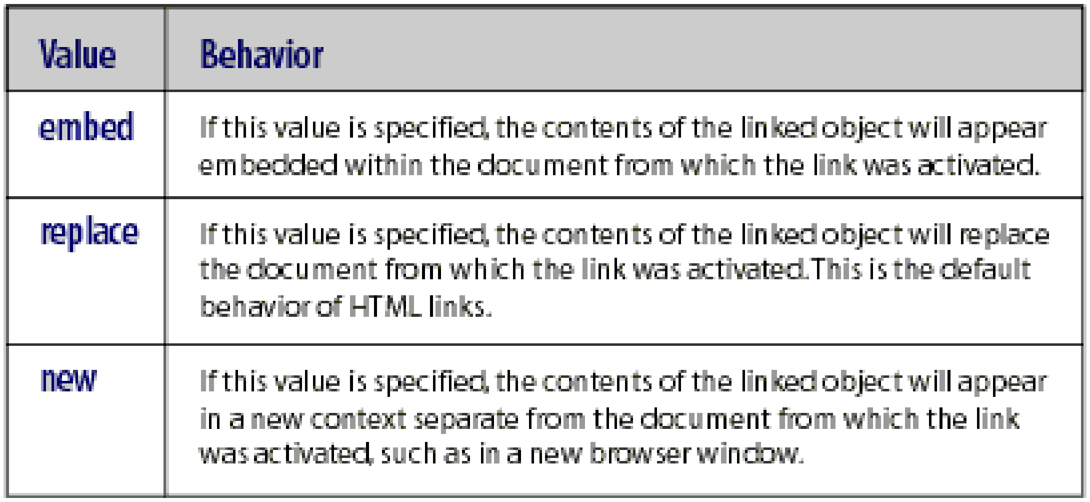
Transcript

Click the Audio button to hear why you shouldn't create documents and DTDs that use an HREF attribute without declaring XML-LINK.

Quiz

Quiz

Click the Quiz button to see what you've learned about links.



Value	Behavior
embed	If this value is specified, the contents of the linked object will appear embedded within the document from which the link was activated.
replace	If this value is specified, the contents of the linked object will replace the document from which the link was activated. This is the default behavior of HTML links.
new	If this value is specified, the contents of the linked object will appear in a new context separate from the document from which the link was activated, such as in a new browser window.

XML Fundamentals

Module 6

Lesson 5
Objective

XPointer

Support links in XML with XPointer.

X*Pointer* is a working draft at the W3C. It's designed to link to portions of a resource without having to link to the entire resource. Imagine that you want to link to a glossary definition without having to link to all the glossary definitions on a single page. XPointer is a proposal designed to address this issue.

In HTML, you can link to any other page, and even link to an internal reference within the page. HTML cannot, however, link to only a certain paragraph in another page, for example. To date, links have been an all-or-nothing proposition.

Under the XPointer proposal, you would be able to link to any element by referencing its name and/or position in the document tree structure. Consider the following example:

```
<?xml version="1.0"?>
<CATALOG>
   <BOOK>
      <TITLE>A Certain Justice</TITLE>
      <AUTHOR>P.D. James</AUTHOR>
      <YEAR-PUBLISHED>1998</YEAR-PUBLISHED>
      <ISBN>0375401091</ISBN>
   </BOOK>
   <BOOK>
      <TITLE>Ashworth Hall</TITLE>
      <AUTHOR>Anne Perry</AUTHOR>
      <YEAR-PUBLISHED>1997</YEAR-PUBLISHED>
      <ISBN>0449908445</ISBN>
   </BOOK>
   <BOOK>
      <TITLE>L.A. Confidential</TITLE>
      <AUTHOR>James Ellroy</AUTHOR>
      <YEAR-PUBLISHED>1997</YEAR-PUBLISHED>
      <ISBN>0446674249</ISBN>
   </BOOK>
   <BOOK>
      <TITLE>Shadow Woman</TITLE>
      <AUTHOR>Thomas Perry</AUTHOR>
      <YEAR-PUBLISHED>1997</YEAR-PUBLISHED>
      <ISBN>0679453024</ISBN>
   </BOOK>
</CATALOG>
```

If you want to link to the fourth book entry, you could do so using the following syntax:

```
href="catalog.xml#child(4,"BOOK")"
```

If you want to link to the author of the fourth book, you could use the following reference:

```
href="catalog.xml#child(4,"BOOK").child(1,"AUTHOR")"
```

You could also use this equivalent syntax:

```
href="catalog.xml#child(4,"BOOK").(1,"AUTHOR")"
```

XPointer proposes to define many more relationships than "parent" and "child" in terms of referencing other elements, including siblings, the next or previous sibling, and so forth.

If you are interested in tracking the developments with the working proposals for XLink and XPointer, visit the W3C's Web site. You can read the latest specifications and track the progress of these as they move toward becoming recommendations.

Exercise

n this exercise, you'll write code using the XLink and XPointer specifications.

XML Fundamentals

Module 6

Links and the DTD

Creating Links attributes in a DTD (Audio transcript)

“ In his book *XML: A Primer*, Simon St. Laurent gives the following strong advice when creating link attributes in a DTD:

"Avoid the temptation to create documents and DTDs that use an HREF attribute without declaring XML-LINK. Even though they might work in HTML browsers, XML-LINK compliant processing applications will ignore the HREF attribute if the XML-LINK attribute is missing." ”

XML Fundamentals

Module 6

Quiz

Link Basics

Each question is worth one point. Some questions ask you to select **the best answer**, others ask you to select **all the correct answers**. To receive credit for questions asking for **all the correct answers**, you must select all the correct answers and **only** the correct answers.

1.XLink is best described as:

Please select **the best answer**.

○ A. The W3C standard for XML Links
○ B. A specification that defines link elements in XML pages
○ C. An XML attribute that creates links and describes their behavior
○ D. The syntax used to create extended links in XML

2.In XLink, a simple link corresponds to the type of link used in HTML. A simple link can:

Please select **the best answer**.

○ A. Express a bi-directional relationship between two objects
○ B. Point to several resources at once
○ C. Create links from other documents to your own
○ D. Express a unidirectional relationship between two objects

3.An XML `inline` attribute with a value of `"false"` indicates:

Please select **the best answer**.

○ A. An out-of-line link; it appears within the text
○ B. An inline link; it does not appear on the page to the user
○ C. An out-of-line link; it does not appear on the page to the user
○ D. An inline link; it appears within the text
○ E. The appearance of a default link

4.XML proposes new behaviors for links, which are defined by attributes such as `show` and `actuate`. The `actuate` attribute can:

Please select **all the correct answers**.

□ A. Specify when a link will be followed
□ B. Specify a new window in which to display the linked object
□ C. Specify that the linked object will replace the document that activated the link

□ D. Specify that the linked object will appear embedded in the linking document

□ E. Specify that the link will be followed when activated by the user

5. To create a valid XML document with links, you must:

Please select **the best answer**.

○ A. Use the HREF attribute without declaring XML-LINK

○ B. Declare XML-LINK without using the HREF attribute

○ C. Use the HREF attribute and declare XML-LINK

○ D. Define the attribute list in the DTD

OK, I'm Done

XML Fundamentals

Module 6

Lesson 6 Objective

Namespaces

Use namespaces to establish clear naming conventions.

Common words sometimes have very different meanings across cultures. In American English, the word "lift" is used in a different context than in the United Kingdom, where "lift" is a common term for what Americans call an elevator.

In XML, tag and attribute names that make perfect sense to one person will make less sense to the next. Although we share a common language, the same word can have multiple meanings in multiple contexts. Consider the list of words below. What do you think these element names might be used for in an XML page?

- TITLE
- MOUSE
- ADDRESS
- ROLE

Even when the choice is obvious to one person, it may not be obvious and may even be contested by another. To resolve differences in naming conventions, the W3C is evaluating a proposal for XML *namespaces*.

Flip through the following frames to see why and how you might use namespaces:

Transcript

Tip

The use of namespace declarations becomes even more valuable when you are using two or more sets of tags and attributes defined by two or more entities. To declare multiple namespaces, just add multiple namespace declarations.

W3C Technical Reports and Publications

W3C Publications:

Recommendations · Proposed Recommendations · Working Drafts · Notes

Related:

Translations of W3C documents · Acknowledged Submissions · W3J

Member only: Newsletter · Newswire

Recent Recommendations

- 7 July: Mathematical Markup Language (MathML") 1.01 Specification. This is a revision of the 7 April 1998 release.
- 29 June: Associating Style Sheets with XML documents
- ...more Recommendations

Proposed Recommendations

- 3 March: Resource Description Framework (RDF) Schemas

Recent Working Drafts

- 19 July: Document Object Model (DOM) Level 2 Specification
- 13 July: Authoring Tool Accessibility Guidelines and Techniques for Authoring Tool Accessibility Guidelines
- 9 July: XML Pointer Language (XPointer)
- 9 July: XSL Transformations (XSLT)
- 9 July: XML Path Language (XPath)
- 6 July: Scalable Vector Graphics (SVG)
- 30 June: XML Fragment Interchange
- 25 June: CSS Namespace Enhancements (Proposal)

- 23 June: XML-Signature Requirements
- 23 June: Color Profiles for CSS3
- 23 June: Multi-column layout in CSS
- 23 June: Paged Media Properties for CSS3
- ... more Working Drafts

Recent Notes

- 24 June: CC/PP exchange protocol based on HTTP Extension Framework
- 24 June: WAP Binary XML Content Format
- 24 June: POIX: Point Of Interest eXchange Language Specification
- 16 June: Accessibility Features of CSS
- 14 June: XFA-template
- 14 June: XFA-FormCalc
- ... more Notes

Recommendations

Mathematical Markup Language (MathML″) 1.01 Specification

7 April 1998, revised 7 July 1999. Patrick Ion, Robert Miner, Stephen Buswell, Nico Poppelier

Associating Style Sheets with XML documents

29 June 1999, James Clark

Web Content Accessibility Guidelines 1.0

5 May 1999, Wendy Chisholm, Gregg Vanderheiden, Ian Jacobs

Resource Description Framework (RDF) Model and Syntax Specification

22 February 1999, Ora Lassila, Ralph R. Swick

WebCGM Profile

21 January 1999, David Cruikshank, John Gebhardt, Lofton Henderson, Roy Platon, Dieter Weidenbrueck

Namespaces in XML

14 January 1999, Tim Bray, Dave Hollander, Andrew Layman

Document Object Model (DOM) Level 1

1 October 1998, Vidur Apparao, Steve Byrne, Mike Champion, Scott Isaacs, Ian Jacobs, Arnaud Le Hors, Gavin Nicol, Jonathan Robie, Robert Sutor, Chris Wilson, Lauren Wood

Synchronized Multimedia Integration Language (SMIL) 1.0 Specification

15 June 1998, Philipp Hoschka

PICS Signed Labels (DSig) 1.0 Specification

27 May 1998, Yang-Hua Chu, Philip DesAutels, Brian LaMacchia, Peter Lipp

Cascading Style Sheets, level 2 (CSS2) Specification

12 May 1998, Bert Bos, Håkon Wium Lie, Chris Lilley, Ian Jacobs

Extensible Markup Language (XML) 1.0 Specification

10 February 1998, Tim Bray, Jean Paoli, C. M. Sperberg-McQueen

PICSRules 1.1 Specification

29 December 1997, Christopher Evans, Clive D.W. Feather, Alex Hopmann, Martin Presler-Marshall, Paul Resnick

HTML 4.0 Specification

18 December 1997, revised 24 April 1998. Dave Raggett, Arnaud Le Hors, Ian Jacobs

HTML 3.2 Reference Specification

14 January 1997, Dave Raggett

Cascading Style Sheets (CSS1) Level 1 Specification

17 December 1996, revised 11 January 1999, Håkon W. Lie and Bert Bos

PICS 1.1 Rating Services and Rating Systems -- and Their Machine Readable Descriptions

31 October 1996, Jim Miller, Paul Resnick and David Singer

PICS 1.1 Label Distribution -- Label Syntax and Communication Protocols

31 October 1996, Tim Krauskopf, Jim Miller, Paul Resnick and Win Treese

PNG (Portable Network Graphics) Specification

1 October 1996, various authors

Proposed Recommendations

Resource Description Framework (RDF) Schemas

3 March 1999, Dan Brickley, R.V. Guha

Working Drafts

The following Working Drafts have been submitted for review by W3C Members and other interested parties. These are ***draft documents*** and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to use W3C working drafts as reference material or to cite them as other than "work in progress".

Working Drafts in Last Call

A document in last call is to be reviewed by Working Groups that rely on or have a vested interest in the technology. The duration of the last call review period is listed in the status section of the document in review.

XHTML" 1.0: The Extensible HyperText Markup Language A Reformulation of HTML 4.0 in XML 1.0

5 May 1999, Steve Pemberton *et al*.

Working Drafts in development

Document Object Model (DOM) Level 2 Specification

19 July 1999, Lauren Wood, Vidur Apparao, Mike Champion, Joe Kesselman, Arnaud Le Hors, Philippe Le Hégaret, Tom Pixley, Jonathan Robie, Peter Sharpe, Chris Wilson

Authoring Tool Accessibility Guidelines and ***Techniques for Authoring Tool Accessibility Guidelines***

13 July 1999, Jutta Treviranus, Jan Richards, Ian Jacobs, Charles McCathieNevile

XML Pointer Language (XPointer)

9 July 1999, Steve DeRose, Ron Daniel

XSL Transformations (XSLT) Specification

9 July 1999, James Clark

XML Path Language (XPath)

9 July 1999, James Clark

Scalable Vector Graphics (SVG)

6 July 1999, Jon Ferraiolo

XML Fragment Interchange

30 June 1999, Paul Grosso, Daniel Veillard

CSS Namespace Enhancements (Proposal)

25 June 1999, Peter Linss

XML-Signature Requirements

23 June 1999, Joseph Reagle Jr.

Color Profiles for CSS3

23 June 1999, Tantek Çelik, Brad Pettit, Chris Lilley

Multi-column layout in CSS

23 June 1999, Håkon Wium Lie

Paged Media Properties for CSS3

23 June 1999, Robert Stevahn

Common Markup for Web Micropayment Systems

9 June 1999, Thierry Michel

Web Characterization Terminology & Definitions Sheet

24 May 1999, Brian Lavoie, Henrik Frystyk Nielsen

XML Information Set

17 May 1999, John Cowan, David Megginson

XML Schema Part 1: Structures

6 May 1999, David Beech, Scott Lawrence, Murray Maloney, Noah Mendelsohn, Henry S. Thompson

XML Schema Part 2: Datatypes

6 May 1999, Paul V. Biron, Ashok Malhotra

Extensible Stylesheet Language (XSL)

21 April 1999, Sharon Adler, Anders Berglund, Jeff Caruso, Stephen Deach, Alex Milowski, Steve Zilles

Platform for Privacy Preferences (P3P) Specification

7 April 1999, Massimo Marchiori, Joseph Reagle

Modularization of XHTML"

6 April 1999, Murray Altheim, Daniel Austin, Frank Boumphrey, Sam Dooley, Shane McCarron, Ted Wugofski

User Agent Accessibility Guidelines

31 March 1999, Jon Gunderson, Ian Jacobs

Ruby

22 March 1999, Marcin Sawicki

International Layout in CSS

22 March 1999, Marcin Sawicki

Character Model for the World Wide Web

25 February 1999, Martin J. Dürst

Scalable Vector Graphics (SVG) Requirements

29 October 1998, Jon Ferraiolo

A P3P Preference Exchange Language (APPEL)

14 August 1998, Marc Langheinrich

SMUX Protocol Specification

10 July 1998, Jim Gettys, Henrik Frystyk Nielsen

HTTP-ng Web Interfaces

10 July 1998, Dan Larner

HTTP-ng Binary Wire Protocol

10 July 1998, Bill Janssen

HTTP-ng Architectural Model

10 July 1998, Bill Janssen, Henrik Frystyk Nielsen, Mike Spreitzer

Requirements for String Identity Matching and String Indexing

10 July 1998, Martin J. Dürst

XSL Requirements Summary

11 May 1998, Norman Walsh

Short- and Long-Term Goals for the HTTP-NG Project

27 March 1998, Michael Spreitzer, Henrik Frystyk Nielsen

XML Linking Language (XLink)

3 March 1998, Eve Maler, Steve DeRose

PEP Specification: an Extension Mechanism for HTTP

21 November 1997, Henrik Frystyk Nielsen, Dan Connolly, Rohit Khare, Eric Prud'hommeaux

P3P Architecture Working Group: General Overview of the P3P Architecture

22 October 1997, Joseph Reagle, Martin Presler-Martin, Melissa Dunn, Philip DesAutels, Lorrie Cranor, Mark Ackerman

P3P Vocabulary Working Group: Grammatical Model and Data Design Model

14 October 1997, Mark Ackerman, Lorrie Cranor, Philip DesAutels, Melissa Dunn, Joseph Reagle, Upendra Shardanand

Positioning HTML Elements with Cascading Style Sheets

19 August 1997, Robert Stevahn, Scott Furman, Scott Isaacs

Web Fonts

21 July 1997, Brad Chase, Chuck Rowe, Chris Lilley, David Meltzer, Glen Rippel, Håkon Lie, Randy Polen, Robert Stevahn, Steve Zilles

Aural Cascading Style Sheets (ACSS) Specification

30 June 1997, Chris Lilley, T.V. Raman

CSS Printing Extensions

26 June 1997, Hakon Wium Lie, Robert Stevahn, Stephen Waters

Digital Signature Label Architecture

10 January 1997, Rohit Khare

Selecting Payment Mechanisms Over HTTP

6 January 1997, Don Eastlake, Rohit Khare, Jim Miller

Notes

The following are W3C Notes that have been published at the Director's discretion. A Note does not represent commitment by W3C to pursue work related to the Note. Notes that were the result of Acknowledged Member Submissions are indicated.

CC/PP exchange protocol based on HTTP Extension Framework

24 June 1999, Hidetaka Ohto, Johan Hjelm

WAP Binary XML Content Format

24 June 1999, Bruce Martin, Bashar Jano

POIX: Point Of Interest eXchange Language Specification

24 June 1999, Hiroyuki Kanemitsu, Tomihisa Kamada

Accessibility Features of CSS

16 June 1999, Ian Jacobs, Judy Brewer

XFA-Template

14 June 1999, Gavin F. McKenzie et al. (XFA Submission)

XFA-FormCalc

14 June 1999, Gavin F. McKenzie et al. (XFA Submission)

Web Architecture: Describing and Exchanging Data

7 June 1999, Tim Berners-Lee, Dan Connolly, Ralph R. Swick

XML Canonicalization Requirements

5 June 1999, James Tauber, Joel Nava

Editing the Web: Detecting the Lost Update Problem Using Unreserved Checkout

10 May 1999, Henrik Frystyk Nielsen, Daniel LaLiberte

Techniques for Web Content Accessibility Guidelines 1.0

5 May 1999, Wendy Chisholm, Gregg Vanderheiden, Ian Jacobs

Web Characterization: From working group to activity

19 March 1999, Jim Pitkow, Johan Hjelm, Henrik Frystyk Nielsen

HTML 4.0 Guidelines for Mobile Access

15 March 1999, Tomihisa Kamada, Takuya Asada, Masayasu Ishikawa, Shin'ichi Matsui

XML XLink Requirements Version 1.0

24 February 1999, Steven J. DeRose

XML XPointer Requirements Version 1.0

24 February 1999, Steven J. DeRose

XPointer-Information Set Liaison Statement Version 1.0

24 February 1999, Steven J. DeRose

Synchronized Multimedia Modules based upon SMIL 1.0

23 February 1999, Patrick Schmitz, Ted Wugofski, Warner ten Kate

XML Information Set Requirements

18 February 1999, David Megginson

XML Schema Requirements

15 February 1999, Ashok Malhotra, Murray Maloney

Personalized Information Description Language (PIDL)

9 February 1999, Yuichi Koike, Tomonari Kamba, Marc Langheinrich (PIDL Submission)

User Agent Authentication Forms

3 February 1999, Scott Lawrence, Paul Leach (Forms Submission)

Universal Commerce Language and Protocol (UCLP)

20 January 1999 (UCLP Submission)

Document Definition Markup Language (DDML) Specification, Version 1.0

19 January 1999, Ronald Bourret, John Cowan, Ingo Macherius, Simon St. Laurent (DDML Submission)

List of suggested extensions to CSS

10 December 1998, Bert Bos

DrawML Specification

3 December 1998, Håkan Lothigius (DrawML Submission)

Composite Capability/Preference Profiles (CC/PP): A user side framework for content negotiation

30 November 1998, Franklin Reynolds, Johan Hjelm, Spencer Dawkins, Sandeep Singhal

XML Fragment Interchange Requirements, Version 1.0

23 November 1998, Paul Grosso

Simple Tree Transformation Sheets 3 (STTS3)

11 November 1998, Daniel Glazman (STTS3 Submission)

The Platform for Privacy Preferences

6 November 1998, Lorrie Faith Cranor, Joseph Reagle

WAP Forum - W3C Cooperation White Paper

30 October 1998, Johan Hjelm, Bruce Martin, Peter King

The Information and Content Exchange (ICE) Protocol

26 October 1998, Neil Webber, Conleth O'Connel, Bruce Hunt, Rick Levine, Laird Popkin, Gord Larose (ICE Submission)

HTML Components

23 October 1998, Chris Wilson (HTML Components Submission)

Extensible Forms Description Language (XFDL) 4.0

2 September 1998, John Boyer, Tim Bray, Maureen Gordon (XFDL Submission)

Schema for Object-oriented XML

30 September 1998, Matt Fuchs, Murray Maloney, Alex Milowski (SOX Submission)

Timed Interactive Multimedia Extensions for HTML (HTML+TIME)

18 September 1998, Patrick Schmitz, Jin Yu, Peter Santangeli (HTML+TIME Submission)

Using XSL and CSS together

11 September 1998, Håkon Lie, Bert Bos

XML-QL: A Query Language for XML

19 August 1998, Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, Dan Suciu (XML-QL Submission)

Document Content Description for XML (DCD)

10 August 1998, Tim Bray, Charles Frankston, Ashok Malhotra (DCD Submission)

A Discussion of the Relationship Between RDF-Schema and UML

4 August 1998, Walter W. Chang

P3P Guiding Principles

21 July 1998, Lorrie Faith Cranor

Displaying SMIL Basic Layout with a CSS2 Rendering engine

20 July 1998, Philipp Hoschka, Chris Lilley

Design of HTTP-ng Testbed

10 July 1998, Daniel Veillard

Action Sheets: A Modular Way of Defining Behavior for XML and HTML

19 June 1998, Vidur Apparao, Brendan Eich, Ramanathan Guha, Nisheeth Ranjan (Action Sheets Submission)

Signed Document Markup Language (SDML)

19 June 1998, Jeff Kravitz (SDML Submission)

Hyper Graphics Markup Language (HGML)

19 June 1998, Mike Evans, Steven Furnell, Andy Phippen, Paul Reynolds, Neil Lilly, John Hammac

Statement on the Intent and Use of PICS: Using PICS Well

1 June 1998, Joseph Reagle, Daniel J. Weitzner

Vector Markup Language (VML)

13 May 1998, Brian Mathews, Daniel Lee, Brian Dister, John Bowler, Howard Cooperstein, Ajay Jindal, Tuan Nguyen, Peter Wu, Troy Sandal (VML Submission)

WebBroker: Distributed Object Communication on the Web

11 May 1998, John Tigue, Jon Lavinder (WebBroker Submission)

XML in HTML Meeting Report

11 May 1998, Dan Connolly, Lauren Wood

Precision Graphics Markup Language (PGML)

10 April 1998, Nabeel Al-Shamma, Robert Ayers, Richard Cohn, Jon Ferraiolo, Martin Newell, Roger K. de Bry, Kevin McCluskey, Jerry Evans (PGML Submission)

Web Schematics on the World Wide Web

31 March 1998, David Duce, Bob Hopgood (Web Schematics Submission)

P3P Protocol and Data Transport Working Group Draft White Paper

24 March 1998, Philip DesAutels, Steve Lucas, Joseph Reagle

Primary Language in HTML

13 March 1998, M.T. Carrasco Benitez

XML Linking Language (XLink) Design Principles

3 March 1998, Eve Maler, Steve DeRose

Web Architecture: Extensible Languages

10 Februray 1998, Tim Berners-Lee, Dan Connolly

Compact HTML for Small Information Appliances

9 February 1998, Tomihisa Kamada (Compact HTML Submission)

Voice Browsers

28 January 1998, Dave Raggett, Or Ben-Nattan

Name Spaces in XML

19 January 1998, Tim Bray, Dave Hollander, Andrew Layman

XML Data

5 January 1998, Andrew Layman, Edward Jung, Eve Maler, Henry S. Thompson, Jean Paoli, John Tigue, Norbert H. Mikula, Steve De Rose (XML Data Submission)

HTML Threading: Conventions for use of HTML in email

5 January 1998, Eric Berman, Pete Resnick, Nick Shelness (HTML Threading Submission)

Client-Specific Web Services by Using User Agent Attributes

30 December 1997, Tomihisa Kamada, Tomohiko Miyazaki

Comparison of SGML and XML

15 December 1997, James Clark

Introduction to RDF Metadata

13 November 1997, Ora Lassila

W3C Data Formats

29 October 1997, Tim Berners-Lee

Amaya Testbed Environments: the Java and ILU versions of Amaya

28 October 1997, Daniel Veillard

PICS Extension for HTTP Cookies

23 October 1997, Daniel Jaye

Simple Tree Transformation Sheets 2

17 October 1997, Daniel Glazman (STTS2 Submission)

Internet Privacy Working Group Privacy Practices for the Web

17 October 1997, Internet Privacy Working Group (IPWG Submission)

Designing a Social Protocol: Lessons Learned from the Platform for Privacy Preferences

2 October 1997, Lorrie Faith Cranor, Joseph Reagle Jr.

Web Interface Definition Language (WIDL)

22 September 1997, Phillip Merrick, Charles Allen (WIDL Submission)

Date and Time Formats

15 September 1997, Misha Wolf, Charles Wicksteed (Date+Time Submission)

A Proposal for XSL

27 August 1997, Sharon Adler, Anders Berglund, James Clark, Istvan Cseri, Paul Grosso, Jonathan Marsh, Gavin Nicol, Jean Paoli,

David Schach, Henry S. Thompson, Chris Wilson (XSL Submission)

Generic Diff Format Specification

21 August 1997, Arthur van Hoff, Jonathan Payne (GDIFF Submission)

The HTTP Distribution and Replication Protocol

21 August 1997, Arthur van Hoff, Jonathan Payne, Mark Hapner, Steve Carter, Milo Medin (DRP Submission)

CSS: Potential Enhancements

19 August 1997, Mike Wexler

The Open Software Description Format (OSD)

13 August 1997, Arthur van Hoff, Hadi Partovi, Tom Thai (OSD Submission)

W3C Activities Related to the "Global Information Networks" Ministerial Conference Meeting in Bonn, Germany

11 July 1997, Sally Khudairi, Josef Dietl

W3C Activities Related to the US "Framework for Global Electronic Commerce"

6 July 1997, Joseph Reagle

Meta Content Framework using XML

24 June 1997, R. V. Guha, Tim Bray (XML Meta Content Submission)

Network Performance Effects of HTTP/1.1, CSS1, and PNG

24 June 1997, Henrik Frystyk Nielsen, Jim Gettys, Anselm Baird-Smith, Eric Prud'hommeaux, Håkon W. Lie, Chris Lilley

Use of CGM as a Scalable Graphics Format

18 June 1997, Roy Platon and Chris Lilley

Navigation through LINK elements in HTML

15 June 1997, Daniel Glazman (Navigation through LINK Submission)

Privacy and Profiling on the Web

2 June 1997, Melissa Dunn, James Gwertzman, Andrew Layman, Hadi Partovi (Web Privacy Submission)

Proposal for an Open Profiling Standard

2 June 1997, Pat Hensley, Max Metral, Upendra Shardanand, Donna Converse, Mike Myers (OPS Submission)

Implementation of OPS Over HTTP

2 June 1997, Pat Hensley, Max Metral, Upendra Shardanand, Donna Converse, Mike Myers

Standard Practices for OPS Systems

2 June 1997, Pat Hensley, Max Metral, Upendra Shardanand, Donna Converse, Mike Myers

White Paper: Joint Electronic Payment Initiative

19 May 1997, Eui-Suk Chung and Daniel Dardailler

Proposal for a Handheld Device Markup Language

9 May 1997, Tim Hyland (HDML Submission)

Handheld Device Markup Language Specification

9 May 1997, Peter King, Tim Hyland

PICS-NG Metadata Model and Label Syntax

14 May 1997, Ora Lassila

HTML DTD with support for Style Sheets

21 April 1997, Dave Raggett

Channel Definition Format (CDF)

10 March 1997, Castedo Ellerman (CDF Submission)

Web Collections using XML

7 March 1997, Alex Hopmann, Scott Berkun, George Hatoun, Yaron Goland, Thomas Reardon, Lauren Antonoff, Eric Berman (XML WebCollections Submission)

An Introduction to Amaya

20 February 1997, Vincent Quint and Irène Vatton

HTTP-based Distributed Content Editing Scenarios

13 December 1996, Ora Lassila

Imagemapped Images and Image-Incapable User Agents

26 November 1996, Ian Graham

Resource Description Messages (RDM)

24 July 1996, Darren Hardy

Frame-based layout via Style Sheets

8 June 1996, Bert Bos, David Raggett and Håkon W. Lie

Jigsaw: An Object Oriented Server

7 June 1996, Anselm Baird-Smith

Jigsaw performance evaluation

7 June 1996, Anselm Baird-Smith

HTML predefined icon-like symbols

4 June 1996, Bert Bos

A Proposed Convention for Embedding Metadata in HTML

2 June 1996, Reported by Stuart Weibel

The ILU Requester: Object Services in HTTP Servers

7 March 1996, Paul Everitt

HTML Dialects: Internet Medial and SGML Document Types

6 March 1996, Dan Connolly

Session Identification URI

21 February 1996, Phillip M. Hallam-Baker, Dan Connolly

Notification for Proxy Caches

21 February 1996, Phillip M. Hallam-Baker

Extended Log File Format

21 February 1996, Phillip M. Hallam-Baker, Brian Behlendorf

A Lexical Analyzer for HTML and Basic SGML

8 February 1996, Dan Connolly

Micro Payment Transfer Protocol (MPTP)

22 November 1995, Phillip M. Hallam-Baker

Giving Information About Other Resources in HTML

20 November 1995, Tim Berners-Lee, David Raggett

Proposals for Gathering Consumer Demographics

6 November 1995, Dan Connolly

"Character Set" Considered Harmful

2 May 1995, Dan Connolly

About W3C Technical Reports

As described in the Process Document, W3C publishes several types of technical reports:

Notes

A Note is a dated, public record of an idea, comment, or document. A Note does not represent commitment by W3C to pursue work related to the Note.

