Foundation

XML for Flash

Sas Jacobs

Foreword by Keith Peters,
co-author of *Extending Flash MX 2004*

Build simple XML-driven Flash applications with Microsoft Office applications, ASP.NET, or PHP

Consume web services and display their results in Flash

friendsof ED
DESIGNER TO DESIGNER™
an Apress® company

Facebook's Exhibit No. 1005
Page 001

GOOGLE EXHIBIT 1005

Foundation XML for Flash

Sas Jacobs

Facebook's Exhibit No. 1005
Page 002

GOOGLE EXHIBIT 1005

QA 76
.76
.H94J333
2006
Copy 1

Foundation XML for Flash

Copyright © 2006 by Sas Jacobs

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN (pbk): 1-59059-543-2

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit www.apress.com.

The information in this book is distributed on an "as is" basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is freely available to readers at www.friendsofed.com in the Source Code section.

Credis

Lead Editor
Chris Mills

Technical Reviewer
Kevin Ruse

Editorial Board
Steve Anglin, Dan Appleman
Ewan Buckingham, Gary Cornell
Tony Davis, Jason Gilmore
Jonathan Hassell, Chris Mills
Dominic Shakeshaft, Jim Sumser

Associate Publisher
Grace Wong

Project Manager
Pat Christenson

Copy Edit Manager
Nicole LeClerc

Copy Editor
Liz Welch

Assistant Production Director
Kari Brooks-Copony

Production Editor
Kelly Winquist

Compositor
Katy Freer

Proofreader
Lori Bring

Indexer
Broccoli Information Management

Artist
Katy Freer

Cover Designers
Corné van Dooren, Kurt Krames

Manufacturing Director
Tom Debolski

LC Control Number

2006 273493

33

For my parents, David and Sherry-Anne, and my sister, Lucy. Thanks for all your support. I feel lucky to have been born into such a terrific family.

Facebook's Exhibit No. 1005
Page 004

GOOGLE EXHIBIT 1005

CONTENTS AT A GLANCE

Foreword xvii

About the Author xix

About the Technical Reviewer xx

About the Cover Image xxi

Acknowledgments xxii

Introduction xxiii

Chapter 1: Flash and XML 1

Chapter 2: Introduction to XML 19

Chapter 3: XML Documents 57

Chapter 4: Using the XML Class 125

Chapter 5: Working with XML in Word 2003 217

Chapter 6: Working with XML in Excel 2003 257

Chapter 7: Working with XML in Access 2003 291

Chapter 8: Using the Data Components with XML 325

Chapter 9: Consuming Web Services with Flash 367

Chapter 10: Using the XMLSocket Class 411

Chapter 11: Which XML Option Is Best for Me? 427

Appendix: Useful Online Resources 439

Index. 445

Facebook's Exhibit No. 1005
Page 005

Page 5 of 140 GOOGLE EXHIBIT 1005

CONTENTS

Foreword . . . . **xvii**

About the Author . . . . **xix**

About the Technical Reviewer . . . . **xx**

About the Cover Image . . . . **xxi**

Acknowledgments . . . . **xxii**

Introduction . . . . **xxiii**

Chapter 1: Flash and XML **1**

Flash 2
XML 3
Multimedia capabilities 4
Visualizing complex information 5
Simplifying the display of information 7
Displaying content from Office 2003 for PCs 7
Displaying content from a web service 9
Accessing your computer 10
Separating content and presentation 12
Specific applications for Flash 15
Flash as a learning tool 15
Creating Flash applications with Flex 17
Summary 17

Chapter 2: Introduction to XML **19**

What is XML? 20
How did XML start? 21
Goals of XML 22
Creating XML documents 22
Elements 23
Attributes 24
Text 26
Entities 26
Comments 27
CDATA 28
An example 28
XML document parts 28
Document prolog 28
XML declaration 29
Processing instructions 29
Document Type Definitions 30
Tree 30
Document root 30
White space 30
Namespaces 31
A simple XML document 32
Requirements for well-formed documents 34
Element structure 35
Elements must be closed 36
Elements must nest correctly 36
Use quotes for attributes 37
Documents that aren't well formed 37
Well-formed XHTML documents 38
Working with XML documents 39
Generating XML content 39
Using XML information 40
XML, HTML, and XHTML 43
Why XML? 46
Simple 47
Flexible 47
Descriptive 47
Accessible 48
Independent 48
Precise 49
Free 49
What can you do with XML? 50
Storing and sharing information 50
Querying and consuming web services 52
Describing configuration settings 52
Interacting with databases 53
Interacting with Office 2003 documents 53
Why is XML important to web developers? 54
Summary 55

Facebook's Exhibit No. 1005
Page 007

 GOOGLE EXHIBIT 1005

Chapter 3: XML Documents 57

Creating XML content 57
Using a text editor 58
XML editors 59
Server-side files 64
Office 2003/2004 66
Word 2003 67
Excel 73
Access 80
InfoPath 86
Office 2003 and data structure 86
Consuming a web service 86
Using web services to interact with Amazon 87
Transforming XML content 91
CSS 91
XSL 94
XPath 94
XSLT 95
Transforming content 96
Sorting content 98
Filtering content 99
Conditional content 99
An example 99
Other methods of applying transformations 101
Determining valid XML 102
Comparing DTDs and schemas 102
Document Type Definitions 103
Elements 104
Attributes 106
Entities 107
A sample DTD 108
XML schemas 109
Simple types 110
Complex types 111
Ordering child elements 113
Element occurrences 113
Creating undefined content 114
Annotations 114
Including a schema 114
An example 115
XML documents and Flash 116
Creating an XML document 116
Creating a schema 120
Linking the schema with an XML document 121
Summary 122

Facebook's Exhibit No. 1005
Page 008

GOOGLE EXHIBIT 1005

Chapter 4: Using the XML Class 125

Loading an XML document into Flash 126
Using the load method 126
Understanding the order of the code 127
Understanding the onLoad function 128
Testing if a document has been loaded 129
Locating errors in an XML file 130
Testing for percent loaded 134
Navigating an XML object 136
Mapping an XML document tree 137
Understanding node types 139
Creating node shortcuts 141
Finding the root node 141
Setting a root node variable 142
Displaying the root node name 142
Locating child nodes 143
Working with specific child nodes 143
Working with the childNodes collection 145
Creating recursive functions 147
Locating siblings 148
Locating parent nodes 150
Extracting information from attributes 150
Putting it all together 152
Loading dynamic XML documents 162
Installing IIS 162
Creating XML content within Flash 175
Creating an XML string 176
Creating XML using methods 178
Creating new elements 178
Creating new text nodes 180
Creating attributes 182
Adding an XML declaration 182
Adding a DOCTYPE declaration 183
Limits of XML methods 183
Putting it all together 184
Modifying XML content within Flash 187
Changing existing values 188
Changing a text node 188
Changing an attribute value 189
Changing a node name 190
Duplicating an existing node 191
Deleting existing content 192
Putting it all together 194
Sending XML content from Flash 197
Using the send method 197
Using the sendAndLoad method 200
Adding custom HTTP headers 202
Putting it all together 204
Limits of the XML class in Flash 208

Facebook's Exhibit No. 1005
Page 009

Page 9 of 140 GOOGLE EXHIBIT 1005

No real-time interaction 208
No validation 209
Flash cannot update external XML documents 209
Security restrictions 209
Creating a cross-domain policy file 209
Proxying external data 210
Tips for working with XML in Flash 210
XML file structure 211
Use the XMLNode class 211
Create XML with a string 211
Validate your XML documents 211
Use the right tool for your dynamic content 212
Summary of the properties, methods, and events of the XML class 212
Summary 215

Chapter 5: Working with XML in Word 2003 217

Why use Microsoft Office? 218
Which Office packages can I use? 219
Understanding data structures 219
XML in Word 2003 220
Opening an existing XML document 220
Transforming the document view 221
Dealing with errors 224
Creating XML content with Save as 225
Understanding WordprocessingML 226
Structuring content within Word 227
Working with the schema library 228
Adding a schema to the library 229
Adding a transformation to a schema 230
Creating a new Word XML document 233
Attaching a schema to a Word document 233
Adding XML tags to the document 235
Adding placeholders for empty XML tags 238
Adding attributes 241
Saving a structured XML document 243
Saving transformed XML content 245
Editing XML content in Word 247
Putting it all together 250
Summary 255

Facebook's Exhibit No. 1005
Page 0010

GOOGLE EXHIBIT 1005

Chapter 6: Working with XML in Excel 2003 257

Opening an existing XML document . . . 258
Opening as a list . . . 258
Dealing with errors . . . 261
Opening as a read-only workbook . . . 262
Opening with the XML Source pane . . . 263
Opening a document with a schema . . . 265
Opening a document with an attached style sheet . . . 265
Dealing with nonrepeating content . . . 266
Dealing with mixed content . . . 267
Dealing with complicated structures in a list . . . 268
Creating XML content with Save As . . . 272
Understanding SpreadsheetML . . . 274
Creating structured XML from an Excel document . . . 275
Creating an XML map in Excel . . . 275
Adding XML elements to Excel data . . . 277
Saving a structured XML document . . . 279
Editing XML content in Excel . . . 284
Using the List toolbar . . . 284
Putting it all together . . . 285
Summary . . . 289

Chapter 7: Working with XML in Access 2003 291

Exporting content as XML . . . 292
Exporting a table object . . . 292
Exporting a query . . . 294
Exporting a report . . . 296
Creating a schema from Access . . . 296
Creating a style sheet from Access . . . 298
Setting export options . . . 300
Exporting linked tables . . . 300
Applying a custom transformation . . . 303
Other export options . . . 306
Importing data from an external file . . . 308
Dealing with import errors . . . 310
Transforming content . . . 312
Using a style sheet to import attributes . . . 314
Putting it all together . . . 316
Access XML resources . . . 322
Summary . . . 323

Facebook's Exhibit No. 1005
Page 0011

Page 11 of 140
GOOGLE EXHIBIT 1005

Chapter 8: Using the Data Components with XML 325

Understanding data components . . . 325
Understanding the XMLConnector . . . 327
Displaying read-only XML data . . . 327
Displaying updatable XML data . . . 328
Configuring the XMLConnector . . . 328
Using the Component Inspector . . . 329
Creating a schema from an XML document . . . 330
Creating a schema by adding fields . . . 331
Understanding schema settings . . . 332
Triggering the component . . . 334
Testing for a loaded XML document . . . 336
Loading an XML document into Flash . . . 336
Binding XML data directly to UI components . . . 338
Adding a binding . . . 338
Configuring the binding . . . 339
Using the DataSet component . . . 344
Binding to a DataSet component . . . 345
Adding an XUpdateResolver component . . . 346
Putting it all together . . . 348
The XMLConnector class . . . 354
Setting the XMLConnector properties . . . 355
Displaying the results . . . 355
Working with the XML class . . . 356
Binding the results to components with ActionScript . . . 357
Including the DataBindingClasses component . . . 357
Creating EndPoints . . . 358
Creating the binding . . . 358
Summary of the properties, methods, and events of the XMLConnector class . . . 363
Summary . . . 365

Chapter 9: Consuming Web Services with Flash 367

Consuming REST web services . . . 369
Using REST in Flash . . . 370
Creating a proxy file . . . 370
Understanding an ASP.NET proxy file . . . 370
Consuming the XML content . . . 371
Consuming an RSS feed . . . 375
Using the WebServiceConnector with SOAP web services . . . 380
Using SOAP in Flash . . . 380
Using the WebServiceConnector . . . 380
Configuring the WebServiceConnector . . . 381
Binding the params . . . 382
Triggering the web services call . . . 384
Binding the results . . . 386
Viewing the Web Services panel . . . 387
Working with XML content from the WebServiceConnector . . . 391

Facebook's Exhibit No. 1005
Page 0012

 GOOGLE EXHIBIT 1005

The WebServiceConnector class 398
Setting the WebServiceConnector properties 398
Sending data to the web service 399
Displaying the results 399
Summary of the properties, methods, and events of the WebServiceConnector class 403
The Web Service classes 404
Creating a WebService object 404
Viewing the raw XML content 405
Logging the details 405
Summary 409

Chapter 10: Using the XMLSocket Class 411

Socket server considerations 412
What socket servers are available? 412
Installing the Unity 2 socket server 413
Downloading the trial version of Unity 2 413
Unpacking the unity files 414
Configuring the server 414
Starting the server 415
Using the XMLSocket class 416
Creating an XMLSocket connection 417
Sending data 419
Responding to data 420
Closing the connection 420
Summary of the methods and event handlers of the XMLSocket class 424
Summary 425

Chapter 11: Which XML Option Is Best for Me? 427

Is XML the best choice? 428
Is response speed an issue? 428
Where is the data stored? 429
XML document 429
Database 429
Other software package 430
Office 2003 document 430
How will the data be maintained? 430
Do you need server-side interaction? 430
Making the decision 431
How should you include the XML content in Flash? 432
Using the XML class 432
Using data components 432
Using the XMLConnector, WebServiceConnector, and Web Service classes 432
Using the XMLSocket class 433

Facebook's Exhibit No. 1005
Page 0013

GOOGLE EXHIBIT 1005

Making the decision 433
Do you need real-time interaction? 433
Is the information time sensitive? 433
Which version of Flash do you own? 434
Which Flash players are you targeting? 434
Do you prefer to work visually? 434
A decision diagram 434
Summary 436

Appendix: Useful Online Resources 439

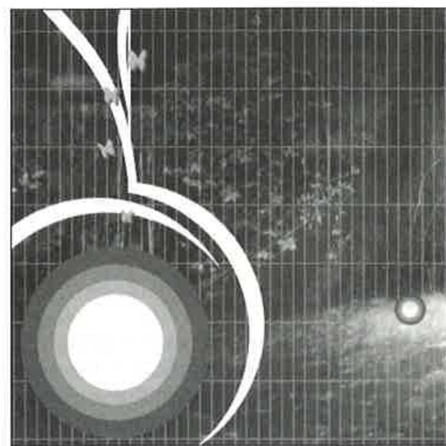
Index. 445

Facebook's Exhibit No. 1005
Page 0014

Page 14 of 140 GOOGLE EXHIBIT 1005

Facebook's Exhibit No. 1005
Page 0015

 GOOGLE EXHIBIT 1005



Chapter 1

FLASH AND XML

How can you create dynamic Macromedia Flash movies that share their content with other software applications and people? How can you store your data so that is simple to use but also adheres to web standards? The answer to both of these questions is to use Extensible Markup Language (XML) with your Flash movies. Storing your data in XML documents provides you with a flexible, platform-independent solution that is simple to implement in Flash.

This book is about using Flash with XML documents. Although it's not a substitute for an XML reference book, you'll learn the important points about working with XML. You'll also learn where and how to use XML documents within your Flash applications.

If you're new to XML, this book helps you to make your Flash movies more dynamic and interactive. Flash developers who've worked with dynamic content benefit by learning more about data binding with XML and Flash web services. Whatever your level, this book provides you with some new insights and ideas about Flash and XML.

Before we learn more about XML and its role within Flash, it's important to understand a little of the history of both areas. In this chapter, I look briefly at this history as well as the reasons why Flash can be a useful tool for working with XML information. I also show you some Flash applications that use XML to provide their data. Chapters 2 and 3 provide you with more detail about XML.

When you develop interactive websites, Flash provides many advantages over HTML web pages. Couple Flash with an external XML data source and you have a flexible and powerful solution for both web and stand-alone applications.

Facebook's Exhibit No. 1005
Page 0016

GOOGLE EXHIBIT 1005

Some of the reasons you might choose Flash to work with XML documents are

- The multimedia capabilities of Flash
- The ability of Flash to visualize and interact with complicated information
- The ability of Flash to simplify the display of information
- The separation of content from presentation that is possible within Flash

Flash

Flash is an amazing piece of software. It began as a simple animation tool and has grown into a sophisticated, high-end application for web developers and designers alike. Most people are familiar with the powerful animation features that are available within Flash. It is a great alternative to animated GIFs and helps web designers avoid some of the cross-browser issues associated with using Dynamic Hypertext Markup Language (DHTML).

Nowadays, the uses for Flash are many and varied. Flash creates everything from simple animations to entire websites and applications, as well as broadcast-quality animations and content for mobile devices. Developers have used Flash for e-learning and online help applications. Flash even challenges Microsoft PowerPoint as a tool for creating online presentations.

The Flash Player is one of the most popular web browser plug-ins in the history of the Internet. In March 2005, Macromedia stated that Flash content had reached 98.3 percent of Internet viewers. This means that designers and developers can rely on some version of the Flash Player being available on most computers. You can find out more about the popularity of Flash compared with other plug-ins at www.macromedia.com/software/player_census/flashplayer/.

People are less familiar with the role of Flash in rich-media applications. Flash creates flexible and stylish front-ends for web applications as well as for stand-alone projects. The more recent releases of Flash include a range of tools for creating complex graphical user interfaces (GUIs). The standard UI components included with Flash make it easy for both designers and developers to create interactive movies. These components are also great for rapid prototyping of applications.

Flash movies can include dynamic content from a number of different sources—databases, text files, and XML documents. A Flash movie can work like a template where you fill in the blanks with the external data. To change the contents of a dynamic Flash movie, simply update the data source. You don't even have to open Flash.

While we've been watching Flash evolve, the area of web development has changed dramatically. We've seen a move from static, brochure-style websites to more interactive sites that offer real functionality to users. Think about the Internet banking applications that are available from most major banks.

Web pages have become increasingly complicated to cope with more sophisticated information and higher expectations from website visitors. The original HTML specification has struggled to meet the demands of modern web pages. As a result, different flavors of HTML have emerged, each tied to specific software packages and versions.

Facebook's Exhibit No. 1005
Page 0017

Page 17 of 140 GOOGLE EXHIBIT 1005

A high proportion of websites are now driven by content management systems. These systems allow website owners to maintain their own content without writing a single line of HTML. Increasingly, these sites draw content from sources such as database and mainframe systems. The role of website designer has changed from updating static web pages to creating systems that allow clients to update their own content.

XML

There have been many other changes in the workings of the World Wide Web, including the introduction of XML in 1998. Since that time, the World Wide Web Consortium (W3C) has released many different recommendations for working with XML documents. XML has become a standard for exchanging electronic data both on and off the Web. Software packages like databases and web browsers now offer the ability to work with XML documents. Even Microsoft Office 2003 for PCs offers support for information in XML format.

The W3C published the first XML specification back in 1998. XML provides a way to create new markup languages and sets out some strict rules for the creation process. In 2000, applying XML rules to the HTML recommendation created a new recommendation, Extensible Hypertext Markup Language (XHTML).

Since that time, XML has filtered into the web development world and its expansion looks set to continue. XML documents provide structured data for both humans and software applications to read. Websites can use XML documents to provide content. Web services allow us to share information across the Internet using an XML format.

XML is a powerful tool for use in building web applications. It is browser and platform independent, and isn't tied to any commercial organization. XML processing software is also available on virtually every platform.

Flash 5 was the first version to introduce XML support. It has remained an important tool in subsequent releases of Flash. The built-in XML parser means that Flash movies can include content from external XML documents. Flash can also generate XML to send to external files.

Flash MX 2004 included data components that automated the process of connecting to an XML document. Developers could incorporate XML content with a single line of ActionScript. The components also allowed for binding between XML content and UI components.

The advanced multimedia and GUI development tools within Flash make it a perfect front-end for applications that use XML documents. You can generate XML from diverse sources, and it's even possible to use Office 2003 on a PC to update the XML content in your Flash movies.

Before we continue, I need to point out that XML is not necessarily the best solution for all dynamic Flash movies. XML documents are a useful option, but other solutions might be more appropriate, such as storing content in a text file or database. Chapter 11 looks at some of the issues you need to consider when deciding which approach to use in your Flash movies.

Facebook's Exhibit No. 1005
Page 0018

GOOGLE EXHIBIT 1005

Multimedia capabilities

Everyone is familiar with the multimedia capabilities within Flash. Flash movies can include sound, video, and animations, and you've probably seen at least one game or application built in Flash that uses these capabilities. One of the strengths of Flash is its ability to produce this multimedia content with relatively small file sizes.

Often, information is easier to understand when it is in a visual form. Everyone knows that a picture is worth a thousand words! The multimedia capabilities of Flash make it ideal for displaying some types of data. Flash also adds a level of interactivity that isn't easy to achieve when you use formats like HTML.

One of the common multimedia applications for Flash movies is the photo gallery. You've probably seen variations on this theme in several places on the Internet. My own website, www.sasjacobs.com, has a Flash photo gallery that I use to display my travel photos. You can see a screenshot in Figure 1-1.

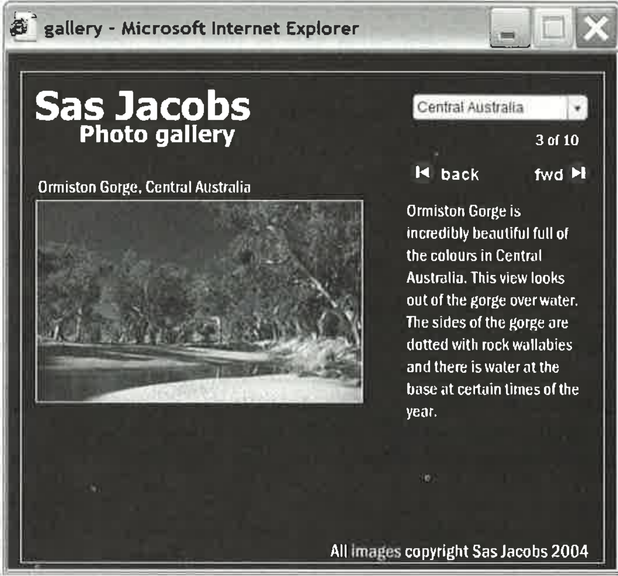


Figure 1-1. A Flash photo gallery application that uses content from an XML document

I use an XML file to store the information about my photos. The XML document includes a file name, category, caption, and description for each photo. The categories display in a ComboBox component. Users select a category and use the back and forward buttons to view each image. As the image loads, the relevant details from the XML document display in the Flash movie.

Each photo is loaded into the gallery only when the user requests the image. This makes the gallery operate much more quickly than if I displayed all photos on a single web page. I've also added a fade-in and fade-out, which would have been very difficult to achieve, cross-browser, with JavaScript and HTML.

I can update the photos in the gallery by changing the XML document. I use an XML editor and type in the changes. You'll learn how to build a similar photo gallery application later in this book.

Facebook's Exhibit No. 1005
Page 0019

Page 19 of 140
GOOGLE EXHIBIT 1005

You can see another variation on the Flash XML Photo gallery application at `www.davidrumsey.com/ticker.html`. Figure 1-2 shows a screenshot.

Figure 1-2. A ticker-style Flash photo gallery application

This is a new take on an old favorite and a clever way to display image thumbnails. The gallery uses a "ticker" to display image thumbnails, similar to a stock ticker. You can choose whether to view an ordered or random display. The information for the images in the ticker comes from an XML file. The application is simple to update and can easily be used for other image collections.

Visualizing complex information

Another strength of Flash is its ability to provide a visual representation of complex information. If you search for **Flash maps** in your favorite search engine, you'll find many commercial products that use Flash to display location maps. Some of these offer XML support for plotting specific points.

In another example, Bernhard Gaul has used Flash as an interface for a global airport weather web service. The information comes from the Cape Science GlobalWeather web service in XML format; Figure 1-3 shows a screenshot.

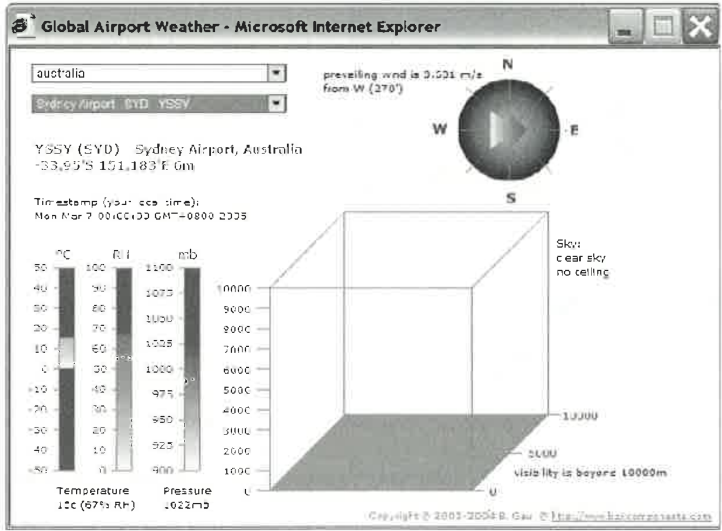


Figure 1-3. Global airport weather information displayed in a Flash movie with content from a web service

Facebook's Exhibit No. 1005
Page 0020

GOOGLE EXHIBIT 1005

Strong visual elements show information such as wind direction, temperature, and pressure. It's much easier to get a sense of the weather from this Flash movie compared with reading a list of figures. You can find out more at `www11.brinkster.com/bgx/webservices/weather.html`.

You can also find this at `www11.brinkster.com/bgx/webservices/weather.html`. Open the website in your favorite browser and click the "View the Flash Visualization" link.

Another example of visualizing information is in the periodic table of elements. We've probably all seen this in high school science classes. Within the periodic table, the position of each element is based on the element's bonding abilities. It is a visual way to represent repeating patterns within elements.

Many web pages provide a visual representation of this table. A Flash representation of the same content makes the information much easier to access. In the Flash version shown in Figure 1-4, rolling the mouse over an element displays the element name. Clicking the element pops up more information about each element. You can see the example at the GalaxyGoo website at `www.galaxygoo.com/chemistry/PeriodicTable.htm`.

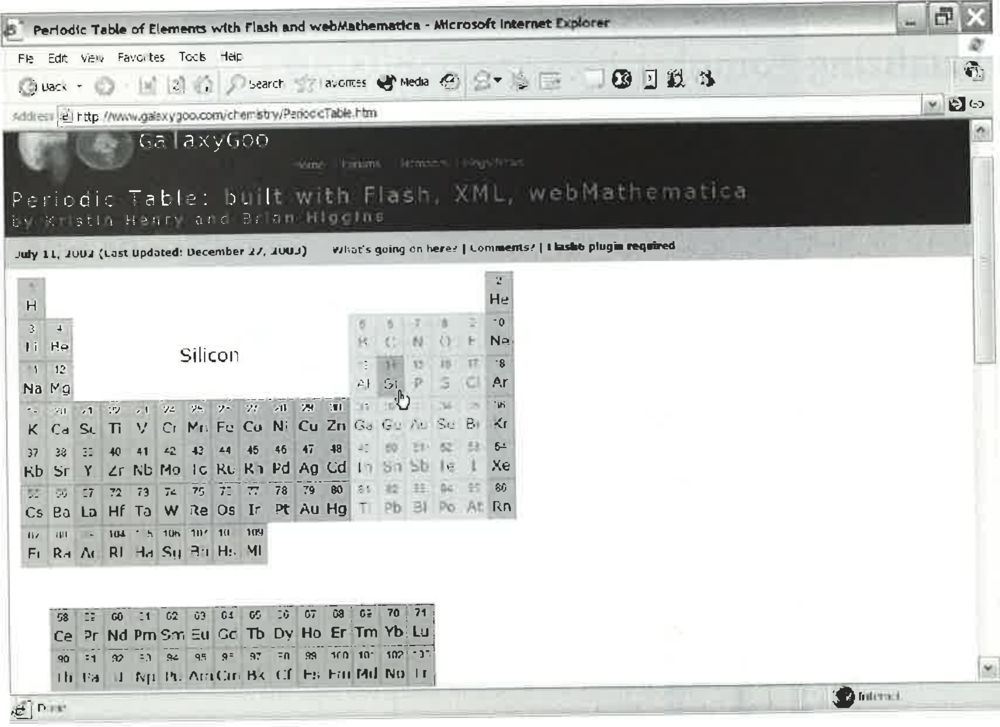


Figure 1-4. An interactive periodic table created in Flash, XML, and webMathematica

Incidentally, this example uses an XML document to provide the basic information about each element. That allows other applications to use the same information. The website also includes a spelling game (see `www.galaxygoo.com/games/tabletoy/tabletoy.html`) that uses the same XML content.

Facebook's Exhibit No. 1005
Page 0021

GOOGLE EXHIBIT 1005

Simplifying the display of information

Information displayed in lists and tables can often be difficult to process. Humans are more comfortable with visual or summarized information that scans easily. Flash is very good at taking complicated information and displaying it in a simplified manner. It can alter the visual appearance by reducing the amount of information that displays. Flash can also add sorting, animation, or other kinds of visual cues to the data.

Displaying content from Office 2003 for PCs

Microsoft Office 2003 is a very popular PC software suite for business and personal users alike. At the time of writing, the latest version of Office for Macintosh users is Office 2004. The Mac version offers limited XML functionality only within Excel, so the example that follows isn't applicable to Macintosh users.

Organizations frequently store complicated tables of information in Excel spreadsheets. If the information is stored in these software packages, how can you make selected parts available to your clients? You may need to simplify a complex Excel structure into something more manageable. You might also have to restrict access to the full worksheet for commercial reasons.

One of my clients, Dura-lite, faced this problem. Dura-lite works with heat transfer products and maintains a complex set of related part numbers within an Excel workbook. The company uses Excel for complicated lookups between two different sets of numbering systems.

Dura-lite needed to make the lookup available to their clients. The existing structure was complicated and contained confidential information so they couldn't just send out their Excel workbook. The clients would have found it difficult to understand and use the content.

Dura-lite also wanted to place this information on their website and on a CD-ROM catalog that could be run offline. However, they felt most comfortable maintaining the data in their Excel workbook. They didn't want to re-create the data in a database and wanted to use the same content for both the website and CD-ROM.

I used Flash to create a catalog for the Excel information. The content for the catalog comes from the Excel workbook via an XML document. Dura-lite updates the workbook and from time to time saves the information from Excel in XML format. The catalog is available on the Dura-lite website in a password-protected area. They also distribute it on a CD-ROM.

The structure of the Excel workbook they use is relatively complex. It contains seven worksheets, each corresponding to a manufacturer. Each worksheet contains information about a model and engine with corresponding part numbers. An additional sheet provides a cross-lookup between part numbers.

Figure 1-5 shows the structure of the workbook. For commercial reasons, I've removed the data, but imagine the structure populated with long part numbers. It would be very difficult to read!

Facebook's Exhibit No. 1005
Page 0022

GOOGLE EXHIBIT 1005

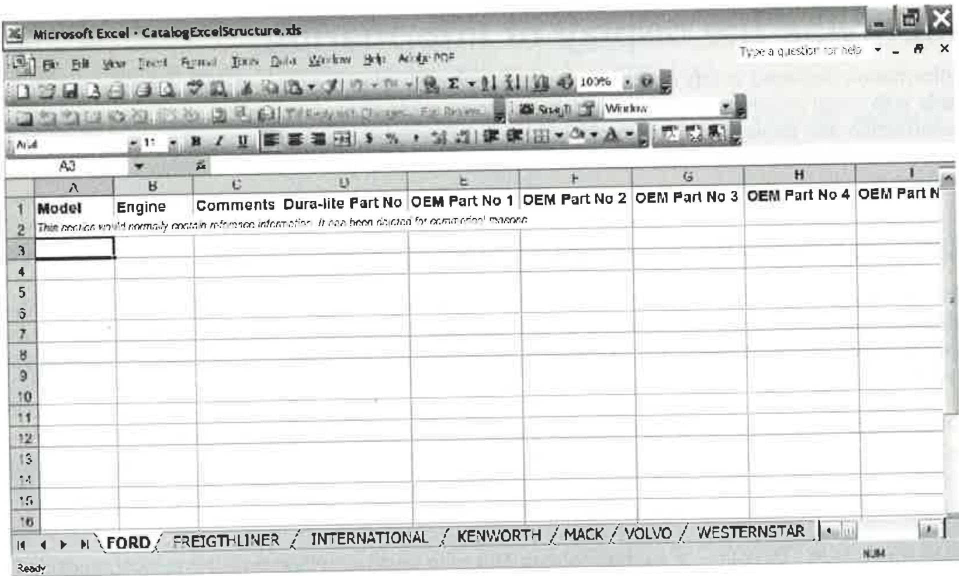


Figure 1-5. The Excel workbook structure for Dura-lite parts

Figure 1-6 shows the same content displayed in the Flash catalog.

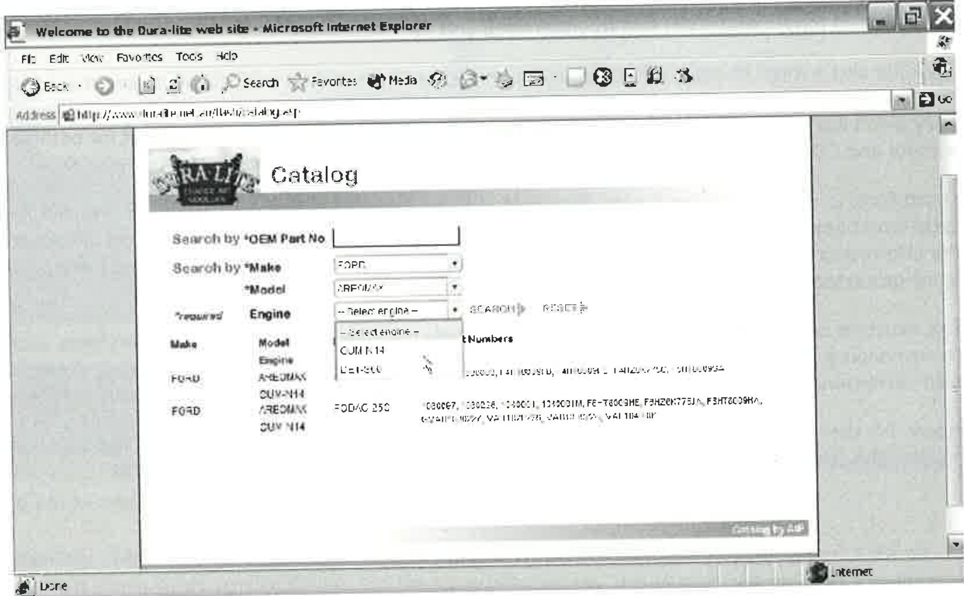


Figure 1-6. The Flash catalog for Dura-lite parts

Facebook's Exhibit No. 1005
Page 0023

There are two ways to use the catalog. If the user knows the original equipment manufacturer (OEM) part number, they enter it and click Search. Otherwise, they select a manufacturer from a drop-down list. This populates the Model drop-down list with all related models. The Engine drop-down list is then populated. The user can optionally choose an engine before they click Search.

The results display in a list below the search form. The text is selectable so that clients can copy and paste the part numbers. You can't see this example on the Web as it is in a password-protected area.

The Flash catalog is much easier to use than the Excel workbook. It simplifies the information available to the user and offers a simple but powerful means of searching for data. An XML document provides the link between Excel and Flash. Dura-lite can update their content at any time using their Excel file, and they don't need to rely on me to make changes for them. Likewise, I don't have to make numerous small updates on their behalf. I can leave control of the content up to Dura-lite and focus on design and development issues.

Displaying content from a web service

Web services provide another example where Flash simplifies the display of complex information. Many organizations make their data available to the public through web services. Data arrives in an XML document, and the built-in XML parser within Flash can translate this document into a simple visual display for users.

Earlier in the chapter, we saw an example of a web service that provides airport weather information. In the next example, we look at how Flash can consume a news service from Moreover. The new headlines update on a daily basis, but the structure of the XML document providing the information never changes. Users can view up-to-date news items any time they open the Flash movie.

Figure 1-7 shows a sample XML document from Moreover displayed in a web browser.

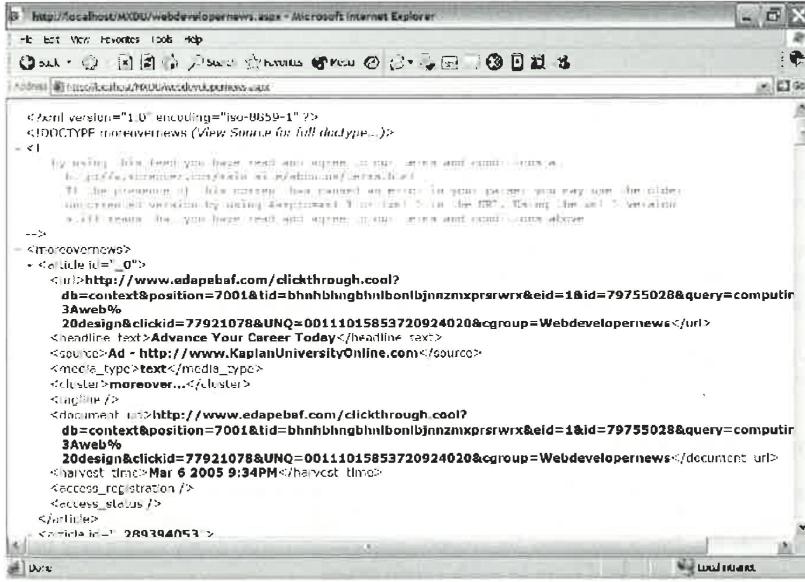


Figure 1-7. An XML document from Moreover news

Facebook's Exhibit No. 1005
Page 0024

Page 24 of 140
GOOGLE EXHIBIT 1005

I requested the daily web developer news and this XML document was provided. It contains a set of articles. Each article has related information such as an id, URL, and headline.

It's not easy for me to scan this document to read the news headlines. The document isn't designed to be read by humans. I can look through the list for the `<headline_text>` element but it's pretty hard work. If I want to see the news item in full, I'll have to copy the `<url>` text and paste it into the address line of a web browser.

Compare the XML document with Figure 1-8, which shows the same information displayed within a Flash movie.

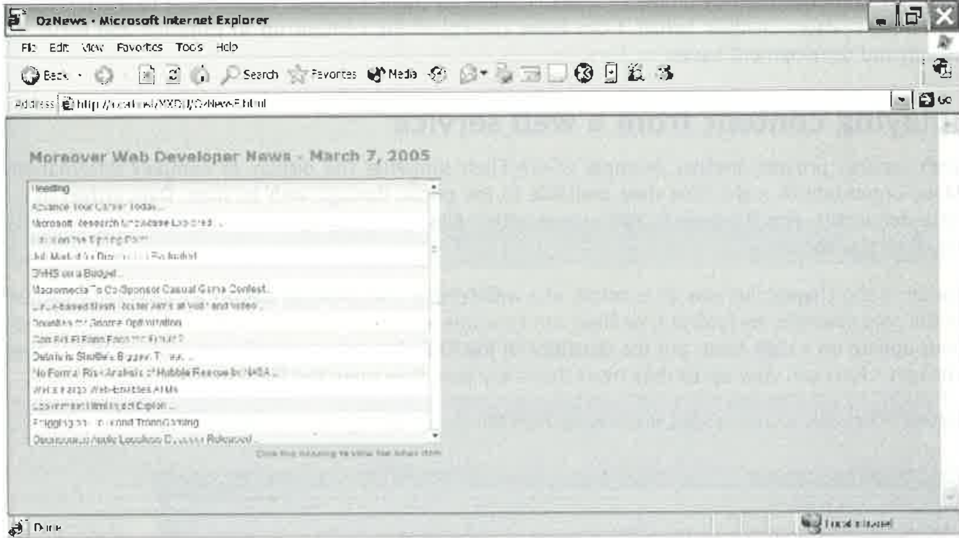


Figure 1-8. Moreover news headlines displayed in a Flash movie

Flash has extracted the relevant information and the headline displays in a DataGrid component. It has also colored every second line to make it easy to view the headlines. Each headline links to an HTML page that displays more detail when clicked.

I could change the display by adding extra columns or by using different colors. I could also make this movie more interactive. For example, the source for the news item could display as the user moves their mouse over the title. I could also add a button that allows me to look up news items from earlier dates.

Flash provides a presentation layer that makes the information much more accessible. You'll build something similar later in this book.

Accessing your computer

Another example of simplifying information with Flash is using it to interact with parts of your computer—for example, the files and folders. Flash can't interact with files and folders directly so you have to use a server-side file. This is important for security reasons—you don't really want a user being able to delete the contents of your hard drive by using a Flash movie.

Facebook's Exhibit No. 1005
Page 0025

GOOGLE EXHIBIT 1005

Server-side languages like ColdFusion, PHP, and VB .NET allow you to work with folders and files on your computer. You could use them to generate a list of characteristics such as file and folder names, file sizes, or the last modified date of a document. You can also use server-side languages to edit and delete files and folders, as well as to change the contents inside text files.

Flash can use a server-side language to work with files and folders on a computer. The server-side file can generate an XML representation of your files and folders. Flash then has access to the information about them in a structured format.

So when would this be useful? Well, any time you wanted to create an up-to-date list of files for use within a Flash movie. In this book, one of the examples we'll look at is an MP3 player.

I've backed up most of my CDs in MP3 format so I can listen to them while I work. I have also built a Flash MP3 player that can play the files. The MP3 player loads the file list from an XML document.

I could type the names of all of my MP3s into an XML file and display the list in Flash. However, given the number of CDs I own, that's likely to take me a long time and I'd have to update the file each time I add new songs.

Instead, I wrote a server-side file that gets the details of the folders and MP3 files in my library, and creates the XML document automatically. However, the XML document doesn't exist in a physical file. Instead, it's generated as a stream of XML information whenever I open the Flash movie.

Figure 1-9 shows the XML document that I have generated from my folders and MP3 files. I used an ASP.NET file to create this document. At the time of writing, there were 542 lines in the file. I'd hate to have to maintain the list manually by typing the content myself!

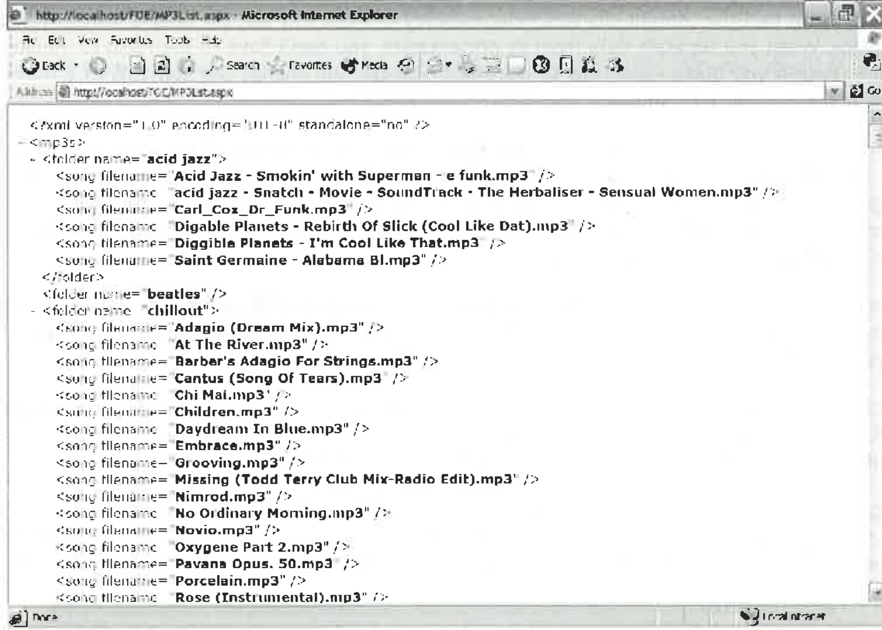


Figure 1-9. An XML document containing a list of MP3 files generated by an ASP.NET file

Facebook's Exhibit No. 1005
Page 0026

GOOGLE EXHIBIT 1005

This XML document powers the Flash MP3 player shown in Figure 1-10.

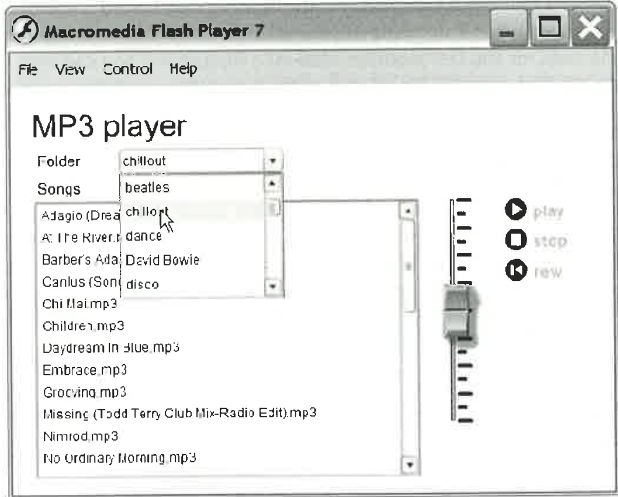


Figure 1-10. An MP3 player that uses an XML document listing the MP3 files

You'll learn how to build a very similar MP3 player a little later in this book.

Separating content and presentation

It's much more flexible for you to build Flash movies that include dynamic content. The term *dynamic content* means that the content changes independently of Flash and that the data is stored in a separate place. When you separate data from its presentation, you don't have to use Flash to update the content. Even if you don't know how to use Flash, you can still change the contents of a movie by changing the external data source.

This means your clients can update their own Flash content. For example, if you are drawing content from an XML or text file, they can open a text editor and edit the contents directly. They can also change the content in a database like MySQL or Access or by using web forms. Your clients are no longer reliant on you every time they want to make a change. All you need to do is provide them with a mechanism for updating the data source.

In the case of XML, clients could update content directly using an XML editor. This would be appropriate for data with a simple structure and confident clients. You can also provide a web form that allows clients to make updates.

You can even set up Office 2003 documents that generate XML content. Clients can make changes in Word, Excel, or Access and export the contents to an XML document. If the Flash movie is part of a website, they can then use a web form to upload the new XML file.

Going back to my Dura-lite example, the client maintains the content of their Flash catalog with Excel. They export the Excel file as an XML document. Figure 1-11 shows how to do this. I can't show the Dura-lite Excel file for commercial reasons, but the screenshot gives you an idea of the process they use.

Facebook's Exhibit No. 1005
Page 0027

GOOGLE EXHIBIT 1005

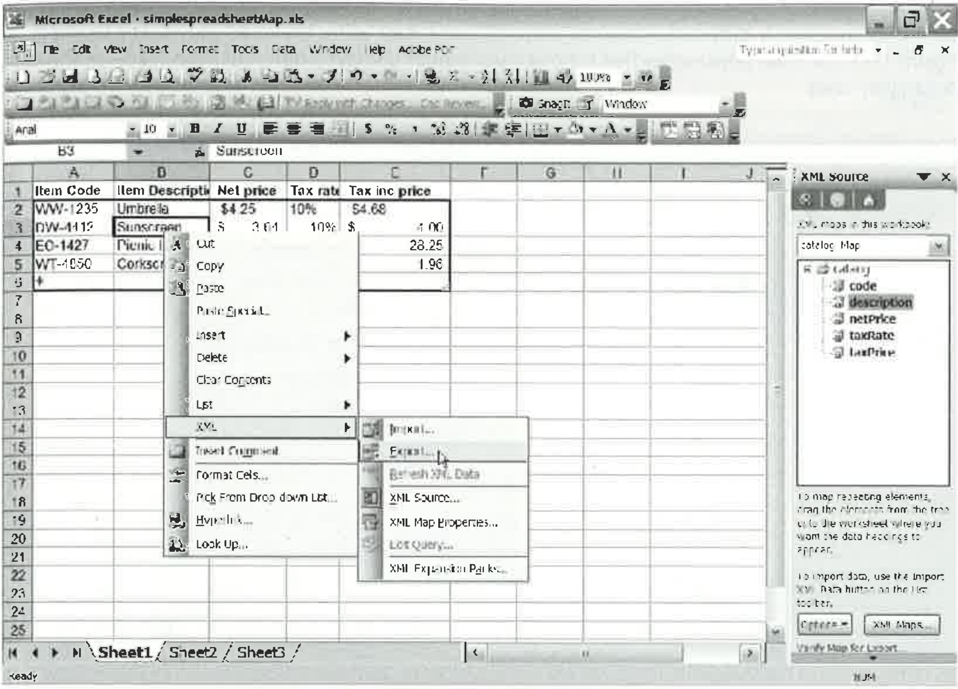


Figure 1-11. Exporting XML data from an Excel spreadsheet

Figure 1-12 shows the web interface that Dura-lite uses to upload a new XML file. This is in a password-protected area so I'm unable to provide a link to the page, but the screenshot should give you the general idea.

Figure 1-12. Dura-lite uses a web interface to upload their new XML files.

Facebook's Exhibit No. 1005
Page 0028

GOOGLE EXHIBIT 1005

Another benefit of separating Flash movies from their content is that you can use the same data structure with completely different movies or view different data using the same movie. For example, in Figure 1-13, the Flash newsreader that I showed you earlier displays a different news feed—in this case Australian news.