Working Drafts

A Working Draft represents work in progress and a commitment by W3C to pursue work in this area. A Working Draft does not imply consensus by a group or W3C.

Proposed Recommendations

A Proposed Recommendation is work that (1) represents consensus within the group that produced it and (2) has been proposed by the Director to the Advisory Committee for review.

Recommendations

A Recommendation is work that represents consensus within W3C and has the Director's stamp of approval. W3C considers that the ideas or technology specified by a Recommendation are appropriate for widespread deployment and promote W3C's mission.

Specifications developed within W3C must be formally approved by the Membership. Consensus is reached after a specification has proceeded through the review stages of Working Draft, Proposed Recommendation, and Recommendation.

Related:

- Translations of W3C Documents. This page provides information about translations of W3C documents, the status of these documents, and how translators can help.
- How to Order W3C Publications
- Contributors to W3C technical reports are required to complete a release form.
- Consult the W3C document distribution and use policy.
- FAQ about W3C documents, including information about translations, mirroring, copying, etc.

World Wide Web Journal (W3J)

The **World Wide Web Journal,** published quarterly by O'Reilly & Associates from 1995 through Winter 1998, was the official journal of the W3C. Please contact O'Reilly & Associates for information regarding availability of back issues.

Tim Berners-Lee, W3C Director

Please send inquiries about this page to Webmaster

$Date: 1999/07/19 22:49:06 $

Exercise

XPointer

XLink and XPointer

Objective

Practice writing code using the XLink and XPointer specification.

Exercise Scoring

Full credit for this exercise is 12 points (4 points for each step). You'll submit your code to the course tutors.

Starter files

To help you complete this exercise, we've supplied a starter file in the 06-05 folder of the course download available from the Resources page.

Instructions

Consider the following table of contents from Cicero Lane's electronic book *Photos from the Road*. This fictitious book consists of various links from the table of contents to different chapters, photos and essays:

```
<ROADPHOTOS-TOC>
Cover Photo
Introduction by Cicero Lane
Chapter 1: Los Angeles
   Venice Beach
   LA County Museum of Art
   Disneyland
Chapter 2: Paris
   Eiffel Tower
   Notre Dame Cathedral
   Le Louvre
Chapter 3: Marshall Islands
   Kwajalein
   Scuba Diving
   WWII Shipwrecks
</ROADPHOTOS-TOC>
```

1. Suppose you are creating an XML file for the Photos from the Road table of contents. Write the code to create simple links to the cover photo, introduction, and Chapter 1 page. Assume the Los Angeles County Museum of Art page links to the LACMA Web site. Write the code to create such a link. Do the same for Disneyland. (Use "lacma.org" and "disneyland.com" in your URLs.) Consider the following points as you code:
   - Because XML has no specified set of tags, you can use element names other than `<A>` to determine your links, as long as you use the `xml:link` attribute. Choose descriptive element names for your links.
   - Consider the item to which you are linking: Is it a page or an image? Choose an appropriate name and file type for each linked item.

   In the text box below, cut and paste your code, and continue on to the next step.

2. For this XML document with links to be valid, you must define the attribute list in the DTD. This syntax is also subject to change, however you can model your syntax after the example presented in the module. Write the code to define the simple link attributes for the first two links you created in Step 1. Consider the following points as you code:
   - Use proper syntax for declaring an attribute list.
   - Remember to specify the name, type and default for each element.

   In the text box below, cut and paste your code, and continue on to the next step.

3. The XPointer specification proposes a scheme to allow you to link to a specific element within a page. Suppose you wanted to link to the third entry in Chapter 2 of the Photos from the Road book. Inferring the document tree structure from the table of contents as shown above, write the single line of code that would link to the LE LOUVRE entry using the XPointer syntax example from this module. Consider the following points as you code:

- The XPointer syntax requires you to reference the root element, then specify child elements by counting to the one you want (e.g., first, second, third and so forth). To specify the exact location you want, you might need to refer to a child of a child element.
- Again, this specification is under discussion, and therefore the syntax is subject to change.

In the text box below, cut and paste your code. When you're ready to submit all your answers to the course tutors, click the OK, I'm Done button.

OK, I'm Done

Module 6

Lesson 7

Objective

Metadata

Specify the namespace from which elements derive meaning.

M*etadata* is data about data. In a database, for example, you could think of the column headers as metadata that describe the contents of each field. In a word processing document, the information pertaining to the author, the subject, the date created and the date last edited is all considered metadata--data about the data within the document itself.

If XML is all about data, then what defines the metadata about the XML data? Netscape submitted a proposal for "Meta Content Framework Using XML" to the W3C in June 1997. That proposal has evolved into the Resource Description Framework (RDF) specification currently under discussion as a working draft at the W3C.

A resource, according to the draft specification, is anything that can be indicated by a Uniform Resource Identifier (URI). The Resource Description Framework is an attempt to codify not the descriptions themselves but the way resource descriptions are described between machines. Just as XML is a language for creating other languages, RDF is a language for specifying resource descriptions.

One resource description scheme gaining attention of late is the Dublin Core initiative. Experts from library, museum, research and networking communities have been working together to create a metadata scheme for describing resources.

The Dublin Core 15-element set breaks down into the following three categories: Content, Intellectual Property and Instantiation. This table indicates which elements relate to each category:

Category	Elements
Content	Title Subject Description Source Language Relation Coverage
Intellectual Property	Creator Publisher Contributor Rights
Instantiation	Date Type Format Identifier

If you wanted to tie your own elements to Dublin Core elements in your XML pages, you could specify that the Dublin Core is the namespace from which these particular elements derive meaning.

Bear in mind that, like everything else in this module, the Dublin Core is an emerging specification still undergoing development. It is likely to change with time.

Exercise

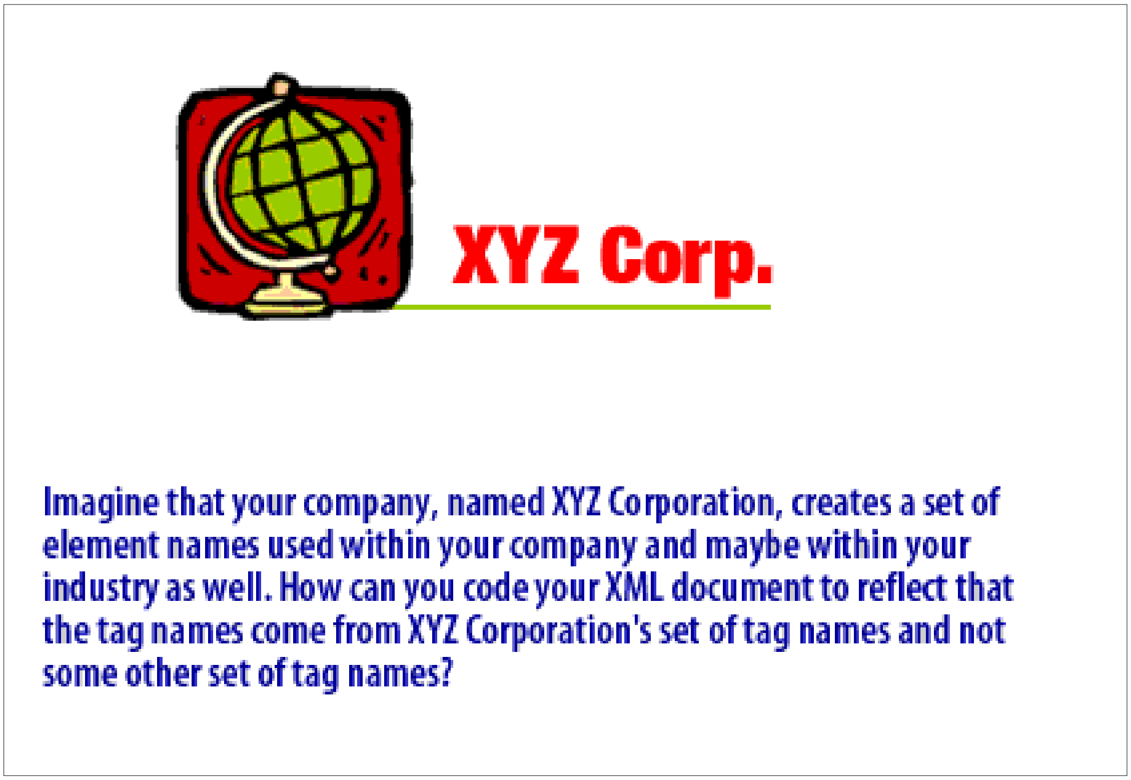
In this exercise, you will practice writing code to use XML namespaces and metadata.

Module 6

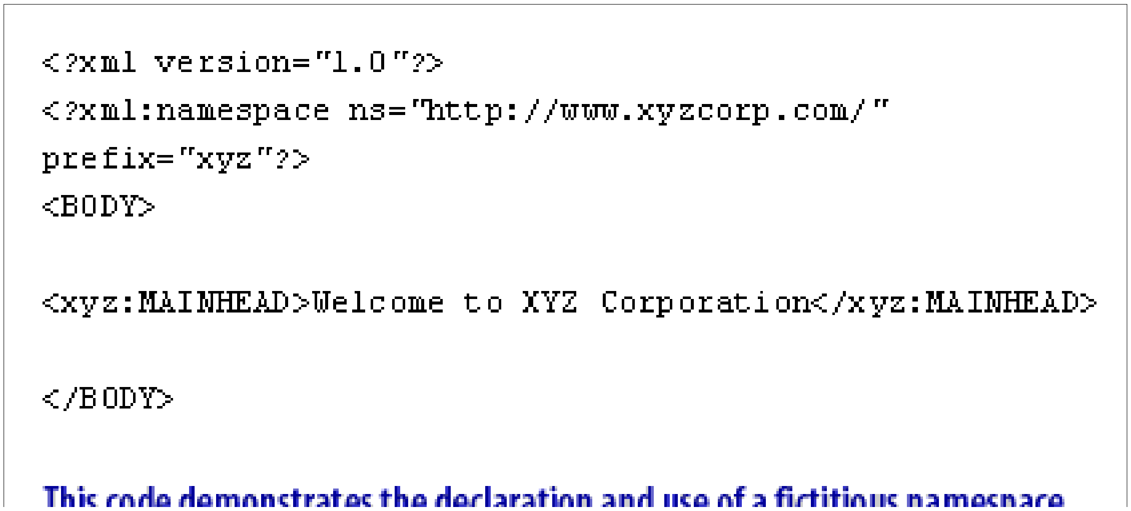
Namespaces

Examples of using namespaces (FlipBook transcript)

An example of why and how you might use namespaces:

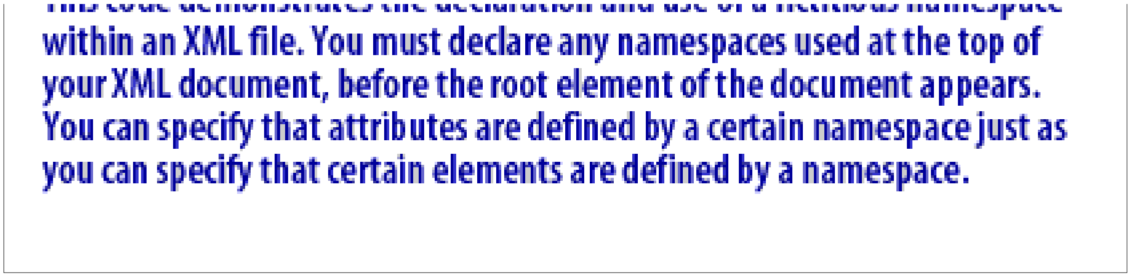


Imagine that your company, named XYZ Corporation, creates a set of element names used within your company and maybe within your industry as well. How can you code your XML document to reflect that the tag names come from XYZ Corporation's set of tag names and not some other set of tag names?

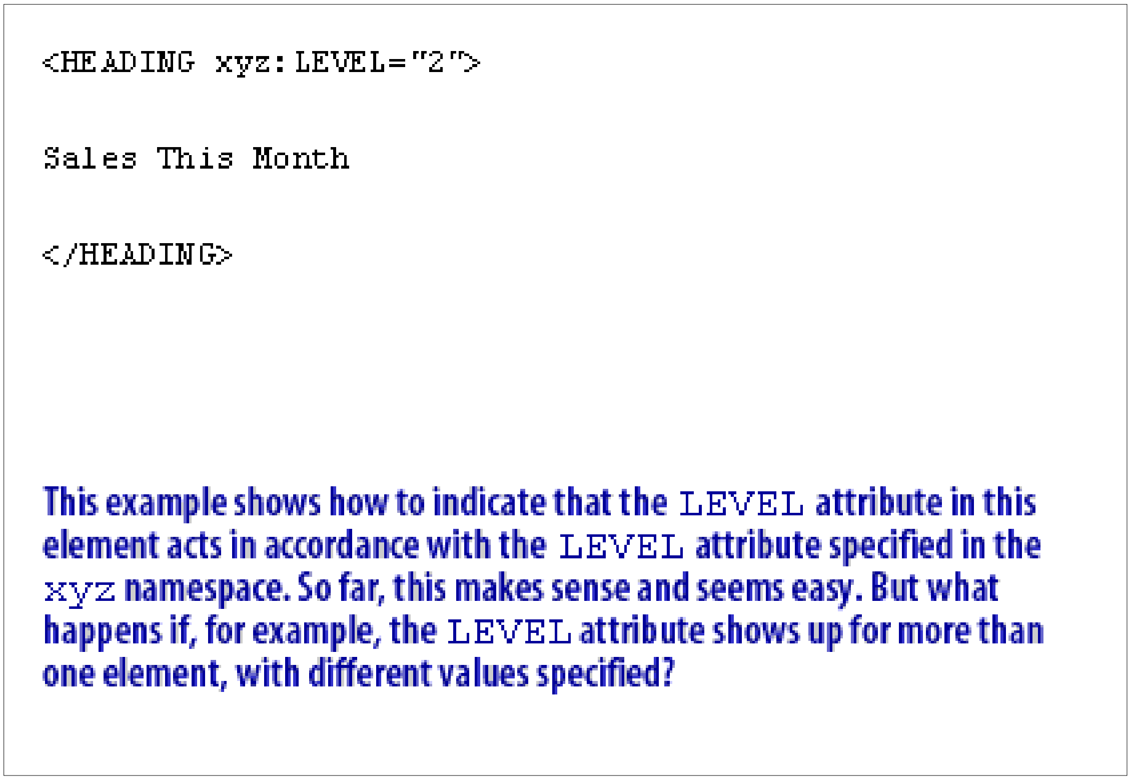


```
<?xml version="1.0"?>
<?xml:namespace ns="http://www.xyzcorp.com/"
prefix="xyz"?>
<BODY>

<xyz:MAINHEAD>Welcome to XYZ Corporation</xyz:MAINHEAD>

</BODY>
```

This code demonstrates the declaration and use of a fictitious namespace

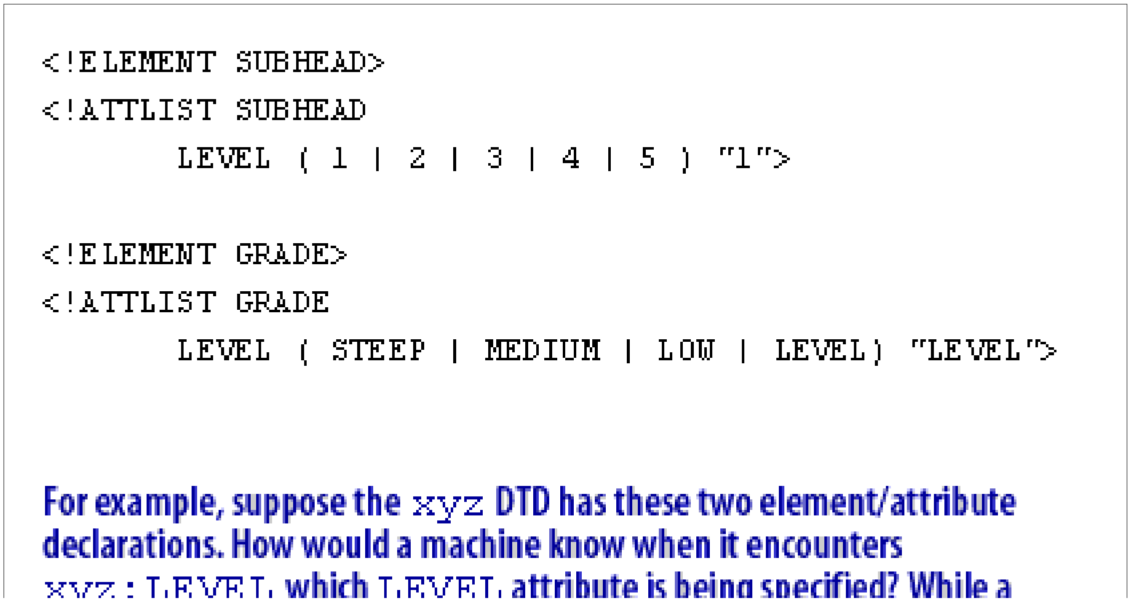


This code demonstrates the declaration and use of a fictitious namespace within an XML file. You must declare any namespaces used at the top of your XML document, before the root element of the document appears. You can specify that attributes are defined by a certain namespace just as you can specify that certain elements are defined by a namespace.



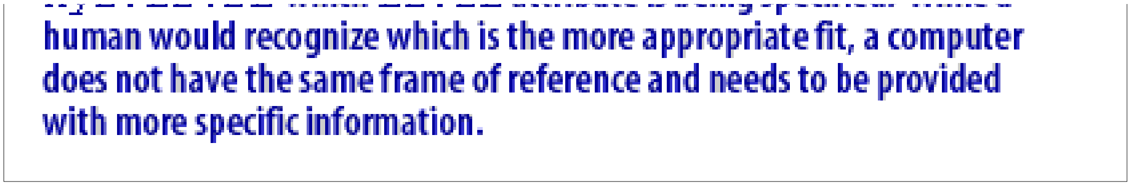
```
<HEADING xyz:LEVEL="2">

Sales This Month

</HEADING>
```

This example shows how to indicate that the `LEVEL` attribute in this element acts in accordance with the `LEVEL` attribute specified in the `xyz` namespace. So far, this makes sense and seems easy. But what happens if, for example, the `LEVEL` attribute shows up for more than one element, with different values specified?

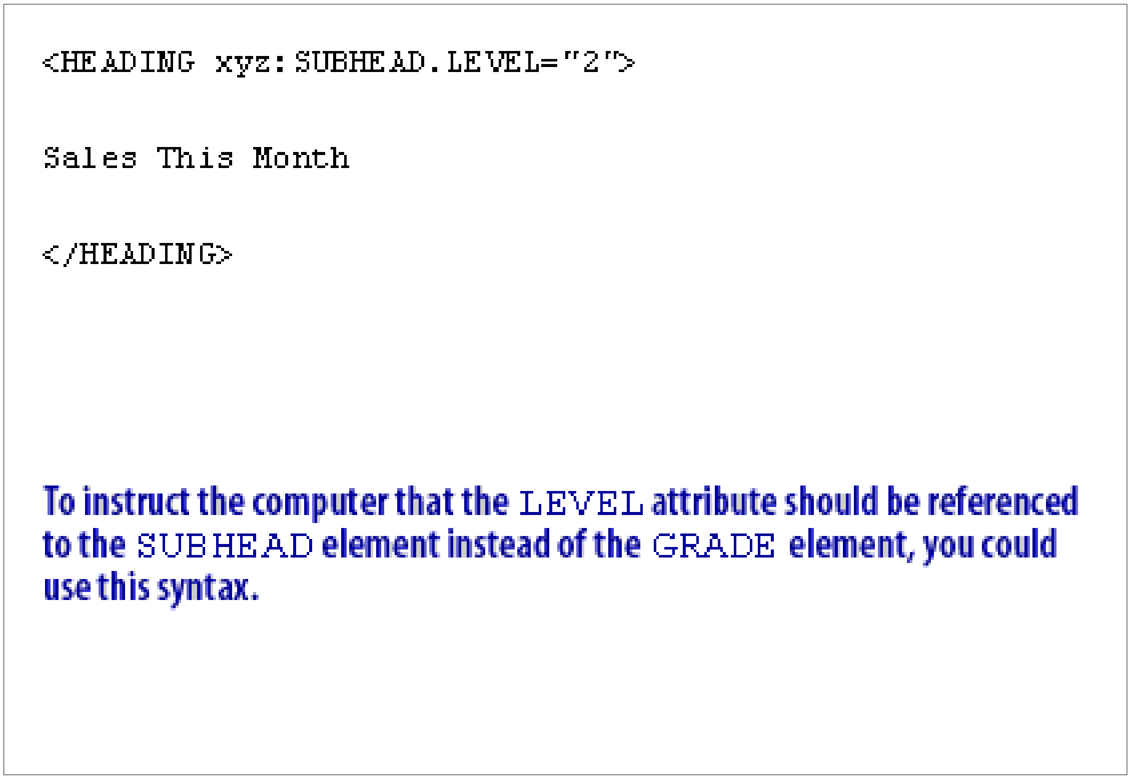


```
<!ELEMENT SUBHEAD>
<!ATTLIST SUBHEAD
       LEVEL ( 1 | 2 | 3 | 4 | 5 ) "1">

<!ELEMENT GRADE>
<!ATTLIST GRADE
       LEVEL ( STEEP | MEDIUM | LOW | LEVEL) "LEVEL">
```

For example, suppose the `xyz` DTD has these two element/attribute declarations. How would a machine know when it encounters `xyz:LEVEL` which `LEVEL` attribute is being specified? While a



human would recognize which is the more appropriate fit, a computer does not have the same frame of reference and needs to be provided with more specific information.



```
<HEADING xyz:SUBHEAD.LEVEL="2">

Sales This Month

</HEADING>
```

To instruct the computer that the LEVEL attribute should be referenced to the SUBHEAD element instead of the GRADE element, you could use this syntax.

Module 6

Lesson 8

XLink, XPointer, namespaces, and metadata

Module wrap-up

The related XML technologies of XLink, XPointer, and namespaces will provide invaluable tools for creating "smarter" documents without losing the aesthetically pleasing elements of style. By using XML, machines will begin to understand Web pages as well as humans do.

In this module, you practiced using these related technologies. You also learned about metadata initiatives.

This module discusses the following terms in relation to XML:

- *XLink*
- *simple link*
- *out-of-line link*
- *XPointer*
- *namespaces*
- *metadata*
- *Dublin Core initiative*

Discussion

Do you have any questions or comments about the material in this module? If so, look in the folder Related XML technologies in the Discussion section, which you can reach by clicking the Discuss button in the toolbar. There's also a folder on current and emerging XML standards.

Quiz

Quiz

Click the Quiz button to test what you've learned in this module.

http://purl.org/dc

DUBLIN CORE METADATA INITIATIVE

Dublin Core topics:

Home | Search | Site Map | What's New | Feedback

Home :

CONTENTS

- Dublin Core Element Set
- About the Dublin Core Metadata Initiative
- News and Publications
- Documents
- Education
- Projects
- Schemas
- Tools
- Working Groups
- Workshop Series

Latest Important Information (updated 1999-07-15):

- 1999-07-02: Dublin Core Elements, Version 1.1 Proposed Recommendation released for comment (deadline 1999-07-31) [More Information] [Announcement]
- 1999-07-02: Bibliographic Citation Working Group Working Draft now available for review (deadline 1999-07-19) [More Information] [Announcement]
- 1999-07-01: Dublin Core Qualifiers now being collected from implementations (deadline 1999-07-31) [More Information] [Announcement]
- 1999-06-21: 7th International Dublin Core Workshop - Call for Participation (deadline 1999-08-01) [More Information] [Announcement]

The Dublin Core: A Simple Content Description Model for Electronic Resources

Metadata for Electronic Resources

The Dublin Core is a metadata element set intended to facilitate discovery of electronic resources. Originally conceived for author-generated description of Web resources, it has attracted the attention of formal resource description communities such as museums, libraries, government agencies, and commercial organizations.

The Dublin Core Workshop Series has gathered experts from the library world, the networking and digital library research communities, and a variety of content specialties in a series of invitational workshops. The building of an interdisciplinary, international consensus around a core element set is the central feature of the Dublin Core. The progress represents the emergent wisdom and collective experience of many stakeholders in the resource description arena. An open mailing list supports ongoing work.

The characteristics of the Dublin Core that distinguish it as a prominent candidate for description of electronic resources fall into several categories:

Simplicity

The Dublin Core is intended to be usable by non-catalogers as well as resource description specialists. Most of the elements have a commonly understood semantics of roughly the complexity of a library catalog card.

Semantic Interoperability

In the Internet Commons, disparate description models interfere with the ability to search across discipline boundaries. Promoting a commonly understood set of descriptors that helps to unify other data content standards increases the possibility of semantic interoperability across disciplines.

International Consensus

Recognition of the international scope of resource discovery on the Web is critical to the development of effective discovery infrastructure. The Dublin Core benefits from active participation and promotion in some 20 countries in North America, Europe, Australia, and Asia.

Extensibility

The Dublin Core provides an economical alternative to more elaborate description models such as the full MARC cataloging of the library world. Additionally, it includes sufficient flexibility and extensibility to encode the structure and more elaborate semantics inherent in richer description standards

Metadata Modularity on the Web

The diversity of metadata needs on the Web requires an infrastructure that supports the coexistence of complementary, independently maintained metadata packages. The World Wide Web Consortium (W3C) has begun implementing an architecture for metadata for the Web. The Resource Description Framework, or RDF, is designed to support the many different metadata needs of vendors and information providers. Representatives of the Dublin Core effort are actively involved in the development of this architecture, bringing the digital library perspective to bear on this important component of the Web infrastructure.

For questions or comments regarding the Dublin Core contact dc@oclc.org

Home | Search | Site Map | What's New | Feedback | About the Dublin Core | News and Publications | Documents | Education | Schemas | Projects | Tools | Working Groups | Workshop Series | Workshop Series Sponsors

XML Fundamentals

Module 6

Metadata

Element usage for the Dublin core scheme

This table describes the appropriate usage for each element:

Element Name	Description
Title	The name of the resource being described. For example, a book, film or painting.
Creator	The primary person or organization responsible for creating the intellectual content. For example, the painter, poet or composer.
Subject	The topic of the resource. Multiple keywords can be used for this element.
Description	A description of the resource. This can include a description of contents, an abstract, or anything else that constitutes a general top-level description of the resource.
Publisher	The person or organization primarily responsible for making the resource available. For example, the publishing company, the corporation, the collection holder.
Contributor	The person or organization who was not primarily responsible for the creation of the intellectual property, but who was a significant contributor in terms of content. For example, the editor, the artist who supplied illustrations, or a secondary author.
Date	The creation date of the resource or the date when it first became available. For example, the date of the painting, the publication date of the article.
Type	The category that best describes the intellectual property. For example, a poem, painting, or music. The group working on the Dublin Core is creating a specific list of acceptable values for `Type`.
Format	The data format of the resource, especially pertaining to hardware or software resources. The Dublin Core group hopes to create a specific list of accepted values for `Format`.
Identifier	A unique string or value identifying the resource. For

	example, ISBN number or URI.
`Source`	The parent resource, if one exists, of the current resource. For example, a magazine name for an article.
`Language`	The language identifier, preferably matching the language code specified in RFC 1766 (http://ds.internic.net/rfc/rfc1766.txt).
`Relation`	The relationship between the element and another source (e.g., `IsVersionOf`, `IsBasedOn`, `IsFormatOf`).
`Coverage`	The range covered by the resource. For example, a city name where the resource resides or the longitude and latitude of the resource.
`Rights`	A statement or link for information regarding intellectual property rights.

Metadata

Dublin Core as the namespace

Here's an example of how *Dublin Core* elements might be used with the book catalog example:

```
<?xml version="1.0"?>
<?xml:namespace  ns="http://purl.oclc.org/dc/"
   prefix="DC"?>
<CATALOG>
   <BOOK>
      <DC:TITLE>A Certain Justice</ DC:TITLE>
      <DC:CREATOR>P.D. James</DC:CREATOR>
      <DC:DATE>1998</DC:DATE>
      <DC:IDENTIFIER>0375401091</DC:IDENTIFIER>
   </BOOK>
</CATALOG>
```

XML Fundamentals

Module 6

Exercise

Metadata

Using namespaces and metadata

Objective

Use XML namespaces and metadata.

Exercise Scoring

Full credit for this exercise is 8 points (2 points for each step). You'll submit your code to the course tutors.

Starter files

To help you complete this exercise, we've supplied a starter file in the 06-07 folder of the course download available from the Resources page.

Instructions

Suppose that *Photos from the Road* I is the first book in an extensive series of travel photo books by Cicero Lane. As the collection grows, Cicero's publishers decide to create an XML namespace for use in cataloging his books. You will help create this namespace.