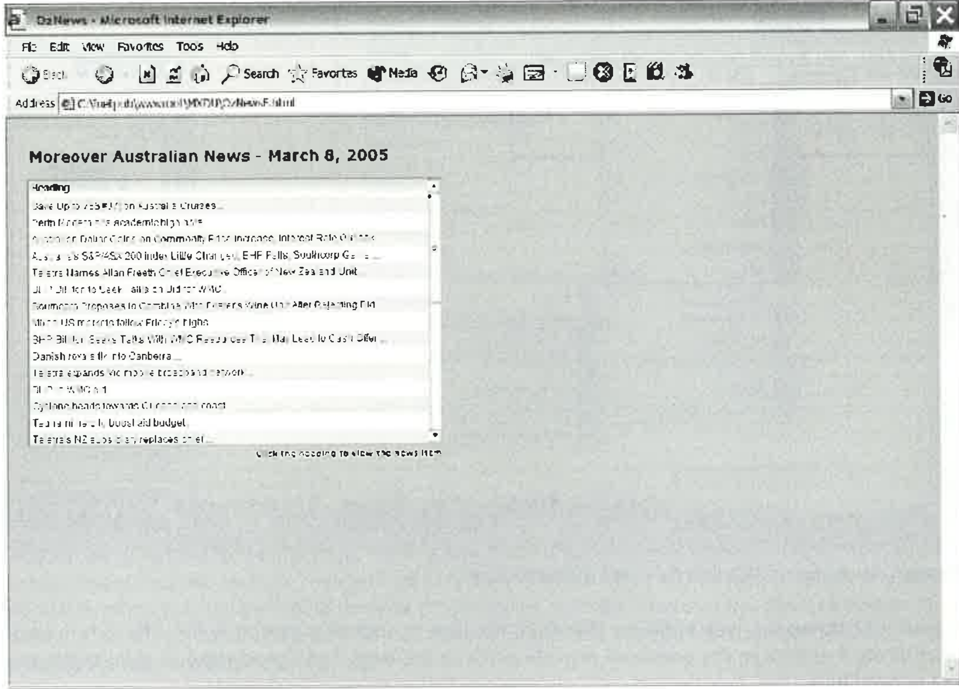


Figure 1-13. The same Flash movie can be used to access multiple sources of information.

If you keep the data structures constant, you can vary the visual appearance by changing Library elements within Flash. You'll be able to use exactly the same ActionScript to load and display the XML content within Flash. That way, you can sell the same solution with different skins.

Figure 1-14 shows the original newsreader with slightly different styling. I make no comment about how attractive the design is!

Facebook's Exhibit No. 1005
Page 0029

GOOGLE EXHIBIT 1005

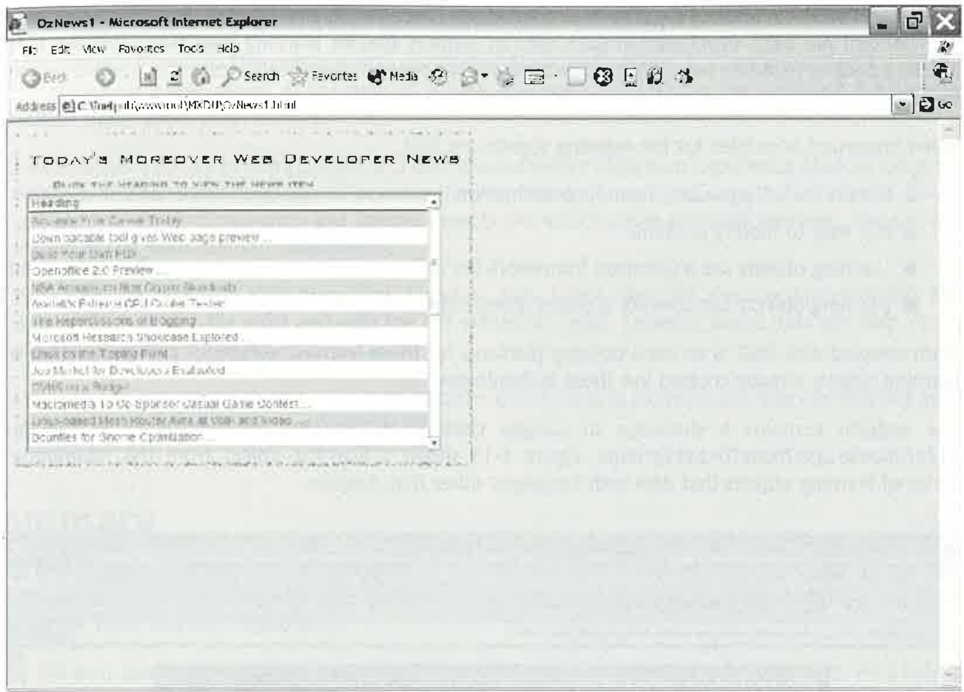


Figure 1-14. The same source of information can be used with different Flash movies.

Specific applications for Flash

Some applications are particularly well suited to Flash. XML-driven maps, stock tickers, and photo gallery examples abound on the Web.

E-learning is another area where Flash is proving very useful. Combining Flash with XML allows distribution of e-learning applications on CD-ROMs. You can run the applications in stand-alone mode without the need for an Internet connection or even a Flash Player. These applications have all the benefits of dynamic data with the flexibility of a portable format.

Flash as a learning tool

In my part of the world, there is a joint project to produce online content for students and teachers. The project, called the Le@rning Federation, is an initiative of the governments of Australia, the Australian states, and New Zealand. You can find out more about the project at `www.thelearningfederation.edu.au/`.

The project works in priority areas such as science, languages other than English, literacy, and numeracy. Content has been developed in each area to support specific learning objectives. The aim is to create a pool of resource materials for teachers and students. Schools can access the content online, through e-learning management systems or servers.

Some important principles for the learning objects are that

- Data is stored separately from its presentation.
- It is easy to modify content.
- Learning objects use a common framework for different contexts, i.e., they can be repurposed.
- Learning objects can operate as stand-alone objects that don't require server interaction.

Flash coupled with XML is an ideal delivery platform for these learning objects. A high proportion of learning objects already created use these technologies.

The website contains a showcase of sample content at `www.thelearningfederation.edu.au/tlf2/showMe.asp?nodeID=242#groups`. Figure 1-15 shows a learning object from the "stampede" series of learning objects that deal with languages other than English.

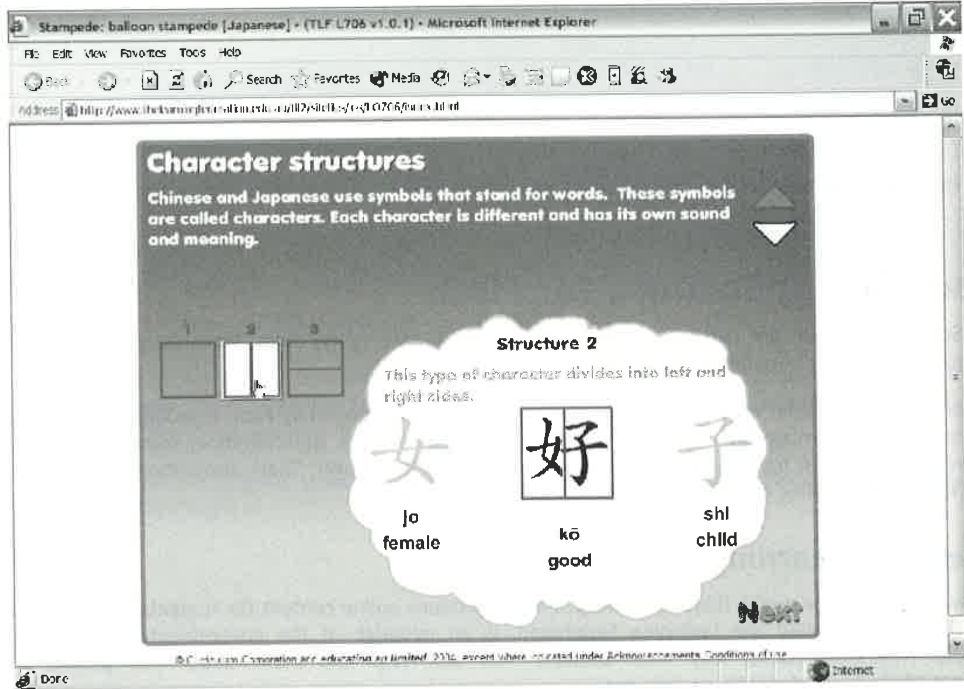


Figure 1-15. A learning object about understanding Chinese and Japanese characters

Facebook's Exhibit No. 1005
Page 0031

Page 31 of 140
GOOGLE EXHIBIT 1005

Creating Flash applications with Flex

Macromedia Flex is an alternative means of creating Flash applications. It is a presentation server that is installed on a web server. At the time of writing, Flex was only available for Sun's Java 2 Enterprise Edition (J2EE) application servers.

Flex includes a library of components and uses Macromedia's Maximum Experience Markup Language (MXML), an XML-based language to describe the interface for an application with ActionScript. MXML lays out the visual components and defines aspects like data sources and data bindings. You can also extend MXML with custom components.

You write the MXML in XML files using a text or XML editor. You can also use Macromedia's Flex Builder. Each MXML file must end with the file extension `.mxml`. Learning about XML will help you to use Flex to create applications.

I haven't covered Flex in this book as it could be an entire book in its own right. You can find out more about it at `www.macromedia.com/software/flex/`.

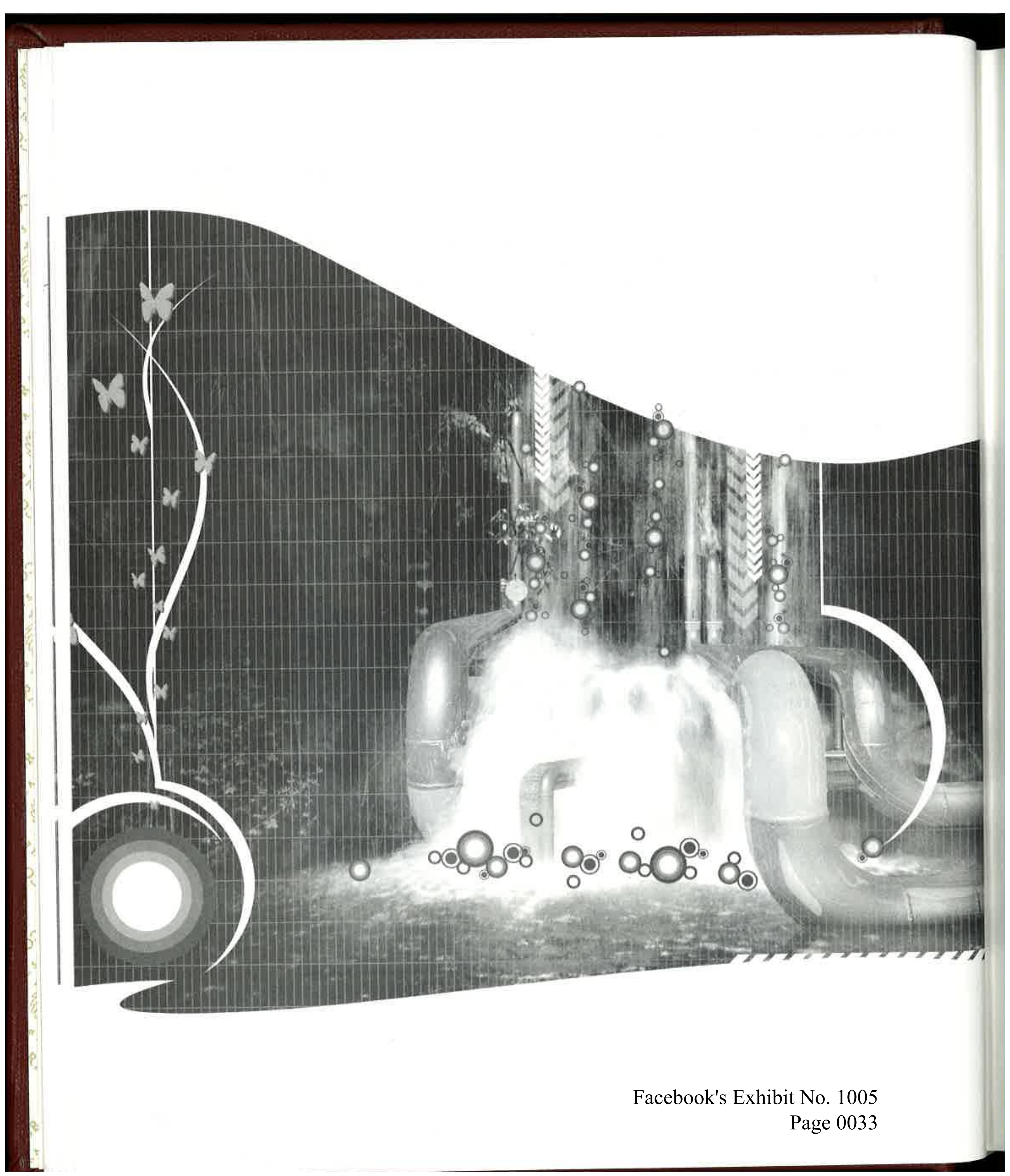
Summary

In this chapter, I covered a brief introduction to Flash and XML. I looked at some of the reasons why developers might use Flash with XML in their applications. I also showed you some sample applications that use Flash and XML together.

In the next chapter, I'll introduce you to XML and explain how to create XML documents. We'll look at the meaning of the word *well formed* and examine the differences between XML, HTML, and XHTML. Chapter 3 will go into more detail about XML documents and we'll look at Document Type Definitions (DTDs), XML schemas, and Extensible Stylesheet Language Transformations (XSLT). If you have experience in working with XML documents, you might want to skip ahead to Chapter 4, where we'll start to build Flash XML applications.

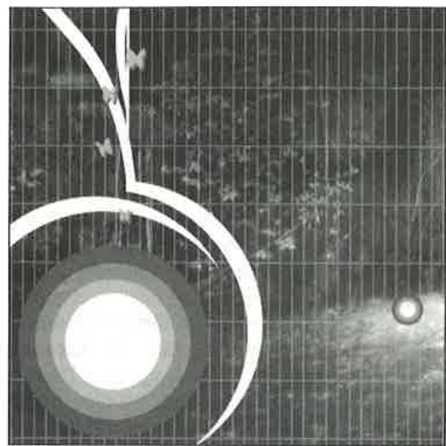
Facebook's Exhibit No. 1005
Page 0032

Page 32 of 140
GOOGLE EXHIBIT 1005



Facebook's Exhibit No. 1005
Page 0033

 GOOGLE EXHIBIT 1005



Chapter 2

INTRODUCTION TO XML

If you work in the web design or development area, you've probably heard of XML. You may have come across it when you were learning how to write web pages or when you started exploring web services. Many software programs share information using XML documents, and Office 2003 for PCs lets you work with XML documents. So what is all the hype about and why should you know about XML?

XML is rapidly becoming the standard for exchanging information between applications, people, and over the Internet. Both humans and computers can read XML documents, and as a format, XML is flexible enough to be adapted for many different purposes.

This chapter introduces you to XML. It explains some of the basic concepts, including the rules governing the structure of XML documents. You'll also learn about some of the uses for XML and the reasons why should you start to use XML in your projects. I show some examples of XML documents, and by the end of the chapter, you'll have a solid understanding of XML and related concepts.

I'll expand on the concepts covered here in Chapter 3, where we'll look at using an XML editor and creating XML content with Office 2003. I'll also look at some related topics—defining XML document rules with schemas and changing the appearance of XML documents with transformations.

Facebook's Exhibit No. 1005
Page 0034

GOOGLE EXHIBIT 1005

What is XML?

Let's start by answering the most basic question: What is XML?

The World Wide Web Consortium (W3C) provides the following definition for XML in their glossary at `www.w3.org/TR/DOM-Level-2-Core/glossary.html`:

Extensible Markup Language (XML) is an extremely simple dialect of SGML. The goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML.

As you can see from this definition, it's very difficult to explain XML in a single sentence or paragraph. To start with, XML stands for **Extensible Markup Language**. Extensible means that you can use XML to create your own languages. The term markup means that the languages you create use tags to surround or mark up text.

XML is not a markup language like HTML. It is a meta-language that you can use to create other markup languages. The languages that you create work with structured data, and you use XML to invent tags that describe your data and the data structures. You can use different tags each time you create an XML document, or you can use the same tags for different documents.

Groups have created their own languages based on XML. This allows them to share information specific to their industry or area of expertise using a common set of markup tags and structures.

One example, Chemical Markup Language (CML), allows scientists to share molecular information in a standardized way. There are specific rules for structuring CML documents and referring to molecular information. MathML is another example of a standard language using XML. XML documents can use MathML to describe mathematical operations.

Extensible HTML (XHTML) is an example that is probably more familiar to you. XHTML was created when HTML was rewritten according to XML rules.

Think about the tags you use in XHTML—<p></p>, <h1></h1>. These tags mark up information on a web page, and you use them in a specific way, according to some predefined rules. For instance, one rule says that you can't include <p></p> tags in the <head> section of a web page.

Being familiar with these rules means that you can open any web page written in XHTML and understand the structure. It also means that any software package that knows the XHTML rules can display a web page.

By itself, XML doesn't do anything other than store information. It's not a programming language in its own right. XML documents need humans or software packages to process the information that they contain.

XML documents work best with structured information such as names and addresses, product catalogs, and lists of documents—anything with a standardized format. You can store hierarchical information within XML documents, a bit like storing information in a database. Instead of breaking the information into tables and fields, you use elements and tags to describe the data.

Facebook's Exhibit No. 1005
Page 0035

 GOOGLE EXHIBIT 1005

This concept is a little easier to explore with an example. Most of us have a phone book that we use to store contact information for our friends and colleagues. You probably have the information in a software package like Microsoft Outlook or Outlook Express.

Your phone book contains many different names but you store the same information about each contact — their name, phone number, and address. The way the information is stored depends on the software package you've chosen. If the manufacturer changed the package or discontinued it, you'd have to find a new way to store information about your contacts.

Transferring the information to a new software program is likely to be difficult. You'd have to export it from the first package, rearrange the contents to suit the second package, and then import the data. Most software applications don't share a standard format for contact data, although some can talk to each other. You have to rely on the standards created by each company.

As an alternative, you can use XML to store the information. You create your own tag names to describe the data; tags like `<contact>`, `<phone>`, and `<address>` provide clear descriptions for your information. Anyone else who looks at the file will be able to understand what information you are storing.

Because your phone book XML document is in a standard format, you can display the details on a web page. Web browsers contain an XML parser to process the XML content. You can also print out your contacts or even build a Flash movie to display and manage your contacts.

Your friends could agree on which tags to use and share their address books with each other. You can all save your contacts in the same place and use tags to determine who has contributed each entry. When you use a standardized structure for storage, the ways that you can work with the information are endless.

How did XML start?

XML has been around since 1998. It is based on Standard Generalized Markup Language (SGML), which in turn was created out of General Markup Language (GML) in the 1960s. XML is actually a simplified version of SGML.

SGML describes how to write languages, specifically those that work with text in electronic documents. SGML is also an international standard—ISO 8879. Interestingly enough, SGML was one of the considerations for HTML when it was first developed.

The first XML recommendation was released in February 1998. Since then, XML has increased in popularity, and it's now a worldwide standard for sharing information. Human beings, databases, and many popular software packages all use XML documents to store and share information. Web services also use an XML format to share information over the Internet.

The W3C developed the XML specification. This organization also works with other recommendations such as HTML and XHTML. Detailed information about the XML specification is available at the W3C's website at `www.w3c.org/XML/`. At the time of writing, the current specification was for XML 1.1. You can view this specification at `www.w3.org/TR/2004/REC-xml11-20040204/`.

Goals of XML

When it created XML, the W3C published the following goals at `www.w3.org/TR/REC-xml/#sec-origin-goals`:

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 which 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.

In other words, XML should be easy to use in a variety of settings, by both people and software applications. The rules for XML documents should be clear so they are easy to create.

So how do we create XML documents?

Creating XML documents

Before we start, it's important to understand what we mean by the term *XML document*. The term refers to a collection of content that meets XML construction rules. When we work with XML, the term document has a more general meaning than with software packages. In Flash, for example, a document is a physical file.

While an XML document can be one or more physical files, it can also refer to a stream of information that doesn't exist in a physical sense. You can create these streams using server-side files; you'll see how this is done later in this book. As long as the information is structured according to XML rules, it qualifies as an XML document.

XML documents contain information and markup. You can divide markup into

- Elements
- Attributes
- Text
- Entities
- Comments
- CDATA

Facebook's Exhibit No. 1005
Page 0037

 GOOGLE EXHIBIT 1005

Elements

Each XML document contains one or more elements. Elements identify and mark up content, and they make up the bulk of an XML document. Some people call elements *nodes*.

Here is an element:

```
<tag>Some text</tag>
```

This element contains two tags and some text. Elements can also include other elements. They can even be empty, i.e., they contain no text.

As in HTML, XML tags start and end with less-than and greater-than signs. The name of the tag is stored in between these signs—`<tagName>`.

The terms element and tag have a slightly different meaning.

A tag looks like this:

```
<tagName>
```

whereas an element looks like this:

```
<tag>Some text</tag>
```

If an element contains information or other elements, it will include both an opening and closing tag—`<tag></tag>`. Empty elements can also be written in a single tag—`<tag/>`—so that

```
<tagname></tagname>
```

is equivalent to

```
<tagname/>
```

There is no preferred way to write empty tags. Either option is acceptable.

You can split elements across more than one line as shown here:

```
<contact>
  Some text
</contact>
```

Each element has a name that must follow a standard naming convention. The names start with either a letter or the underscore character. They can't start with a number. Element names can contain any letter or number, but they can't include spaces. Although it's technically possible to include a colon (`:`) character in an element name, it's not a good idea as these are used when referring to namespaces. You'll understand what that means a little later in the chapter.

You usually give elements meaningful names that describe the content inside the tags. The element name

```
<fullName>Sas Jacobs</fullName>
```

is more useful than

```
<axbjd>Sas Jacobs</axbjd>
```

You can't include a space between the opening bracket < and the element name. You are allowed to include space anywhere else, and it's common to include a space before the /> for empty elements. In the early days of XHTML, older browsers required the extra space for tags such as <br /> and <hr />.

When an element contains another element, the container element is called the *parent* and the element inside is the *child*.

```
<tagname>
  <childTag>Text being marked up</childTag>
</tagname>
```

The family analogy continues with *grandparent* and *grandchild* elements as well as *siblings*.

You can also mix the content of elements, i.e., they contain text as well as child elements:

```
<tagname>
  Text being <childTag>marked up</childTag>
</tagname>
```

The first element in an XML document is called the *root element*, *document root*, or *root node*. It contains all the other elements in the document. Each XML document can have only one root element. The last tag in an XML document will nearly always be the closing tag for the root element.

XML is case sensitive. For example, <phoneBook> and </phonebook> are not equivalent tags and can't be used in the same element. This is a big difference from HTML.

Elements serve many functions in an XML document:

- Elements mark up content. The opening and closing tags surround text.
- Tag names provide a description of the content they mark up. This gives you a clue about the purpose of the element.
- Elements provide information about the order of data in an XML document.
- The position of child elements can show their importance.
- Elements show the relationships between blocks of information. Like databases, they show how one piece of data relates to others.

Attributes

Attributes supply additional information about an element. They provide information that clarifies or modifies an element.

Attributes are stored in the start tag of an element after the element name. They are pairs of names and related values, and each attribute must include both the name and the value:

```
<tagname attributeName="attributeValue">
  Text being marked up
</tagname>
```


Facebook's Exhibit No. 1005
Page 0039

GOOGLE EXHIBIT 1005

Attribute values appear within quotation marks and are separated from the attribute name with an equals sign. You can use either single or double quotes around the attribute value. Interestingly enough, you can also mix and match your quotes in the same element:

```
<tagname attribute1="value1" attribute2='value2'>
```

You might choose to use double quotes where a value contains an apostrophe:

```
<person name="o'mahoney">
```

You would use single quotes where double quotes make up part of the value:

```
<photo caption='It was an "interesting" day'>
```

Keep in mind that tags can't be included within an attribute.

An XHTML image tag provides an example of an element that contains attributes:

```
<img src="logo.gif" width="20" height="15" alt="Company logo"/>
```

There is no limit to number of attributes within an element, but attributes inside the same element must have unique names. When you are working with multiple attributes in an element, the order isn't important.

Attribute names must follow the same naming conventions as elements. You can't start the name with a number, and you can't include spaces in the name. Some attribute names are reserved, and you shouldn't use them in your XML documents. These include

- `xml:lang`
- `xml:space`
- `xml:link`
- `xml:attribute`

You can rewrite attributes as nested elements. The following

```
<contact id="1">
  <name>Sas Jacobs</name>
</contact>
```

could also be written as

```
<contact>
  <id>1</id>
  <name>Sas Jacobs</name>
</contact>
```

There is no one right way to structure elements and attributes. The method you choose depends on your data. The way you're going to process the XML document might also impact on your choices. Some software packages find it harder to work with attributes compared with elements.

Text

Text refers to any information contained between opening and closing element tags. In the line that follows, the text Sas Jacobs is stored between the <fullName> and </fullName> tags:

```
<fullName>Sas Jacobs</fullName>
```

Unless you specify otherwise, the text between the opening and closing tags in an element will always be processed as if it was XML. This means that special characters such as < and > have to be replaced with the entities < and >. The alternative is to use CDATA to present the information, and I'll go into that a little later.

I've listed the common entities that you'll need to use in Table 2-1.

Table 2-1. Entities commonly used in XML documents

Character	Entity
<	<
>	>
'	'
"	"
&	&

Entities

Character entities are symbols that represent a single character. In HTML, character entities are used for special symbols such as an ampersand (&) and a nonbreaking space ( ).

Character entities replace reserved characters in XML documents. All tags start with a less-than sign so it would be confusing to include another one in your code.

```
<expression>3 < 5</expression>
```

This code would cause an error during processing. If you want to include a less-than sign in text, you can use the entity <:

```
<expression>3 &lt; 5</expression>
```

Some entities use Unicode numbers. You can use numbers to insert characters that you can't type on a keyboard. For example, the entity é creates the character **é**—an e with an acute accent. The number 233 is the Unicode number for the character **é**.

You can also use a hexadecimal number to refer to a character. In that case, you need to include an x in the number so the reference would start with &#x. The hexadecimal entity reference for **é** is é.

Facebook's Exhibit No. 1005
Page 0041

GOOGLE EXHIBIT 1005

The Character Map in Windows tells you what codes to use. Open it by choosing Start ➤ All Programs ➤ Accessories ➤ System Tools ➤ Character Map. Figure 2-1 shows the Character Map dialog box.

The bottom left of the window shows the hexadecimal value. Don't forget to remove the trailing zeroes and add &#x to the beginning of the value. The right side shows the Unicode number. Again, you'll need to remove the first 0 from the code.

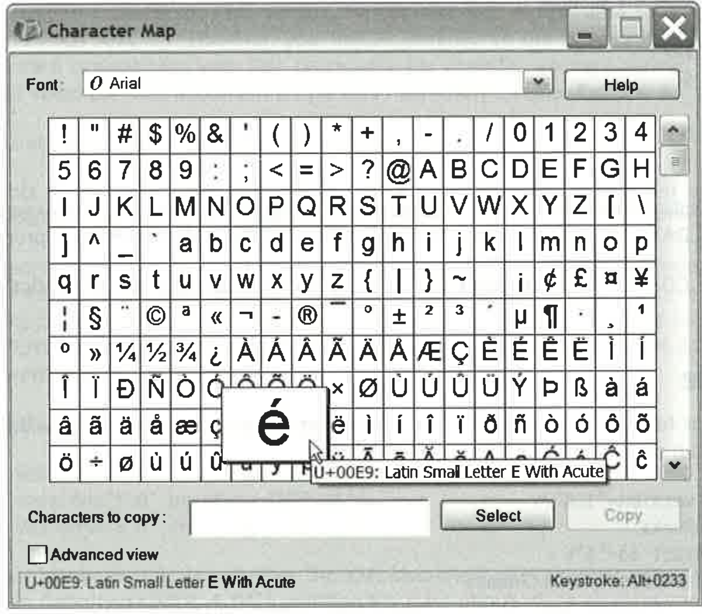


Figure 2-1. The character map in Windows displaying the small letter e with an acute accent

Comments

Comments in XML work the same as in HTML. They begin with the characters <!-- and end with -->:

```
<!-- here is a commented line -->
```

Comments are a useful way to leave messages for other users of an XML document without affecting the way the XML document is processed. In fact, processing software always ignores comments in XML documents. You can also use comments to hide a single line or a block of code.

The only requirements for comments in XML documents are that

- A comment can't appear before the first line XML declaration.
- Comments can't be nested or included within tag names.
- You can't include --> inside a comment.
- Comments shouldn't split tags, i.e., you shouldn't comment out just a start or ending tag.

Facebook's Exhibit No. 1005
Page 0042

GOOGLE EXHIBIT 1005

CDATA

CDATA stands for character data. CDATA blocks mark text so that it isn't processed as XML. For example, you could use CDATA for information containing characters such as < and >. Any < or > character contained within CDATA won't be processed as part of a tag name.

CDATA sections start with <![CDATA and finish with]>. The character data is contained within square brackets [ ] inside the section:

```
<![CDATA[
  3 < 5
  or
  2 > 0
]]>
```

Entities will display literally in a CDATA section so you shouldn't include them. For example, if you add < to your CDATA block it will display the same way when the XML document is processed.

The end of a CDATA section is marked with the]]> characters so you can't include these inside CDATA.

An example

The listing that follows shows a simple XML document. I'll explain this in detail a little later in the chapter. You can see elements, attributes, and text:

```
<?xml version="1.0"?>
<phoneBook>
  <contact id="1">
    <name>Sas Jacobs</name>
    <address>123 Some Street, Some City, Some Country</address>
    <phone>123 456</phone>
  </contact>
</phoneBook>
```

XML document parts

An XML document contains different parts. It will always start with a *prolog*. The remainder of the XML document is contained within the document root or root element.

Document prolog

The document prolog appears at the top of an XML document and contains information about the XML document as a whole. It must appear before the root element in the document. The prolog is a bit like the <head> section of an HTML document. It can also include comments.

Facebook's Exhibit No. 1005
Page 0043

 GOOGLE EXHIBIT 1005

XML declaration

The prolog usually starts with an XML declaration, although this is optional. If you do include a declaration, it must be the first line of your XML document. The declaration tells software applications and humans that the content is an XML document:

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

The XML declaration includes an XML version, in this case 1.0. At the time of writing, the latest recommendation was XML 1.1. However, you should continue to use the `version="1.0"` attribute value for backward compatibility with XML processors. For example, adding a version 1.1 declaration causes an error when the XML document is opened in Microsoft Internet Explorer 6.

The XML declaration can also include the `encoding` and `standalone` attributes.

XML documents contain characters that follow the Unicode standard, maintained by the Unicode Consortium. You can find out more at `www.unicode.org/`.

Encoding determines the character set for the XML document. You can use Unicode character sets UFT-8 and UTF-16 or ISO character sets like ISO 8859-1, Latin-1 Western Europe. If no encoding attribute is included, it is assumed that the document uses UTF-8 encoding. Languages like Japanese and Chinese need UTF-16 encoding. Western European languages often use ISO 8859-1 to cope with the accents that aren't part of the English language.

The `encoding` attribute must appear after the `version` attribute:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml version="1.0" encoding="UTF-16"?>
<?xml version="1.0" encoding="ISO-8859-1">
```

The `standalone` attribute indicates whether the XML document uses external information, such as a Document Type Definition (DTD). A DTD specifies the rules about which elements and attributes to use in the XML document. It also provides information about the number of times each element can appear and whether an element is required or optional.

The `standalone` attribute is optional but must appear as the last attribute in the declaration. The value `standalone="no"` can't be used when you are including an external DTD or style sheet.

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

Processing instructions

The prolog can also include processing instructions (PI). These instructions pass information about the XML document to other applications.

Processing instructions start with `<?` and finish with `?>`. The first item in a PI is a name, called the PI target. PI names that start with *xml* are reserved.

A common PI is the inclusion of an external XSLT style sheet. This PI must appear before the document root:

```
<?xml-stylesheet type="text/xsl" href="listStyle.xsl"?>
```


Facebook's Exhibit No. 1005
Page 0044

Page 44 of 140
GOOGLE EXHIBIT 1005

Processing instructions can also appear in other places in the XML document.

Document Type Definitions

Document Type Definitions (DTDs), or DOCTYPE declarations, appear in the prolog. These are rules about the elements and attributes within the XML document. A DTD provides information about which elements are legal in an XML document and tells you which elements are required and which are optional. In other words, a DTD provides the rules for a valid XML document.

The prolog can include a set of declarations about the XML document, a reference to an external DTD, or both. This code shows an external DTD reference:

```
<?xml version="1.0"?>
<!DOCTYPE phoneBook SYSTEM "phoneBook.dtd">
```

We'll look at DTDs in more detail in Chapter 3.

Tree

Everything that isn't in the prolog is contained within the document tree. This includes the elements, attributes, and text in a hierarchical structure. The root node is the trunk of the tree. You call the child elements of the root node branches.

As we've seen, elements can include other elements or attributes. They can also contain text values or a mixture of both. HTML provides good examples of mixed content.

```
<p>This is a paragraph element with an element <br/> inside</p>
```

This distinction becomes important when you use a schema to describe the structure of the document tree.

Document root

An XML document can have only one root element. All of the elements within an XML document are contained within this root element.

The root element can have any name at all, providing that it conforms to the standard element naming conventions. In HTML documents, you can think of the <html> tag as the root element.

White space

XML documents include white space so that humans can read them more easily. White space refers to spaces, tabs, and returns that space out the content in the document. The XML specification allows you to include white space anywhere within an XML document except before the XML declaration.

XML processors do take notice of white space in a document, but many won't display the spaces. For example, Internet Explorer won't display more than one space at a time when it displays an XML or XHTML document.

Facebook's Exhibit No. 1005
Page 0045

Page 45 of 140
GOOGLE EXHIBIT 1005

If white space is important, maybe for poetry or a screenplay, you can use the xml:space attribute in an element. There are two possible values for this attribute: default and preserve. Choosing the default value is the same as leaving out the attribute.

You can add the xml:space="preserve" attribute to the root node of a document to preserve all space within the document tree:

```
<phoneBook xml:space="preserve">
```

Namespaces

XML documents can get very complicated. One XML document can reference another XML document, and different rules may apply for each. When this happens, it's possible that two different XML documents will use the same element names.

In order to overcome this problem, we use *namespaces*. Namespaces associate XML elements with an owner. A namespace ensures that each element name is unique within a document, even if other elements use the same name.

You can find out more about namespaces by reading the latest recommendation at the W3C website. At the time of writing, this was the "Namespaces in XML 1.1" recommendation at www.w3.org/TR/2004/REC-xml-names11-20040204/.

It isn't compulsory to use namespaces in your XML documents, but it can be a good idea. Namespaces are also useful when you start to work with schemas and style sheets. We'll look at some examples of schemas and style sheets in the next chapter.

Each namespace includes a reference to a Uniform Resource Identifier (URI). A URI is an Internet address, and each URI must be unique in the XML document. The URIs used in an XML document don't have to point to anything, although they often will.

You can define a namespace using the xmlns attribute within an element. Each namespace usually has a prefix that you use to identify elements belonging to that namespace. You can't start your prefixes with *xml*, and they shouldn't include spaces.

```
<FOE:fullName xmlns:FOE="http://www.friendsofed.com/">
  Sas Jacobs
</FOE:fullName>
```

In the preceding element, the FOE prefix refers to the namespace http://www.friendsofed.com/. I've prefixed the element <fullName> with FOE, and I can use it with other elements and attributes.

```
<FOE:address>
  123 Some Street, Some City, Some Country
</FOE:address>
```

I'll then be able to tell that the <address> element also comes from the http://www.friendsofed.com/ namespace.

You can also define a namespace without using a prefix. If you do this, the namespace will apply to all elements that don't have a prefix or namespace defined.

Facebook's Exhibit No. 1005
Page 0046

GOOGLE EXHIBIT 1005

The following listing shows how to use a namespace with no prefix in an XML element:

```
<contact id="1" xmlns="http://www.friendsofed.com/">
  <name>Sas Jacobs</name>
  <address>123 Some Street, Some City, Some Country</address>
  <phone>123 456</phone>
</contact>
```

The namespace applies to all the child elements of the <contact> element so the <name>, <address>, and <phone> elements will use the default namespace http://www.friendsofed.com/.

Namespaces will become clearer when we start working with schemas and style sheets in Chapter 3.

A simple XML document

So far, we've covered some of the rules for creating XML documents. We've looked at the different types of content within an XML document and seen some XML fragments. Now it's time to put these rules together to create a complete XML document.

The following listing shows a simple XML document based on the phone book that I talked about earlier. I use the example throughout the rest of this chapter.

```
<?xml version="1.0"?>
<phoneBook>
  <contact id="1">
    <name>Sas Jacobs</name>
    <address>123 Some Street, Some City, Some Country</address>
    <phone>123 456</phone>
  </contact>
  <contact id="2">
    <name>John Smith</name>
    <address>4 Another Street, Another City, Another Country</address>
    <phone>456 789</phone>
  </contact>
</phoneBook>
```

I've saved this document in the resource file address.xml.

The first line declares the document as an XML document. The declaration is not required, but it's good practice to include it. A software package that opens the file will immediately identify it as an XML document.

The remaining lines of the XML document contain elements. The first element, <phoneBook>, contains the other elements: <contact>, <name>, <address>, and <phone>. There is a hierarchical relationship between these elements.

There are two <contact> elements. They share the same *parent*, <phoneBook>, and are *child nodes* of that element. They are also *siblings* to each other.

Facebook's Exhibit No. 1005
Page 0047

GOOGLE EXHIBIT 1005

The <contact> tag is a container for the <name>, <address>, and <phone> elements, and they are *child elements* of the <contact> tag. The <name>, <address>, and <phone> elements are *grandchildren* of the <phoneBook> element.

You'll notice that the last line is a closing </phoneBook> tag written with exactly the same capitalization as the first tag.

Each <contact> tag has a single attribute—id. Attributes normally provide extra information about a specific element, in this case a unique identifier for each <contact>.

As we saw earlier, the element

```
<contact id="1">
```

can be rewritten as

```
<contact>
  <id>1</id>
</contact>
```

In this document tree, the trunk of the tree is the <phoneBook> tag. Branching out from that are the <contact> tags, and each <contact> has <name>, <address>, and <phone> branches.

Figure 2-2 shows the relationship between the elements in the phone book XML document.

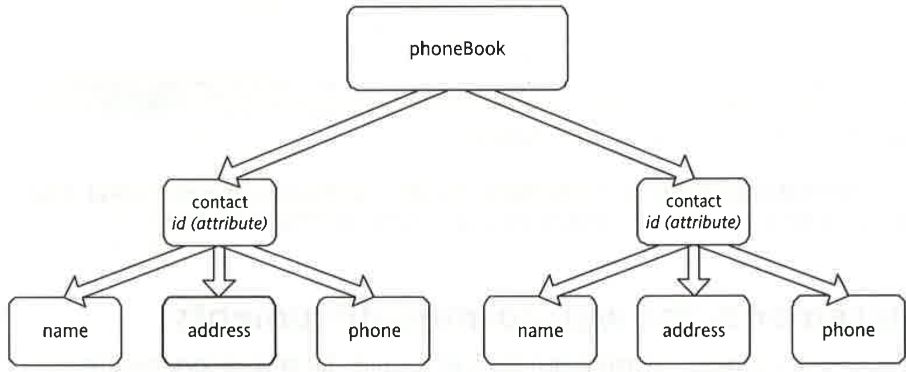


Figure 2-2. The hierarchy of elements within the phone book XML document

In this example, I've created my own tag names. The names I've chosen tell you about the type of information that I'm working with so it's easy to figure out what I'm describing.

If I want to share the rules for my phone book XML document with other people, I can create a DTD or XML schema to describe how to use the tags. Adding a reference to the DTD or schema will ensure that any XML documents that I create follow the rules. This process is called *validating* an XML document. I'll look at working with DTDs and schemas in the next chapter.

Facebook's Exhibit No. 1005
Page 0048

Page 48 of 140

GOOGLE EXHIBIT 1005

I can view an XML document by opening it in a web browser. Figure 2-3 shows `address.xml` displayed in Internet Explorer.

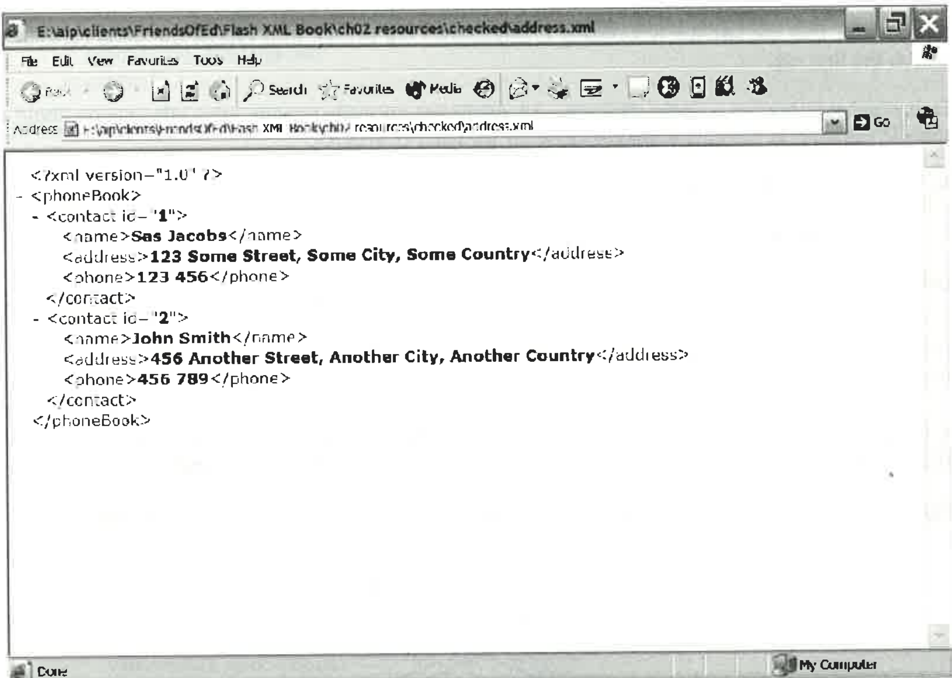


Figure 2-3. An XML document opened in Internet Explorer

You can see that Internet Explorer has formatted the document to make it easier to read. It has also added some minus signs that I can click to collapse branches of the document tree.

Requirements for well-formed documents

In the preceding sections, I've mentioned some of the rules for creating XML documents. In this section, we look at these rules in more detail. Documents that meet the requirements are said to be *well formed*.

XHTML provides us with a standard set of predefined tags. We have to use the `<ul> <li> </li> </ul>` tags when we want to create a list. Because there are no predefined tags in XML documents, it's important that the rules for creating documents are strict. You can create any tags you like, providing that you stick to these rules.

Facebook's Exhibit No. 1005
Page 0049

 GOOGLE EXHIBIT 1005

Well-formed documents meet the following criteria:

- The document contains one or more elements.
- The document contains a single root element, which may contain other nested elements.
- Each element closes properly.
- Start and end tags have matching case.
- Elements nest correctly.
- Attribute values are contained in quotes.

I'll look at each of these rules in a little more detail.

Element structure

An XML document must have at least one element: the document root. It doesn't have to have any other content, although in most cases it will.

The following XML document is well formed as it contains a single element <phoneBook>:

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

Of course, this document doesn't contain any information so it's not likely to be very useful.

It's more likely that you'll create an XML document where the root element contains other elements. The following listing shows an example of this structure:

```
<?xml version="1.0"?>
<phoneBook>
  <contact id="1">
    <name>Sas Jacobs</name>
    <address>123 Some Street, Some City, Some Country</address>
    <phone>123 456</phone>
  </contact>
</phoneBook>
```

As long as all of the elements are contained inside a single root element, the document is well formed.

This listing shows a document without a root element. This document is not well formed.

```
<?xml version="1.0"?>
<contact id="1">
  <name>Sas Jacobs</name>
</contact>
<contact id="2">
  <name>John Smith</name>
</contact>
```

Elements must be closed

You must close all elements correctly. The way you do this depends on whether or not the element is empty, i.e., whether it contains text or other elements.

You can close empty elements by adding a forward slash to the opening tag:

```
<name/>
```

In the case of a nonempty element, you have to add a closing tag, which must appear after the opening tag:

```
<name>Sas Jacobs</name>
```

You can also write empty elements with a closing tag:

```
<name></name>
```

As XML is case sensitive, start and end tag names must match exactly. The following examples are incorrect:

```
<name>Sas Jacobs</Name>
<Name>Sas Jacobs</name>
```

You would rewrite them as

```
<name>Sas Jacobs</name>
```

The following example is also incorrect:

```
<name>Sas Jacobs
<name>John Smith
```

The elements have an opening tag but no corresponding closing tag. This rule also applies to XHTML. In XHTML, you can't use the following code, which was acceptable in HTML:

```
<p>A paragraph of information.
<p>Another paragraph.
```

I'll talk about the differences between XML, HTML, and XHTML a little later in this chapter.

Elements must nest correctly

You must close elements in the correct order. In other words, child elements must close before their parent elements.

This line is incorrect:

```
<contact><name>Sas Jacobs</contact></name>
```

and should be rewritten as

```
<contact><name>Sas Jacobs</name></contact>
```

Facebook's Exhibit No. 1005
Page 0051

GOOGLE EXHIBIT 1005

Use quotes for attributes

All attribute values must be contained in quotes. You can either use single or double quotes; these two lines are equivalent:

```
<contact id="1">
<contact id='1'>
```

If your attribute value contains a single quote, you have to use double quotes, and vice versa:

```
<contact name="O'Malley"/>
```

or

```
<contact nickname='John "Bo bo" Smith'/>
```

You can also replace the quote characters inside an attribute value with character entities:

```
<contact name="O&apos;Malley"/>
contact nickname='John &quot;Bo bo&quot; Smith'/>
```

Documents that aren't well formed

If you try to view an XML document that is not well formed, you'll see an error. For example, opening a document that isn't well formed in a web browser will cause an error message similar to the one shown in Figure 2-4. This is quite different from HTML documents; most web browsers will ignore any HTML errors such as missing </p> tags.

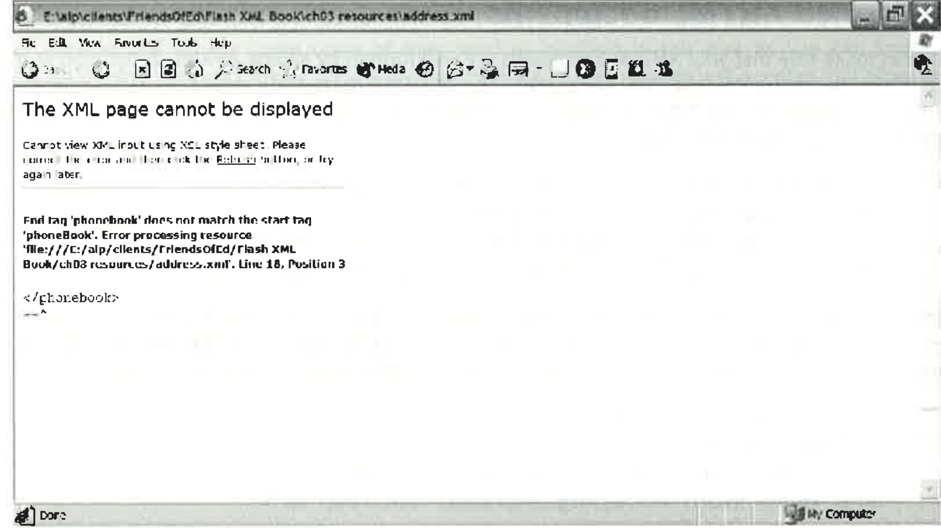


Figure 2-4. Internet Explorer displaying a document that isn't well formed

Facebook's Exhibit No. 1005
Page 0052

GOOGLE EXHIBIT 1005

An XML editor such as XMLSpy often provides more detailed information about the error. You can see the same XML document displayed in XMLSpy in Figure 2-5.

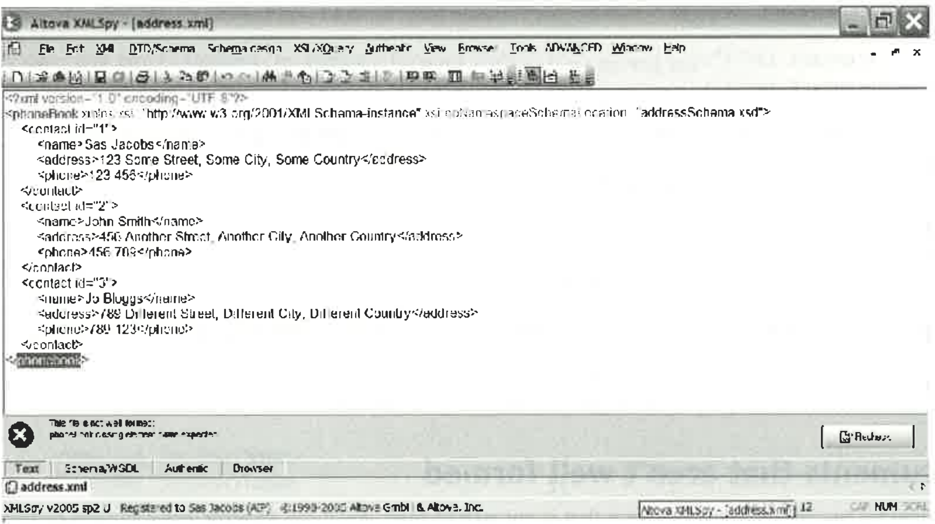


Figure 2-5. A document that is not well formed displayed in XMLSpy

The error message shows that the closing element name </phoneBook> was expected.

Well-formed XHTML documents

You can make sure that your XHTML documents are well formed by adding an XML declaration at the top of the file before the DOCTYPE declaration. The DOCTYPE declaration should contain a reference to the appropriate XHTML DTD. The DTD can specify strict or transitional conformance by including one of the following two declarations:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Strictly conforming documents must meet the mandatory requirements in the XHTML specification. If you are declaring a strictly conforming document, you should include a namespace in the <html> tag. The following listing shows the W3C recommendation for well-formed XHTML documents:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
```

You can see this W3C recommendation at www.w3.org/TR/xhtml11/conformance.html.