Here is the code which categorizes the elements in the table of contents:

```
<ROADPHOTOS>
<TOC>
<TITLE> Photos from the Road I </TITLE>
<COVER> Cover Photo </COVER>
<INTRO> Introduction by Cicero Lane </INTRO>
<DESTINATION> Los Angeles
   <TOUR> Venice Beach </TOUR>
   <TOUR> LA County Museum of Art </TOUR>
   <TOUR> Disneyland </TOUR>
</DESTINATION>
<DESTINATION> Paris
   <TOUR> Eiffel Tower </TOUR>
   <TOUR> Notre Dame Cathedral </TOUR>
   <TOUR> Le Louvre </TOUR>
```

```
</DESTINATION>
<DESTINATION> Marshall Islands
   <TOUR> Kwajalein </TOUR>
   <TOUR> Scuba Diving </TOUR>
   <TOUR> WWII Shipwrecks </TOUR>
</DESTINATION>
</TOC>
</ROADPHOTOS>
```

1. Write the declaration for the *Photos from the Road I* namespace. This line of code must appear in any XML file using this namespace. Consider the following points as you code.
   - Assume that the namespace can be found on the *Photos from the Road Web* site at www.roadphotos.com/namespace.
   - Use the prefix `ropho` in all element declarations. This prefix works well because it is identifiable but brief.
   - Include the root element in your code to demonstrate the proper placement of your namespace declaration. Also, be sure to declare your document as XML.
   - Though the XML namespaces proposal is still being evaluated and is subject to change, use the syntax demonstrated in this module.

   In the text box below, cut and paste your code, and continue on to the next step.

2. Now add code with namespace attributes for the rest of the document, using the element tags given. Specify attributes (including values) for at least two of your elements. Consider the following points as you code:
   - Many of your entries have the same element names. These elements are good candidates for adding attributes to differentiate their contents.
   - You can choose any attributes you like, but consider how they might be useful to someone searching for information in this document series.
   - Be sure you use correct syntax for your attributes and values.

   In the text box below, cut and paste your code, and continue on to the next step.

3. Using the element attributes you declared in Step 2, write the appropriate element/attribute-list declarations for this document's DTD. Consider the following points as you code:
    - Determine whether an attribute can have multiple values or only one. Be sure to use proper attribute-list declaration syntax.
    - Consider whether any of your attributes for different elements use the same name. If so, include a line or two of the syntax you could use for specifying which namespace element your attribute should reference.

   In the text box below, cut and paste your code, and continue on to the next step.

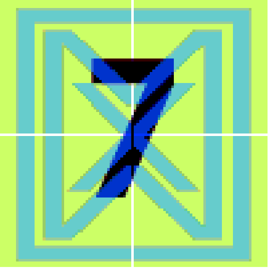
4. The Dublin Core scheme categorizes resource works such as books. Suppose you want to add "*Photos from the Road I*" to a library catalog. Using the Dublin Core initiative's elements, write a brief declaration to catalog Cicero Lane's "*Photos from the Road I*." Consider the following points as you code:
    - You must reference the Dublin Core as your namespace to tie to their elements.
    - Reference identification information for this book; you can assume some of this information.
    - Write code for at least three Dublin Core elements of "*Photos from the Road*."
    - The Dublin Core namespace can be accessed at www.purl.org/metadata/dublin_core#.

   In the text box below, cut and paste your code, and continue on to the next step.

OK, I'm Done

Module 7

Lesson 1



Styling XML files

Module introduction

As much as we care about the content and structure of XML documents, they will be of little value until they can be rendered and perceived in an aesthetically pleasing manner. For this, the Extensible Style Language (XSL) has been proposed in addition to the Cascading Style Sheet (CSS) method.

In this module, you will learn about these emerging standards and how they relate to XML. You will see examples of how both CSS and XSL can be used with XML.

Module learning objectives

After completing the module, you will have the skills and knowledge necessary to:

- Create a style sheet for XML files
- Link a style sheet to an XML file
- Evaluate the capability of XSL
- Use the construction rules, the basic units of XSL files
- Define patterns to represent relationships
- Use wildcards to increase flexibility and decrease repetition
- Use qualifier attributes
- Specify a pattern by referencing an attribute value for an element
- Specify the selection precedence for patterns
- Use objects to define the visual and behavioral characteristics of elements
- Define default characteristics and add literal text to a style sheet
- Use XSL with IE 5.0

XML Fundamentals

Module 6

Quiz

XLink, XPointer, namespaces and metadata

Each question is worth one point. Some questions ask you to select **the best answer**, others ask you to select **all the correct answers**. To receive credit for questions asking for **all the correct answers**, you must select all the correct answers and **only** the correct answers.

1.XPointer can best be described as:

Please select **the best answer**.

○ A. The W3C standard for XML link references

○ B. A proposal to link portions of an XML resource rather than the entire resource

○ C. An XML attribute that links elements in the document tree structure

○ D. The syntax used to create internal references within a page in XML

2.In XML, a namespace is used to:

Please select **the best answer**.

○ A. Create names for your XPointer sibling links

○ B. Give multiple contexts and meanings to common names

○ C. Define two or more sets of tags with multiple entities

○ D. Establish clear naming conventions

3.Metadata is data about the data within a document. A specification currently under discussion as a working draft at the W3C proposes a language for specifying metadata about XML data. This language is called:

Please select **the best answer**.

○ A. Resource Description Framework (RDF)

○ B. Uniform Resource Identifier (URI)

○ C. Dublin Core (DC)

○ D. Meta Content Framework (MCF)

4.The Dublin Core is:

Please select **all the correct answers**.

□ A. A group of experts at the national university in Ireland who proposed the RDF specification

□ B. A resource description scheme developed by experts from various academic communities
□ C. A namespace from which its elements derive meaning
□ D. An established resource specification endorsed by the W3C.
□ E. A 15-element scheme that places elements in three categories

5. According to the Dublin Core initiative, which element should you use to describe the person or organization responsible for producing a work's intellectual content (such as the composer, poet or painter)?

Please select **the best answer**.

○ A. `Author`
○ B. `Source`
○ C. `Creator`
○ D. `Developer`
○ E. `Primary`

OK, I'm Done

XML Fundamentals

Module 7

Lesson 2
Objective

Creating a style sheet

Create a style sheet for XML files.

Because XML contains no display information, it will be imperative that you create or use a *style sheet* for any XML files you want to display. Without this information, a machine will have no way of determining how to render your text. Creating style sheets will be a mandatory part of working with XML files.

For SGML files, *DSSSL* (pronounced "Dissell") provides display information. For HTML files, the Cascading Style Sheet specification (CSS) performs this task. For XML, a new language has been proposed that takes the best of DSSSL and CSS and extends them. This new language is called the Extensible Style Language or *XSL*.

While CSS has also been proposed as a style language for XML files, you will see that XSL promises much greater power and control over your pages, and will likely become the preferred language for styling XML pages. XSL is not expected to replace CSS, but rather to enhance and extend style capabilities. For very simple, straightforward XML documents, CSS might be all the style you need. For more complex documents, XSL will likely be the more robust candidate for handling style information for your XML file.

You can embed style information directly within an HTML document. However, because XML's goal is to continue to separate content from presentation information, the style rules *must* reside in a separate file called a style sheet for XML documents.

Both CSS and XSL depend on certain style properties for functionality.

Lesson 3 Objective

CSS and XML

Link a style sheet to an XML file.

Cascading Style Sheets are text files containing one or more rules or definitions for the style characteristics of a particular element. One proposal for linking a style sheet to an XML file uses the following syntax:

```
<?xml:stylesheet type="text/css"
href="filename.css"?>
```

This FlipBook shows how a CSS file and an XML file might be linked together:

Transcript

Remember that these examples are theoretical. They should give you an idea of how CSS could be used with XML if a proposal is agreed upon that supports this model.

One key reason that CSS is likely to be less popular with XML developers than the other proposed language, *XSL*, is that CSS is not itself an XML application, whereas XSL is. The two are not, however, mutually exclusive. You will be able to use both CSS and XSL as needs dictate.

Module 7

Creating a style sheets

The CSS1 property list

The following is a list of the style properties supported by CSS1. They are grouped by property category. CSS2 adds additional properties and supports different media types. CSS1 is currently designed expressly for affecting the visual display of the elements.

The CSS1 property list:

Style	Properties	
Font	font-family font-size font-weight	font-variant font-style font
Color and background	color background-color background-image background-repeat	background-attachment background-position background
Classification	display white-space list-style-type	list-style-image list-style-position list-style
Text	text-align vertical-align text-indent line-height	text-decoration text-transform letter-spacing word-spacing
Box	width height float clear margin margin-top margin-right margin-bottom margin-left padding padding-top padding-right padding-bottom	padding-left border border-top border-right border-bottom border-left border-color border-style border-width border-top-width border-right-width border-bottom-width border-left-width

XML Fundamentals

Module 7

Lesson 4
Objective

XSL

Evaluate the capability of XSL.

The current XML 1.0 specification defines Cascading Style Sheets (CSS) as the preferred method for applying formatting information to XML documents. However, Extensible Stylesheet Language or *XSL* is the W3C Working Draft (WD) technology that will become the preferred method for applying styles to XML documents.

XSL was created from both SGML and CSS in an effort to provide maximum flexibility without maximum complexity for formatting XML pages. XSL is a much more robust language than CSS. CSS can only define style properties for elements. XSL, on the other hand, allows you to specify style relationships based on a user-definable hierarchy. In addition, XSL provides you with the opportunity to include script, text and images in a style rule.

XSL is much more powerful and consequently much more difficult to learn than CSS. Whereas CSS simply specifies style characteristics for elements, XSL specifies relationships, style characteristics and structure options. XSL can take XML input and output it to an HTML format, providing a useful transforming utility while we await the development of browsers that will natively display XML information.

XSL is expected to become the preferred method for formatting XML data because its origin is the same as XML's: Both are derived from Standard Generalized Markup Language (SGML). And unlike CSS, XSL has the ability to display XML data in arbitrary output structure.

Designing an XSL stylesheet requires knowledge of the XML document to be formatted. With that knowledge, you can create a template that uses a series of patterns to define the XML data's structure and format. The XML document references the XSL file, resulting in a formatted display in the browser window. This implementation allows for pure presentation of XML documents.

Warning

The XSL proposals that IE 4.x supports use the implementation of DSSSL flow objects. This module will introduce you to XSL concepts using familiar objects. The syntax you will learn is appropriate for IE 4.x browsers. At the end of the module, you will find out how to use XSL with IE 5.0 in accordance with the W3C WD2 dated December 1998.

Quiz

Quiz

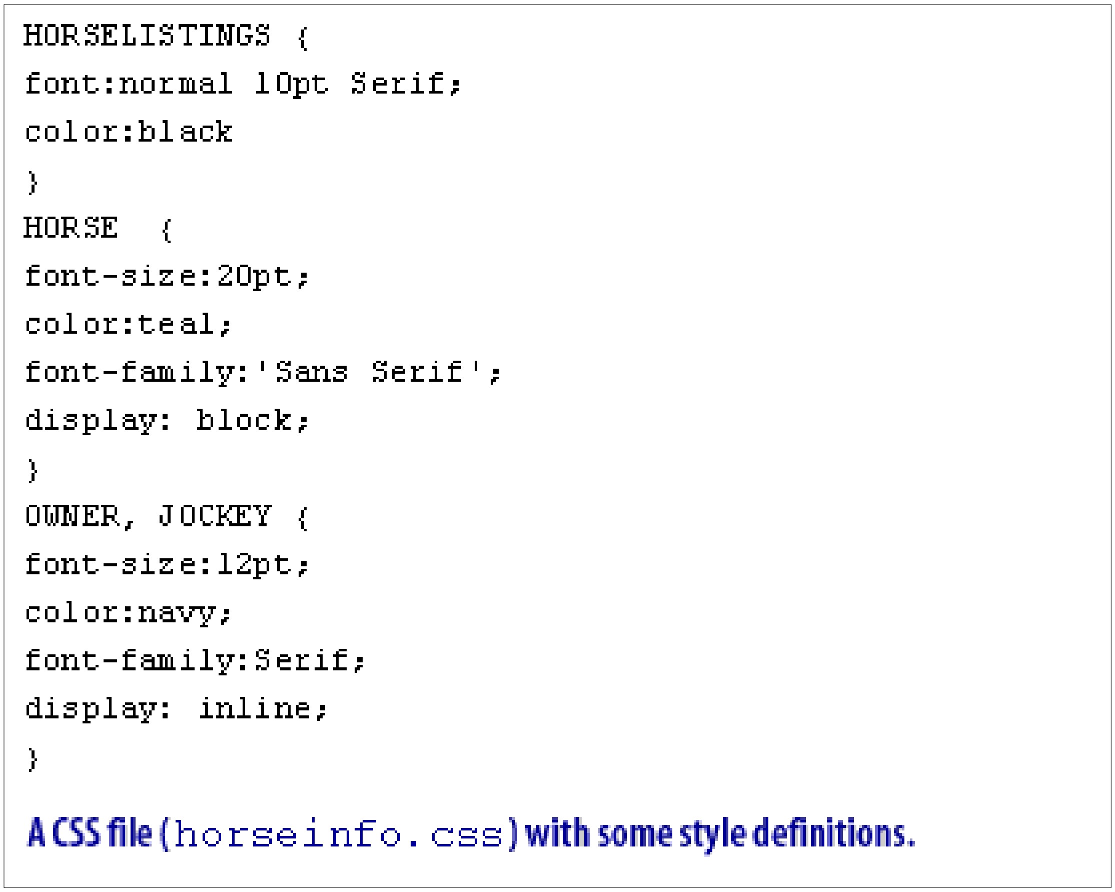
Click the Quiz button to see what you've learned about styling XML files.

Module 7

CSS and XML

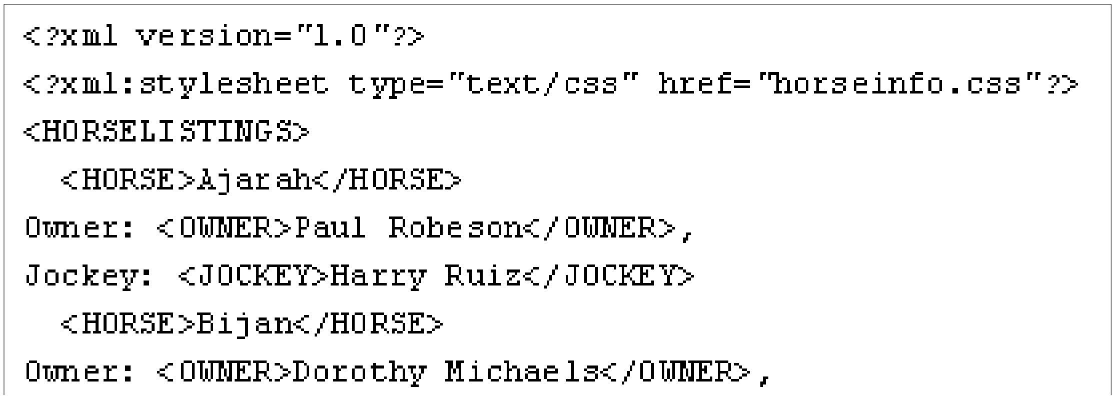
An example of linked files (FlipBook transcript)

How a CSS file and an XML file might be linked together:

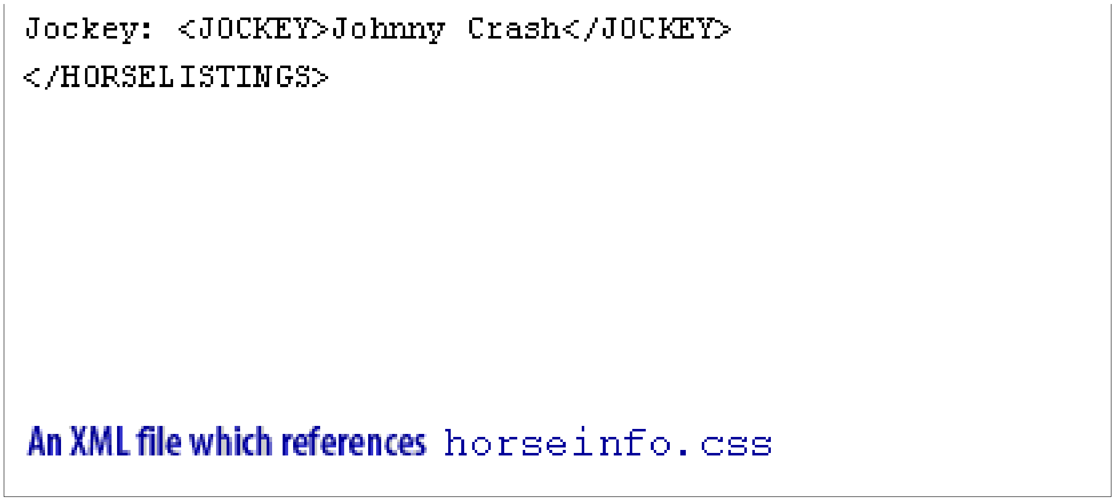


```
HORSELISTINGS {
font:normal 10pt Serif;
color:black
}
HORSE  {
font-size:20pt;
color:teal;
font-family:'Sans Serif';
display: block;
}
OWNER, JOCKEY {
font-size:12pt;
color:navy;
font-family:Serif;
display: inline;
}
```

A CSS file (horseinfo.css) with some style definitions.



```
<?xml version="1.0"?>
<?xml:stylesheet type="text/css" href="horseinfo.css"?>
<HORSELISTINGS>
  <HORSE>Ajarah</HORSE>
Owner: <OWNER>Paul Robeson</OWNER>,
Jockey: <JOCKEY>Harry Ruiz</JOCKEY>
  <HORSE>Bijan</HORSE>
Owner: <OWNER>Dorothy Michaels</OWNER>,
```



```
Jockey: <JOCKEY>Johnny Crash</JOCKEY>
</HORSELISTINGS>
```

An XML file which references horseinfo.css

XML Fundamentals

Module 7

Lesson 5
Objective

XSL construction rules

Use the construction rules, the basics units of XSL files.

The basic units of XSL files are the *construction rules*. These rules determine which elements will be affected when, and in what way. Construction rules have two components: a pattern definition and an action. The pattern specifies not only the formatting for an element, but the context in which that formatting rule will apply. For example, are you specifying a construction rule for all headings, or only for the first heading within a section? In the action section, you specify how element contents will be presented.

Perhaps the easiest way to demonstrate how a construction rule works is to view a comparison between styled HTML text and the same text styled in the context of XSL:

Transcript

Together, the pattern and the action make up the construction rule. The general syntax for creating a construction rule is as follows:

```
<rule>
<!--pattern-->
target-element reference
<!--action-->
<flow object and style information>
   <children/>
</flow object>
</rule>
```

Exercise

In this exercise, you'll practice writing code to create style sheets for use with an XML document.

Leading the Web to its Full Potential...

World Wide Web Consortium Supports HTTP/1.1 Reaching IETF Draft Standard

W3C is pleased to recognize that HTTP/1.1 has been approved as an IETF Draft Standard, and to have contributed to its development and implementation. "Products which use HTTP/1.1 have been demonstrated to run significantly faster than those which do not. I urge everyone to check for HTTP/1.1 compliance when choosing software," says **Tim Berners-Lee**, W3C Director. (Press Release, Testimonials, RFC 2616, RFC 2617)

Other HTTP/1.1 materials from W3C:

- "Network Performance Effects of HTTP/1.1, CSS1, and PNG", which was presented at ACM SIGCOMM '97. An overview of performance benefits is also available.
- "Editing the Web: Detecting the Lost Update Problem Using Unreserved Checkout".
- W3C software implementing HTTP/1.1: the Libwww protocol library and the Jigsaw Web server.

Other news:

- Mathematical Markup Language (MathML[tm]) 1.01 Specification. This is a revision of the 7 April 1998 release.
- Associating Style Sheets with XML documents Recommendation (Press Release, Testimonials).
- Amaya 2.1 available for Windows and Unix

...past W3C news

Technical Reports **About W3C**

User Interface Domain

HTML

Style Sheets: CSS, XSL

Document Object Model: DOM

Synchronized Multimedia: SMIL

Math: MathML

Graphics: SVG, WebCGM

Voice Browser

Internationalization

Mobile Access

Technology and Society Domain

Electronic Commerce

Metadata: RDF, PICS

Privacy: P3P

XML Signature

Architecture Domain

HTTP, HTTP-NG

Television and the Web

Web Characterization

XML

Web Accessibility Initiative

WAI Accessibility Guidelines

WAI International Program Office

WAI Technical Activity

W3C Open Source Software

Amaya browser/editor

Jigsaw Web server

Libwww protocol library

HTML Tidy Utility

W3C Services

About the Web

Web History

W3C's specifications, including current Recommendations, Working Drafts, and Notes. Translations of technical reports.

Contact W3C / Join W3C
The Members / The Team

Member Area

For W3C Member employees only. Get a personal password.

Press Information

Contacts, news releases, W3C in the press

Search W3C site and public mailing lists archives

Submit

HTML Validation Service

CSS2 Package:

· W3C Core Styles

· CSS2 Validation Service

· CSS Test Suite

Mailing Lists

W3C Web Site

Search

Help navigating the site

Historical

Web Journal

Virtual Library

CERN Server

W3C is hosted by the Laboratory for Computer Science at MIT, by INRIA and Keio University with support from DARPA and the European Commission.

W3C HTML 4.0 Webmaster

XML Fundamentals

Module 7

Quiz

Defining XSL

Each question is worth one point. Some questions ask you to select **the best answer**, others ask you to select **all the correct answers**. To receive credit for questions asking for **all the correct answers**, you must select all the correct answers and **only** the correct answers.

1.Because XML provides no display information, you must use another technology to provide style information to your XML documents. Which technology is most likely to be preferred for styling XML pages?

Please select **the best answer**.

○ A. CSS
○ B. CSS2
○ C. DSSSL
○ D. XSL
○ E. XMLS

2.When you provide display instructions for an XML file, you must:

Please select **the best answer**.

○ A. Embed the style information within an HTML file
○ B. Embed the style information within an XML file
○ C. Place style rules in a separate file called a style sheet
○ D. Place style rules in a separate file called a CSS

3.A Cascading Style Sheet is:

Please select **the best answer**.

○ A. An XML document that contains rules for the style characteristics of an element
○ B. An XSL document that contains rules for the style characteristics of an element
○ C. A text file that contains rules for the style characteristics of an element
○ D. An HTML file that contains rules for the style characteristics of an element
○ E. A DSSSL file that contains rules for the style characteristics of an element

4. Which of the following are key reasons that CSS is likely to be less popular than XSL for styling XML documents?

Please select **all the correct answers**.

□ A. CSS specifies style characteristics; XSL specifies characteristics, relationships and structure options.

□ B. CSS cannot be used with XSL.

□ C. XSL itself is an XML application.

□ D. CSS itself is an XML application.

□ E. XSL promises greater power and control over pages than CSS.

OK, I'm Done

Module 7

Lesson 6 Objective

View Code

Defining a pattern

Define patterns to represent relationships.

Consider this combination of an XML and XSL file.

The pattern as represented in this example indicates that every occurrence of the `MainHead` element should receive the action. But what if you wanted to format only those `MainHead` elements that fell within `NewSection` elements? You could define a pattern to represent that relationship as follows:

```
<element type="NewSection">
<target-element type="MainHead"/>
</element>
```

In this example, the target-element is still `MainHead`, but `MainHead` will only receive the action of the rule when it is within a `NewSection` element.

The next example demonstrates how to specify a pattern based not on an ancestor element, but on a descendent element. In this case, `MainHead` will be the target-element only when it contains a `NewSection` element.

```
<target-element type="MainHead">
<element type="NewSection"/>
</target-element>
```

You can specify more than one target-element if you want two or more elements to share the same style characteristics. For example, the following would define a pattern whereby the `MainHead` and `NewSection` elements would receive the same formatting action:

```
<target-element type="MainHead"/>

<target-element type="NewSection"/>
```

Because neither element contains the other, both of these elements would receive the same action regardless of whether `MainHead` or `NewSection` came first.

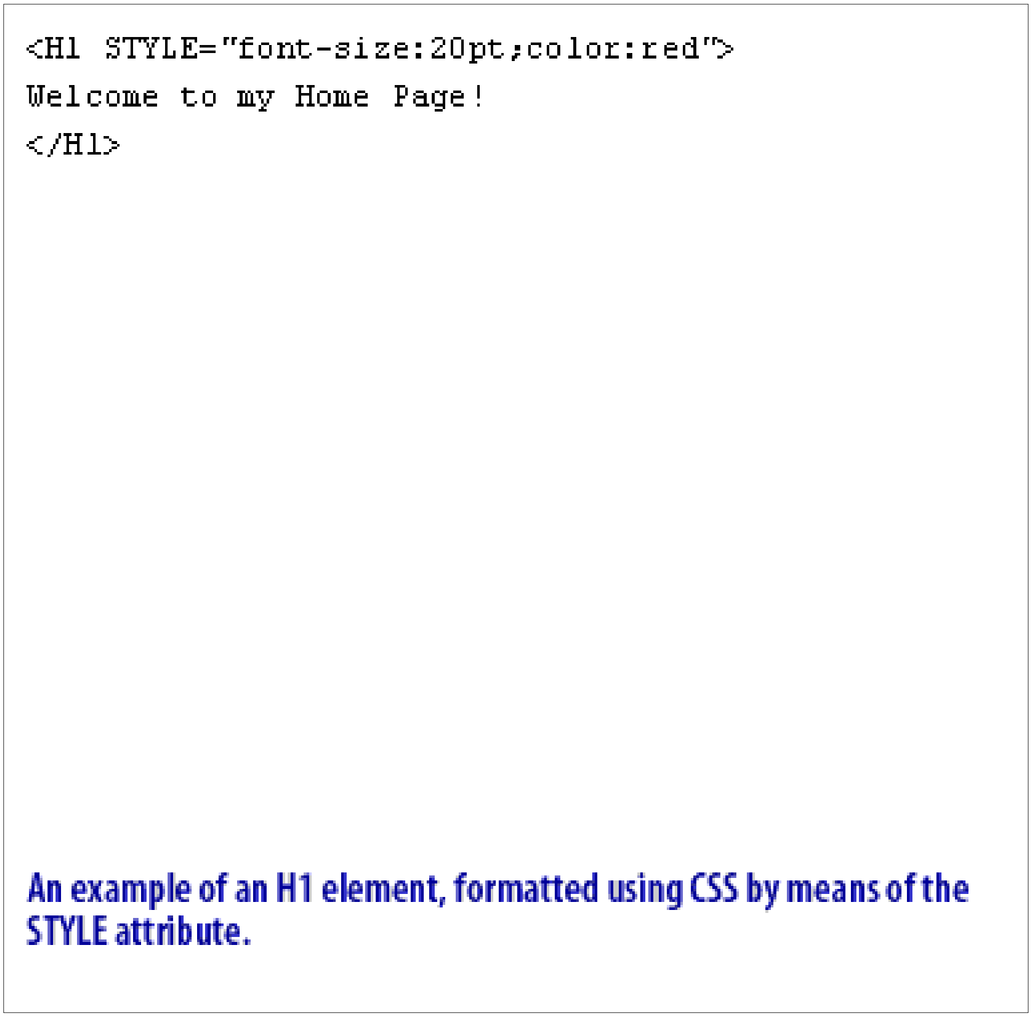
XML Fundamentals

Module 7

XSL construction rules

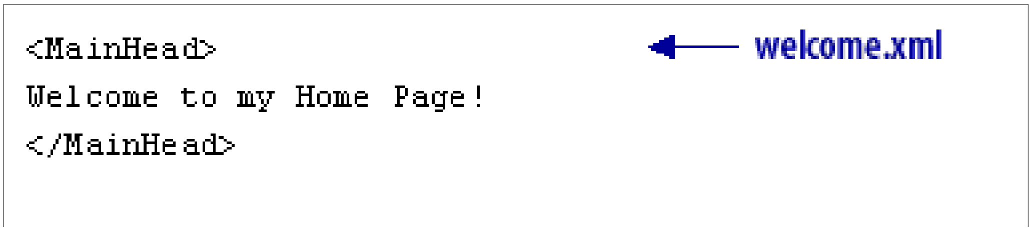
An example of using XSL (FlipBook transcript)

A comparison between styled HTML text and the same text styled in the context of XSL:

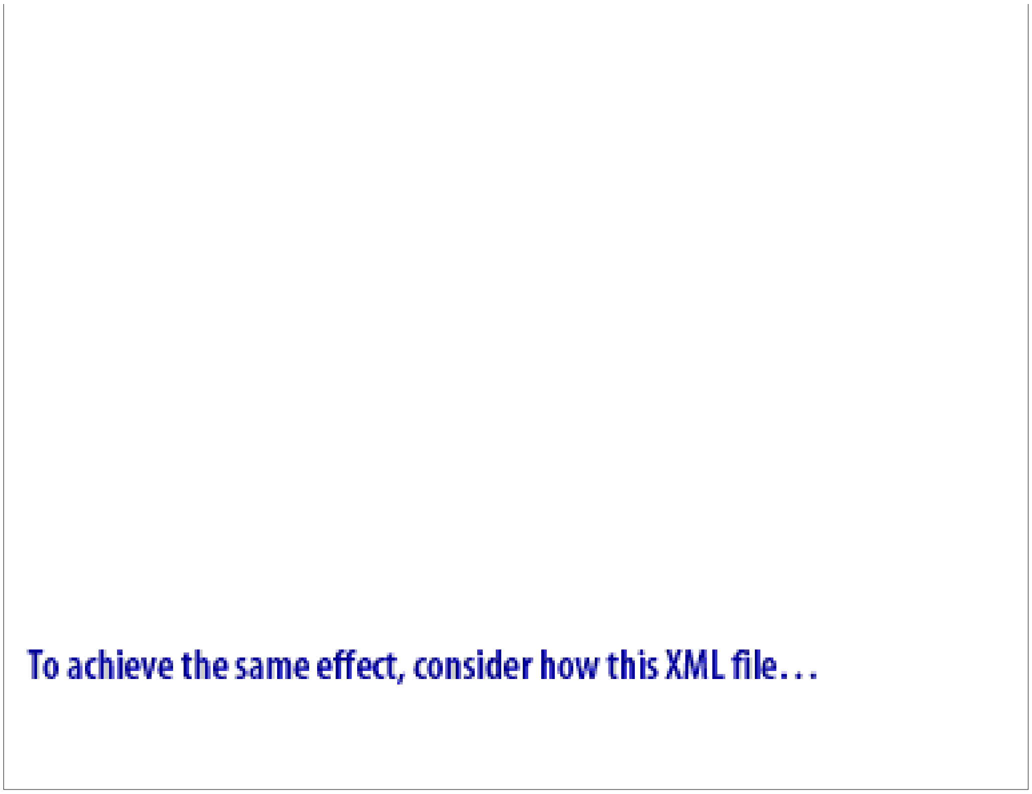


```
<H1 STYLE="font-size:20pt;color:red">
Welcome to my Home Page!
</H1>
```

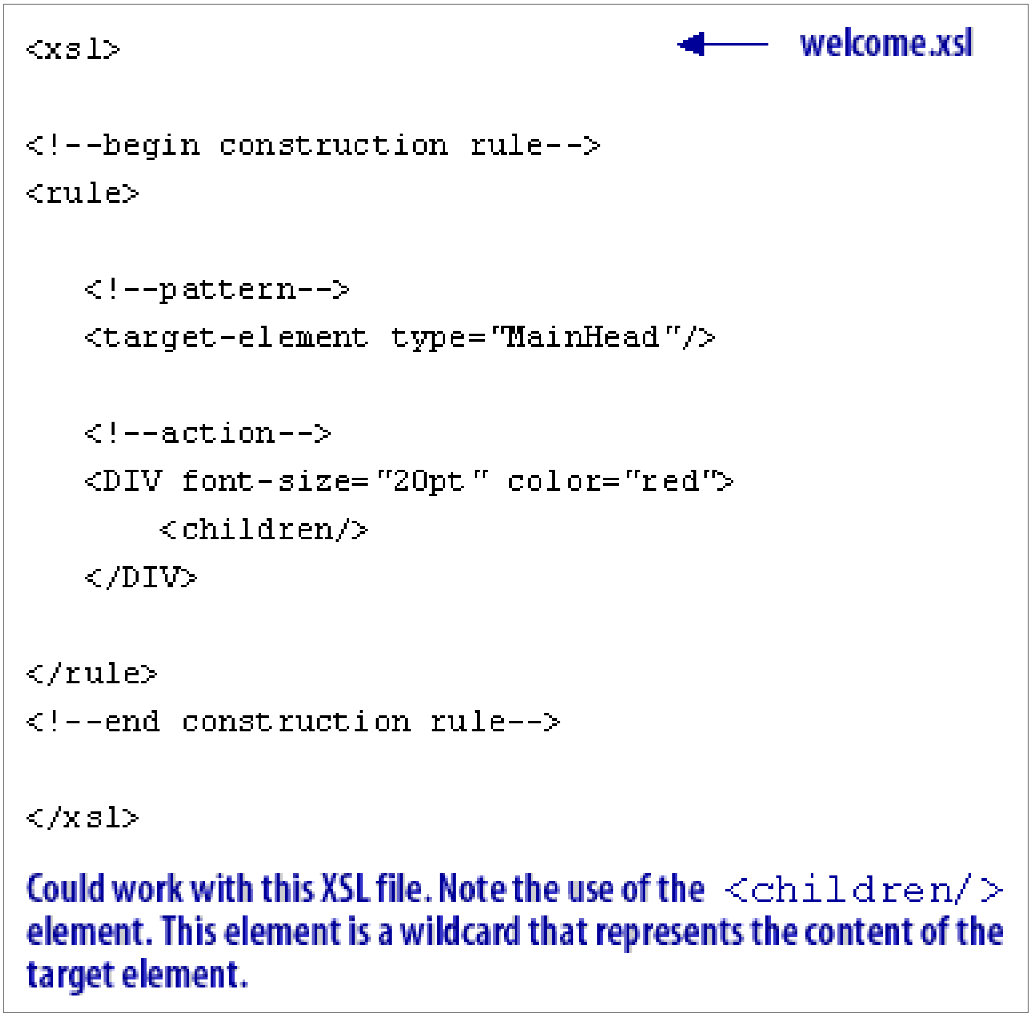
An example of an H1 element, formatted using CSS by means of the STYLE attribute.



```
<MainHead>
Welcome to my Home Page!
</MainHead>
```



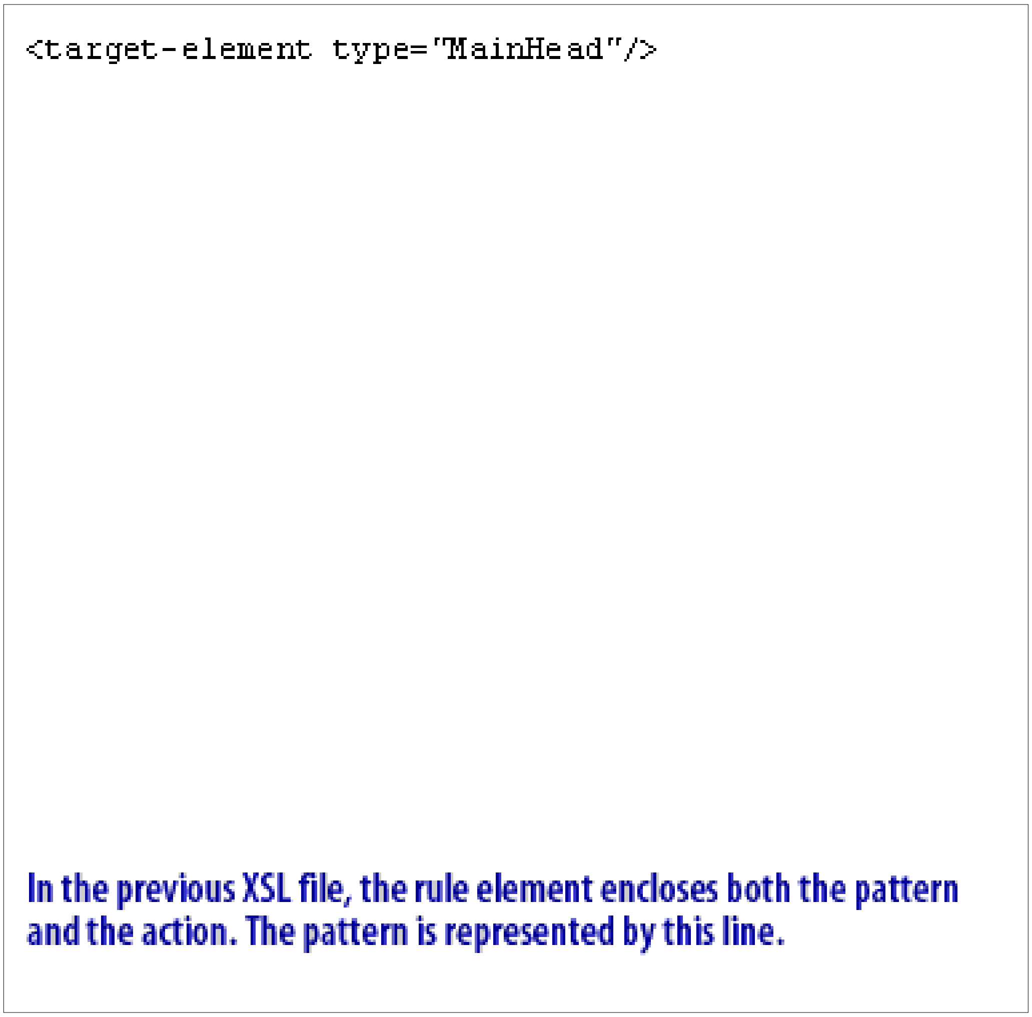
To achieve the same effect, consider how this XML file…



welcome.xsl

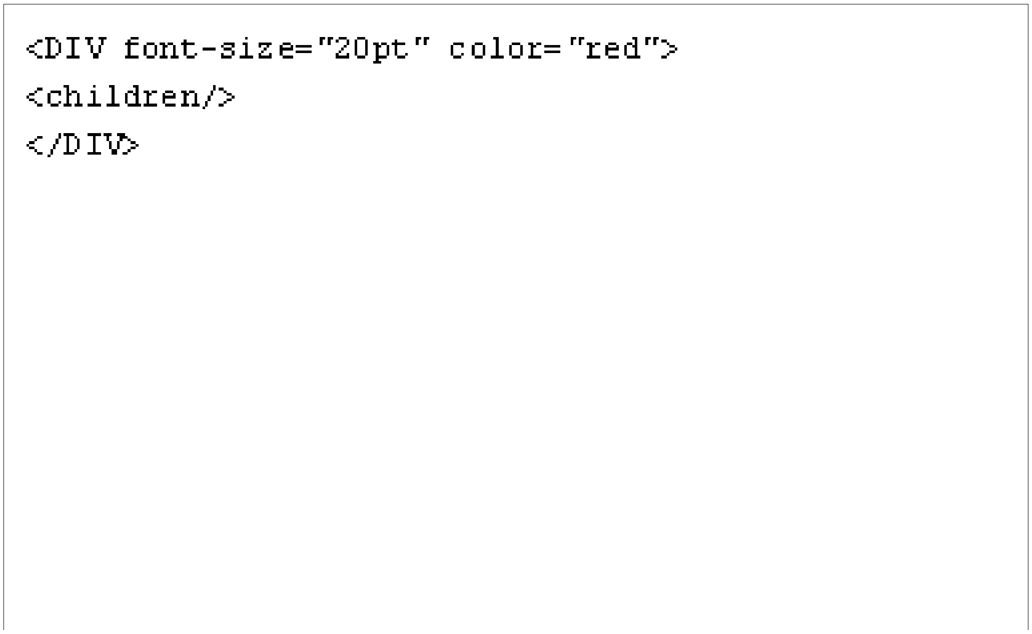
```
<xsl>

<!--begin construction rule-->
<rule>

   <!--pattern-->
   <target-element type="MainHead"/>

   <!--action-->
   <DIV font-size="20pt" color="red">
       <children/>
   </DIV>

</rule>
<!--end construction rule-->

</xsl>
```

Could work with this XSL file. Note the use of the `<children/>` element. This element is a wildcard that represents the content of the target element.

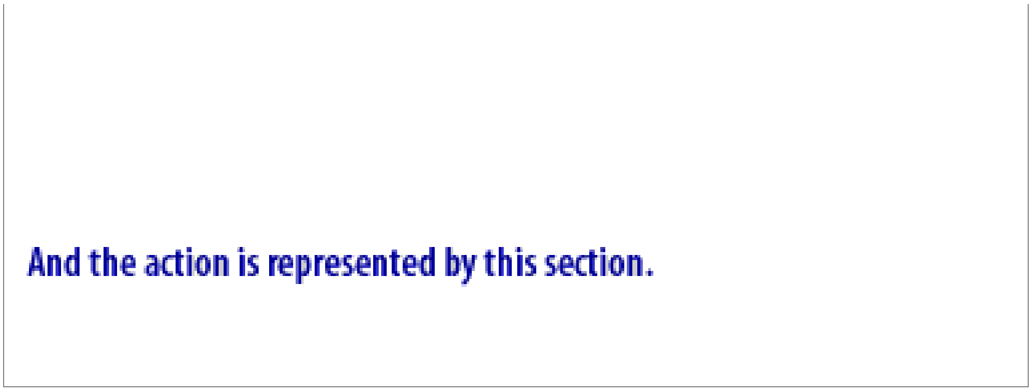


```
<target-element type="MainHead"/>
```

In the previous XSL file, the rule element encloses both the pattern and the action. The pattern is represented by this line.



```
<DIV font-size="20pt" color="red">
<children/>
</DIV>
```



And the action is represented by this section.

XML Fundamentals

Module 7

Exercise

XSL construction rules

Creating style sheets

Objective

Practice writing code to create style sheets for use with an XML document.

Exercise Scoring

Full credit for this exercise is 6 points (3 points for each step). You'll submit your code to the course tutors.

Starter files

To help you complete this exercise, we've supplied a starter file in the 07-05 folder of the course download available from the Resources page.

Instructions

In this exercise, you will practice writing code to create style sheets for use with an XML document.

Consider the following electronic catalog, which lists the major works by artist Samson Lane:

```
Samson Lane -- A Retrospective
Paintings
   "Mischief No. 19"  (1997)
   "Pants in Blue"  (1999)
   "Evil with a Grin"  (1997)
Sculptures
   "Lick the Hand"  (1999)
   "Freedom"  (1998)
   "Territorial Creature"  (1998)
```

Here is the same catalog with elements tags to categorize its subject matter into a well-formed XML document:

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

```
<ARTISTCAT>
<ARTIST> Samson Lane </ARTIST>
<COLLECTION>
<COLLECTION-TITLE> A Retrospective </COLLECTION-TITLE>
<PAINTINGS> Paintings
   <TITLE> "Mischief No. 19"
      <DATE> 1997 </DATE>
   </TITLE>
   <TITLE> "Pants in Blue"
      <DATE> 1999 </DATE>
   </TITLE>
   <TITLE> "Evil with a Grin
      <DATE> 1997 </DATE>
   </TITLE>
</PAINTINGS>
<SCULPTURES> Sculptures
   <TITLE> "Lick the Hand"
      <DATE> 1999 </DATE>
   </TITLE>
   <TITLE> "Freedom"
      <DATE> 1998 </DATE>
   </TITLE>
   <TITLE> "Territorial Creature"
      <DATE> 1998 </DATE>
   </TITLE>
</SCULPTURES>
</COLLECTION>
</ARTISTCAT>
```

1. Format the Samson Lane catalog with style instructions so that it will be aesthetically pleasing when displayed in browsers. Write the declaration to link a style sheet to your XML file using the syntax given in this module. Then, write some code for the CSS file to which you declared a link. Consider the following points as you code:
   - Your style sheet declaration should link to a CSS text file. Choose a descriptive name for your style sheet, and consider that you might link it to other similar artist catalogs in the future.
   - Your CSS file should include at least two formatting instructions for each element. You can choose any style properties you like, but consider how your style choices might affect the information's visual organization when it appears in a browser. For example, people generally expect a topic heading to appear in larger type than its subordinate information.

   In the text box below, cut and paste your code, and continue on to the next step.

2. XSL is still in the proposal stage. However, this module introduced the proposed syntax for creating XSL construction rules. Using the XML and CSS files you created in Steps 1 and 2, write the code for an XSL file that specifies the construction rule for your first XML element (which can be your root element). Consider the following points as you code:
    - Open and close your XSL file with simple <XSL> tags.
    - Remember that you must specify the beginning and end of a construction rule with the proper syntax.
    - Recall that an XSL construction rule has two components: a pattern and an action. Use comment syntax to call out these two components in your construction rule.
    - The general syntax for creating an XSL construction rule includes a target element (which is empty) in the pattern definition, and a flow object in the action definition.
    - You should also include the empty children element in your action definition. This wildcard represents the content of your target element.

In the text box below, cut and paste your code. When you're ready to submit all your answers to the course tutors, click the OK, I'm Done button.

OK, I'm Done

Lesson 7
Objective

Using wildcards

Use wildcards to increase flexibility and decrease repetition.

There will be times when you will want to use a wildcard representation in your style files to allow for increased flexibility and decreased repetition when defining your elements.

Flip through these frames to see an example of using wildcards:

Transcript

An XML and XSL file combination

`welcome.xml`

```
<MainHead>
Welcome to my Home Page!
</MainHead>
```

`welcome.xsl`

```
<xsl>

<!--begin construction rule-->
<rule>

<!--pattern-->
<target-element type="MainHead"/>

<!--action-->
<DIV font-size="20pt" color="red">
<children/>
</DIV>

</rule>
<!--end construction rule-->

</xsl>
```

Lesson 8

Objective

Qualifier attributes

Use qualifier attributes.

Two special qualifier attributes are available to `target-element`: `position` and `only`. Each has a set of accepted values, enumerated in the table below:

Attribute	Accepted Values

position	`first-of-type`
	`last-of-type`
	`first-of-any`
	`last-of-any`

only	`of-type` `of-any`

This FlipBook demonstrates how to use these attributes:

Transcript

Quiz

Quiz

Click the Quiz button to see what you've learned about using XSL.

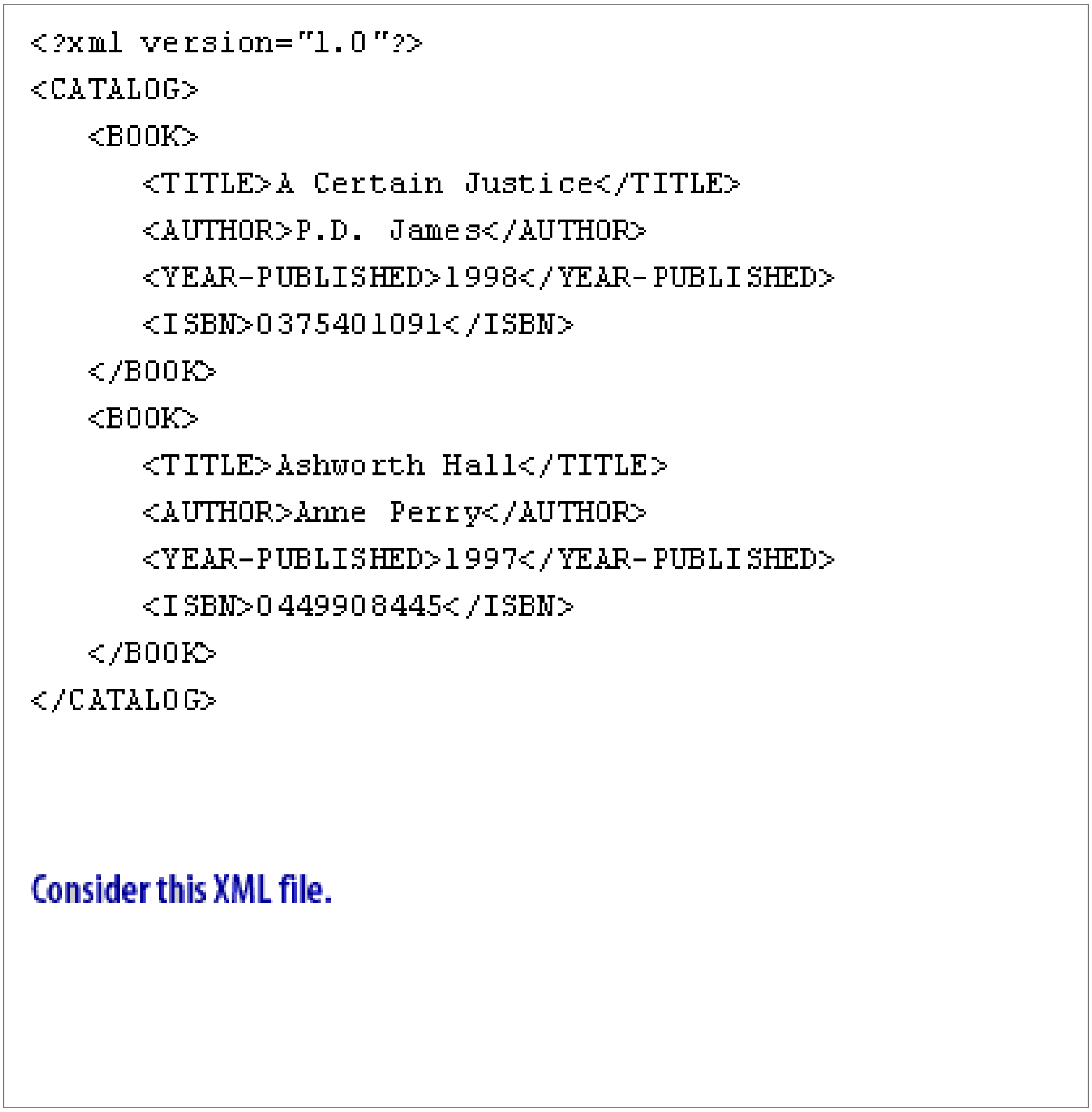
XML Fundamentals

Module 7

Using wildcards

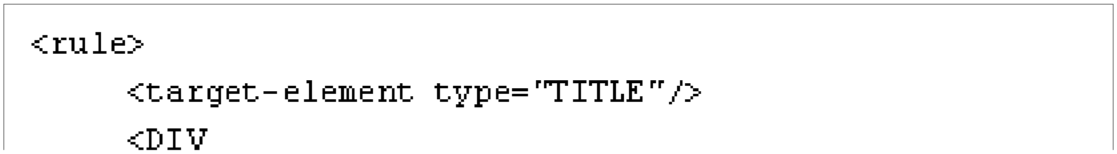
An example of using wildcards (FlipBook transcript)

These frames show an example of using wildcards:

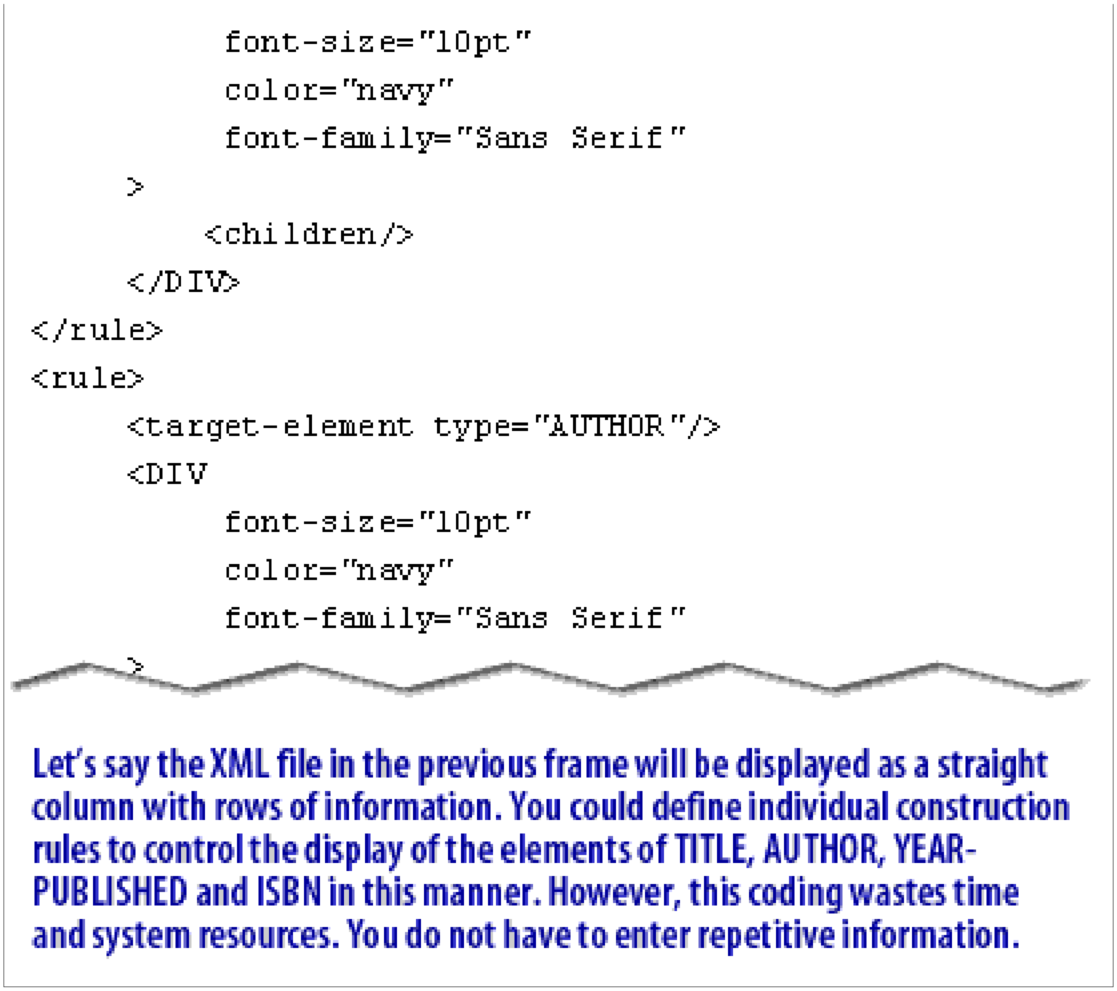


```
<?xml version="1.0"?>
<CATALOG>
   <BOOK>
      <TITLE>A Certain Justice</TITLE>
      <AUTHOR>P.D. James</AUTHOR>
      <YEAR-PUBLISHED>1998</YEAR-PUBLISHED>
      <ISBN>0375401091</ISBN>
   </BOOK>
   <BOOK>
      <TITLE>Ashworth Hall</TITLE>
      <AUTHOR>Anne Perry</AUTHOR>
      <YEAR-PUBLISHED>1997</YEAR-PUBLISHED>
      <ISBN>0449908445</ISBN>
   </BOOK>
</CATALOG>
```

Consider this XML file.

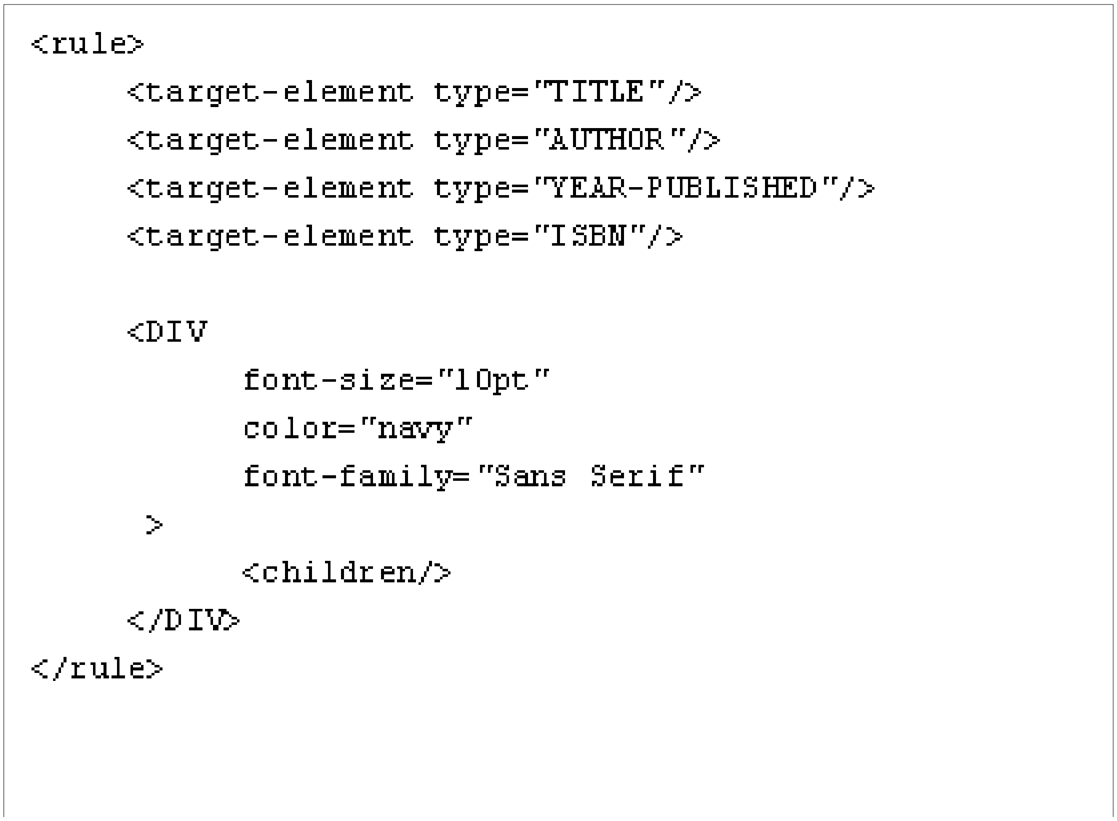


```
<rule>
     <target-element type="TITLE"/>
     <DIV
```

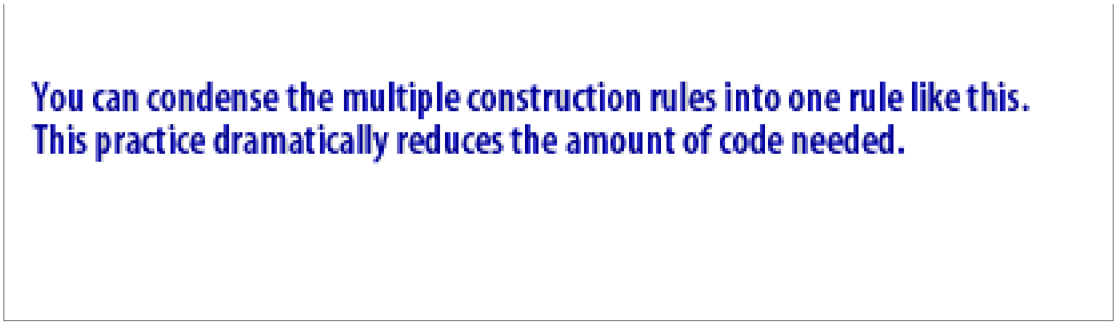


```
          font-size="10pt"
          color="navy"
          font-family="Sans Serif"
     >
         <children/>
     </DIV>
</rule>
<rule>
     <target-element type="AUTHOR"/>
     <DIV
          font-size="10pt"
          color="navy"
          font-family="Sans Serif"
     >
```