Facebook's Exhibit No. 1005
Page 0053

GOOGLE EXHIBIT 1005

Working with XML documents

You can work with XML documents in many different ways. To start with, you need to figure out how to create the document. For example, you can write the document yourself in a text editor or have it generated automatically by a software package. You can even create a stream of XML information by running a server-side web page through a web server.

You also need to consider how to work with your XML documents. Will you view them in a web browser? Will you display and update the document in Office 2003? Maybe you'll create a Flash movie that displays and updates the XML document.

Generating XML content

The simplest way to create an XML document is by typing the tags and data in your favorite text editor and saving it with an `.xml` extension. At the very minimum, you must follow the rules for well-formed XML content. You can also create a DTD or schema to describe the rules for your elements. This will allow you to ensure that your content is valid.

You can also use an XML editor to help create content. Many commercial XML editing tools are on the market. Search for **XML editors** in your favorite search engine to see a current list. I like to use XMLSpy, and I'll show you more about it in the next chapter. One advantage of XML editors is that they will color-code XML documents as well as provide code hints.

Software packages can generate XML documents automatically. Files written in PHP, ColdFusion, ASP.NET, or any other server-side language can generate XML from a database or another source. Microsoft Office 2003 for PCs also allows you to save an Office document in XML format.

Whenever you *consume* or use a web service, you'll receive the information that you request in an XML document. A web service is like an application that you can use across the Internet. Companies like Amazon and Google make some of their services available in this way. As you can imagine, Amazon doesn't want you poking around in their database so they provide web services that allow you to carry out various searches. They protect their data but still give you access.

To use a web service, you need to send a request to the provider. The request is formatted in a specific way, and there will be different requirements for each web service that you consume. You get the results back in XML format. We'll find out more about web services later in this book.

If you've used a news feed, the RSS (Rich Site Summary or Really Simple Syndication) format is an XML document. RSS uses XML to describe website changes and is really a type of web service. There are different RSS versions that you can use to provide a news feed from a website or organization. The specification for the most recent version, RSS 2.0, is at `http://blogs.law.harvard.edu/tech/rss`.

One example of an RSS feed is the news service from Macromedia. You can find out about the news feed from `www.macromedia.com/devnet/articles/xml_resource_feed.html` and view the content at `www.macromedia.com/devnet/resources/macromedia_resources.rdf`. This news feed provides information from the Macromedia Developer Center using RSS 1.0.

Facebook's Exhibit No. 1005
Page 0054

GOOGLE EXHIBIT 1005

Using XML information

Software programs access the information in an XML document through the XML Document Object Model (DOM). The DOM is a W3C specification for programming languages that you use with XML documents. At the time of writing, version 3.0 of the DOM specification was available at `www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/`.

The general term for any software package that processes XML documents is an *XML processor*. Many different software packages fall into this category. You can use them to view, extract, display, and validate your XML data. Word, Office, and Excel 2003 can exchange XML information and so are examples of XML processors.

XML parsers are one category of XML processors. Parsers can read through the content of an XML document and extract individual elements, attributes, and text. Parsers need to be able to separate processing instructions from elements, attributes, and text. Flash has a built-in XML parser that allows you to work with XML documents and include them in Flash movies.

XML parsers first check to see if a document is well formed. They can't use documents that aren't well formed. Earlier we saw an error message from Internet Explorer when it tried to open a document that wasn't well formed.

XML parsers fall into two categories—nonvalidating and validating parsers. All parsers check to see if a document is well formed. Validating parsers also compare the structure of the document with a DTD or XML schema to check that it is constructed correctly. A document that meets the rules listed within a DTD or schema is considered *valid* by the validating parser.

Most Web browsers are capable of displaying an XML file so that you can see the structure and content. They contain built-in parsers to process and display the XML document appropriately.

Both Internet Explorer and Mozilla Firefox contain nonvalidating XML parsers. They allow you to open and display an XML file just as you would any web page. When you open an XML file, the information structure displays using the default settings of the web browser.

Internet Explorer 6 contains version 3 of the MSXML Parser. You saw the file `address.xml` file as it appears when opened in Internet Explorer earlier in the chapter. Figure 2-6 shows the same file open in Firefox.

Facebook's Exhibit No. 1005
Page 0055

 GOOGLE EXHIBIT 1005

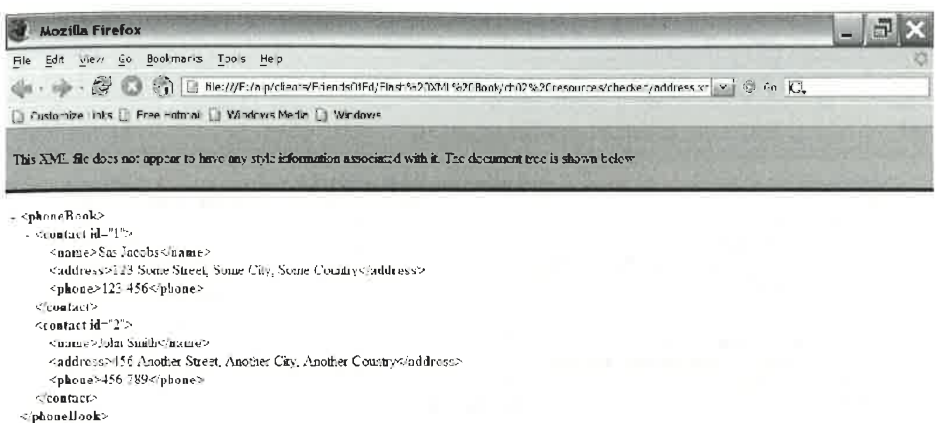


Figure 2-6. An XML file open in Firefox

You can download tools for Internet Explorer that will allow you to validate an XML file against an embedded schema. The download is called **Internet Explorer Tools for Validating XML and Viewing XSLT Output**; visit the Microsoft Download Center at `www.microsoft.com/downloads/search.aspx` for the relevant files.

By default, the tools install in a folder called `IEXMLTLS`. You'll need to right-click the `.inf` files in the folder and choose the Install option before the tools will be available within Internet Explorer.

After you have installed the IE XML tools, you can right-click an XML page that's open in Internet Explorer. The shortcut menu will have two extra options: Validate XML and View XSL Output. Figure 2-7 shows the context menu.

Facebook's Exhibit No. 1005
Page 0056

GOOGLE EXHIBIT 1005

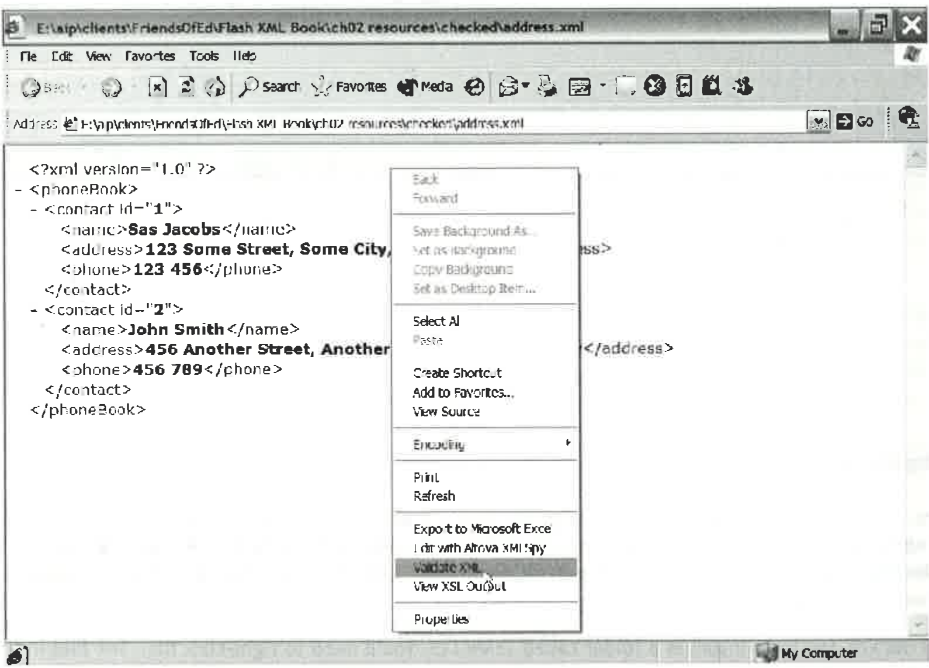


Figure 2-7. After installing the Internet Explorer XML tools, you can right click in the browser window to validate the XML or view XSL output.

Firefox contains a tool called the DOM Inspector, which displays information about the structure of the XML document. Choose Tools ➤ DOM Inspector to view the structure. Figure 2-8 shows this tool.

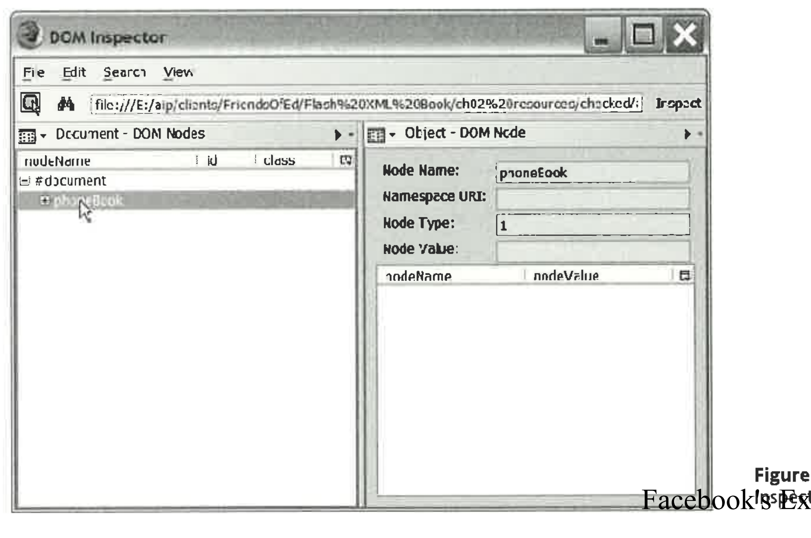


Figure 2-8. The DOM Inspector in Firefox

Facebook's Exhibit No. 1005
Page 0057

Page 57 of 140
GOOGLE EXHIBIT 1005

Flash contains a nonvalidating XML parser. This means that Flash won't check whether the XML document meets the rules set down in a DTD or XML schema. When you read in an XML document, Flash is able to convert the information into a document tree. You can then move around the tree to extract information for your Flash movie.

XML, HTML, and XHTML

Although the terms XML, HTML, and XHTML all sound similar, they're really quite different. XML and HTML are not competing technologies. They are both used for managing and displaying online information but both do different things. XML doesn't aim to replace HTML as the language for web pages. XHTML is a hybrid of the two languages.

HTML tags deal with both the information in a web page and the way it displays. In other words, HTML works with both presentation and content. It doesn't deal with the structure of the information or the meaning of different pieces of data. You can't use HTML to transform the display of information into a completely different layout. If you store information in a table, you can't easily change it into a list.

HTML was designed as a tool for sharing information online in web pages. The complex designs that appear in today's web pages weren't part of the original scope of HTML. As a result, designers often use HTML in ways that were never dreamed of when the language was first created.

The rules for using HTML aren't terribly strict. For example, you can add headings by using the tags `<h1>` to `<h6>`. The `<h1>` tag is the first level of heading, but there is no requirement to include heading tags in any particular order. The first heading in your HTML page could actually be enclosed in an `<h3>` or `<h4>` tag.

Web pages written in HTML can contain errors that don't affect the display of the information. For example, in many browsers, you could include two `<title>` tags and the page would still load. You can also forget to include a closing `</table>` tag and the table will still be rendered.

HTML is supposed to be a standard, but it works differently across web browsers. Most web developers know about the problems in designing a website so it appears the same way in Internet Explorer, Opera, Firefox, and Netscape Browser for both PCs and Macs.

Like XML, HTML comes from the Standard Generalized Markup Language (SGML). Unlike XML, HTML is not extensible. You're stuck with a standard set of tags that you can't change or extend in any way.

XML only deals with content. It describes the structure of information without concerning itself with the appearance of that information. An XML document can show relationships in your data just like a database. This just isn't possible in an HTML document.

XML content is probably easier to understand than HTML. The names of tags normally describe the data they mark up. In the example file `address.xml`, tag names such as <address> and <phone> tell you what data is contained in the element.

XML may be used to display information directly in a web page. It's more likely, though, that you'll use the XML document behind the scenes. It will probably provide the content for a web application or a Flash movie.

Facebook's Exhibit No. 1005
Page 0058

GOOGLE EXHIBIT 1005

Compared with HTML, XML is much stricter about the way markup is used. There are rules about how tags are constructed, and we've already seen that XML documents have to be well formed. A DTD or schema can also provide extra rules for the way that elements are used. These rules can include the legal names for tags and attributes, whether they're required or optional, as well as the number of times that each element must appear. In addition, schemas specify what data type must be used for each element and attribute.

XML documents don't deal with the display of information. If you need to change the way XML data looks, you can change the appearance by using Cascading Style Sheets (CSS) or Extensible Stylesheet Language (XSL). XSL transformations offer the most power; you can use them to create XHTML from an XML document or to sort or filter a list of XML elements.

XHTML evolved so that the useful features of XML could be applied to HTML. The W3C says that XML *reformulated* HTML into XHTML. XHTML documents have much stricter construction rules and are generally more robust than their HTML counterparts.

The HTML specification provides a list of legal elements and attributes within XHTML. XML governs the way that the elements are used in documents. For example, in XHTML, you must close all tags. The HTML `<br>` tag has to be rewritten as `<br/>` or `<br></br>`. In XHTML, web designers can't use a single `<p>` tag to create a paragraph break as they could in HTML.

Another change is that you must write attribute values in full. For example

```
<input type="radio" value="JJJ" checked/>
```

has to be written as

```
<input type="radio" value="JJJ" checked="checked"/>
```

You can find the XHTML specification at `www.w3.org/TR/xhtml1/`. It became a recommendation in 2000 and was revised in 2002.

I've summarized the main changes from HTML to XHTML:

- You should include a DOCTYPE declaration specifying that the document is an XHTML document.
- You can optionally include an XML declaration.
- You must write all tags in lowercase.
- All elements must be closed.
- All attributes must be enclosed in quotation marks.
- All tags must be correctly nested.
- The `id` attribute should be used instead of `name`.
- Attributes can't be minimized.

The following listing shows the previous `address.xml` document rewritten in XHTML. I've done this so you can compare XHTML and XML documents.

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```


Facebook's Exhibit No. 1005
Page 0059

GOOGLE EXHIBIT 1005

```
<html>
<body>
  <table>
  <tr>
    <td>Sas Jacobs</td>
    <td>123 Some Street, Some City, Some Country</td>
    <td>123 456</td>
  </tr>
  <tr>
    <td>John Smith</td>
    <td>4 Another Street, Another City, Another Country</td>
    <td>456 789</td>
  </tr>
</table>
</body>
</html>
```

Notice that the file includes both an XML declaration and a DOCTYPE declaration. You can see the content in the resource file `address.html`.

You're probably used to seeing information like this in web pages. A table displays the content and lists each contact in a separate row. Figure 2-9 shows this document opened in Internet Explorer.

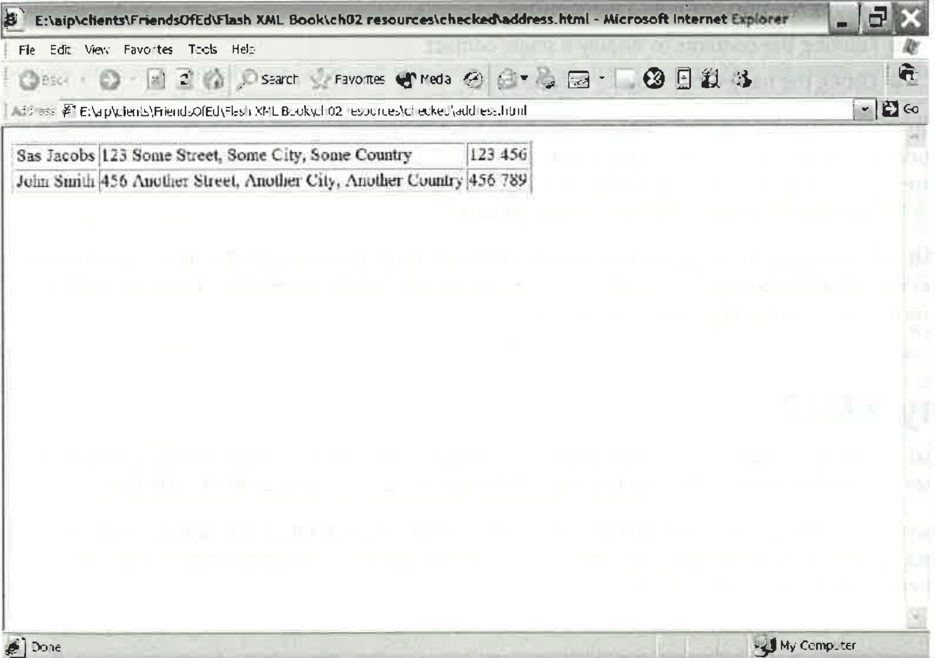


Figure 2-9. An HTML file displayed in Internet Explorer

Facebook's Exhibit No. 1005
Page 0060

GOOGLE EXHIBIT 1005

I've rewritten the content in XHTML so that it conforms with the stricter rules for XML documents. However, the way the document is constructed may still cause some problems. Each piece of information about my contacts is stored in a separate cell within a table. The <td> tags don't give me any clue about what the cell contains. I get a better idea when I open the page in a web browser.

It would be difficult for me to use a software program to extract the content from the web page. I could remove the <td> tags and add the content to a database, but if the order of the table columns changed, I might end up with the wrong data in the wrong database field. There's no way to associate the phone number with the third column.

The web page controls the display of information. Although I can make some minor visual adjustments to the table using style sheets, I can't completely transform the display. For example, I can't remove the table and create a vertical listing of all entries without completely rewriting the XHTML.

Each time I print the document, it will look the same. I can't exclude information such as the address column from my printout. I don't have any way to filter or sort the information. I am not able to extract a list of contacts in a specific area or sort into contact name order.

Compare this case with storing the information in an XML document. I can create my own tag names and write a schema that describes how to use these tags. When I view the document in a web browser, the tag names make it very clear what information they're storing.

I can apply a transformation to change the appearance of an XML document, including

- Sorting the document into name order
- Filtering the contents to display a single contact
- Listing the names in a table or bulleted list

XML isn't a replacement for XHTML documents, but it certainly provides much more flexibility for working with data. You're likely to use XML documents differently from XHTML documents. XML documents are a way to store structured data that may or may not end up in a web page. You normally use XHTML only to display content in a web browser.

XML offers many advantages compared with other forms of data storage. Before I explore what you can do with XML documents, I think it's important to understand the benefits of working with XML. I'll look at this more closely in the next section.

Why XML?

XML is simple, flexible, descriptive, accessible, independent, precise, and free! Using it in Flash will save you maintenance time. What more incentive could you need to start working with it?

You've seen the advantages that XML offers over HTML and XHTML when working with structured data. Given the strong support for XML in Flash, there's bound to be some project in the near future where you'll need to use XML data.

Facebook's Exhibit No. 1005
Page 0061

GOOGLE EXHIBIT 1005

Simple

The rules for creating XML documents are simple. You just need a text editor or another software package capable of generating XML. The only proviso is that you follow some basic rules so that the XML document is well formed.

Reading an XML document is also simple. Tag names are normally descriptive so you can figure out what data each element contains. The hierarchical structure of elements allows you to work out the relationships between each piece of information. When you use XML documents, you don't have to separate out extra style elements when reading an XML document.

Flexible

One key aspect of XML is its flexibility. As long as you follow some simple rules, you can structure an XML document in any way you like. The choice of tag names, attributes, and structures is completely flexible so you can tailor it to suit your data.

Unless you're working with an existing XML-based language such as XHTML, you are not restricted to a standard list of tags. For example, in XHTML, you have to use an `<h1>` tag to display a title on your web page; you can't create your own tag `<pageTitle>`.

You can share information about your XML-based language with other people by using a DTD or schema to describe the "grammar," or rules, for the language. While both types of documents serve the same purpose, schemas use XML to describe the syntax. So if you know XML, you know the basic rules for writing schemas.

Software programs can also use DTDs and schemas. This allows them to map XML elements and work with specific parts of XML documents. For example, Excel 2003 for PCs uses schemas when exporting XML documents. The schema describes the name for each tag, the type of data it will contain, and the relationships among each of the elements.

XML documents provide data for use in different applications. You can generate an XML document from a corporate software package, transform it to display on a website, share it with staff on portable devices, use it to create PDF files, and provide it to other software packages. You can reuse the same data in several different settings. The ability to repurpose information is one of XML's key strengths.

The way XML information displays is also flexible. You can display any XML document in a web browser to see the structure of elements. You can also use other technologies or software packages to change the display quite dramatically. For example, you could transform your phone book XML document into

- A printed list of names and numbers sorted into name order
- A web page displaying the full details of each entry in a table
- A Flash movie that allows you to search for a contact

I'm sure you can think of many more ways to use a phone book XML document.

Facebook's Exhibit No. 1005
Page 0062

GOOGLE EXHIBIT 1005

Descriptive

Because you can choose your own tag names, your XML document becomes a description of your data. Some people call XML documents *self-describing*.

It's easy for humans to understand the content of an XML document just by looking at the tag names. It's also unambiguous for computers, providing they know the rules and structures in the XML document.

In our XHTML page, we could only describe each table cell using the tag `<td>`. The corresponding XML document used tags like `<name>`, `<address>`, and `<phone>`, so it was easy to determine what information each element contained.

The hierarchy in elements means that XML documents show relationships between information in a similar way to a database. The hierarchies in the phone book document tell me that each contact has a name, address, and phone number and that I can store many different contacts.

Accessible

XML documents separate data from presentation so you can have access to the information without worrying about how it displays. This makes the data accessible to many different people, devices, and software packages at the same time. For example, my phone book XML document could be

- Read aloud by a screen reader
- Displayed on a website
- Printed to a PDF file
- Processed automatically by a software package
- Viewed on a mobile phone

XML documents use Unicode for their standard character sets so you can write XML documents in any number of languages. A Flash application could offer multilingual support simply by using different XML documents within the same movie. Switch to a different XML document to display content in an alternative language. The Le@rning Federation example referred to in Chapter 1 does exactly that.

Independent

XML is platform and device independent. It doesn't matter if you view the data on a PC, Macintosh, or handheld computer. The data is still the same and people can exchange it seamlessly. Programmers can also use XML to share information between software packages that otherwise couldn't communicate with each other.

You don't need a specific software package to work with XML documents. You can type the content in just about any package capable of receiving text. The document can be read in a web browser, text editor, or any other XML processor. XML documents can query databases to provide a text-based alternative. In the case of web services, XML is an intermediary between you and someone else's database.

Facebook's Exhibit No. 1005
Page 0063

 GOOGLE EXHIBIT 1005

XML doesn't have "flavors" that are specific to a single web browser, version, or operating system. You don't have to create three different versions of your XML document to cater for different viewing conditions.

Precise

XML is a precise standard. If you want your XML document to be read by an XML parser, it must be well formed. Documents that aren't well formed won't display. Compare this with HTML files. Even when it contains fundamental errors, the web page will still display in a web browser.

When a schema or DTD is included within an XML document, you can validate the content to make sure that the structure conforms to the rules you've set down. Less strict languages like HTML don't allow you to be this precise with your content. XML documents with schemas provide standards so there is only one way that the data they contain can be structured and interpreted.

Free

XML is a specification that isn't owned by any company or commercial enterprise. This means that it's free to use XML—you don't have to buy any special software or other technology. In fact, most major software packages either support XML or are moving so that they will support it in the future.

XML is a central standard in a whole family of related standards. These recommendations work together to create an independent framework for managing markup languages. Table 2-2 shows some of the other related recommendations from the W3C.

Table 2-2. Some of the main XML-related recommendations from the W3C

Recommendation	Purpose
XML Schema Definition (XSD)	Schemas describe the structure and syntax of an XML document.
Extensible Stylesheet Language (XSL)	XSL determines the presentation of XML documents. It uses XSL Transformations (XSLT), XML Path Language, and XSL Formatting Objects (XSL-FO).
XSL Transformations (XSLT)	XSLT transforms one XML document into another XML document.
XML Path Language (XPath)	XPath navigates or locates specific parts of XML documents.
XSL Formatting Objects (XSL-FO)	XSL-FO specifies formatting to be applied to an XML document.
XML Linking Language (XLink)	XLink describes the links between XML documents.
XML Pointer Language (XPointer)	XPointer describes references between XML documents so you can use them in links or in other documents.

Continued

Facebook's Exhibit No. 1005
Page 0064

GOOGLE EXHIBIT 1005

Table 2-2. *Continued*

Recommendation	Purpose
XML Query (XQuery)	XQuery queries XML documents to extract information. At the time of writing, it was a working draft rather than a recommendation of the W3C.
XForms	XForms are an XML-based replacement for XHTML forms.
Simple Object Access Protocol (SOAP)	SOAP is a standard protocol for requesting information from a web service.
Web Services Description Language (WSDL)	WSDL describes web services using an XML structure.