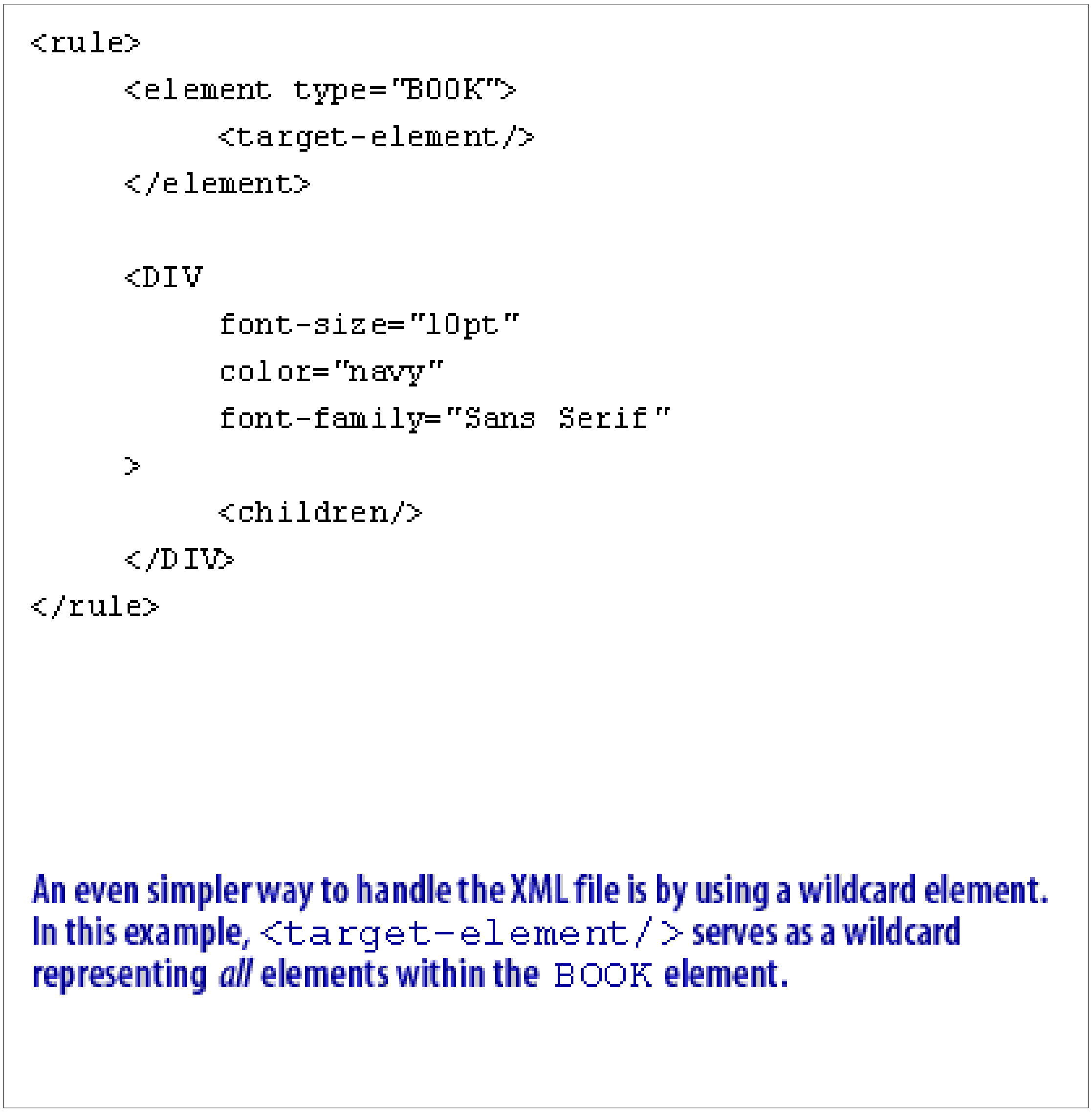
Let's say the XML file in the previous frame will be displayed as a straight column with rows of information. You could define individual construction rules to control the display of the elements of TITLE, AUTHOR, YEAR-PUBLISHED and ISBN in this manner. However, this coding wastes time and system resources. You do not have to enter repetitive information.



```
<rule>
     <target-element type="TITLE"/>
     <target-element type="AUTHOR"/>
     <target-element type="YEAR-PUBLISHED"/>
     <target-element type="ISBN"/>

     <DIV
           font-size="10pt"
           color="navy"
           font-family="Sans Serif"
      >
           <children/>
     </DIV>
</rule>
```

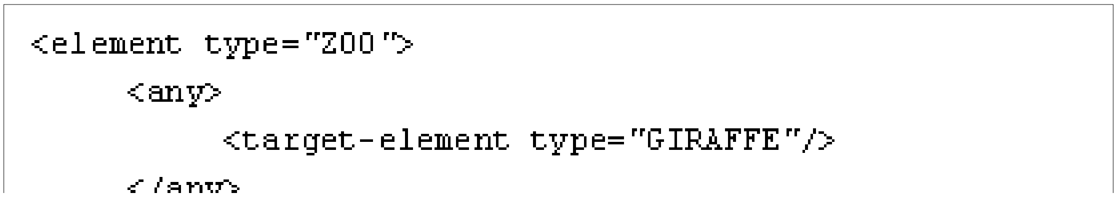


You can condense the multiple construction rules into one rule like this. This practice dramatically reduces the amount of code needed.

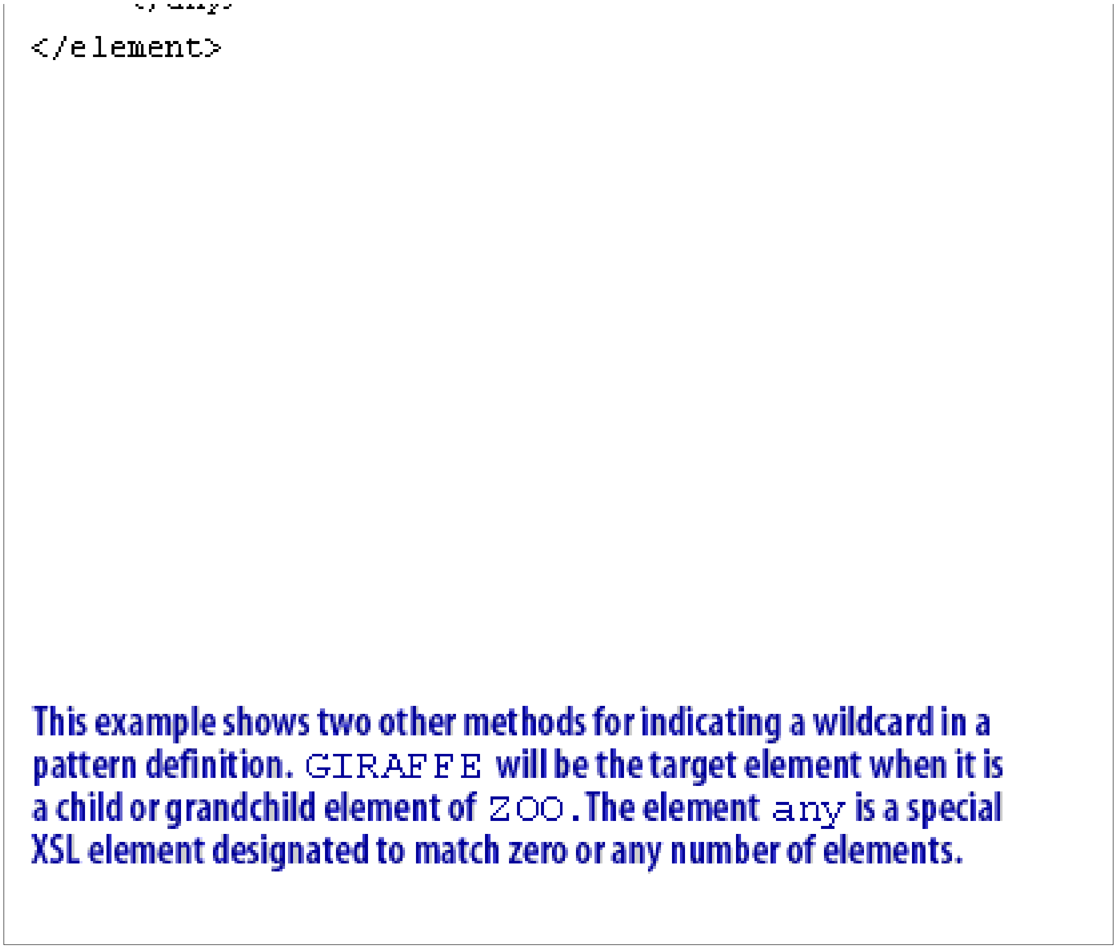


```
<rule>
     <element type="BOOK">
          <target-element/>
     </element>

     <DIV
          font-size="10pt"
          color="navy"
          font-family="Sans Serif"
     >
          <children/>
     </DIV>
</rule>
```

An even simpler way to handle the XML file is by using a wildcard element. In this example, `<target-element/>` serves as a wildcard representing *all* elements within the `BOOK` element.

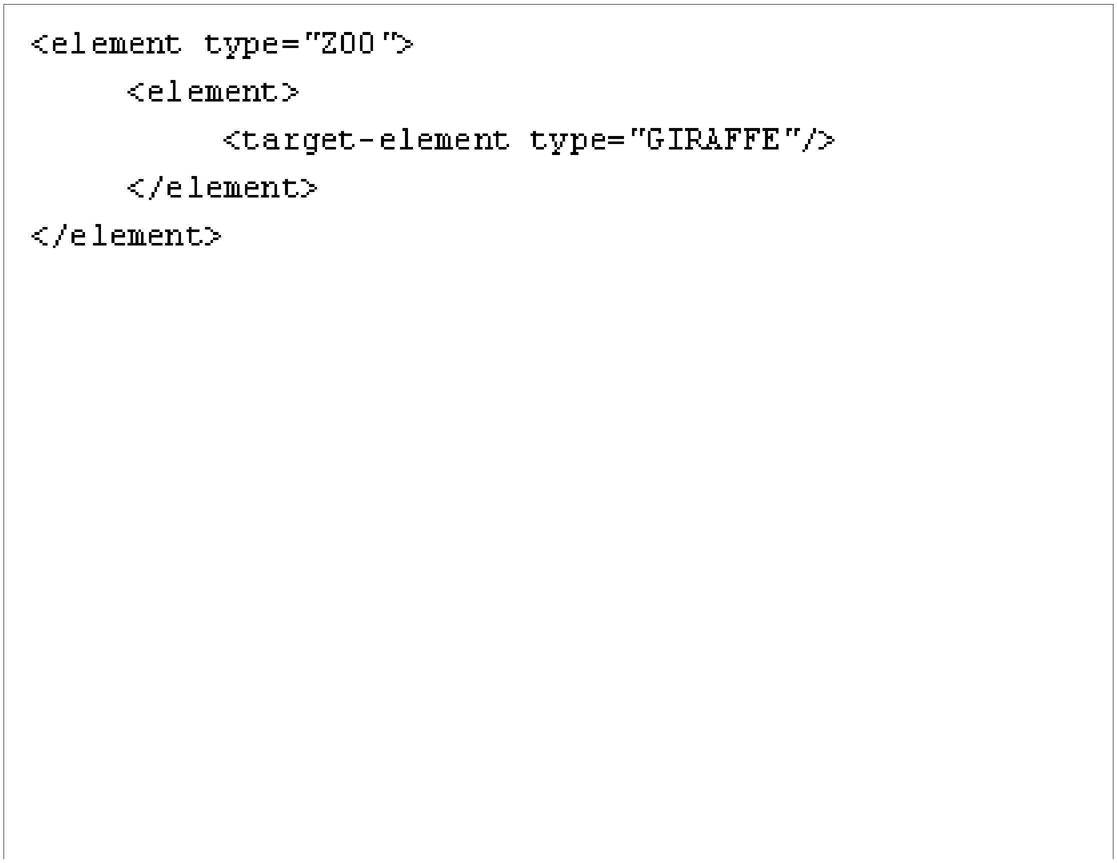


```
<element type="ZOO">
     <any>
          <target-element type="GIRAFFE"/>
     </any>
```

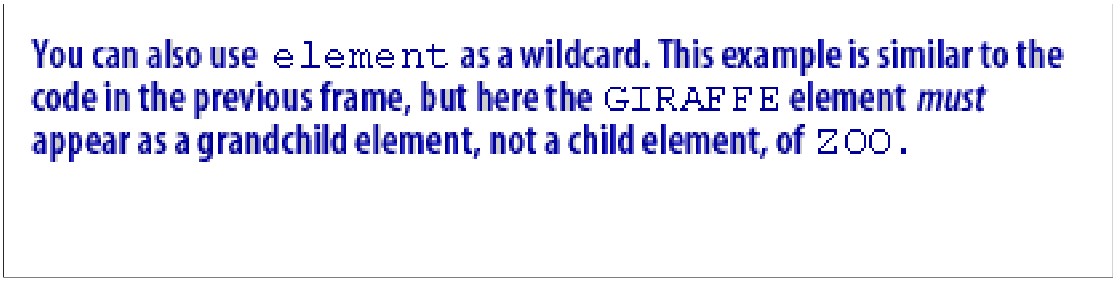


```
     </any>
</element>
```

This example shows two other methods for indicating a wildcard in a pattern definition. `GIRAFFE` will be the target element when it is a child or grandchild element of `ZOO`. The element `any` is a special XSL element designated to match zero or any number of elements.



```
<element type="ZOO">
     <element>
          <target-element type="GIRAFFE"/>
     </element>
</element>
```



You can also use `element` as a wildcard. This example is similar to the code in the previous frame, but here the `GIRAFFE` element *must* appear as a grandchild element, not a child element, of `ZOO`.

Lesson 9
Objective

Attributes

Specify a pattern by referencing an attribute value for an element.

You can specify a pattern by referencing an attribute value for an element:

MouseOver

Transcript

However, the pattern need not select an attribute from the target-element. The match can be made with an attribute from a parent, child or other element:

MouseOver

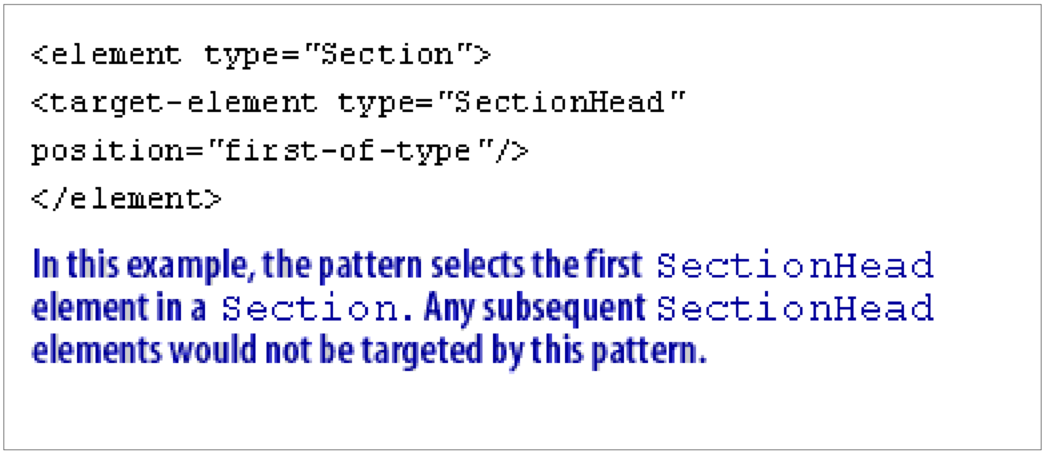
Transcript

Module 7

Qualifier attributes

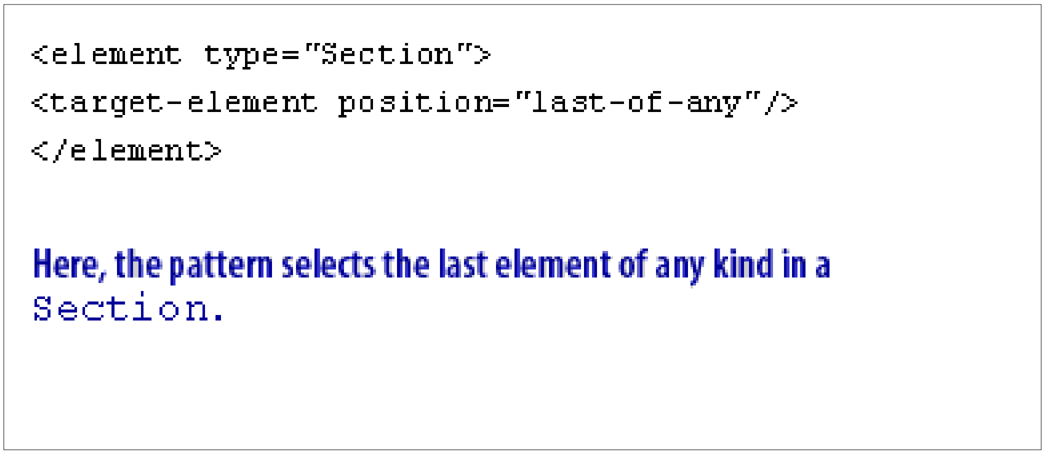
An example of using qualifier attributes (FlipBook transcript)

How qualifier attributes might be used:



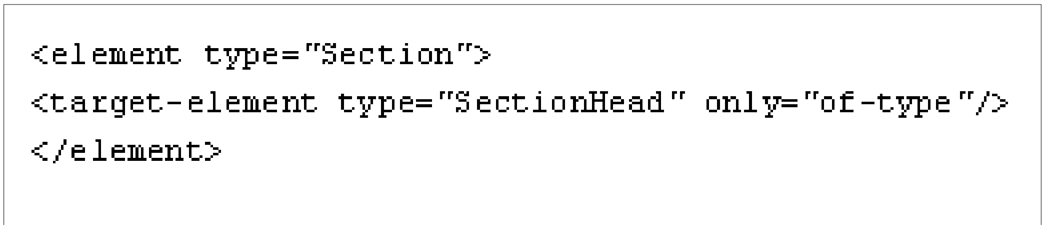
```
<element type="Section">
<target-element type="SectionHead"
position="first-of-type"/>
</element>
```

In this example, the pattern selects the first `SectionHead` element in a `Section`. Any subsequent `SectionHead` elements would not be targeted by this pattern.

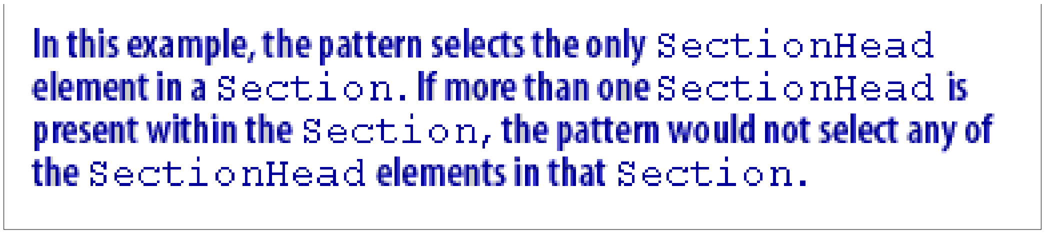


```
<element type="Section">
<target-element position="last-of-any"/>
</element>
```

Here, the pattern selects the last element of any kind in a `Section`.



```
<element type="Section">
<target-element type="SectionHead" only="of-type"/>
</element>
```



In this example, the pattern selects the only `SectionHead` element in a `Section`. If more than one `SectionHead` is present within the `Section`, the pattern would not select any of the `SectionHead` elements in that `Section`.

Module 7

Quiz

Using XSL

Each question is worth one point. Some questions ask you to select **the best answer**, others ask you to select **all the correct answers**. To receive credit for questions asking for **all the correct answers**, you must select all the correct answers and **only** the correct answers.

1.The basic units of XSL files are called:

Please select **the best answer**.

○ A. Element tags
○ B. Construction rules
○ C. Style sheets
○ D. Target elements

2.What are the two components that make up a construction rule?

Please select **the best answer**.

○ A. An action and a target element
○ B. A target element and a style attribute
○ C. A pattern definition and an action
○ D. A target element and a pattern definition
○ E. A style attribute and a value

3.If you want more than one element to share the same style characteristics, you:

Please select **the best answer**.

○ A. Can specify more than one target element in the pattern definition.
○ B. Can specify more than one pattern definition in the construction rule.
○ C. Must create a separate construction rule.
○ D. Must create a separate style sheet.

4.You can use a wildcard in your style files to:

Please select **all the correct answers**.

□ A. Increase repetition.
□ B. Increase flexibility.
□ C. Decrease repetition.
□ D. Decrease flexibility.
□ E. Save time and system resources.

OK, I'm Done

XML Fundamentals

Module 7

Lesson 10 Objective

Rule arbitration

Specify the selection precedence for patterns.

When an element is matched by more than one pattern, the most specific pattern will be given precedence by an XSL processor. In other words, rules do not "cascade" as they do in CSS styles. You must specify the appropriate style information in the rule that will be processed when more than one rule will apply.

The selection precedence is proposed to rank in the order that follows:

1. The pattern with the highest importance value (as specified by an `importance` attribute).
2. The pattern with the greatest number of `id` attributes.
3. The pattern with the greatest number of `class` attributes.
4. The pattern with the greatest number of `element or target-element` elements with a `type` attribute value specified.
5. The pattern with the least number of wildcard elements.
6. The pattern with the highest priority as specified through a `priority` attribute in the rule.
7. The pattern with the greatest number of `only` qualifiers.
8. The pattern with the greatest number of `position` qualifiers.
9. The pattern with the greatest number of `attribute` specifications.

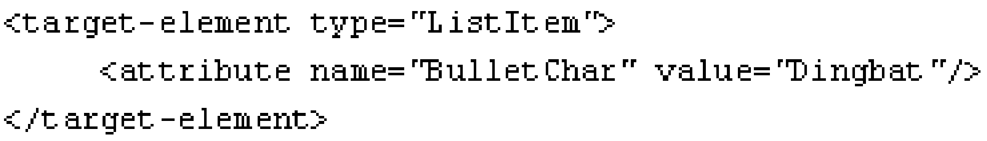
10. If a conflict arises where two rules are determined to be of equal weight in the order of precedence, the rule closer to the end of the XSL file will be used.

XML Fundamentals

Module 7

Attributes

Referencing an attribute value (MouseOver transcript)



```
<target-element type="ListItem">
     <attribute name="BulletChar" value="Dingbat"/>
</target-element>
```

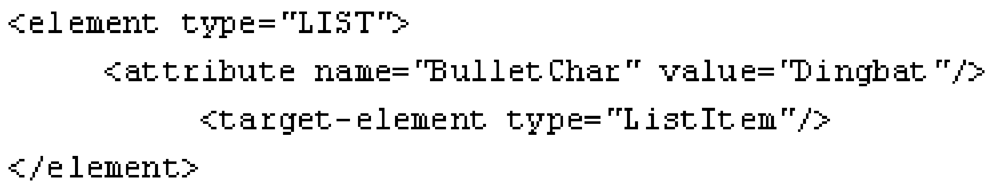
This code would select the `ListItem` element if the `BulletChar` attribute had a value of `"Dingbat."`

XML Fundamentals

Module 7

Attributes

Selecting other attributes (MouseOver transcripts)



```
<element type="LIST">
     <attribute name="BulletChar" value="Dingbat"/>
          <target-element type="ListItem"/>
</element>
```

Similar to the previous code, but this time the `BulletChar` attribute appears in the parent `LIST` element.

XML Fundamentals

Module 7

Lesson 11

Objective

Defining the action with flow objects

Use flow objects to define the visual and behavioral characteristics of elements.

The action part of the construction rule uses what are called *flow objects*--objects from either DSSSL or HTML that help define the visual and behavioral characteristics of the element being styled.

In XSL, flow objects are elements that have predefined meaning in terms of display characteristics and specific usage. You can use XSL to map XML data to an HTML-like format.

According to a submission before the W3C, there is a list of flow objects from HTML that can be used with XSL.

You will want to be sure that any flow object you use conforms to the XML well-formedness constraints. For example, if you want to use the `BR` element as a flow object, you will have to write the tag with the / at the end to indicate that it is an empty tag (`<BR/>`).

XSL also uses several DSSSL flow objects. More flow objects may be introduced that are not derived from HTML or DSSSL. Because the audience for this course will be more familiar with HTML flow objects than DSSSL flow objects, the discussion here will be limited to using HTML flow objects.

In this FlipBook, you'll see how to use flow objects to specify formatting:

Transcript

XML Fundamentals

Module 7

Lesson 12

Objective

The root rule and literal text

Define default characteristics and add literal text to a style sheet.

XSL provides a special `root` element that can be used to define default characteristics of a page and its elements. The following example shows how a default font is set for all the elements on the page:

```
<rule>
   <root/>
   <HTML>
   <BODY font-size="11pt"
      font-weight="normal" color="navy">
   <children/>
   </BODY>
   </HTML>
</rule>
```

Whenever `root` is specified, it represents the target of the action in that rule.

Adding literal text to a style sheet

While allowing for the different contextual selection patterns, what you have seen so far are ways of applying formatting information to elements in a way similar to the application of CSS styles. But XSL provides more power than CSS because it allows you to include additional elements as part of the style definition. You can, for example, add literal text to a *construction rule*, as shown in bold type in the example below:

```
<rule>
   <target-element type="TITLE">

   <DIV
      font-size="10pt"
      color="navy"
      font-family="Sans Serif"
   >
   <literal>The name of this book is: </literal>
      <children/>
   </DIV>
</rule>
```

Exercise

Exercise

In this exercise, you'll practice writing code to create XSL style rules for an XML document.

Defining the action with flow objects

Elements in HTML

Think of the differences between the `SPAN`, `DIV`, and `TD` elements from HTML. Each of these elements sits differently within an HTML-like page. A `DIV` element is usually a block element that occupies the document margin to margin, although it can be "floated" left or right, causing text to wrap around it and creating a different "flow."

The `SPAN` element is an inline element that has no effect on the flow of text on the page, and is used simply to span formatting information across a character or set of characters. The `TD` element is a special kind of flow object, creating a table cell that may be preceded or followed by another table cell.

XML Fundamentals

Module 7

Defining the action with flow objects

HTML flow objects available to XSL

The following is a list of flow objects from HTML that can be used with XSL, according to one submission before the W3C:

- `HTML`
- `TITLE`
- `META`
- `BASE`
- `SCRIPT`
- `BODY`
- `PRE`
- `DIV`
- `BR`
- `SPAN`
- `A`
- `FORM`
- `INPUT`
- `SELECT`
- `TEXTAREA`
- `TABLE`
- `CAPTION`
- `COL`
- `COLGROUP`
- `THEAD`
- `TBODY`
- `TFOOT`
- `TR`
- `TD`
- `HR`
- `IMG`
- `MAP`
- `AREA`
- `OBJECT`
- `PARAM`

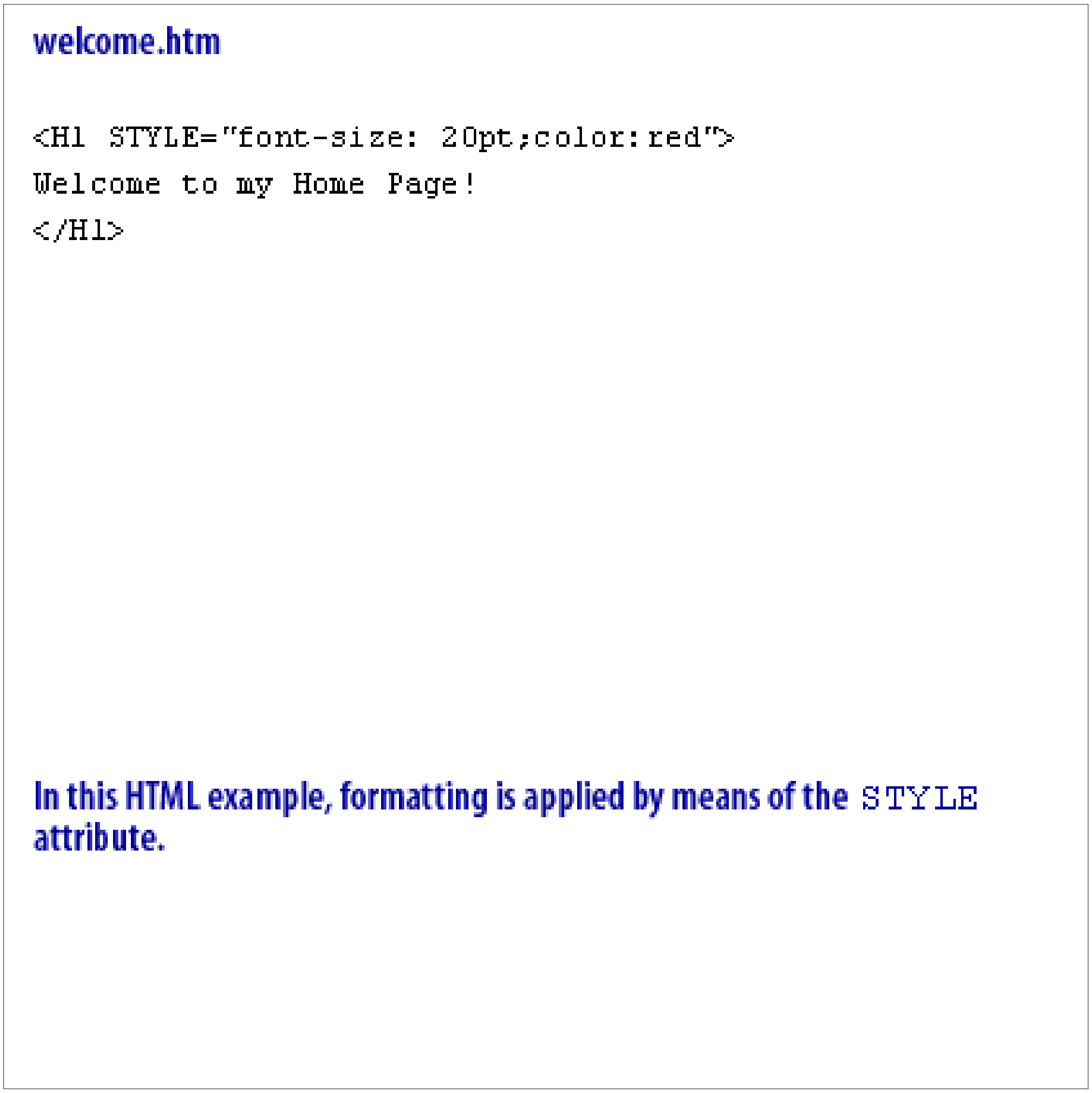
XML Fundamentals

Module 7

Defining the action with flow objects

An example of using flow objects (FlipBook transcript)

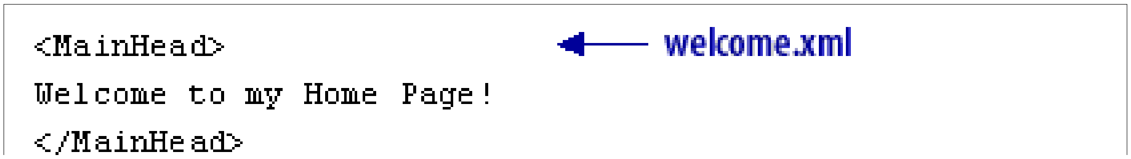
Here's an example of how flow objects can be used to specify formatting:



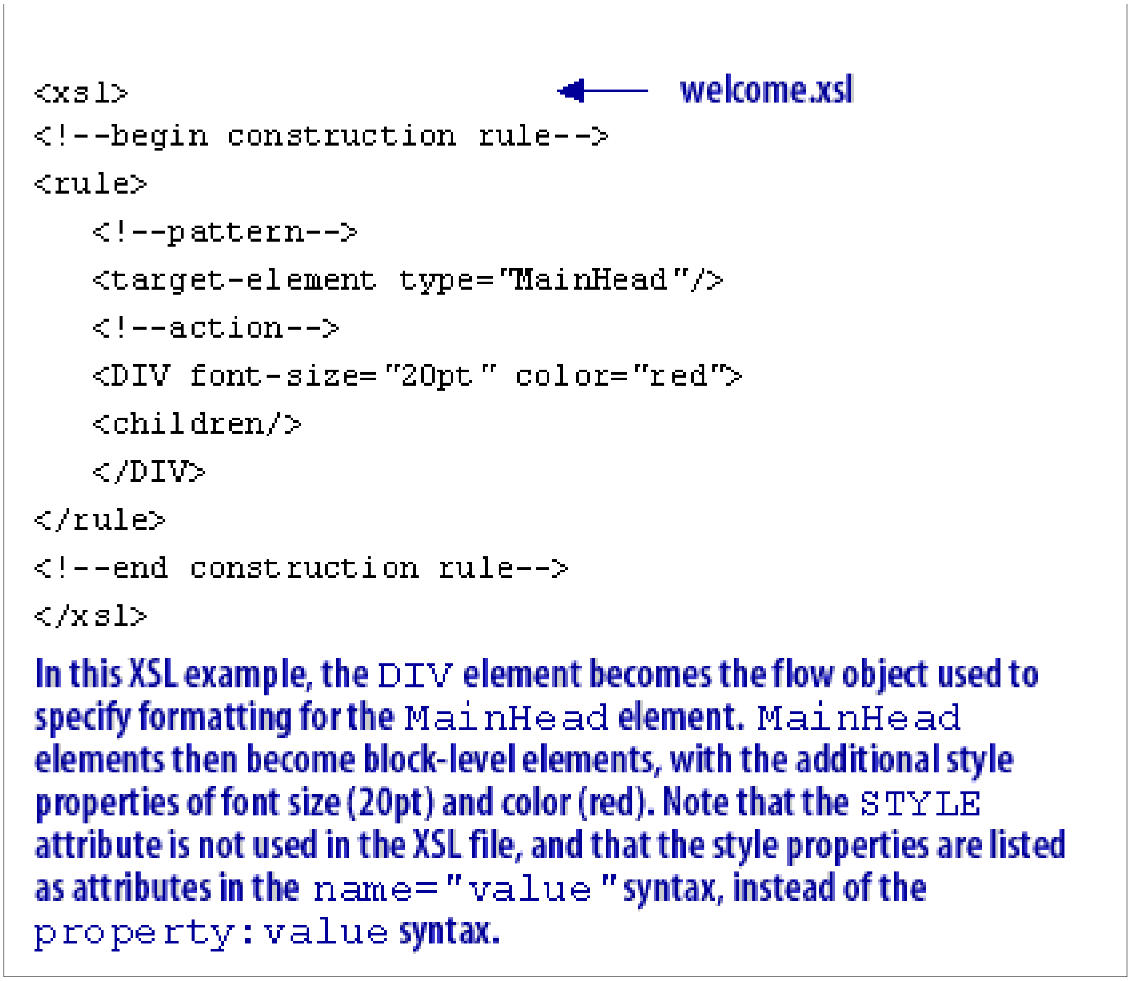
welcome.htm

```
<H1 STYLE="font-size: 20pt;color:red">
Welcome to my Home Page!
</H1>
```

In this HTML example, formatting is applied by means of the STYLE attribute.



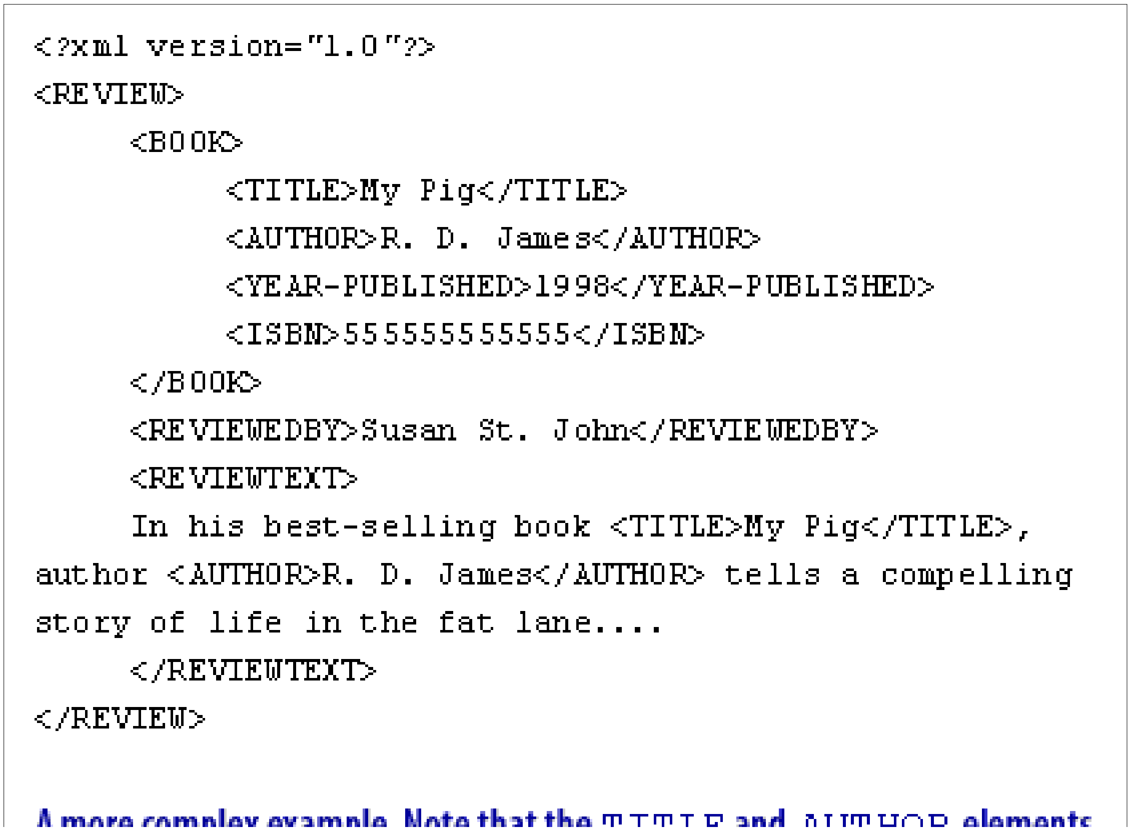
```
<MainHead>
Welcome to my Home Page!
</MainHead>
```



welcome.xsl

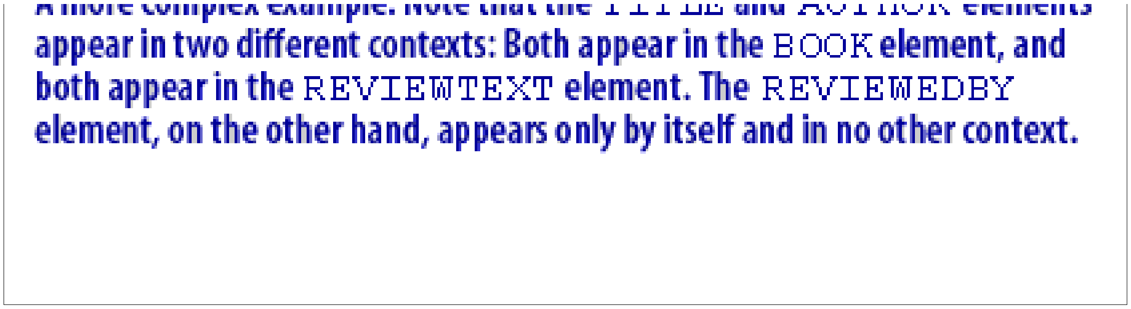
```
<xsl>
<!--begin construction rule-->
<rule>
   <!--pattern-->
   <target-element type="MainHead"/>
   <!--action-->
   <DIV font-size="20pt" color="red">
   <children/>
   </DIV>
</rule>
<!--end construction rule-->
</xsl>
```

In this XSL example, the `DIV` element becomes the flow object used to specify formatting for the `MainHead` element. `MainHead` elements then become block-level elements, with the additional style properties of font size (20pt) and color (red). Note that the `STYLE` attribute is not used in the XSL file, and that the style properties are listed as attributes in the `name="value"` syntax, instead of the `property:value` syntax.

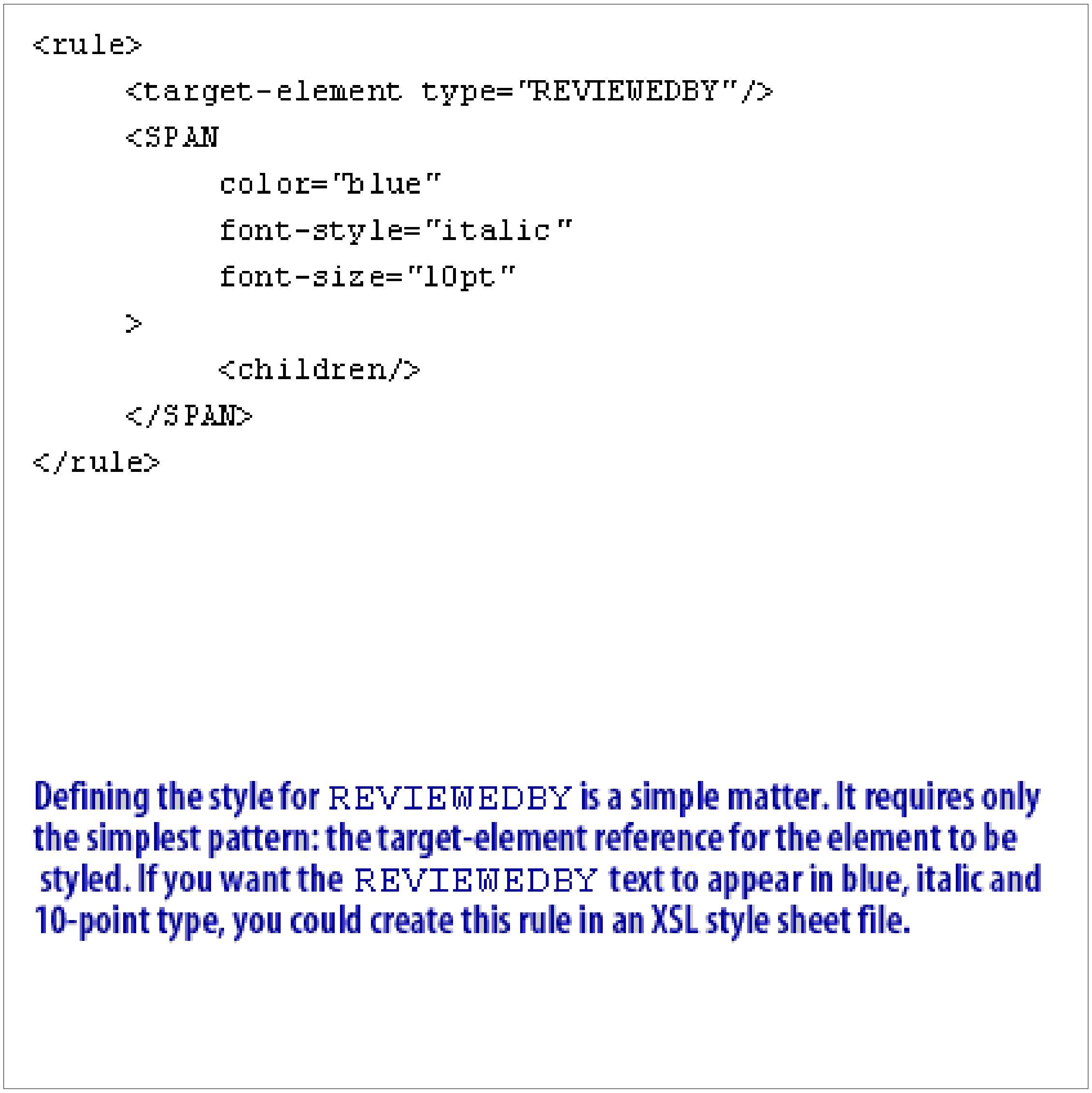


```
<?xml version="1.0"?>
<REVIEW>
     <BOOK>
          <TITLE>My Pig</TITLE>
          <AUTHOR>R. D. James</AUTHOR>
          <YEAR-PUBLISHED>1998</YEAR-PUBLISHED>
          <ISBN>555555555555</ISBN>
     </BOOK>
     <REVIEWEDBY>Susan St. John</REVIEWEDBY>
     <REVIEWTEXT>
     In his best-selling book <TITLE>My Pig</TITLE>,
author <AUTHOR>R. D. James</AUTHOR> tells a compelling
story of life in the fat lane....
     </REVIEWTEXT>
</REVIEW>
```

A more complex example. Note that the `TITLE` and `AUTHOR` elements

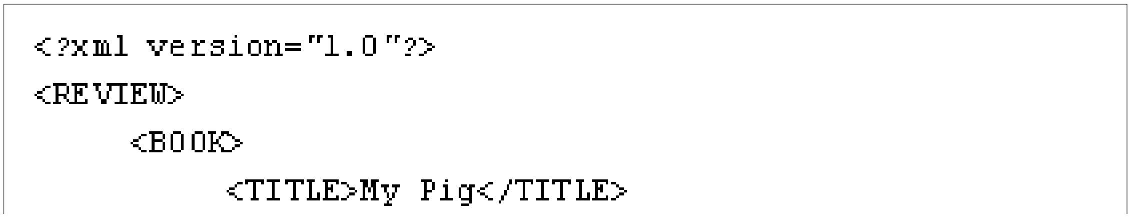


A more complex example. Note that the TITLE and AUTHOR elements appear in two different contexts: Both appear in the BOOK element, and both appear in the REVIEWTEXT element. The REVIEWEDBY element, on the other hand, appears only by itself and in no other context.

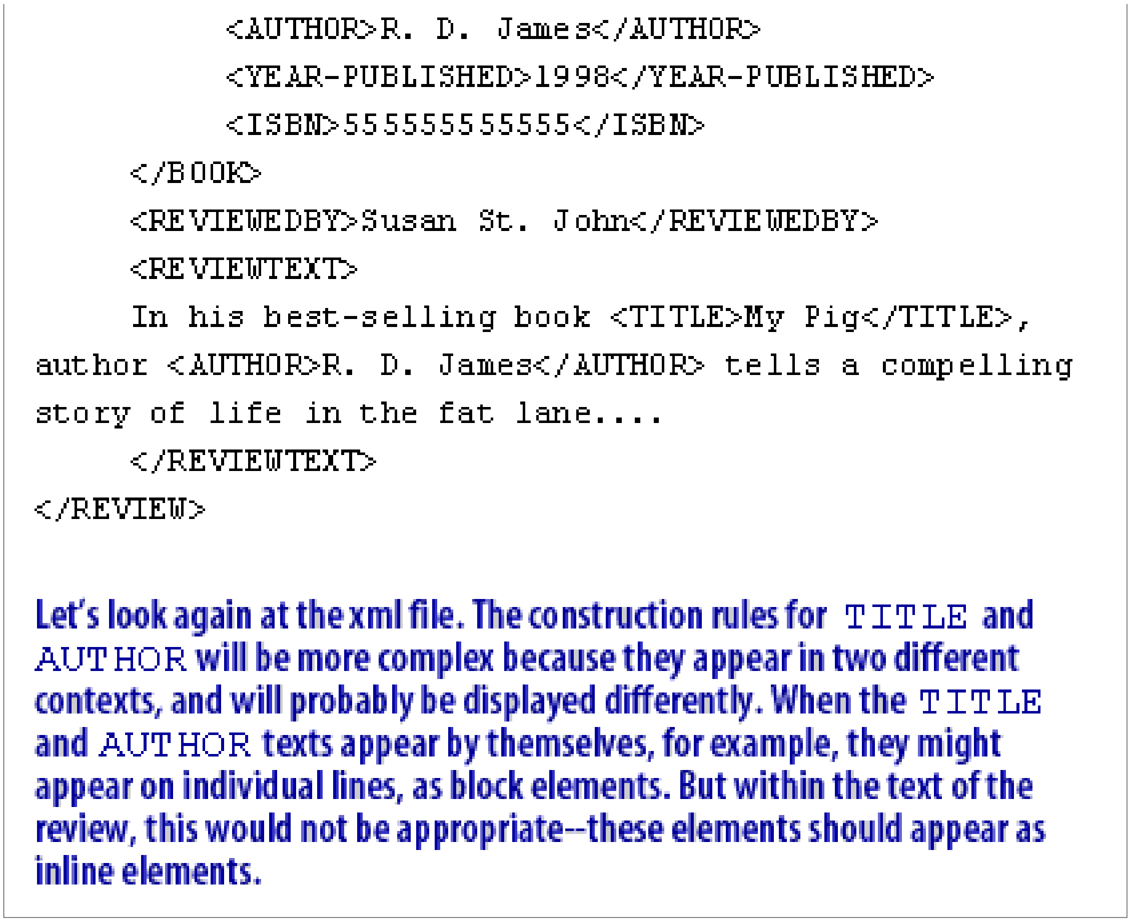


```
<rule>
     <target-element type="REVIEWEDBY"/>
     <SPAN
          color="blue"
          font-style="italic"
          font-size="10pt"
     >
          <children/>
     </SPAN>
</rule>
```

Defining the style for REVIEWEDBY is a simple matter. It requires only the simplest pattern: the target-element reference for the element to be styled. If you want the REVIEWEDBY text to appear in blue, italic and 10-point type, you could create this rule in an XSL style sheet file.



```
<?xml version="1.0"?>
<REVIEW>
     <BOOK>
          <TITLE>My Pig</TITLE>
```



```
          <AUTHOR>R. D. James</AUTHOR>
          <YEAR-PUBLISHED>1998</YEAR-PUBLISHED>
          <ISBN>555555555555</ISBN>
     </BOOK>
     <REVIEWEDBY>Susan St. John</REVIEWEDBY>
     <REVIEWTEXT>
     In his best-selling book <TITLE>My Pig</TITLE>,
author <AUTHOR>R. D. James</AUTHOR> tells a compelling
story of life in the fat lane....
     </REVIEWTEXT>
</REVIEW>
```

Let's look again at the xml file. The construction rules for TITLE and AUTHOR will be more complex because they appear in two different contexts, and will probably be displayed differently. When the TITLE and AUTHOR texts appear by themselves, for example, they might appear on individual lines, as block elements. But within the text of the review, this would not be appropriate--these elements should appear as inline elements.

Lesson 13
Objective

XSL and Internet Explorer 5.0

Use XSL with IE 5.0.

Internet Explorer 5.0 fully supports the W3C WD2 dated December 1998. It also supports CSS for XML formatting. Under this new proposal, XSL enables you to create formatting templates comparable to CSS rules, which can be applied to XML data to define format.

View MouseOver
Transcript

Click the View MouseOver button to explore an XML and XSL file combination coded for IE 5.0. This MouseOver will show you how to use XSL with IE 5.0 in accordance with the W3C WD2.

View Image

Click the View Code button to see how the XML and XSL file combination would render in IE 5.0.

XML Fundamentals

Module 7

Exercise

The root rule and literal text

Using XSL

Objective

Practice writing code to create XSL style rules for an XML document.

Exercise Scoring

Full credit for this exercise is 18 points (3 points for each step). You'll submit your code to the course tutors.

Starter files

To help you complete this exercise, we've supplied a starter file in the 07-12 folder of the course download available from the Resources page.

Instructions

In this exercise, you will practice writing code to create XSL style rules for an XML document. XSL is still in the proposal stage. However, this module introduced the proposed syntax for creating XSL construction rules.

Consider the following XML document:

```
<?xml version="1.0"?>
<ARTISTCAT>
<ARTIST> Samson Lane </ARTIST>
<COLLECTION>
<COLLECTION-TITLE> A Retrospective </COLLECTION-TITLE>
<PAINTINGS> Paintings
   <TITLE> "Mischief No. 19"
      <DATE> 1997 </DATE>
   </TITLE>
   <TITLE> "Pants in Blue"
      <DATE> 1999 </DATE>
   </TITLE>
   <TITLE> "Evil with a Grin
      <DATE> 1997 </DATE>
   </TITLE>
</PAINTINGS>
<SCULPTURES> Sculptures
   <TITLE> "Lick the Hand"
      <DATE> 1999 </DATE>
   </TITLE>
   <TITLE> "Freedom"
      <DATE> 1998 </DATE>
   </TITLE>
   <TITLE> "Territorial Creature"
      <DATE> 1998 </DATE>
   </TITLE>
</SCULPTURES>
</COLLECTION>
</ARTISTCAT>
```

1. In another exercise, you created an XSL construction rule for your first XML element using the syntax proposed in this module. Now, write code for an XSL file that specifies construction rules for the XML elements in this document. Consider the following points as you code:
   - Open and close your XSL file with simple `<XSL>` tags.
   - Remember that you must specify the beginning and end of a construction rule with the proper syntax. You should have a construction rule for each of your elements.
   - Recall that an XSL construction rule has two components: a pattern and an action. Use comment syntax to call out these two components in your first construction rule.
   - The general syntax for creating an XSL construction rule includes a target element (which is empty) in the pattern definition, and a flow object in the action definition.
   - You should also include the empty `children` element in your action definition. This wildcard represents the content of your target element.
   - Many flow objects are available through HTML. Feel free to use any that seem appropriate.

   In the text box below, cut and paste your code, and continue on to the next step.

2. The pattern you defined in Step 1 for the `TITLE` element indicates that every occurrence of `TITLE` will receive the action specified. Suppose you want to apply the action only to those titles that fall within the `PAINTINGS` element. Write the code for a construction rule that indicates your style properties apply only to `TITLE` elements within the `PAINTINGS` element. Consider the

following points as you code:

- Your target element should remain the same, but indicate that it must be within the `PAINTINGS` element to receive the action of the rule.
- Write the code for this *entire* construction rule.

In the text box below, cut and paste your code, and continue on to the next step.

3. The pattern you defined in Step 2 indicates that you want to apply the action only to `TITLE` elements that fall within the `PAINTINGS` element. Now suppose you want to write code that specifies a pattern based on a descendant element rather than an ancestor. Write the code for the construction rule that indicates to show that the indicated style properties apply to `TITLE` elements only when they contain a `DATE` element. Consider the following points as you code:
   - `TITLE` will be your target element, but it will receive the action of the rule only when the `DATE` element is present within it.
   - Write the code for this *entire* construction rule.

   In the text box below, cut and paste your code, and continue on to the next step.

4. The `PAINTINGS` and `SCULPTURES` elements in this XML document carry equivalent weight. Therefore, these two elements probably use the same style properties. Write the code to specify both `PAINTINGS` and `SCULPTURES` as the target elements so they can share one set of style characteristics. Consider the following points as you code:
   - You can specify one set of style properties for multiple target elements in a single construction rule.
   - As long as neither element contains the other, the order in which you specify the target elements does not matter. Both will receive the formatting instructions.
   - Write the code for this *entire* construction rule.

In the text box below, cut and paste your code, and continue on to the next step.

5. In Step 4 you specified two target elements in your pattern to receive the action. Now rewrite that construction rule to use a wildcard element that reduces the amount of code needed for the same action. Consider the following points as you code:
    - Both the `PAINTINGS` and `SCULPTURES` elements are contained within another element. Recall the code you used in Step 2 to specify a target element within another element.
    - Consider that you want to specify all elements within another element.
    - Think about how the `CHILDREN` element is used to represent the content of your target element. The syntax for this operation is similar.

In the text box below, cut and paste your code, and continue on to the next step.

6. XSL allows you to include additional elements (such as literal text or script) as part of your style definition. Write the code to add some literal text to the construction rule for the `TITLE` or `DATE` element. Consider the following points as you code:
    - Choose simple text that will apply to all `TITLE` elements or `DATE` elements.
    - Assume that the literal text will be inserted before the element text.
    - Consider whether a literal text element is added to the pattern definition or the action definition, and where it is placed in relation to other properties.

In the text box below, cut and paste your code. When you're ready to submit all your answers to the course tutors, click the OK, I'm Done button.

OK, I'm Done

XML Fundamentals

Module 7

Lesson 14

Styling XML files

Module wrap-up

In this module, you learned how to define simple rules that specify patterns of data to receive formatting. You also learned formatting actions defined by HTML flow objects.

In addition, you glimpsed the potential power of XSL in that it allows the addition of literal text and scripting functions within the style construction rules. What you have learned so far will carry you a long way toward creating functional and attractively displayable XML files.

This module discusses the following terms in relation to XML:

- *style sheet*
- *XSL*
- *construction rule*
- *flow objects*

Discussion

Do you have any questions or comments about the material in this module? If so, don't forget about the Related XML technologies folder in the Discussion section, which you can reach by clicking the Discuss button in the toolbar.

Quiz

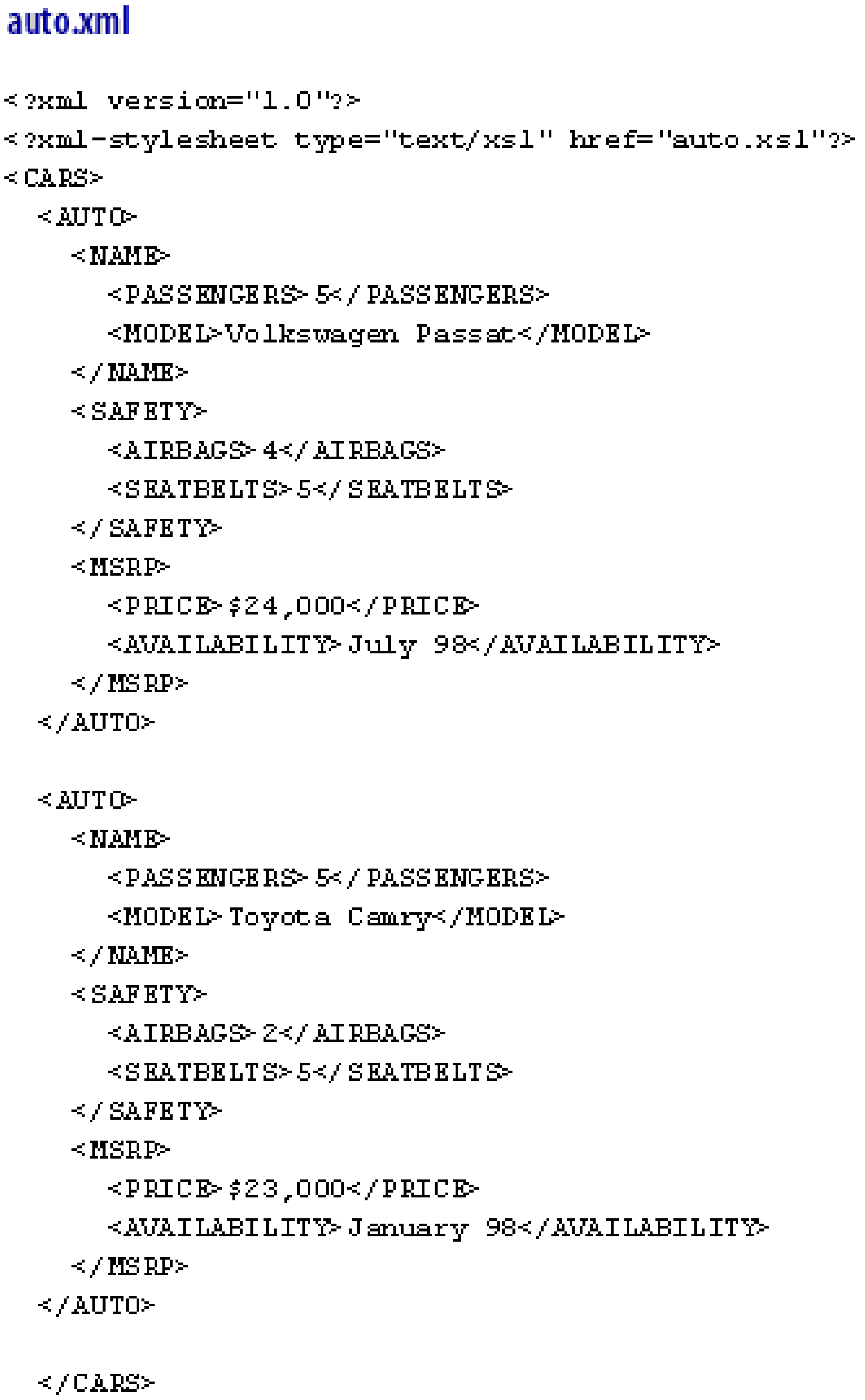
Quiz

Click the Quiz button to test what you've learned in this module.