I'll look a little more closely at DTDs, XML schemas, and XSLT in Chapter 3 of this book.

What can you do with XML?

So far I've introduced you to XML and given you some background information on how to construct XML documents. You've also seen how XML is different from HTML and XHTML. Now it's time to explore how you can use XML documents.

Remember that the primary purpose for XML documents is the storage of data. XML allows people to share information using a self-describing document. The data is easy to read and interpret. Software packages can also read XML documents and use them as a medium for information exchange.

An XML document is portable and doesn't require the purchase of any specific software or technology. You can use the same XML documents for many different purposes. Best of all, XML documents are completely platform independent.

Common uses for XML documents include

- Storing and sharing information
- Querying and consuming web services
- Describing configuration settings
- Interacting with databases
- Interacting with Office 2003 documents

Storing and sharing information

The most important use for XML documents is in storing information. XML documents provide a way to describe structured data within a text file. The advantage of XML over other storage formats is that it is a standard so you can use the same content in many different ways. The same XML file could pro-

Facebook's Exhibit No. 1005
Page 0065

GOOGLE EXHIBIT 1005

vide content for a website, a Flash movie, and a printed document. You save time because you need to create the XML file only once to use it in these varied settings.

Each XML document that you create will probably have a different structure designed to meet the needs of the people or software who will use the information. The element names will describe the data they contain, and the element structures will show how blocks of information relate to each other.

An XML document doesn't need any specific software or operating system, which means you can share it with other people and software applications. XML documents can also provide information to other web-based applications, including websites and Flash applications.

If you are working in an industry group, you can design your own language for sharing information. By creating DTDs or schemas, you ensure that everyone understands how the language works and that their XML documents conform to a standard set of rules. CML and MathML are good examples of common languages.

The listing that follows shows a MathML document taken from the examples at www.mathmlcentral.com/Tools/FromMathML.jsp. The listing shows how **sin(x^2)** could be described using MathML elements.

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <mrow>
    <mi>sin</mi>
    <mo>&#8289;</mo>
    <mo>(</mo>
    <msup>
      <mi>x</mi>
      <mn>2</mn>
    </msup>
    <mo>)</mo>
  </mrow>
</math>
```

You can find out more about MathML at www.w3.org/Math/.

Another useful standard relates to graphics. Scalable Vector Graphics (SVG) is an XML-based language that describes two-dimensional graphics. If you want to find out more, the SVG recommendation is at www.w3c.org/Graphics/SVG/.

The following listing shows a sample SVG document. I've saved the document as shapes.svg (in your resource files) if you want to have a closer look. The elements describe a yellow rectangle, blue ellipse, and green triangle.

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
"http://www.w3.org/TR/SVG/DTD/svg10.dtd">
<svg>
  <desc>Shapes</desc>
  <rect x="5" y="5" width="100" height="50" fill="yellow"/>
  <ellipse cx="200" cy="100" rx="100" ry="40" fill="blue"/>
  <polygon points="110,140 40,300 120,250" fill="green"/>
</svg>
```


Facebook's Exhibit No. 1005
Page 0066

Page 66 of 140 GOOGLE EXHIBIT 1005

Figure 2-10 shows this file displayed in Internet Explorer.

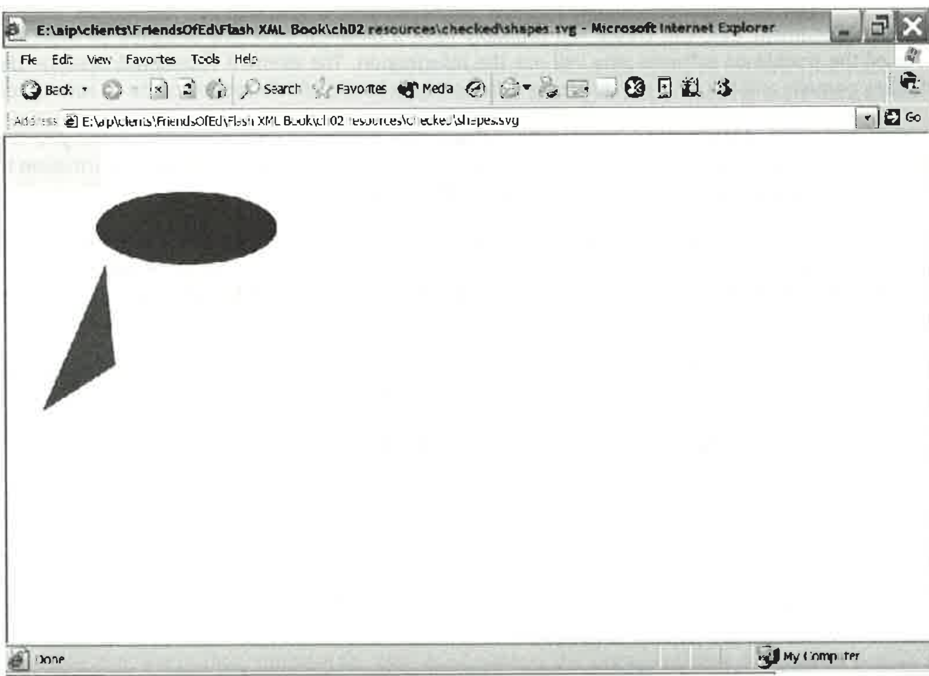


Figure 2-10. The sample SVG document displayed in Internet Explorer

Querying and consuming web services

XML documents are the standard way to share information through web services. Web services are public functions that organizations make available. For example, you can use web services to calculate currency exchange transactions, look up weather details, read news feeds, and perform searches at Amazon or Google.

When you send a request to a web service, you'll often use SOAP, an XML format. You'll also receive the information from the web service in an XML document. We'll look at web services in more detail later in this book.

Describing configuration settings

Many software packages use XML documents to describe their configuration settings. For example, an XML document format is used to configure .NET applications. The settings for a .NET application are stored in a file called `web.config`. The file uses standard XML elements to store settings such as debugging, authentication, error handling, and global variables. The following listing shows a sample `web.config` file. You can also see the saved `web.config` file within your resources.

Facebook's Exhibit No. 1005
Page 0067

Page 67 of 140 GOOGLE EXHIBIT 1005

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration>
  <appSettings>
    <add key="fileSaveLocation" value="D:\Hosting\website\images\"/>
  </appSettings>
  <system.web>
    <customErrors mode="Off"/>
    <compilation debug="true"/>
  </system.web>
</configuration>
```

The file contains a global variable or *key* location for saving files: `fileSaveLocation`. There are also some settings for customized errors and debugging.

Interacting with databases

Many common databases allow you to work with XML documents. SQL Server and Oracle both offer support for XML interaction. XML documents can query a database and return results. For example, in SQL Server, you can construct a SELECT statement that returns the results as an XML fragment:

```
SELECT * FROM BOOKS FOR XML AUTO
```

XUpdate is an XML-based language that describes updates to an XML document tree. It is not a W3C recommendation but uses the XPath specification. XUpdate is one way to manage XML document and database updates. Flash uses XUpdate in the XUpdateResolver data component.

You can find out more about XUpdate at `http://xmldb-org.sourceforge.net/xupdate/xupdate-wd.html`. There are some useful examples of XUpdate statements at `www.xmldatabases.org/projects/XUpdate-UseCases/`.

Interacting with Office 2003 documents

One exciting new application for XML is its role in Microsoft Office 2003 documents. For PC users, this means that you can save Word, Excel, and Access 2003 documents in XML format. Office 2003 can generate XML documents that you can use in other software packages, such as Flash. You can also display and update XML documents in Office 2003.

The Save As command converts Word and Excel 2003 documents into XML format using either **WordprocessingML** or **SpreadsheetML**. These are XML-based languages created by Microsoft to describe Word and Excel structures and formatting. You can also apply your own schema so that you can modify the XML documents produced by Office 2003.

Unfortunately, this functionality is only available for PC users. There is limited XML functionality in Excel 2004 for Macintosh users. Office XP for PCs also offers some XML support, but it is limited compared with Office 2003.

Facebook's Exhibit No. 1005
Page 0068

GOOGLE EXHIBIT 1005

Why is XML important to web developers?

XML is an important tool for all web developers, even those who don't use Flash. XML provides the basis for much of the content on the Internet, and its importance will only increase over time. Many people consider XML the *lingua franca* of the Internet as it provides the standards for data exchange between humans and machines in many different settings.

Web developers use XML to create their own languages to store, structure, and name data. XML content is the perfect mechanism for self-describing data. This makes XML documents ideal for sharing with other developers and IT specialists.

As a developer, you can use the same XML content for many different purposes. For example, you could use a single XML document to power a .NET application as well as a Java version. You could also transform the content into an XHTML document, a Flash movie, or a PDF file.

XML-related technologies also let you sort and filter the data within an XML document. Style sheet transformations allow you to reshape your data any way you want. You can then show the transformed content in a web browser, read it aloud, print it out, or send it to a mobile phone.

A physical XML document provides portability over and above that of a database. Creating an XML document enables you to distribute database content offline. For example, you can use the XML file with a stand-alone Flash movie and distribute it on a CD-ROM. In addition, providing an XML layer between a user and a database is a good way to prevent access to sensitive corporate data.

The built-in support for XML within Flash allows Flash developers to use XML data within any Flash movie. Using the content from XML documents enables you to update your movies without ever having to open Flash. You can store your Flash application settings in an XML file so that you can configure the application with simple changes to the document. If you're not comfortable with ActionScript, you can use the data components to add XML content to your movies. You can work with the panels in Flash so that you don't have to write any ActionScript.

You can save maintenance time by allowing your clients to manage their own content. It's often not practical for them to learn how to use Flash, and in reality, you probably don't want to give your clients access to the Flash movies that you've created. Instead, you provide mechanisms for clients to update an XML file and the Flash movie will update accordingly.

I have clients who update the content of their Flash movies using Office 2003. They make the changes within Word, Access, or Excel 2003; export the content in XML format; and replace the existing XML file with the new file they've just created. They can use a web page to upload the new XML file to their website, or they can burn it to a CD-ROM with a stand-alone Flash file.

These clients have the flexibility to change their Flash movie content whenever they like, and I've found that most clients are very comfortable working this way. It also saves me from continually editing their Flash movies each time the content changes.

Facebook's Exhibit No. 1005
Page 0069

GOOGLE EXHIBIT 1005

Summary

In this chapter, you've learned about XML and the contents of XML documents. You've also learned about the differences between XML, HTML, and XHTML. As a developer or designer, I hope I've shown you the advantages of working with XML in your applications.

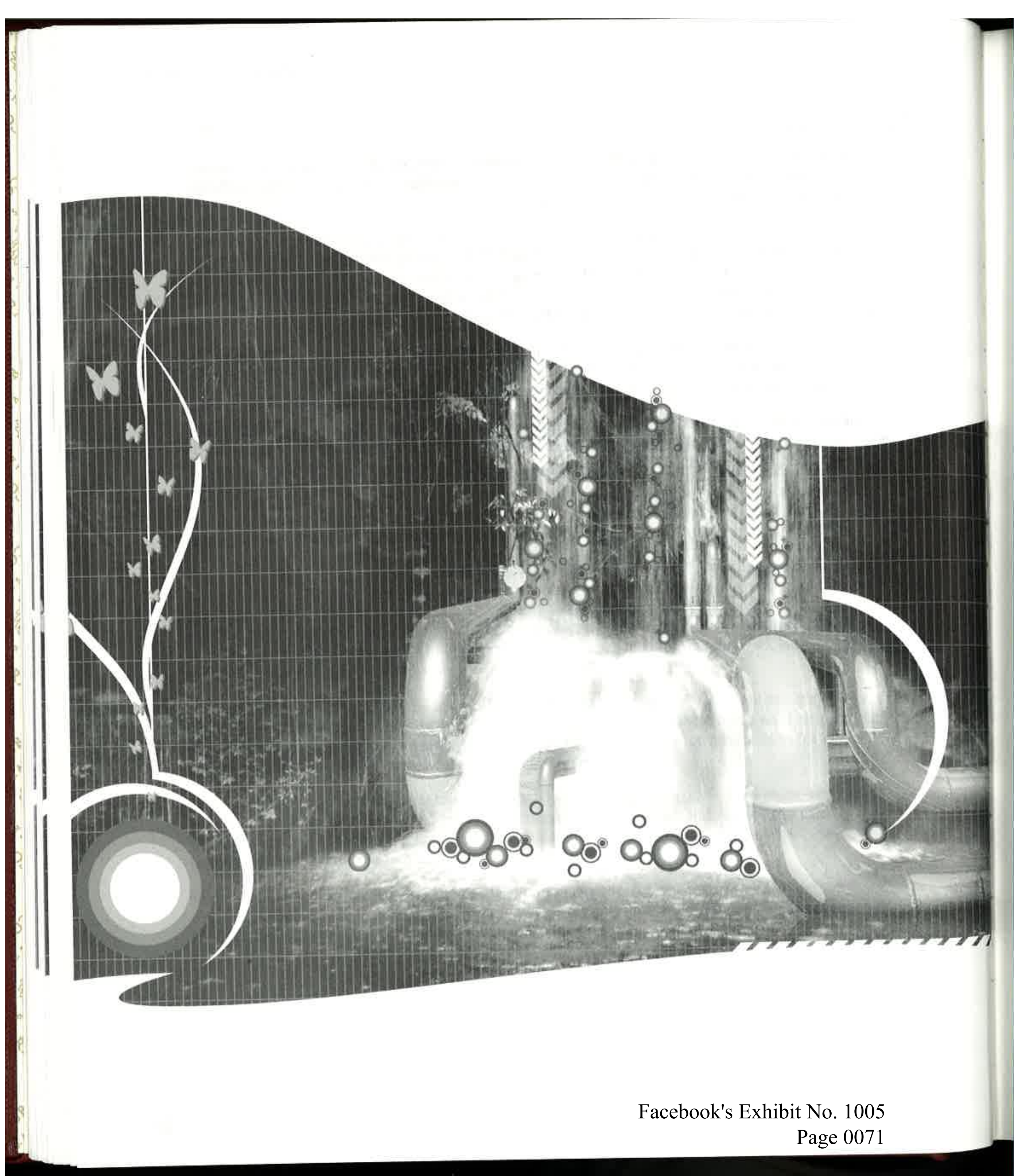
The importance of XML cannot be overstated. As a technology, it allows organizations to create their own mechanisms for sharing information. At its simplest, XML provides a structured, text-based alternative to a database. More complex uses of XML might involve data interactions between corporate systems and outside consumers of information. The most important thing to remember is that an XML document can provide a data source for many different applications.

The widespread adoption of XML by major software companies such as Microsoft and Macromedia ensure its future. Most of the popular database packages provide XML support. If it's not already there, expect XML to become part of most software packages in the near future.

The next chapter looks at working with XML content in XML editors and in Office 2003. It also looks more closely at consuming web services. I'll cover creating DTDs and XML schemas as well as transforming XML documents with XSLT. We'll finish by creating an XML document and schema from scratch.

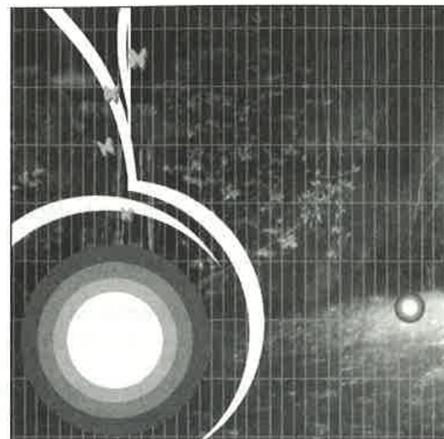
Facebook's Exhibit No. 1005
Page 0070

Page 70 of 140 GOOGLE EXHIBIT 1005



Facebook's Exhibit No. 1005
Page 0071

 GOOGLE EXHIBIT 1005



Chapter 3

XML DOCUMENTS

Before you can start working with XML content in Flash, you have to create the XML documents that you'll be using. This chapter looks at the different ways you can do this. I'll show you how you can generate content in a text or XML editor, from Office 2003 and by consuming web services. I'll have a quick look at querying Amazon and Google and receiving XML responses. You'll build applications that work with web services later in the book.

I'll also look at how you can transform XML documents using CSS and XSL style sheets. I'll cover creating Document Type Definitions (DTDs) and schemas that describe the rules for your XML documents. At the end of the chapter, we'll create an XML document and schema that we'll use in an application in the next chapter.

Remember that an XML document doesn't have to be a physical file. There's nothing to stop you from creating a text file with an `.xml` extension to store your XML content, and we'll look at different ways to do this. However, the term *XML document* can also refer to XML information that comes from a software package or web application.

Creating XML content

You can create XML content in many different ways, including

- Typing XML content in a text or XML editor
- Generating XML content with a server-side file
- Extracting XML content from software such as Office 2003
- Consuming XML generated by a web service or news feed

Facebook's Exhibit No. 1005
Page 0072

Page 72 of 140
GOOGLE EXHIBIT 1005

Each of the XML documents that you create will have different content and structure. The only thing they'll have in common is the rules that you use to create them. At the very minimum, all XML documents must be well formed. Later on, we'll look at creating valid documents with a DTD or schema.

Using a text editor

You can use a text editor like Notepad or SimpleText to type your XML content. You'll need to enter every line using your keyboard, which could take a long time if you're working with a large document. When you've finished, save the file with an `.xml` extension and you'll have created an XML document.

You can also use a text editor to create a DTD, schema, or XSL style sheet. Just remember to use the correct file extension—`.dtd` for DTDs, `.xsd` for schemas, and `.xsl` for XSL style sheets.

Don't forget that if you're using Notepad, you'll probably need to change Save as type to All Files before you save the document. Otherwise, you could end up with a file called `address.xml.txt` by mistake. Figure 3-1 shows the correct way to do this.

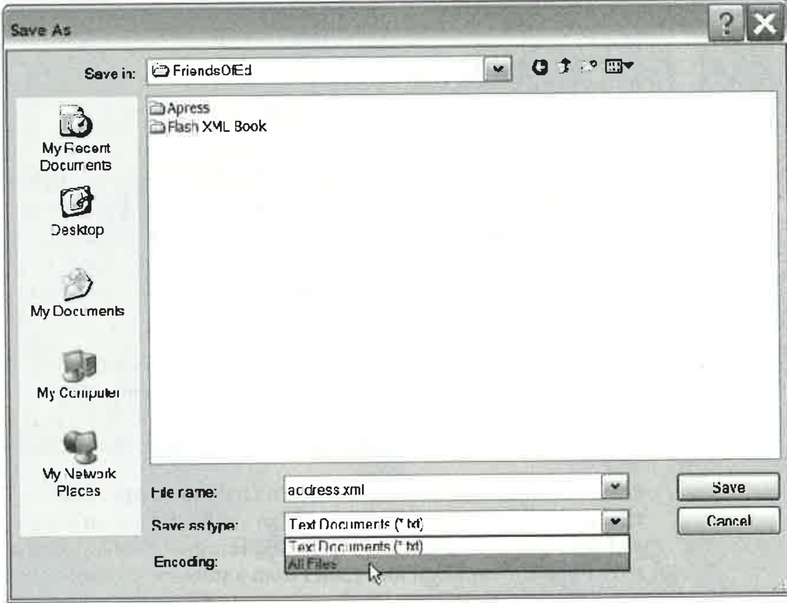


Figure 3-1. Saving an XML document in Notepad

Text editors are easy to use, but they don't offer any special functionality for XML content. Text editors won't tell you if your tag names don't match, if you've mixed up the cases of your element names, or if you've nested them incorrectly. There are no tools to check if your XML document meets the rules set down in a DTD or schema. Text editors don't automatically add color to your markup. In fact, you may not find any errors in your XML documents until you first try to use an XML parser.

Facebook's Exhibit No. 1005
Page 0073

Page 73 of 140 GOOGLE EXHIBIT 1005

You can also use HTML editors like HomeSite and BBEdit to create XML documents. The advantage of these over text editors is that they can automate the process a little. HTML editors often come with extensions for working specifically with XML documents. For example, they can add the correct declarations to the file and auto-complete your tag names. They'll also add coloring to make it easier to read your content.

However, you'll still have to type in most of your content line by line. Again, most HTML editors don't include tools to validate content and to apply transformations. You can only expect that functionality from an XML editor.

XML editors

An XML editor is a software program designed to work specifically with XML documents. Most XML editors include tools that auto-complete tags, check for well-formedness, and validate XML documents. You can use XML editors to create XSL style sheets, DTDs, and schemas.

The category "XML editors" includes both free and for-purchase software packages. With such a range of great XML tools available, you'd have to wonder why people would want to create XML documents with a text or HTML editor.

Common XML editors include

- Altova XMLSpy
- SyncRO Soft <oXygen/>
- WebX Systems UltraXML
- XMLEditPro (freeware)
- RustemSoft XMLFox (freeware)

You can find a useful summary of XML editors and their features at `www.xmlsoftware.com/editors.html`.

Although it isn't mandatory to use an XML editor when creating XML documents, it's likely to save you time, especially if you work with long documents.

Altova XMLSpy 2005 is one of the most popular XML editors for PCs. You can download a free home user edition of the software from `www.altova.com/download_spy_home.html`. You can also purchase a version with additional professional level features.

As we'll be using XMLSpy in this section of the book, it's probably a good idea to download it and install it on your computer. If you're working on a Macintosh, you'll need to get access to a PC if you want to try out the examples.

You can work with any type of XML content in XMLSpy, including XHTML documents. It includes a text editor interface as well as graphical features. XMLSpy offers features such as checking for well-formedness and validity. It also helps out with tag templates if you've specified a DTD or schema.

You can use XMLSpy to create DTDs and schemas as well as XSL style sheets. It also allows you to apply style sheets to preview transformations of your XML documents.

We'll look at some of the features of this software package in a little more detail as an illustration of what's possible with XML editing software.

To start with, when you create a new document, XMLSpy allows you to choose from many different types. Figure 3-2 shows you some of the choices.

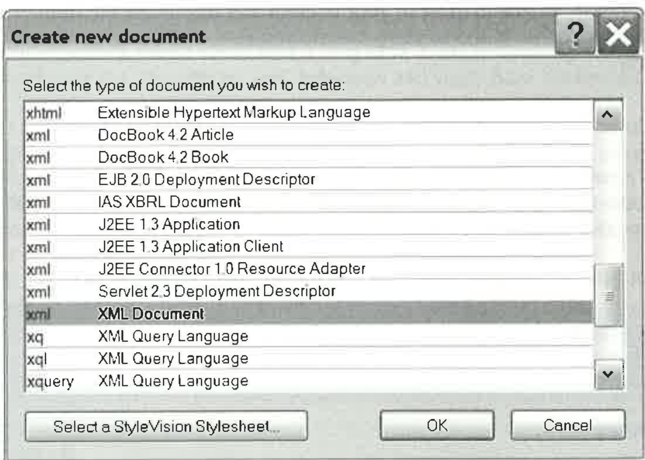


Figure 3-2. Options available when creating a new document with XMLSpy

Depending on the type of document you choose, XMLSpy automatically adds the appropriate content. For example, choosing the type XML Document automatically adds the following line to the new file:

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

When you create a new XML document, XMLSpy will ask you if you want to use an existing DTD or schema. Figure 3-3 shows the prompt.

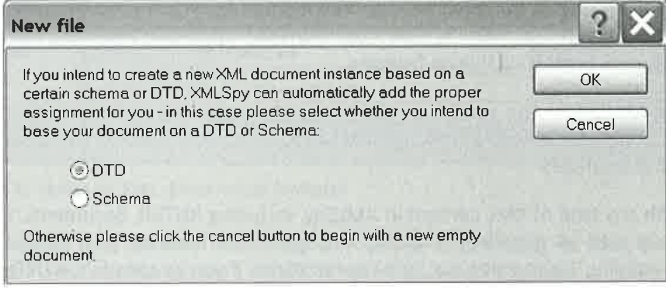


Figure 3-3. When you create a new XML document, XMLSpy prompts for a DTD or schema reference.

Facebook's Exhibit No. 1005
Page 0075

Page 75 of 140 GOOGLE EXHIBIT 1005

If you choose either a DTD or schema and select a file, XMLSpy will create a reference to it in your XML document:

```
<phoneBook xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="addressSchema.xsd">
```

If you don't include a DTD or schema reference, you can always add one later by using the DTD/Schema menu.

You can use XMLSpy in Text view, like a text editor, or in Authentic view, which has WYSIWYG features. Schemas can also use the Schema/WSDL view, a graphical presentation that simplifies the creation process. The final option is Browser view, which simulates how a document would display in a web browser.

XML documents with a referenced DTD or schema will show you extra information when you work in Text view. Clicking on an element or attribute in the main window will display information about it in the Info panel on the left side. You can see this in Figure 3-4.

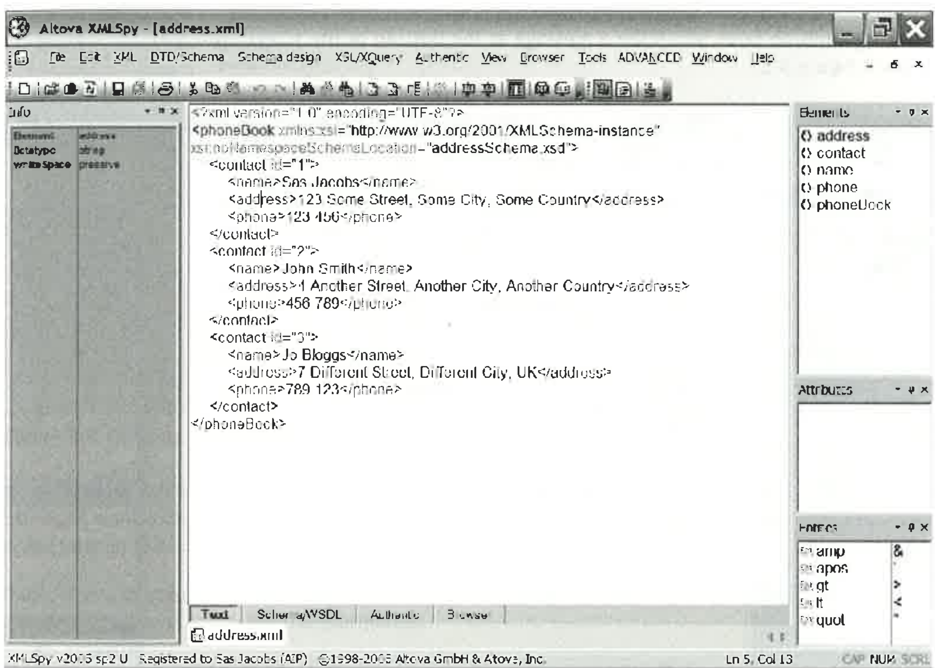


Figure 3-4. When a schema or DTD is referenced, XMLSpy displays information about the selected element or attribute. Entry Helpers are included on the right side of the screen.

The Entry Helpers panel on the right shows a list of the available elements. The panel also shows you common entities. One very useful feature is the ability to add an element template to the main window from the Elements panel.

Position your cursor in the XML document, double-click the appropriate tag name, and XMLSpy adds an element template to the code. This is very handy if the element you've chosen contains child elements as XMLSpy adds the complete tree from that point, including attributes.

Open the resource file `address.xml` in XMLSpy to test these features. Click to the left of the closing </phoneBook> tag and press *Enter*. Position your cursor in the blank line and double-click the `<contact>` element in the Elements panel. XMLSpy will insert a `<contact>` element, complete with child elements, into the document.

Another feature of XMLSpy is checking whether an XML document is well formed. If you are using a text editor, you'd have to do this by loading the document into an XML parser and checking for errors. Not only is this time consuming, but the error messages are often not as detailed as you'd like them to be!

In XMLSpy, you can check the document by clicking the button with the yellow tick or by using the *F7* key. XMLSpy then checks all the requirements for well-formed documents, including a single root node, tag case, element ordering, and quotes on attributes. I covered the requirements for well-formed documents in Chapter 2.

If XMLSpy finds an error, you'll see a message at the bottom of the screen with a Recheck button, as shown in Figure 3-5.

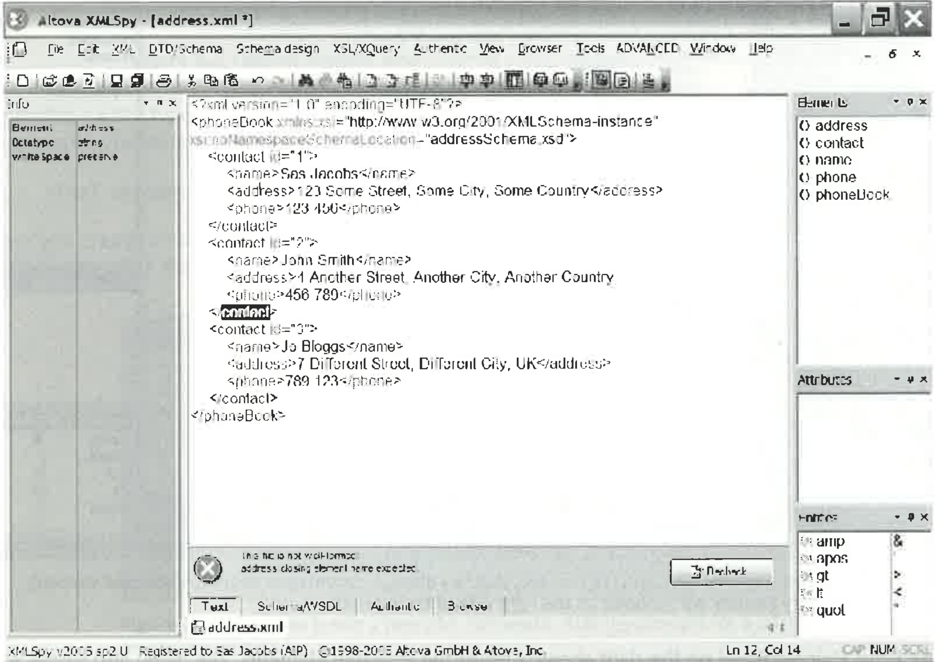


Figure 3-5. Using XMLSpy to check whether a document is well formed

Facebook's Exhibit No. 1005
Page 0077

GOOGLE EXHIBIT 1005

If you want to see it in action, change the `address.xml` file to introduce a deliberate mistake and check it again for well-formedness. You could change the case of one of the closing tags or remove the apostrophes from an attribute. You'll see a detailed error message that will help you to pinpoint where you went wrong.

XMLSpy can also check if an XML document is valid against a DTD or schema. Click the button with the green tick or use the *F8* key. Figure 3-6 shows an invalid document after it's been checked in XMLSpy.

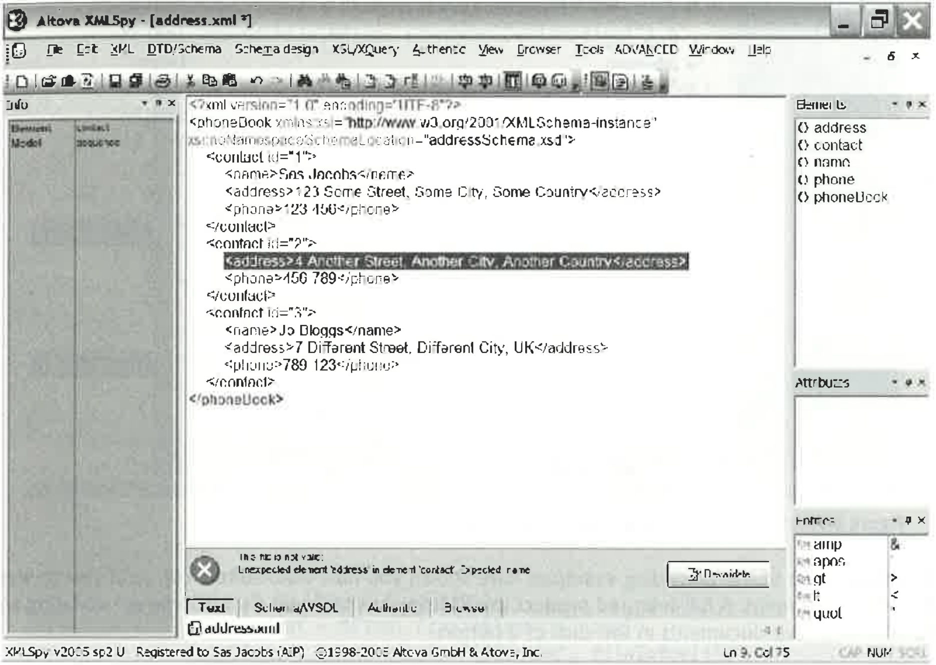


Figure 3-6. Checking validity in XMLSpy

You can test this feature by checking if `address.xml` is valid against its schema `addressSchema.xsd`. You might want to open up the schema file to have a look at the content. It will make a lot more sense to you later in the book!

Finally, if you're going to transform your XML document with XSLT, you can use XMLSpy to create the style sheet and to preview the transformation.

Once you've added a style sheet reference to your XML document, use the *F10* key to apply the transformation. XMLSpy will create an `XSLOutput.html` file and display your transformed content.

You can add a style sheet reference by choosing XSL/XQuery ➤ Assign XSL and selecting the file `listStyle.xsl`. Make sure you check the Make path relative to address.xml check box before clicking OK. XMLSpy adds the style sheet reference to the XML document.

```
<?xml-stylesheet type="text/xsl" href="listStyle.xsl"?>
```

Press the *F10* key to see the transformation. Figure 3-7 shows the `XSLOutput.htm` file created by XMLSpy.

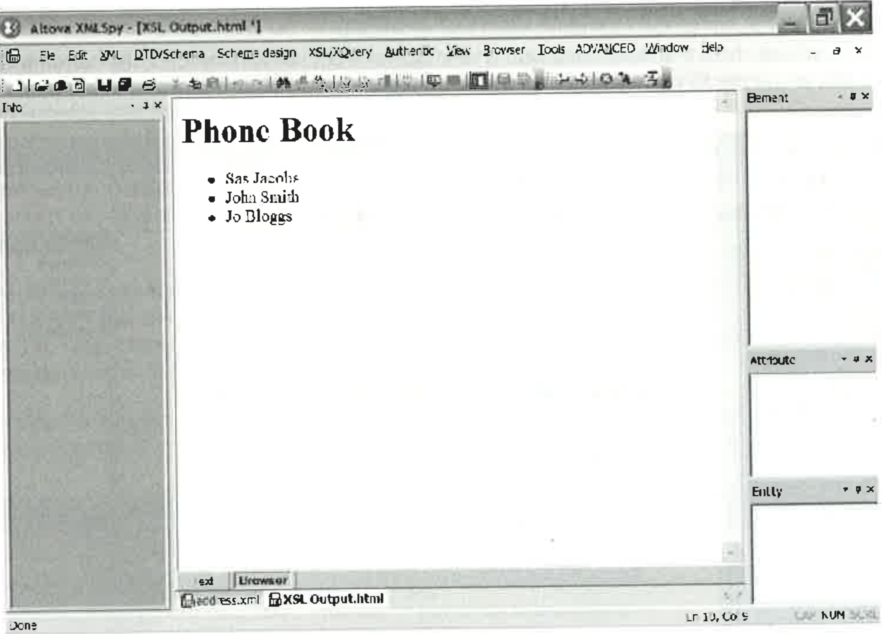


Figure 3-7. A transformed XML document in XMLSpy

Hopefully, some of the preceding examples have shown you how XML editors can help you to work with XML documents. A full-featured product like XMLSpy can save you a lot of time by validating and transforming your documents in the click of a button.

Server-side files

You can use content from any server-side file that generates XML. That means you can use a ColdFusion, PHP, or .NET file to create the XML content for you dynamically. For example, you might query a database and receive the response as an XML document. You might also use a server-side file to query the files and folders within your computer. Server-side code can create an XML document that describes the folder structures and file names.

The following listing shows some VB .NET code that generates a list of folders and files in XML format. The resource file `MP3List.aspx` contains the complete listing.

```
<%@ Page Language="vb" Debug="true" %>
<%@ import Namespace="System" %>
<%@ import Namespace="System.IO" %>
<%@ import Namespace="System.XML" %>
<script runat="server">
```

Facebook's Exhibit No. 1005
Page 0079

 GOOGLE EXHIBIT 1005

```
Dim strDirectoryLocation as String = "e:\mp3z\"
Dim dirs As String(), fileInfos as String()
Dim i as Integer, j as Integer
  sub Page_Load
    Dim MP3Xml as XmlDocument = new XmlDocument()
    Dim folderElement as XMLElement
    Dim songElement as XMLElement
    Dim writer As New XmlTextWriter(Console.Out)
    writer.Formatting = Formatting.Indented
    MP3Xml.AppendChild(MP3Xml.CreateXmlDeclaration("1.0", "UTF-8", ➥
    "no"))
    Dim RootNode As XmlElement = MP3Xml.CreateElement("mp3s")
    MP3Xml.AppendChild(RootNode)
    if Directory.Exists(strDirectoryLocation) then
      dirs = Directory.GetDirectories(strDirectoryLocation)
      for i = 0 to Ubound(dirs)
        dirs(i) = replace(dirs(i), strDirectoryLocation, "")
      next
      Array.sort(dirs)
      for i=0 to Ubound(dirs)
        folderElement = MP3Xml.CreateElement("folder")
        folderElement.SetAttribute("name", dirs(i))
        RootNode.AppendChild(folderElement)
        fileInfos = Directory.GetFiles(strDirectoryLocation & ➥
        dirs(i) & "\", "*.mp3")
        for j = 0 to Ubound(fileInfos)
          fileInfos(j) = replace(fileInfos(j), strDirectoryLocation ➥
          & dirs(i) & "\", "")
        next
        Array.sort(fileInfos)
        for j = 0 to Ubound(fileInfos)
          songElement = MP3xml.CreateElement("song")
          songElement.SetAttribute("filename", fileInfos(j))
          folderElement.AppendChild(songElement)
        next
      next
    End If
    dim strContents as String = MP3Xml.outerXML
    response.write (strContents)
  end sub
</script>
```

The server-side file returns a list of folders and MP3 files in an XML document. Figure 3-8 shows how the file looks when viewed in a web browser. Note that because the file contains server-side code, you'll have to run it through a web server like Microsoft Internet Information Services (IIS). If you check the address bar in the screenshot, you'll see that the file is running through `http://localhost/`.

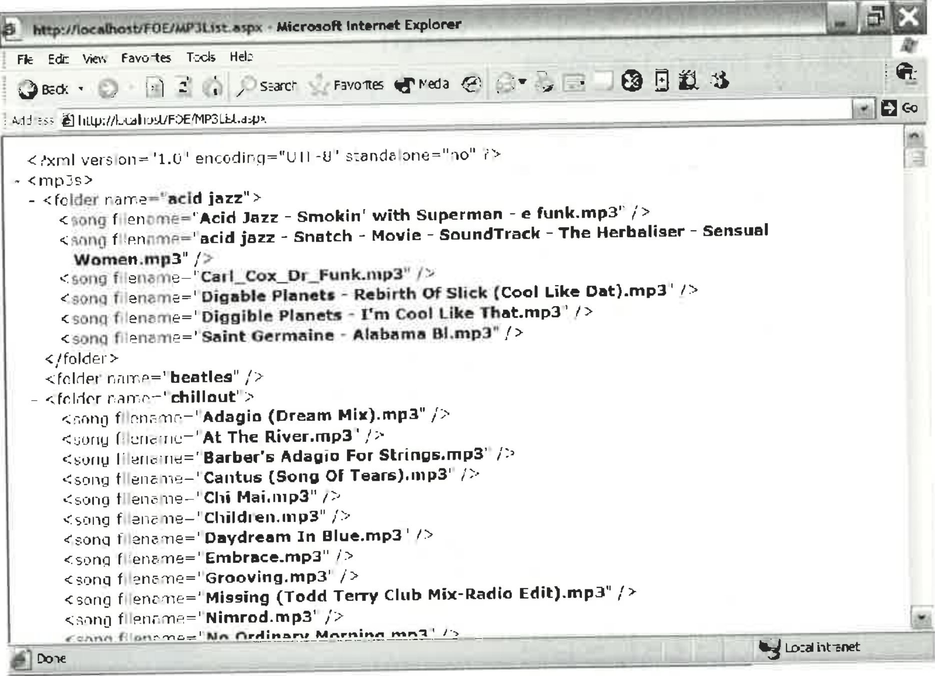


Figure 3-8. XML content generated by a server-side file, displayed in Internet Explorer

This is an example of an XML document that doesn't exist in a physical sense. I didn't save a file with an `.xml` extension. Instead, the server-side file creates a stream of XML data. The VB .NET file transforms the file system into an XML document.

Office 2003/2004

Believe it or not, Microsoft Office can be a source of XML content. For PCs, Microsoft Office 2003 has built-in XML support within Word, Excel, and Access. Unfortunately for Macintosh users, Office 2004 doesn't provide the same level of support. Macintosh users can use Excel 2004 to read and write XML documents, but they can't use schemas and style sheets.

Most people wouldn't think of Office documents as containers for structured XML information. Normally, when we work with Office documents we are more concerned with the appearance of data. Word, Excel, and Access 2003 all offer support for information exchange via XML. These applications can open, generate, and transform XML documents.

Word 2003 creates **WordprocessingML** (previously called WordML) while Excel writes **SpreadsheetML**. Both are markup languages that conform to the XML specification. You can find out more about these languages at `www.microsoft.com/office/xml/default.mspx`.

Whenever you use Save as and select XML format in Word or Excel, you're automatically generating one of those markup languages. Unfortunately, both languages are quite verbose as they include tags

Facebook's Exhibit No. 1005
Page 0081

GOOGLE EXHIBIT 1005

for everything—document properties and styling as well as the data itself. The resulting XML document can be quite heavy.

An alternative is to use a schema or XSL style sheet to format the output. You can extract the data to produce a much more concise XML document. Applying a schema to Word or Excel allows other people to update the content in Office without seeing a single XML tag.

Access also allows you to work with data in XML format, but it doesn't have its own built-in XML language. You just export straight from a table or query into an XML document that replicates the field structure.

In this section, I'll show you how to generate XML from Office 2003. The examples use sample files from the book's resources, so you can open them and follow along if you'd like. They are illustrations of the functionality that is available in Office 2003 rather than step-by-step tutorials. We'll do some more hands-on work with Office 2003 XML in Chapters 5, 6, and 7.

Word 2003

The stand-alone and professional versions of Word 2003 provide tools that you can use to work with XML documents. The trial edition of Word doesn't give you the same functionality. Let's look at the different ways that you can create and edit XML information in Word.

Creating an XML document using Save As

The simplest way to generate an XML document from Word 2003 is to use the File ➤ Save As command and choose XML Document as the type. Figure 3-9 shows how to do this.

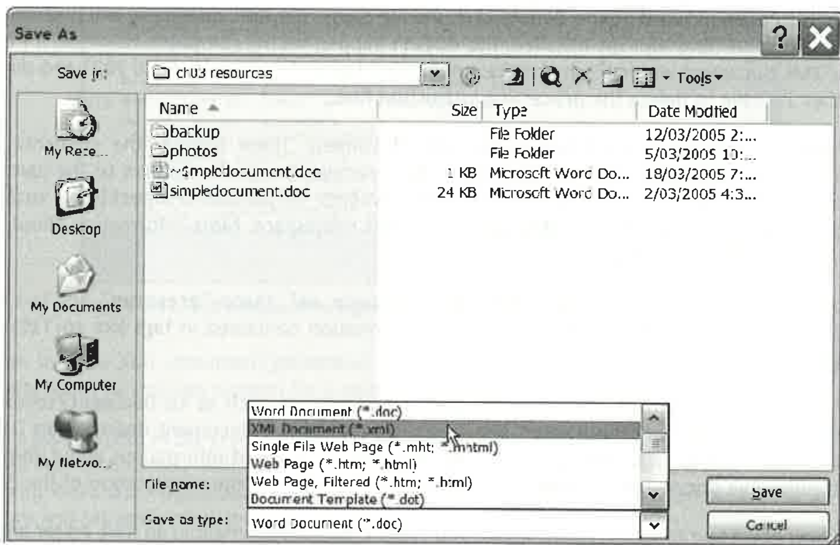


Figure 3-9. Using Save as type to generate an XML document

You can see a before and after example in your resource files. I've saved the Word document simpledocument.doc as simpledocument.xml. The source Word file contains three lines, each a different heading type. You can open simpledocument.xml in Notepad or an XML editor, to see the WordprocessingML generated by Word.

The following listing shows the first few lines of simpledocument.xml:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<?mso-application progid="Word.Document"?>
<w:wordDocument xmlns:w="http://schemas.microsoft.com/office/word/
2003/wordml" xmlns:v="urn:schemas-microsoft-com:vml"
xmlns:w10="urn:schemas-microsoft-com:office:word"
xmlns:sl="http://schemas.microsoft.com/schemaLibrary/2003/core"
xmlns:aml="http://schemas.microsoft.com/aml/2001/core"
xmlns:wx="http://schemas.microsoft.com/office/word/2003/auxHint"
xmlns:o="urn:schemas-microsoft-com:office:office"
xmlns:dt="uuid:C2F41010-65B3-11d1-A29F-00AA00C14882"
w:macrosPresent="no" w:embeddedObjPresent="no" w:ocxPresent="no"
xml:space="preserve">
<o:DocumentProperties><o:Title>Heading 1</o:Title>
<o:Author>Sas Jacobs</o:Author>
```

The listing I've shown doesn't display all of the content of the Word document; it only lists the introductory declarations. Feel free to repeat the test yourself to see the enormous amount of XML generated by Word.

You'll notice that there is a processing instruction on the second line of the XML document that instructs it to open in Word. If you double-click the file name, the XML document will probably open in Word 2003. As I have XMLSpy installed, this doesn't happen on my computer. However, if I tried to use this XML document within Flash, the document would probably open in Word 2003 and skip Flash altogether. I'd have to delete the processing instruction first.

A number of namespaces are listed in the XML document. These identify the elements in the document. Each namespace has a unique prefix. For example, the prefix o refers to the namespace urn:schemas-microsoft-com:office:office. The elements <o:DocumentProperties>, <o:Title>, and <o:Author> use the prefix o so they come from this namespace. More information about namespaces is available in Chapter 2.

The document also includes a declaration to preserve space: xml:space="preserve". The last lines in the listing are elements, and you'll recognize the information contained in tags like <o:Title> and <o:Author>.

Scroll through the document and you'll see that it has sections such as <o:DocumentProperties>, <w:fonts>, <w:styles>, and <w:docPr>. The actual content of the document doesn't start until the <w:body> tag. WordprocessingML is descriptive, but contains a lot of information about the styling applied within the document. It is concerned with both the data and the presentation of the data.

If you knew how to write WordprocessingML, you could create a document in an XML editor and open it in Word. You could also edit the WordprocessingML from the Word document in your XML editor as an alternative way to make changes to the document.

Facebook's Exhibit No. 1005
Page 0083

 GOOGLE EXHIBIT 1005

Working with the XML Toolbox

You can download a tool to work with XML directly in Word 2003. It is a plug-in called XML Toolbox, which you can download from the Microsoft website at `www.microsoft.com/downloads/details.aspx?familyid=a56446b0-2c64-4723-b282-8859c8120db6&displaylang=en`. You'll need to have a full version of Word 2003 and the .NET Framework installed before you can use the Toolbox. Installing the plug-in is very simple. You need to accept the license agreement and click the Install button.

Once you've installed the Toolbox, you'll have an extra toolbar called the Word XML Toolbox. Figure 3-10 shows this toolbar. You can use XML Toolbox to view the XML elements within a document or to add your own content.

Figure 3-10. The XML Toolbox toolbar in Word 2003

Choose the View XML command from the XML Toolbox drop-down menu to see the WordprocessingML from within Word 2003. Figure 3-11 shows the XML source.

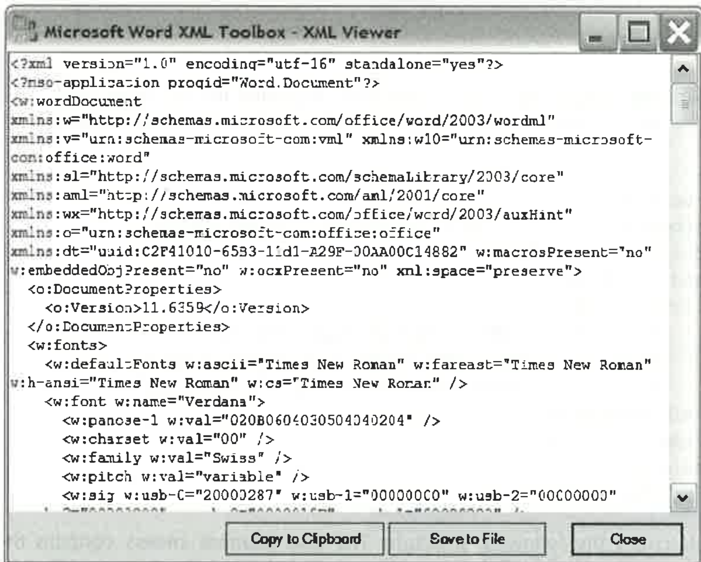


Figure 3-11. The WordprocessingML viewer in Word 2003

You can use the XML document generated by Word 2003 in other applications. For example, you could use Word to manage content for a web application or a Flash movie.

You're a little limited in the types of XML documents that Word 2003 can produce. Word doesn't handle data that repeats very well. You'd be better off to use Excel 2003 or Access 2003 instead. It's better to use Word 2003 documents as a template or form for XML data. You can create the document structure and set aside blank areas for the data.

Creating XML content by using a schema

If you have the stand-alone or professional versions of Word 2003, you'll be able to use schemas to ensure that an XML document created in Word is valid according to your language rules. A schema will also allow you to reduce the number of XML elements created from the document.