MouseOver

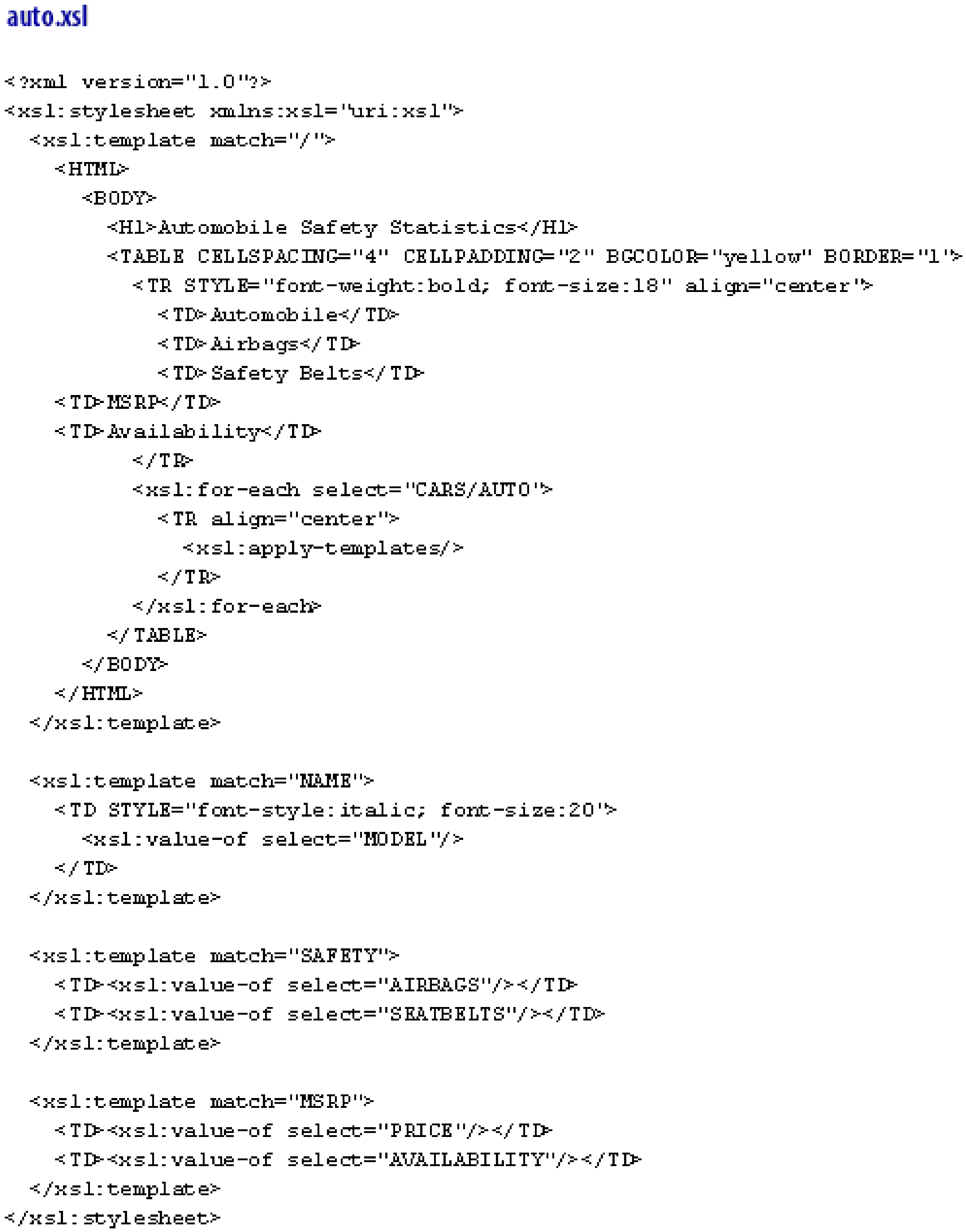
XSL and Internet Explorer 5.0

An XML and XSL file combination for IE 5.0 (MouseOver transcript)



auto.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="auto.xsl"?>
<CARS>
  <AUTO>
    <NAME>
      <PASSENGERS>5</PASSENGERS>
      <MODEL>Volkswagen Passat</MODEL>
    </NAME>
    <SAFETY>
      <AIRBAGS>4</AIRBAGS>
      <SEATBELTS>5</SEATBELTS>
    </SAFETY>
    <MSRP>
      <PRICE>$24,000</PRICE>
      <AVAILABILITY>July 98</AVAILABILITY>
    </MSRP>
  </AUTO>

  <AUTO>
    <NAME>
      <PASSENGERS>5</PASSENGERS>
      <MODEL> Toyota Camry</MODEL>
    </NAME>
    <SAFETY>
      <AIRBAGS>2</AIRBAGS>
      <SEATBELTS>5</SEATBELTS>
    </SAFETY>
    <MSRP>
      <PRICE>$23,000</PRICE>
      <AVAILABILITY>January 98</AVAILABILITY>
    </MSRP>
  </AUTO>

  </CARS>
```



auto.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="uri:xsl">
  <xsl:template match="/">
    <HTML>
      <BODY>
        <H1>Automobile Safety Statistics</H1>
        <TABLE CELLSPACING="4" CELLPADDING="2" BGCOLOR="yellow" BORDER="1">
          <TR STYLE="font-weight:bold; font-size:18" align="center">
            <TD>Automobile</TD>
            <TD>Airbags</TD>
            <TD>Safety Belts</TD>
    <TD>MSRP</TD>
    <TD>Availability</TD>
          </TR>
          <xsl:for-each select="CARS/AUTO">
            <TR align="center">
              <xsl:apply-templates/>
            </TR>
          </xsl:for-each>
        </TABLE>
      </BODY>
    </HTML>
  </xsl:template>

  <xsl:template match="NAME">
    <TD STYLE="font-style:italic; font-size:20">
      <xsl:value-of select="MODEL"/>
    </TD>
  </xsl:template>

  <xsl:template match="SAFETY">
    <TD><xsl:value-of select="AIRBAGS"/></TD>
    <TD><xsl:value-of select="SEATBELTS"/></TD>
  </xsl:template>

  <xsl:template match="MSRP">
    <TD><xsl:value-of select="PRICE"/></TD>
    <TD><xsl:value-of select="AVAILABILITY"/></TD>
  </xsl:template>
</xsl:stylesheet>
```

auto.xsl (title)

This entire example applies four templates: one each for the ROOT element, the NAME element, the SAFETY element and the MSRP element.

<xsl:stylesheet xmlns:xsl="uri:xsl">

Note the declaration of "xmlns". This declaration refers to the XSL namespace, and it is required.

<xsl:template match="/">

This element uses the slash (/) for pattern-matching criteria, which will apply formatting to the root element (in this case, CARS).

<xsl:for-each select="CARS/AUTO">

This code directs the template to create a pattern for each instance of AUTO and apply the declared formatting.

<xsl:apply-templates/>

This element is an empty tag used as a data flow object, allowing data to be inserted in its place.

<xsl:template match="NAME">

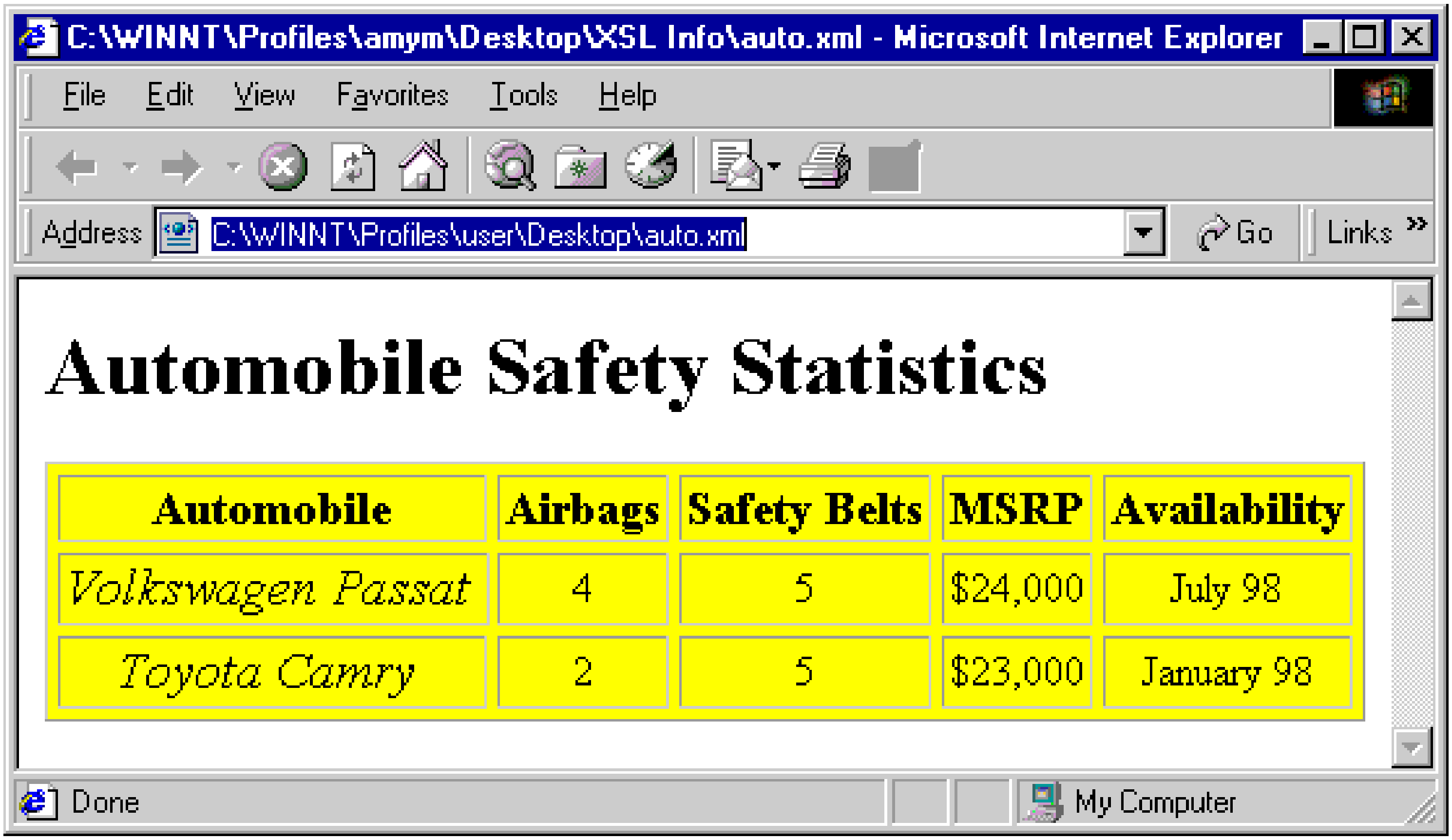
This tag replaces the construction rule as specified in WD2. In auto.xsl, this element looks for the pattern of NAME as a child element of AUTO. It then inserts the value of the MODEL element, which is a child of the NAME element, by calling <xsl:value-of select="MODEL"/>.

<xsl:template match="SAFETY">

This element looks for the SAFETY pattern of each AUTO. The SAFETY child elements, which are AIRBAGS and SEATBELTS, are then called for their values by the <xsl:value-of select="AIRBAGS"/> and <xsl:value-of select="SEATBELTS"/> tags.

<xsl:value-of select="AIRBAGS"/>

This tag is the flow object (as opposed to stand alone <children/>)


C:\WINNT\Profiles\amym\Desktop\XSL Info\auto.xml - Microsoft Internet Explorer
File
Edit
View
Favorites
Tools
Help
Address
C:\WINNT\Profiles\user\Desktop\auto.xml
Go
Links
Done
My Computer

Automobile Safety Statistics

Automobile	Airbags	Safety Belts	MSRP	Availability
Volkswagen Passat	4	5	$24,000	July 98
Toyota Camry	2	5	$23,000	January 98

The five columns of information result from the four templates that were declared and applied. Each template applies a series of looking "for-each" instance (a pattern) and gathering the "value-of" each element. After these processes are complete, the style sheet is closed.

XML Fundamentals

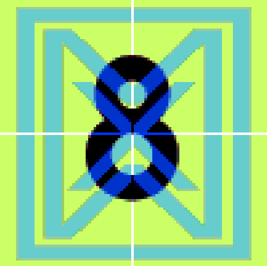
Module 8

Lesson 1

XML today and in the future

Module introduction

This module shows you how XML is used today, and discusses how it's likely to be used in the future.



Module learning objectives

By the end of the module, you will have the skills and knowledge necessary to:

- Explore current support for XML
- Use simple databinding to display contents of an XML file.
- Describe examples of XML use
- Consider a popular adoption scenario for the future of XML
- Assess XML's impact on HTML
- Consider potential e-commerce uses for XML
- Explain how XML provides information interchange between databases and information systems.

Module 7 Quiz

Styling XML files

Each question is worth one point. Some questions ask you to select **the best answer**, others ask you to select **all the correct answers**. To receive credit for questions asking for **all the correct answers**, you must select all the correct answers and **only** the correct answers.

1.What happens when an element is matched by more than one pattern?

Please select **all the correct answers**.

□ A. The rules "cascade" as they do in CSS styles.

□ B. Both rules will be applied to the element simultaneously.

□ C. Selection precedence is established based on a proposed order of rank.

□ D. The XSL processor gives precedence to the first rule it sees.

□ E. The XSL processor gives precedence to the most specific pattern.

2.In XSL, a "flow object" is an element that:

Please select **all the correct answers**.

□ A. Came from either DSSSL or HTML

□ B. Can be used to map XML data to an HTML-like format

□ C. Is used in the action part of the construction rule to define content

□ D. Is used in the pattern part of the construction rule to define visual or behavioral characteristics.

□ E. Has user-defined meaning in terms of display characteristics and usage

3.XSL provides a special element called `root` that is used to:

Please select **the best answer**.

○ A. Contain all other elements within the style sheet

○ B. Define several elements in a style sheet all at once

○ C. Serve as the father for all child elements in a style sheet

○ D. Define default style characteristics for a page and its elements

4.XSL provides more power than CSS because it allows you to:

Please select **all the correct answers**.

□ A. Include literal text in a construction rule

- □ B. Add formatting information to elements
- □ C. Include script as part of the style definition
- □ D. Define multiple style properties in the same style sheet

OK, I'm Done

XML Fundamentals

Module 8

Lesson 2

Objective

Supporting XML

Explore current support for XML.

Netscape Communicator 5.0: Native Support for XML

Netscape Communications has opened a site at http://www.mozilla.org in an effort to put its popular browser code in the public domain. When the first release of the code for what would be Communicator 5.0 was made public, industry observers noted that Netscape has already incorporated native support for XML. The browser includes an XML parser, as well as support for XLink and the namespaces initiative. In addition, Netscape has designed its parser to work with the CSS method of describing style in conjunction with XML, instead of the proposed XSL method. You can follow developments on Netscape's efforts at the aforementioned site.

Microsoft Internet Explorer 5.0

According to Microsoft, the 5.0 version of the Internet Explorer browser includes significant updates to XML-related components offered in earlier beta releases and in the 4.0 version.

Microsoft's latest browser does natively support XML 1.0 and XSL Working Draft 2. However, Microsoft developed both an applet and an ActiveX control that enabled the display of XML files within Internet Explorer 4.0. Microsoft also provided an XSL ActiveX control to enable the use of XSL style sheets with XML files in IE 4.x.

Document Object Model (DOM)

The W3C is currently reviewing the *Document Object Model* (DOM) for HTML, which will affect the DOM for XML as well. Once it is specified, and once XSL is accepted as a standard, the path for widespread acceptance of XML as the native language for exchanging data across the Web will be clear.

XML Fundamentals

Module 8

Lesson 3

Objective

Databinding XML to HTML

Use simple databinding to display contents of an XML file.

Internet Explorer 4.0 ships with a Java applet (called XMLDSO) that allows you to bind the contents of your XML file to an HTML file. Once the data is bound, you can display the information through the mechanism of HTML. This is perhaps the simplest way to view a straightforward set of XML data within IE 4.x. In this lesson's optional exercises, you can use simple databinding to display the contents of an XML file.

You can also use simple databinding to display the contents of an XML file in Internet Explorer 5.0. The XMLDSO Java applet is not required to display XML in 5.0.

Warning

If you are using a Netscape browser, you can complete either exercise, but the contents of the XML files will not render in your browser. We've provided graphics so you can see how the documents would appear in IE.

Exercise

Exercise
Use simple databinding to display the contents of an XML file in Internet Explorer 4.0.

Exercise

Exercise
Use simple databinding to display the contents of an XML file in Internet Explorer 5.0.

- **The Mozilla Organization**
  - Our Mission
  - Who We Are
  - Getting Involved
  - Community
  - Editorials
  - What's New
  - Newsbot
- **Development**
  - Roadmap
  - Module Owners
  - Blue Sky
  - Projects
  - Status
  - Tools
- **Products**
  - Source Code
  - Binaries
  - Documentation
  - License Terms
  - Bug Reports
  - Quality
- **Search**
- **Feedback**

Status Update — 18 July 99

This week's update contains news on NGLayout, Mail/News, Javascript, and XPConnect Please be sure to check out the just added XPToolkit update from the previous week.

more...

Mozilla News — 16 Jul 99

M8 On The Wire

Milestone 8 binaries have been released. Thanks to Duncan Wilcox and Pete Collins for providing BeOS and FreeBSD packages. As usual, use bugzilla to report bugs.

Mozilla for BeOS

The BeZilla project has just released a development kit based on M7.

BugAThon 300

Looking for a way to help? Join the BugAThon 300! There are over 300 open layout bugs. Help out by decomposing these bugs into simple test cases. Win valuable prizes!

New Newsgroup Announcements

A newsgroup/mailing-list pair for the discussion of Seamonkey (the v5.0 Browser) has been created. Additionally, a forum for the discussion of (non-crypto-related) security was created a while back, but not splashed here. Here is a link to more details about all the Mozilla discussion groups.

more...

XML Fundamentals

Module 8

Supporting XML

Comparing IE browser versions 4.0 and 5.0

Internet Explorer 5.0 supports the following key features:

- Direct browsing of native XML
- High-performance, validating XML engine
- Extensible Style Language (XSL) support
- XML Document Object Model (DOM)
- XML Schema

Consider the following table, which compares the functionality of these XML components in Microsoft Internet Explorer versions 4.0 and 5.0.

XML Function	Internet Explorer 4.0	Internet Explorer 5.0
Users can view XML documents in browser using XSL or CSS.	Required the DSO applet or XSL ActiveX control from Microsoft	Capability is built in
High-performance XML engine provides XML-processing capabilities.	Required a separate parsing application for validation	Enhanced engine performs validation inherently; full support of W3C XML and XML Namespaces Recommendations; Microsoft Windows will include native XML support to read and manipulate data between components and applications.
Extensible Style Language (XSL) allows developers to apply style sheets to XML data to format display.	Required the XSL ActiveX control	XSL processor is built in. Style sheets can format dynamic and flexible XML display based on the Dec. 18, 1998 XSL Working Draft. Users can search for information within an XML data set on client or server using the XSL Pattern syntax.
The XML	Included support of	Includes full support of the W3C

Document Object Model (DOM) is a standard object application programming interface that gives XML document developers control over content, structure and other aspects of the data.	Working Draft XML DOM when combined with the DSO applet or the XSL ActiveX control	XML DOM Level 1 Recommendation (accessible from script, the Microsoft Visual Basic development system, C++ and other languages).

XML Fundamentals

Module 8

Supporting XML

Using an XSL style sheet with XML in IE 4.x

Microsoft provides an ActiveX control to allow IE 4.x users to display XML files according to XSL instructions. To use this control, you would type the following code below the opening body tag of your XML file:

```
<OBJECT ID="XSLControl"
   CLASSID="CLSID:2BD0D2F2-52EC-11D1-8C69-0E16BC000000"
   CODEBASE="msxsl.cab"
   STYLE="display:none">
      <PARAM NAME="documentURL" VALUE=XMLFileURL>
      <PARAM NAME="styleURL" VALUE=XSLFileURL>
</OBJECT>
```

To use this control, you need the `msxsl.cab` file in the same directory as your XML file. This `.cab` file is provided in the course download for your use.

IE 5.0 natively supports XSL, which eliminates the need for the Active X control required by IE 4.x.

Module 8

Lesson 4
Objective

Using Dynamic HTML with XML

Examine a Dynamic HTML version of an XML file.

In this simulation, you will examine a dynamic HTML version of a catalog that will display the information in a single column or a table format.

Transcript

View Code

Click the View Code button to see how this catalog would be created for IE 4.x using the XMLDSO Java applet.

View Code

The IE 5.0 browser allows you to do direct databinding, so you don't have to call the XMLDSO applet. Click the View Code button to see how the same catalog would be created for IE 5.0.

These files are also provided in the demos directory of the course download, which is available from the Resources page.

XML Fundamentals

Module 8

Exercise

Databinding XML to HTML

Databinding XML to HTML in IE 4.x

Objective

Use simple databinding to display the contents of an XML file in IE 4.x.

Exercise Scoring

This exercise is not scored.

Starter files

The files for this exercise can be found in the 08-03 directory of the course download, which is available from the Resources page.

Overview

Data binding allows you to take content from a remote source such as a separate file or database and merge it with an existing XML document structure. In this exercise, you will walk through the process of using simple databinding to display the contents of an XML file.

Instructions

1. Open the file `catalog1.xml` and examine its contents.
2. Open the file `books.htm`.

   Add the XMLDSO applet shown below to `books.htm`. Insert the code just below the `<BODY>` tag.

   ```
   <applet code="com.ms.xml.dso.XMLDSO.class" STYLE="width:100%;height:25px"
   id="xmldso"
   MAYSCRIPT="true">
   <param name="url" value="catalog1.xml">
   </applet>
   ```
3. Now that you have added the XML datasource object applet, you will want to specify that the contents of this object are to be used within the table. To do this, add `datasrc="#xmldso"` to the `<TABLE>` tag as indicated as shown below:

   ```
   <TABLE id="table" datasrc="#xmldso" border="2" width="100%" cellpadding="5" >
   ```
4. Add the heading information to the table to indicate which columns will fall where in the table. Then, add the `<SPAN>` element to span information from the datasource object in the table data

cells as indicated:

```
<TABLE id="table" datasrc="#xmldso" border="2" width="100%"
cellpadding="5">
<THEAD>
<TR>
<TH>Author</TH>
<TH>Title</TH>
<TH>Year</TH>
<TH>ISBN</TH>
</TR>
</THEAD>
<TBODY>
<TR>
<TD VALIGN="top"><SPAN datafld="AUTHOR"></TD>
<TD VALIGN="top"><SPAN datafld="TITLE"></TD>
<TD VALIGN="top"><SPAN datafld="YEAR-PUBLISHED"></TD>
<TD VALIGN="top"><SPAN datafld="ISBN"></TD>
</TR>
</TBODY>
</TABLE>
```

5. Save the file.
6. Open `books.htm`. Click the View Image button to see how the file should render.

   View Image

7. To understand the relevance of the `THEAD` and `TBODY` elements in this example, remove them temporarily. Leave the rest of the table intact as follows:

```
<TABLE id="table" datasrc="#xmldso" border="2" width="100%"cellpadding="5">
<TR>
<TH>Author</TH>
<TH>Title</TH>
<TH>Year</TH>
<TH>ISBN</TH>
</TR>
<TR>
<TD VALIGN="top"><SPAN datafld="AUTHOR"></TD>
<TD VALIGN="top"><SPAN datafld="TITLE"></TD>
<TD VALIGN="top"><SPAN datafld="YEAR-PUBLISHED"></TD>
<TD VALIGN="top"><SPAN datafld="ISBN"></TD>
</TR>
</TABLE>
```

8. Save the file.
9. Reload the file. You should see a very different display of the data.

View Image

OK, I'm Done

Exercise

Databinding XML to HTML

Databinding XML to HTML in IE 5.0

Objective

Use simple databinding to display the contents of an XML file in IE 5.0.

Exercise Scoring

This exercise is not scored.

Starter files

The files for this exercise can be found in the 08-03 directory of the course download, which is available from the Resources page.

Overview

Data binding allows you to take content from a remote source such as a separate file or database and merge it with an existing XML document structure. In this exercise, you will walk through the process of using simple databinding to display the contents of an XML file in IE 5.0.

Instructions

1. Open the file `catalog1.xml`. Examine and copy its contents.
2. Open the file `books.htm`. Paste the code you just copied into the file as shown below.

```
<HTML>
<HEAD>
<TITLE>XML Databinding Example</TITLE>
</HEAD>

<BODY>

<xml ID="catalog">

<CATALOG>
<BOOK>
<TITLE>A Certain Justice</TITLE>
<AUTHOR>P.D. James</AUTHOR>
<YEAR-PUBLISHED>1998</YEAR-PUBLISHED>
<ISBN>0375401091</ISBN>
</BOOK>
```

```
<BOOK>
<TITLE>Ashworth Hall</TITLE>
<AUTHOR>Anne Perry</AUTHOR>
<YEAR-PUBLISHED>1997</YEAR-PUBLISHED>
<ISBN>0449908445</ISBN>
</BOOK>
<BOOK>
<TITLE>L.A. Confidential</TITLE>
<AUTHOR>James Ellroy</AUTHOR>
<YEAR-PUBLISHED>1997</YEAR-PUBLISHED>
<ISBN>0446674249</ISBN>
</BOOK>
<BOOK>
<TITLE>Shadow Woman</TITLE>
<AUTHOR>Thomas Perry</AUTHOR>
<YEAR-PUBLISHED>1997</YEAR-PUBLISHED>
<ISBN>0679453024</ISBN>
</BOOK>
</CATALOG>

</xml>

<TABLE BORDER="2" WIDTH="100%" CELLPADDING="5">
<THEAD>
<TR>
<TH></TH>
<TH></TH>
<TH></TH>
<TH></TH>
</TR>
</THEAD>
<TBODY>
<TR>
<TD VALIGN="top"></TD>
<TD VALIGN="top"></TD>
<TD VALIGN="top"></TD>
<TD VALIGN="top"></TD>
</TR>
</TBODY>
</TABLE>

</BODY>
</HTML>
```

3. Modify the table as shown below:

```
<TABLE DATASRC="#catalog" WIDTH="100%" BORDER="1">
```

```
<THEAD>
<TR BGCOLOR="yellow">
<TH>AUTHOR</TH>
<TH>TITLE</TH>
<TH>YEAR</TH>
<TH>ISBN</TH>
</TR>
</THEAD>
<TBODY>
<TR>
<TD VALIGN="top"><DIV DATAFLD="AUTHOR"></DIV></TD>
<TD VALIGN="top"><DIV DATAFLD="TITLE"></DIV></TD>
<TD VALIGN="top"><DIV DATAFLD="YEAR-PUBLISHED"></DIV></TD>
<TD VALIGN="top"><DIV DATAFLD="ISBN"></DIV></TD>
</TR>
</TBODY>
</TABLE>
</BODY>
</HTML>
```

4. Save the file.
5. Open `books.htm` in Internet Explorer 5.0. Click the View Image button to see how the file should render.

   View Image

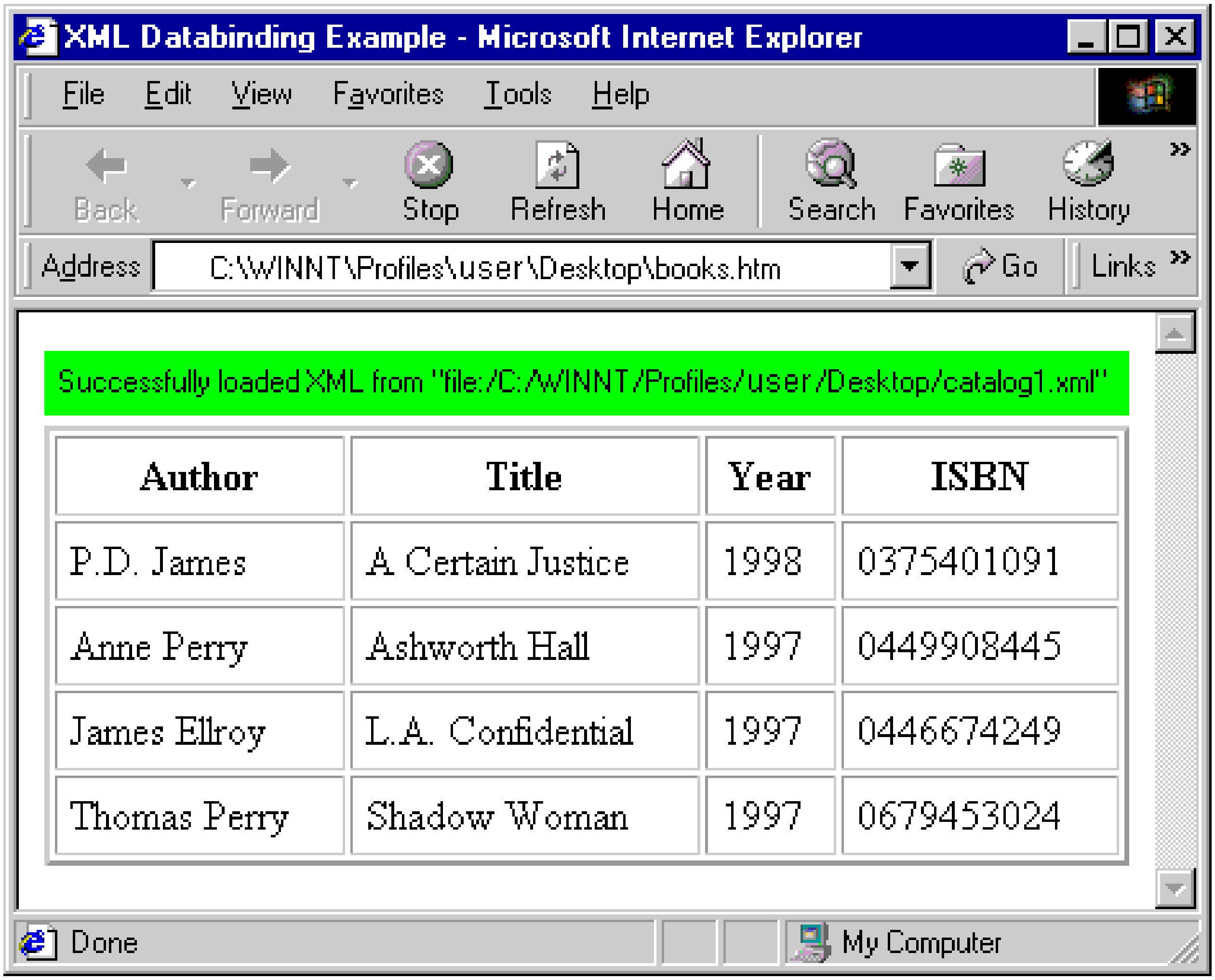
6. To understand the relevance of the `THEAD` and `TBODY` elements in this example, remove them temporarily. Leave the rest of the table intact as shown:

```
<TABLE BORDER="1" DATASRC="#catalog" WIDTH="100%">
<TR BGCOLOR="yellow">
<TH>AUTHOR</TH>
<TH>TITLE</TH>
<TH>YEAR</TH>
<TH>ISBN</TH>
</TR>
<TR>
<TD VALIGN="top"><DIV DATAFLD="TITLE"></DIV></TD>
<TD VALIGN="top"><DIV DATAFLD="AUTHOR"></DIV></TD>
<TD VALIGN="top"><DIV DATAFLD="YEAR-PUBLISHED"></DIV></TD>
<TD VALIGN="top"><DIV DATAFLD="ISBN"></DIV></TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

7. Save the file.
8. Reload the file. You should see a very different display of the data, similar to the figure below:

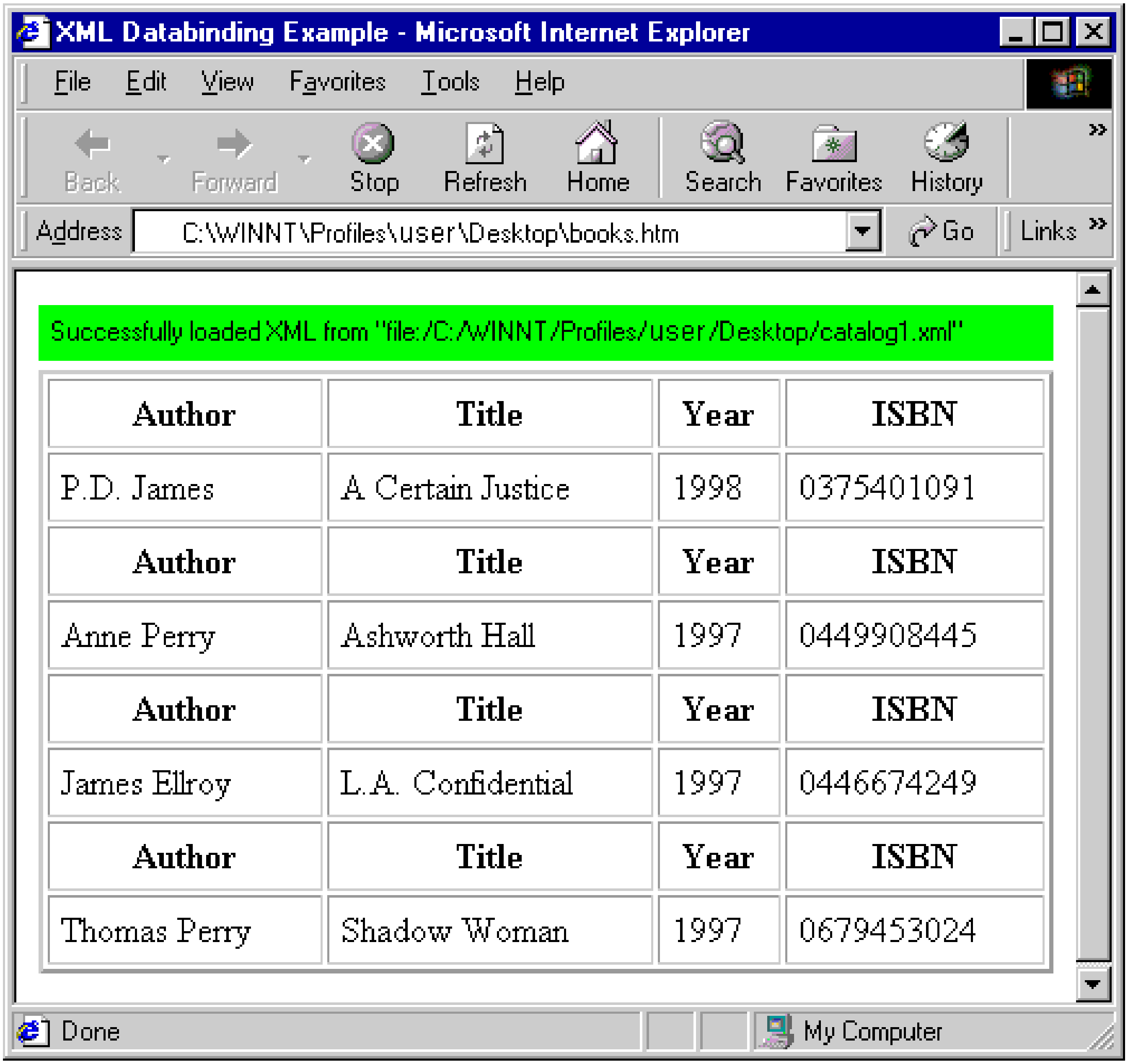
   View Image

OK, I'm Done


XML Databinding Example - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Forward
Stop
Refresh
Home
Search
Favorites
History
Address
C:\WINNT\Profiles\user\Desktop\books.htm
Go
Links
Done
My Computer

Successfully loaded XML from "file:/C:/WINNT/Profiles/user/Desktop/catalog1.xml"

Author	Title	Year	ISBN
P.D. James	A Certain Justice	1998	0375401091
Anne Perry	Ashworth Hall	1997	0449908445
James Ellroy	L.A. Confidential	1997	0446674249
Thomas Perry	Shadow Woman	1997	0679453024



XML Databinding Example - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Forward
Stop
Refresh
Home
Search
Favorites
History
Address
Go
Links
Done
My Computer

C:\WINNT\Profiles\user\Desktop\books.htm

Successfully loaded XML from "file:/C:/WINNT/Profiles/user/Desktop/catalog1.xml"

Author	Title	Year	ISBN
P.D. James	A Certain Justice	1998	0375401091
Author	**Title**	**Year**	**ISBN**
Anne Perry	Ashworth Hall	1997	0449908445
Author	**Title**	**Year**	**ISBN**
James Ellroy	L.A. Confidential	1997	0446674249
Author	**Title**	**Year**	**ISBN**
Thomas Perry	Shadow Woman	1997	0679453024

XML Fundamentals

Module 8

Lesson 5
Objective

An interactive demo

Experiment with an XML demonstration file.

In this simulation, you'll experiment with an XML demonstration file. Though this application is small, hopefully it will spur your creativity and enable you to see where this technology can take you.

Transcript

The files for this application can be found in the demos folder in the course download, which is available from the Resources page. These files can be viewed with IE 4.x.

Using Dynamic HTML with XML

A catalog built using DHTML and XML (Simulation transcript)

1. Here's an example of a book catalog created using Dynamic HTML with XML. Click the button that allows you to view books in a table format.
2. A table of books is displayed. Click the button that allows you to view books as blocks of information.
3. A column of data is displayed. You've reached the end of the simulation. Click the Exit button to return to the course. If you'd like to experiment with this application on you own, look for the files in the demos folder in the course download.

```
<HTML>
<HEAD>
<TITLE>
XMLDSO/DHTML Example
</TITLE>
<SCRIPT>

function showTable() {
   ByTable.style.display=""
   ByChunk.style.display="none"
}

function showList() {
   ByTable.style.display="none"
   ByChunk.style.display=""

}



</SCRIPT>
</HEAD>
<BODY>
<H1 STYLE="text-align: center;font-family:Arial;color:blue"
>Book Catalog</H1>
<P STYLE="text-align: center">
<BUTTON onclick="showTable()">
<IMG SRC="table.gif"><BR>
View Books in a Table Format</BUTTON>
&nbsp;&nbsp;&nbsp;
<BUTTON onclick="showList()"><IMG SRC="book.gif"><BR>
View Books as Blocks of Information</BUTTON>
<P>



<applet code="com.ms.xml.dso.XMLDSO.class"
   width="0" height="0" id="xmldso" MAYSCRIPT="true">
<PARAM NAME="url" VALUE="catalog1.xml">
</applet>

<DIV ID="ByTable" STYLE="display:none">
<TABLE id="table" STYLE="border:thick solid blue;width: 100%"
datasrc="#xmldso" cellpadding="5">
<THEAD>
<TR>
<TH STYLE="border: thin solid blue">Author</TH>
<TH STYLE="border: thin solid blue">Title</TH>
```

```html
<TH STYLE="border: thin solid blue">Year</TH>
<TH STYLE="border: thin solid blue">ISBN</TH>
</TR>
</THEAD>
<TBODY>
<TR>
<TD VALIGN="top" STYLE="border: thin solid blue"><SPAN
datafld="AUTHOR" dataformatas="HTML"></TD>
<TD VALIGN="top" STYLE="border: thin solid blue"><SPAN
datafld="TITLE" dataformatas="HTML"></TD>
<TD VALIGN="top" STYLE="border: thin solid blue"><SPAN
datafld="YEAR-PUBLISHED" dataformatas="HTML"></TD>
<TD VALIGN="top" STYLE="border: thin solid blue"><SPAN
datafld="ISBN" dataformatas="HTML"></TD>
</TR>
</TBODY>
</TABLE>
</DIV>
<BR><BR>


<DIV ID="ByChunk" STYLE="display:none">
<TABLE datasrc="#xmldso">
<TR>
<TD>
<SPAN datafld="TITLE" dataformatas="HTML"
STYLE="color:blue;font-family:arial;font-weight:bold"><BR>
</TD></TR>
<TR><TD>
<SPAN datafld="AUTHOR" dataformatas="HTML"
STYLE="font-style:italic">
</TD></TR>
<TR><TD>
Published in <SPAN datafld="YEAR-PUBLISHED"
dataformatas="HTML"><BR>
</TD></TR>
<TR><TD>
ISBN: <SPAN datafld="ISBN" dataformatas="HTML">
</TD></TR>
<TR><TD>&nbsp;</TD></TR>
</TABLE>
</DIV>
```

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

```
<XML ID=xmlData SRC="catalog1.xml"></XML>
<HTML>
<HEAD>
<TITLE>
XML/DHTML Example
</TITLE>
<SCRIPT>

function showTable() {
  ByTable.style.display=""
  ByChunk.style.display="none"
}

function showList() {
  ByTable.style.display="none"
  ByChunk.style.display=""

}



</SCRIPT>
</HEAD>
<BODY>
<H1 STYLE="text-align: center;font-family:Arial;color:blue">Book Catalog</H1>
<P STYLE="text-align: center">
<BUTTON onclick="showTable()"><IMG SRC="table.gif"><BR>
View Books in a Table Format</BUTTON>
&nbsp;&nbsp;&nbsp;
<BUTTON onclick="showList()"><IMG SRC="book.gif"><BR>
View Books as Blocks of Information</BUTTON>
<P>

</P>




<DIV ID="ByTable" STYLE="display:none">
<TABLE id="table" STYLE="border:thick solid blue;width: 100%"
datasrc="#xmldata" cellpadding="5">
<THEAD>
<TR>
<TH STYLE="border: thin solid blue">Author</TH>
<TH STYLE="border: thin solid blue">Title</TH>
<TH STYLE="border: thin solid blue">Year</TH>
<TH STYLE="border: thin solid blue">ISBN</TH>
</TR>
```

```
</THEAD>
<TBODY>
<TR>
<TD VALIGN="top" STYLE="border: thin solid blue">
<SPAN datafld="AUTHOR" dataformatas="HTML"></TD>
<TD VALIGN="top" STYLE="border: thin solid blue">
<SPAN datafld="TITLE" dataformatas="HTML"></TD> <TD VALIGN="top" STYLE="border: thin
solid blue">
<SPAN datafld="YEAR-PUBLISHED" dataformatas="HTML"></TD> <TD VALIGN="top"
STYLE="border: thin solid blue">
<SPAN datafld="ISBN" dataformatas="HTML"></TD>
</TR>
</TBODY>
</TABLE>
</DIV>
<BR><BR>


<DIV ID="ByChunk" STYLE="display:none">
<TABLE datasrc="#xmldata">
<TR>
<TD>
<SPAN datafld="TITLE" dataformatas="HTML"
STYLE="color:blue;font-family:arial;font-weight:bold"><BR>
</TD></TR>
<TR><TD>
<SPAN datafld="AUTHOR" dataformatas="HTML"
STYLE="font-style:italic">
</TD></TR>
<TR><TD>
Published in <SPAN datafld="YEAR-PUBLISHED"
dataformatas="HTML"><BR>
</TD></TR>
<TR><TD>
ISBN: <SPAN datafld="ISBN" dataformatas="HTML">
</TD></TR>
<TR><TD>&nbsp;</TD></TR>
</TABLE>
</DIV>




</BODY>
</HTML>
```

Lesson 6
Objective

XML adoption scenario

Consider a popular adoption scenario for the future of XML.

How will XML be used in the immediate future? While any prediction must necessarily be taken with a grain of salt, some groups have spent a great amount of time and money to answer this very question. A commonly held perception on this matter has been summarized as follows:

Phase 1: XML is separate from HTML

XML is created and manipulated apart from HTML. Developers learn how to build XML files and work with parsers. XML is only about data, and display mechanisms are unimportant in this phase.

Phase 2: XML begins to replace HTML

In this phase, it is expected that authors will begin creating pure XML pages, using style sheets as a display mechanism in browsers that natively support XML. XSL, the proposed style language companion for XML, will gain ground in this phase, and will perhaps both complement, and in some cases, replace Cascading Style Sheets as the primary formatting mechanism.

Phase 3: Write once, output everywhere

Industry observers have predicted that XML will be the "next big thing" not only for the Internet, but for all types of document interchange.

Audio

Transcript

XML has received unprecedented early attention from major industry players. Click the Audio button to hear what Adam Bosworth, general manager of the Internet Platform and Tools Division at Microsoft, had to say about XML.

XML Fundamentals

Module 8

An interactive demo

An XML demonstration file (Simulation transcript)

1. Here's an example of an employee phone list created with the use of XML. Enter the extension 5444 in the Enter Phone Ext. text field.
2. Click OK.
3. Click Show All Employees.
4. Close the window by clicking the X in the top right hand corner.
5. Click the arrow next to Judy Abbott name to see who else is listed.
6. Select Joseph Conrad.
7. Click the Get Phone Ext for Employee button.
8. Here's Joseph Conrad's extension. Close the window (by clicking the X in the top right hand corner).
9. You've reached the end of the simulation. If you'd like to experiment with this application on you own, look for the files in the demos folder in the course download.

XML Fundamentals

Module 8

Lesson 7
Objective

The future of HTML

Assess XML's impact on HTML.

Many developers are asking about their current investments in HTML. Will those disappear? XML has not been slated to replace HTML. For temporary documents and small vendor pages, HTML is robust enough to provide the visibility and ease of creation and use that has made the rapid growth of Web sites possible. HTML is also a wonderfully small transport format for sending simple mail messages, for example.

Vendors with a large amount of information to be searched, accessed, and in any way reused will find great benefits from migrating existing HTML pages to the XML format. The key is the reuse aspect. If the data will be used once, there may be no valid business reason to spend time tagging the data in an XML-compliant manner. But if the data will be used, reused, sought after and applied, then storing the data either in a database and using XML as a middleware application, or storing the data in a native XML format, makes good business sense.

Module 8

XML adoption scenario

Phase 1: XML is separate from HTML

We are already in this phase. Most XML pages are stored as objects that are then referenced by an applet or ActiveX control. XML is used only to structure data. HTML is still used to represent the data through some form of object linkage.

Discussions have ensued at the W3C about how to incorporate well-formed XML blocks of data in an HTML page. No formal proposal has come from these talks at the time of this writing.

XML Fundamentals

Module 8

XML adoption scenario

Phase 2: XML begins to replace HTML

We have already started and will continue to see the development of industry languages, specific vocabularies generated from XML for specific uses. One such language is Microsoft's Channel Definition Format (CDF) language, used to create channels to which Web users can subscribe to receive push content. Other languages include CML, the Chemical Markup Language, and MathML, the markup language for representing mathematical equations.

XML adoption scenario

Phase 3: Write once, output everywhere

Those already working with SGML documents can switch to the simpler, more lightweight XML language for a means of defining and creating interoperable data. Industries with large volumes of text that need to be publicly accessible will create XML repositories, large database-like collections of XML data.

In this phase, analysts foresee the ability to output a single set of XML data into multiple media formats. For example, a catalog of computer parts could be made available in published, paper-based form; in both a low-bandwidth and high-bandwidth HTML form; in XML/XSL form; on a CD-ROM; on an audio tape; in Braille; and on an interactive kiosk. The ability to generate content in one form and then use it in a host of others is one reason for the growing excitement over the possibilities of XML.

Lastly, in this phase, XML is expected to become the *de facto* standard for document interchange, enabling financial transactions, the sharing of medical records between hospitals, the sharing of detailed inventories with design engineers, and much more. Full XML applications are predicted for deployment across the Internet in this stage.

XML Fundamentals

Module 8

XML analyzed (Audio transcript)

“ In a recent interview by Jeff Walsh for *InfoWorld* magazine, Adam Bosworth, general manager of the Internet Platform and Tools Division at Microsoft, had this to say:

"When I started working on this officially at Microsoft, which was in December 1996, I told Microsoft that I thought it was going to be important enough that I was going to create a team and start working on it. The comment I got back mostly was, `What's XML? Tell me more,' but not, `Oh yeah, we understand this is a core strategic part of our direction.'

"I think XML has taken off so much because, to be blunt, the demand is there. XML made it possible for [many people] to open up their applications and have their applications interact with other components and applications without them having to write lots of complicated platform-specific code. That idea is so exciting to people, that you see people just running with it." ”

XML Fundamentals

Module 8

Lesson 8

Objective

XML and e-commerce

Consider potential e-commerce uses for XML.

XML's greatest potential may be realized in the field of e-commerce. E-commerce depends on standardized transmissions of data sets that are labeled in a consistent, machine-coherent manner.

To date, the prominent method of exchanging business transaction information between entities has been EDI, Electronic Data Interchange. In the past, using EDI meant working with specially tailored software that was usually platform-specific. The current standard, called EDIFACT (for the United Nations Standard Messages Directory for Electronic Data Interchange For Administration, Commerce and Transport), is used for business-to-business communications.

XML opens up new possibilities for interoperability that will enable dissimilar systems to more easily share and understand electronic transactions.

One proposal under development is XML/EDI. You can read about this at http://www.xmledi.com. The W3C has also convened a group to discuss XML/EDI.

Warning

It's important to remember that despite its potential to greatly enhance the intelligent use of information across barriers, XML will not solve all document interchange problems. It is not a one-format-cures-all solution.

Lesson 9
Objective

XML and databases

Explain how XML provides information interchange between databases and information systems.

By design, XML's purpose is to allow described data interchange between Web servers and browsers. Additionally, XML provides information interchange between databases. Because XML is machine-intelligible, it allows intelligent information exchange between databases within an organization, as well as exchange with other organizations' information systems.

For example, financial institutions have standardized a set of XML elements and their described meanings. This standard will enable all financial institutions that use XML to exchange information in a common and intelligible language. Consider the following example of a tag used in this manner:

```
<credit-card type="visa"
exp="00/00">4123-1234-1234-1234</credit-card>
```

This standard syntax allows all institutions to properly identify the content within the tag as a Visa credit card number and its expiration date. Establishing such conformity eliminates the need for each institution to convert information to its own proprietary format, and ensures that all will have consistent and descriptive content to manage.

As in HTML, the tag essentially becomes a container for the information, which can then be appropriately processed, stored or passed on to the final destination with a clear understanding of its meaning. This container can be considered its own unique piece of information that can be stored or called individually, at random or as need requires. HTML, however, can only format the information.

One future concern involves the types of databases that will handle XML information. Some experts say that to achieve the full functionality of XML, special servers and databases will be required.

Another distinct difference in XML-compliant databases is how the information is stored. Traditional databases store information in table-and-column format to create structure. By design, XML references information based on hierarchy and child levels. XML blurs the former distinction between structured data and unstructured documents because XML can describe text documents as well as fields in traditional relational database tables.

An XML data server will not provide an immediate solution for all organizations. A transitional period must take place, similar to the move from mainframe data storage to the relational database storage schema. However, as we move forward and the use of XML becomes more common, many solutions will emerge to better take advantage of XML's full functionality.

CLick Here to Enter

Module 8

Lesson 10

XML today and in the future

Module wrap-up

In this module, you saw examples of how the various pieces of technology can be fit together and displayed today.

In addition, you learned about some of the expected near-future developments for XML. You saw the deployment cycle for XML, and you discovered some of the emerging XML vocabularies that will help enable e-commerce as well as a host of other applications. You were also pointed to a Web site through which you can monitor the development of tools for creating XML and XSL files.

This module discusses the following term in relation to XML:

Document Object Model (DOM)

Discussion

Don't forget to investigate the Discussion section for XML Fundamentals-especially the folder on current and emerging XML standards. You can reach the Discussion section by clicking the Discuss button in the toolbar.

Quiz

Quiz

Click the Quiz button to check what you've learned in this module.

<!XML Fundamentals>

Module 8

XML and databases

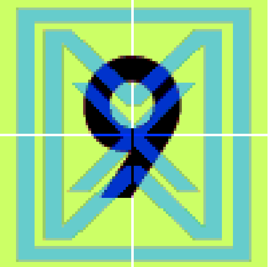
Existing XML solutions

Industry leaders such as Oracle already offer XML solutions. However, these solutions integrate as an additional processing layer on top of existing relational databases. This configuration requires the translation of XML commands to a typical database query, such as a SQL statement, to process the information. In contrast, a database that natively supports XML should not be required to use SQL, but instead will have a special query structure--an open standard--that can use and support XML to its full potential. The company Software AG Inc. claims to have developed such a server-and-database solution.

XML Fundamentals

Module 9

Lesson 1



Course conclusion

You have received a comprehensive introduction to creating and working with the Extensible Markup Language. You should now be ready to create your own projects using XML and its related technologies.

It's important to understand that by the time you complete this course, significant developments in XML and its related family of technologies are likely to have occurred. If you will be involved in XML development or will propose an XML solution to management, be sure that you get the latest information on the developing standard.

Module 8

Quiz

XML today and in the future

Each question is worth one point. Some questions ask you to select **the best answer**, others ask you to select **all the correct answers**. To receive credit for questions asking for **all the correct answers**, you must select all the correct answers and **only** the correct answers.

1. A commonly held perception regarding the future of XML summarizes its adoption and use into phases. Which of the following phases are included in this XML adoption scenario?

   Please select **all the correct answers**.

   □ A. XML is used with HTML.
   □ B. XML is separate from HTML.
   □ C. XML begins to replace SGML.
   □ D. XML begins to replace HTML.
   □ E. Write here, format there.
   □ F. Write once, output everywhere.

2. In Phase 1 of the XML adoption scenario presented in this module, XML is:

   Please select **all the correct answers**.

   □ A. Used only to structure data.
   □ B. Used to structure data and format pages by linking to XSL.
   □ C. Used to retrieve HTML objects by referencing an applet or ActiveX control.
   □ D. Used to represent data through some form of object linkage.
   □ E. Infused with blocks of HTML data in its pages.

3. In Phase 2 of the XML adoption scenario presented in this module, XML will:

   Please select **all the correct answers**.

   □ A. Be used to create channels for push content.
   □ B. Generate specific industry vocabularies for specific uses.
   □ C. Use Cascading Style Sheets as its primary formatting mechanism.
   □ D. Require a browser plug-in to display XML pages.
   □ E. Be used to create new forms of HTML for math and science disciplines

4. In Phase 3 of the XML adoption scenario presented in this module, XML will:

Please select **all the correct answers**.

- □ A. Allow those with SGML documents to migrate to the more comprehensive, heavyweight XML.
- □ B. Provide a means for defining and creating interoperable data.
- □ C. Enable industries with large text volumes to access XML repositories and databases.
- □ D. Probably become the de facto standard for document interchange.
- □ E. Allow users to output data in either paper-based or CD-ROM format.

OK, I'm Done

XML Fundamentals

Module 9

Lesson 2

We'd love your feedback

Please take a couple of minutes to complete the simple course evaluation on the next page. In return, we'll provide you with a priority code good for a **10 percent discount** on any DigitalThink course (not applicable to corporate customers where volume discounts already apply). This is your chance to let the DigitalThink team know how you think we can improve our courses--we'll **really** use your input!

Your comments appreciated

If you have any comments about DigitalThink in general, you can also send us your feedback via email.

Other DigitalThink courses

For a complete listing of DigitalThink courses, please visit our catalog. DigitalThink offers courses on a variety of topics, including Java programming and Microsoft certification preparation.

XML Fundamentals

Module 9

Lesson 3

Help us help you learn!

We hope that you found *XML Fundamentals* a valuable and enjoyable learning experience.

Please take a couple of minutes to answer several questions regarding the overall organization, presentation, and quality of the course material. Your input will enable us to improve this course on a continual basis and deliver the content that *you* want to see in a fashion that is best for *you*.

On a scale of 1 to 10 (10 being the highest), overall, how do you rate the course you've just taken: *XML Fundamentals*?

10

Please read each of the following statements and then select the sentiment that most closely matches your reaction. Following the statements is the most important part of this short survey--a space for you to provide feedback in your own words.

The DigitalThink Web site

1. The pages on the DigitalThink Web site were quick to load.
○ strongly agree
○ agree
○ neutral
○ disagree
○ strongly disagree

2. The DigitalThink site is easy to navigate through.
○ strongly agree
○ agree
○ neutral
○ disagree
○ strongly disagree

3. I was able to learn all I needed to know about DigitalThink courses from the information on the site.

○ strongly agree
○ agree
○ neutral
○ disagree
○ strongly disagree

4. The design of the DigitalThink site struck me as confusing and unclear.
○ strongly agree
○ agree
○ neutral
○ disagree
○ strongly disagree

5. I have a clear understanding of what DigitalThink does and how Web-based training at DigitalThink works.
○ strongly agree
○ agree
○ neutral
○ disagree
○ strongly disagree

Your course experience

6. The course covered the materials that I expected it would.
○ strongly agree
○ agree
○ neutral
○ disagree
○ strongly disagree

7. I found it easy to navigate through the course.
○ strongly agree
○ agree
○ neutral
○ disagree
○ strongly disagree

8. The course Syllabus window was a useful tool for moving through the course.
○ strongly agree
○ agree
○ neutral
○ disagree
○ strongly disagree

9. There was too much text to read in most of the lessons.

○ strongly agree
○ agree
○ neutral
○ disagree
○ strongly disagree

10. The quizzes were an important part of my overall course experience.
○ strongly agree
○ agree
○ neutral
○ disagree
○ strongly disagree

11. The course audio worked well and I could easily understand the instructor's spoken comments.
○ strongly agree
○ agree
○ neutral
○ disagree
○ strongly disagree

12. The interactive Chat and Discussion areas were an important and fun part of my overall course experience.
○ strongly agree
○ agree
○ neutral
○ disagree
○ strongly disagree

Learning via the Web

13. Web-based learning at DigitalThink enabled me to learn the course materials quickly and conveniently.
○ strongly agree
○ agree
○ neutral
○ disagree
○ strongly disagree

14. My DigitalThink course was a more effective learning method than I thought it would be.
○ strongly agree
○ agree
○ neutral
○ disagree

○ strongly disagree

15. I would rather learn from a book instead of taking another DigitalThink course.

○ strongly agree
○ agree
○ neutral
○ disagree
○ strongly disagree

16. I would rather take a CD-ROM-based training course than learn from DigitalThink.

○ strongly agree
○ agree
○ neutral
○ disagree
○ strongly disagree

17. I would rather take a classroom-based training course with a live instructor than learn via the Web using DigitalThink.

○ strongly agree
○ agree
○ neutral
○ disagree
○ strongly disagree

18. I would be interested in taking another Web-based course from DigitalThink.

○ strongly agree
○ agree
○ neutral
○ disagree
○ strongly disagree

19. If I needed to learn a new skill for my job, I would check with DigitalThink to see if they offered a Web-based course on the topic.

○ strongly agree
○ agree
○ neutral
○ disagree
○ strongly disagree

20. The best feedback we get comes in the form of your own words. In two sentences or more, how would you describe your DigitalThink course experience to a friend?

Thank you for taking the time to help us make the DigitalThink experience the greatest way to learn!

submit form reset all answers

DigitalThink

COMPANY

About DigitalThink | Management Team | Jobs | Contact Us | Visit Us | Newsroom

Locker Login

Course Catalog

Company

Corporate Training

Newsroom

Events

Support Services

Training Resources

Site Map

Home

Search Our Site

Find

Contact Us

Use these email addresses to direct your questions or comments to the appropriate department at DigitalThink.

We also have information regarding our physical address and a map to our offices.

If you're interested in working at DigitalThink, please visit our Jobs area.

sales@digitalthink.com
Our Sales Team can set up a Corporate Account to provide Web-based training to your entire company. We can also produce and deliver custom courses to help your organization train employees, sales people, or customers.

support@digitalthink.com
If you're having trouble using your credit card to buy a course, or if you can't log into a course you're registered for, please contact us.

suggestions@digitalthink.com
If you'd like to suggest a course topic or an improvement to our site, send an email to this address.

info@digitalthink.com
Use this address to get more general information about our company.

graphics@digitalthink.com
How do we look? Comments and questions about our site design or graphics should be sent here.

bugs@digitalthink.com
If you've discovered a bug (for example, a broken link or an image that won't load), let us know. The more details you can give us about the problem, the better.

Home | Courses | Corporate Training | Support | Events | Newsroom | Training Resources | Company | Site Map