You need to follow these steps to create an XML document in Word 2003 using a schema:

1. Create a schema for the XML document.
2. Create a Word 2003 template that uses the schema.
3. Create a new document from the template and save the data in XML format.

The result is a valid XML document that is much smaller than its WordprocessingML relative.

Let's look at this more closely in an example. Chapter 5 provides you with the step-by-step instructions that you'll need to work through an example. The next section gives you an overview of the main steps and isn't intended as a tutorial.

Creating the schema

I used the following schema to describe the XML structure for my news item. The resource file newsSchema.xsd contains the complete schema. You'll learn how to create schemas a little later in this chapter.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="news">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="newsDate" type="xsd:string"/>
        <xsd:element name="newsTitle" type="xsd:string"/>
        <xsd:element name="newsContent" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

The schema describes the following structure. The root element <news> contains the <newsDate>, <newsTitle>, and <newsContent> elements. There can only be one of each of those elements, and they must be included in the order specified. The elements all contain string data.

Creating the Word 2003 template

I've created a simple template called newsXML.dot to show a news item. It is made up of three form fields to capture the date, title, and content of the news item. If you have Word 2003, you can open the file to see how it looks. Use the *Ctrl-Shift-X* shortcut key to toggle the display of the XML tags.

This template already has the schema applied, but I've included the instructions here in case you want to re-create it yourself. We'll cover this in more detail in Chapter 5. After you open the template, you'll need to unlock it if you want to make any changes. Choose Tools ➤ Unprotect Document.

Facebook's Exhibit No. 1005
Page 0085

GOOGLE EXHIBIT 1005

To apply the schema to a Word 2003 template, choose Tools ➤ Templates and Add-Ins and select the XML Schema tab. Click the Add Schema button and navigate to the schema file. Enter a URI or namespace for the schema and an alias, as shown in Figure 3-12.

Figure 3-12. Entering schema settings in Word 2003

When you've finished, the schema alias should appear in the Templates and Add-Ins dialog box, as shown in Figure 3-13.

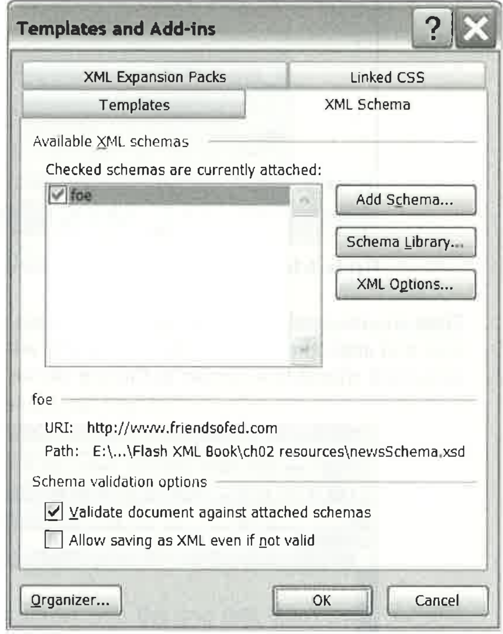


Figure 3-13. Attached schemas in Word 2003

To streamline the XML produced by this document, click the XML Options button and choose the Save Data Only option. This excludes formatting information from the output. Make sure that Validate document against attached schemas is also checked.

You can only apply the XML tags if you have selected the Show XML tags in the document option in the Task Pane. If you can't see the Task Pane, choose View ➤ Task Pane and choose XML Structure from the drop-down menu at the top.

First, you need to apply the root element to the entire document. Select all of the content, right-click, and choose Apply XML element. Select the news element. When prompted choose Apply to Entire Document. You should see the content surrounded by a shaded tag, as shown in Figure 3-14.

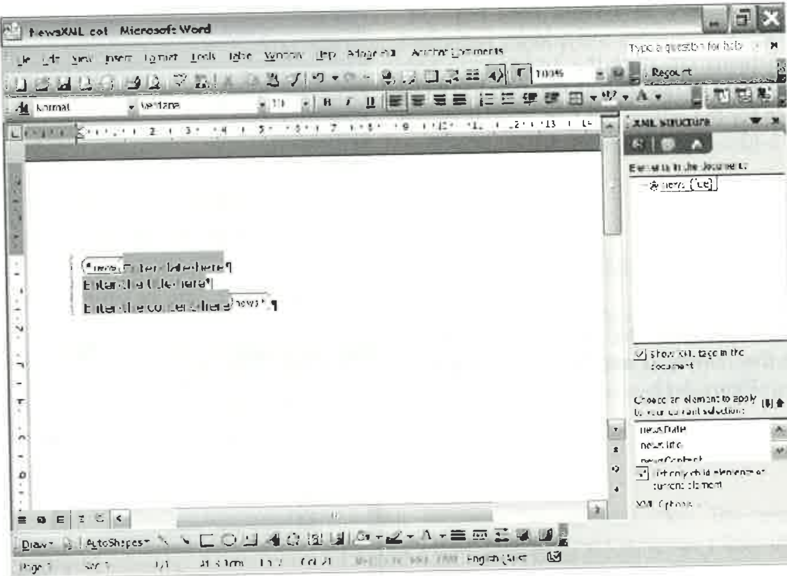


Figure 3-14 The content marked up in Word 2003

Then you can apply the other elements to each part of the Word document. Select the fields, one by one, and apply the tags by right-clicking and selecting Apply XML element. When you've finished, the document should look similar to the one shown in Figure 3-15.

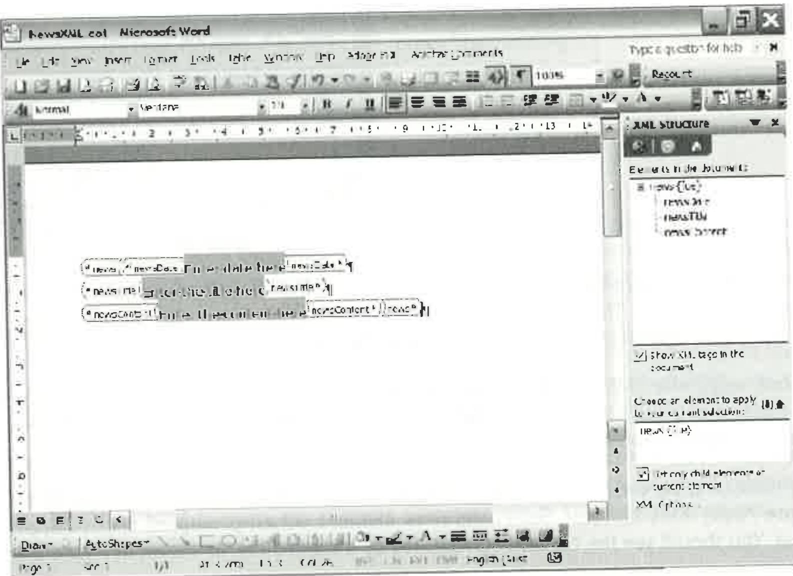


Figure 3-15. The completed template in Word 2003

Facebook's Exhibit No. 1005
Page 0087

GOOGLE EXHIBIT 1005

The result is a template that maps to an XML schema. Don't forget to lock the fields before you save and close the template. Choose View ➤ Toolbars ➤ Forms and click the padlock icon.

Creating XML content from a new document

Once you've created the template, you can generate XML content from documents based on this template. Choose File ➤ New and select the news template. When the new document is created, all you have to do is fill in the fields. You can hide the XML tags by deselecting the Show XML tags in the document option in the Task Pane.

Output the XML by choosing File ➤ Save and selecting the XML document type. Make sure you check the Save data only option before you save. You'll see the warning shown in Figure 3-16. Click Continue.

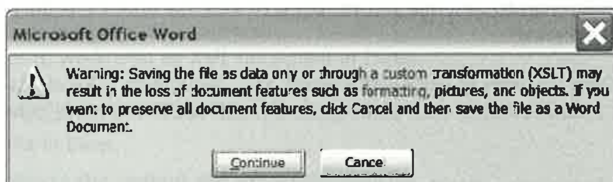


Figure 3-16. You'll see this warning when saving in XML format with a schema in Word 2003.

The resource file `NewsItem.xml` contains the competed XML document from Word 2003. The following listing shows the content:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<news xmlns="newsSchema">
  <newsDate>July 4, 2005</newsDate>
  <newsTitle>Fireworks extravaganza!</newsTitle>
  <newsContent>US expats in Australia celebrated the 4th of July
  with firework demonstrations throughout the country.</newsContent>
</news>
```

Compare the structure and content of this XML document with the one that didn't use a schema, `simpledocument.xml`. The tag names in this document are more descriptive, and it is significantly shorter than the WordprocessingML document. It would be very easy to use this XML document within a Flash movie. Using `simpledocument.xml` would be much harder.

We'll cover the step-by-step instructions for creating XML from Word 2003 in much more detail in Chapter 5.

Excel

If you own Excel Professional or Enterprise edition, you'll be able to work with XML documents. Again, you can't use the trial edition of Excel 2003. As with Word, you can save an Excel file in XML format so that you can use it on the Web or in Flash. You can also use Excel to open an XML document so that you can update or analyze the information.

Facebook's Exhibit No. 1005
Page 0088

GOOGLE EXHIBIT 1005

Excel document structures are very rigid. They always use a grid made up of rows and columns. This means that the structure of XML data generated from Excel will match this format. In Word, it's possible for you to include elements within other elements or text. For example, you could display this XML structure using Word:

```
<title>
  This is a title by
  <author>Sas Jacobs</author
</title>
```

In Excel, the smallest unit of data that we can work with is a cell. Cells can't contain other cells, so our XML document structure with mixed content can't display properly in Excel. Any XML document generated from Excel will include grid-like data.

Excel uses a document map to describe the structure of XML documents. A document map is like a simpler version of a schema.

In this section, I'll show you how to work with existing XML documents in Excel. It's an overview of the functionality that's available rather than a complete tutorial. You'll find more detailed information in Chapter 6.

Creating an XML document using Save As

As with Word, the easiest way to create an XML document from Excel is to save it using the XML document type. Choose File ➤ Save As and select XML in the Save as type drop-down box. I've done this with the file simplespreadsheet.xls; you can see the resulting XML document saved as simplespreadsheet.xml.

You'll notice that a simple Excel document has created a large XML document. This listing shows the first few lines of the XML document:

```
<?xml version="1.0"?>
<?mso-application progid="Excel.Sheet"?>
<Workbook xmlns="urn:schemas-microsoft-com:office:spreadsheet"
xmlns:o="urn:schemas-microsoft-com:office:office"
xmlns:x="urn:schemas-microsoft-com:office:excel"
xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet"
xmlns:html="http://www.w3.org/TR/REC-html40">
<DocumentProperties xmlns="urn:schemas-microsoft-com:office:office">
  <Author>Sas Jacobs</Author>
  <LastAuthor>Sas Jacobs</LastAuthor>
```

The second line of the file is a processing instruction that instructs the file to open in Excel:

```
<?mso-application progid="Excel.Sheet"?>
```

As with Word, a number of namespaces are referenced in the XML document. The element names <DocumentProperties> and <Author> are self-explanatory. The XML document includes information about each sheet in a <Worksheet> element. There are descriptions for <Table>, <Column>, <Row>, <Cell>, and <Data>, and Excel methodically describes the contents of each worksheet by column and by row. This is how Excel translates the grid style of Excel documents into XML. What we're most interested in is the contents of the <Data> elements; they contain the values from each cell.

Facebook's Exhibit No. 1005
Page 0089

GOOGLE EXHIBIT 1005

As with Word, you'll notice that Excel generates a long XML document. It's hard for humans to read, and extracting the content would be a lengthy process. Again, using a schema will reduce the quantity of data generated by Excel 2003.

You can use Excel to open an existing XML document. Before displaying the data, Excel will ask you how you want to open the file, as shown in Figure 3-17. The process will be a little different depending on whether the document references a schema.

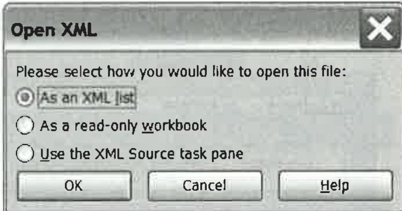


Figure 3-17. Excel 2003 asks how an XML file should be opened.

Opening an XML document with a schema

You use these steps to work with an XML document in Excel:

1. Optionally create a schema for the XML document.
2. Open the file in Excel.
3. Make changes to the content and export the XML file.

If you open the file as an XML list, Excel will use any related schema to determine how to display data. The following listing shows `address.xml`. It uses the schema `addressSchema.xsd`. Figure 3-18 shows how this file translates when opened in Excel.

```
<?xml version="1.0" encoding="UTF-8"?>
<phoneBook xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="addressSchema.xsd">
  <contact id="1">
    <name>Sas Jacobs</name>
    <address>123 Some Street, Some City, Some Country</address>
    <phone>123 456</phone>
  </contact>
  <contact id="2">
    <name>John Smith</name>
    <address>4 Another Street, Another City, Another Country</address>
    <phone>456 789</phone>
  </contact>
  <contact id="3">
    <name>Jo Bloggs</name>
    <address>7 Different Street, Different City, UK</address>
    <phone>789 123</phone>
  </contact>
</phoneBook>
```


Facebook's Exhibit No. 1005
Page 0090

Page 90 of 140 GOOGLE EXHIBIT 1005

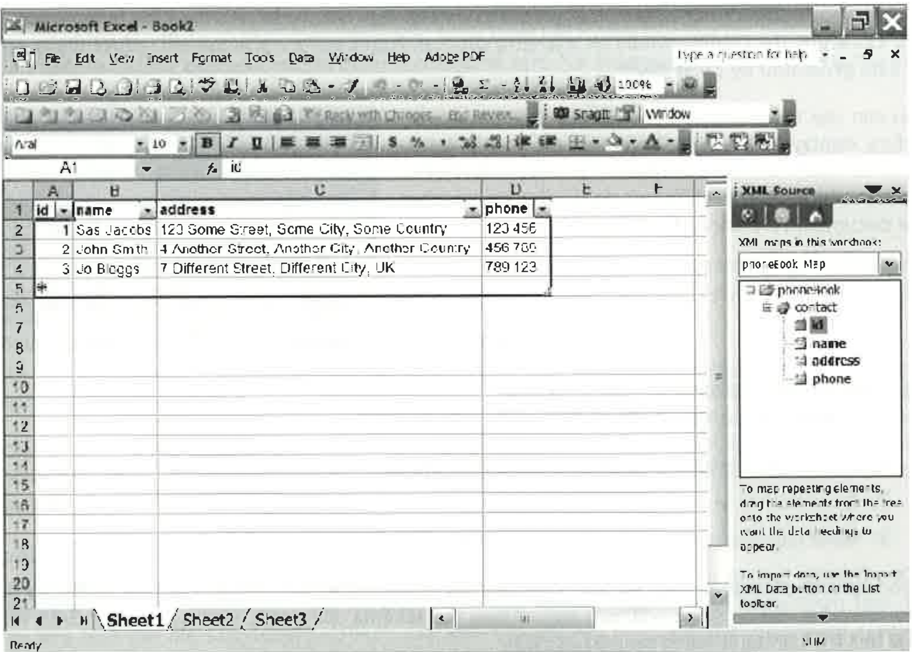
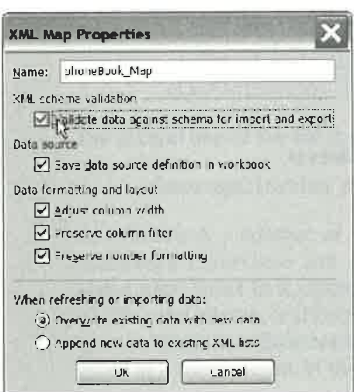


Figure 3-18. An XML document displayed in Excel 2003

Excel automatically creates a document map for the elements from the schema. You can see the document map in the XML Source Task Pane. Excel has also added an automatic filter to the column headings. You can select specific content from the XML document by choosing values from the drop-down lists.

You can make changes to the existing data in Excel or even add new data. Be careful how you generate the XML document. If you use Save As and choose the XML type, you'll re-create the current content using SpreadsheetML. It will produce a large document that doesn't match your schema. Instead, you should export the data as shown in the next section.



Exporting XML data with a document map

Before exporting the data, you'll want to make sure the changes you've made are valid against the schema. Right-click inside your data and choose XML ➤ XML Map Properties. Check the Validate data against schema for import and export option. This option isn't checked by default. You can also find XML Map Properties in the Data ➤ XML menu. Figure 3-19 shows the XML Map Properties dialog box.

Figure 3-19. The XML Map Properties dialog box in Excel 2003

Facebook's Exhibit No. 1005
Page 0091

GOOGLE EXHIBIT 1005

To export the XML document, right-click in the data and choose XML ➤ Export. Enter a file name, choose a location, and click Export. Excel will generate an XML document that is valid according to your schema.

I used Excel to update the `address.xml` file and exported the data to the resource file `addressExportedFromExcel.xml`. If you look at the XML structure, you'll see that it's almost identical to that of the `address.xml` file. Figure 3-20 shows them side by side in XMLSpy.

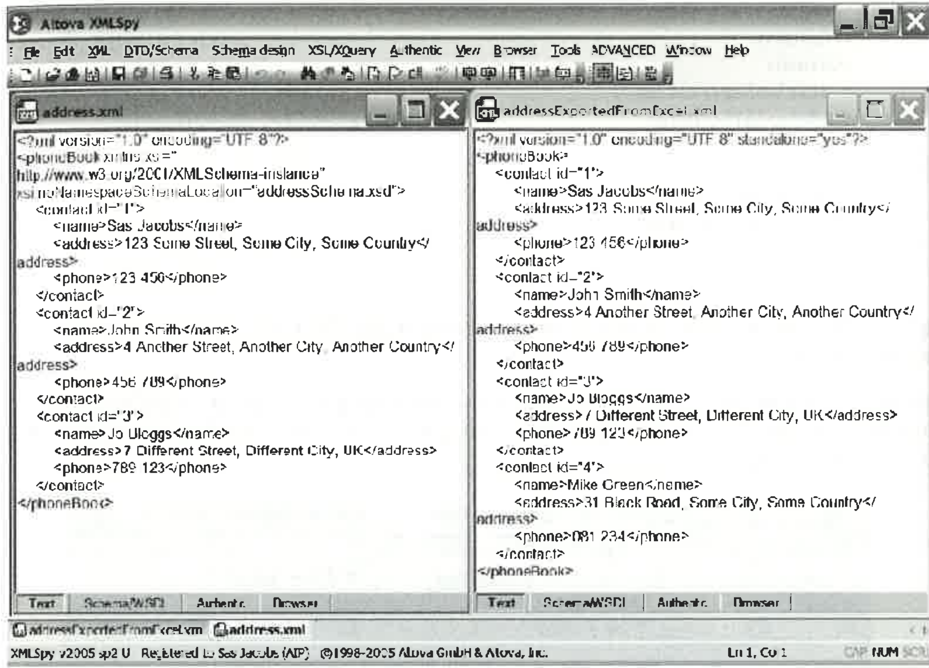


Figure 3-20. The original and updated XML documents open in XMLSpy

Opening an XML document without a schema

If you open an XML document that doesn't specify a schema, Excel will create one based on the data. Figure 3-21 shows the warning that Excel will display.

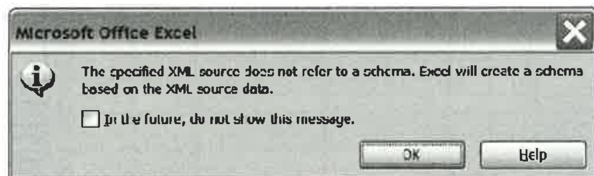


Figure 3-21. Excel 2003 will create a document map for an XML file if a schema doesn't exist.

When the data is imported, Excel creates a document map and figures out how to display the data. You can try this with the resource file `excelImport.xml`. This listing shows a simple XML document without a schema:

```
<?xml version="1.0">
<ImportData>
  <Column>
    <title>Jan</title>
    <data>1234</data>
  </Column>
  <Column>
    <title>Feb</title>
    <data>5678</data>
  </Column>
  <Column>
    <title>Mar</title>
    <data>9123</data>
</Column>
</ImportData>
```

Figure 3-22 shows the XML document after importing it into Excel. I've saved the imported file as resource file `excelImport.xls`.

The document map created by Excel displays in the XML Source Task Pane. If it's not visible, you can show it by choosing View ➤ Task Pane and selecting XML Source from the drop-down menu at the top of the Task Pane.

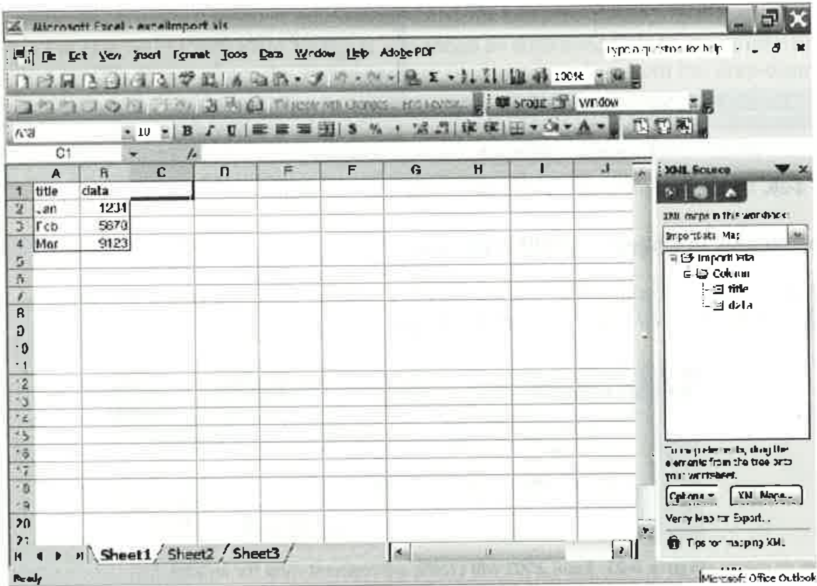


Figure 3-22. An XML file without a schema imported into Excel 2003

Facebook's Exhibit No. 1005
Page 0093

GOOGLE EXHIBIT 1005

Working with mixed elements

If you use Excel to open an existing XML document, make sure that it conforms to a grid structure. Excel will have difficulty interpreting the structure of an XML document that contains text and child elements together in the same parent element.

This listing shows the file addressMixedElements.xml. As you can see, this document includes mixed content in the <address> element. It contains both text and a child element, <suburb>.

```
<?xml version="1.0" encoding="UTF-8"?>
<phoneBook>
  <contact id="1">
    <name>Sas Jacobs</name>
    <address>123 Some Street,
      <suburb>Some City</suburb>
    , Some Country</address>
    <phone>123 456</phone>
  </contact>
  <contact id="2">
    <name>John Smith</name>
    <address>4 Another Street,
      <suburb>Another City</suburb>
    , Another Country</address>
    <phone>456 789</phone>
  </contact>
</phoneBook>
```

Figure 3-23 shows the file opened in Excel 2003. The <address> element and text is missing; only the child element <suburb> displays.

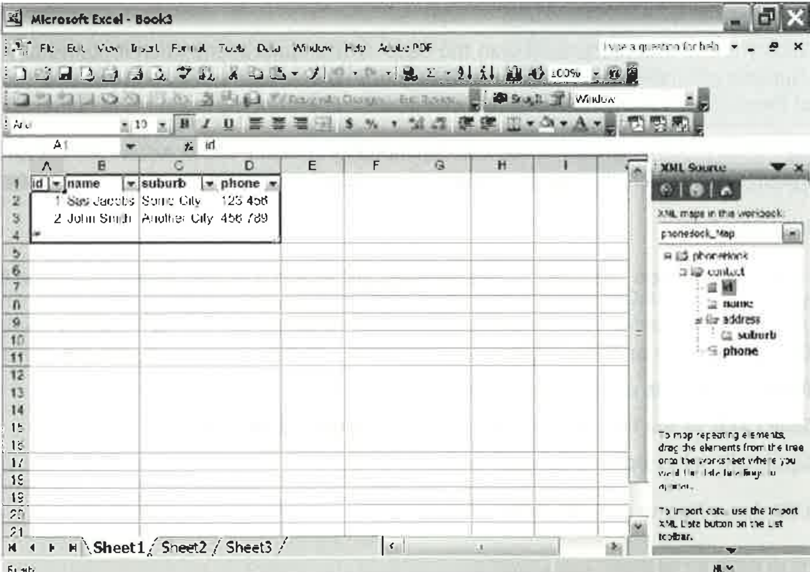


Figure 3-23. An XML document with mixed elements doesn't display correctly in Excel 2003.

If you export the data to an XML document, Excel will only save the elements displayed. The following listing shows the exported file addressMixedElementsExported.xml:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<phoneBook xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <contact id="1">
    <name>Sas Jacobs</name>
    <address>
      <suburb>Some City</suburb>
    </address>
    <phone>123 456</phone>
  </contact>
  <contact id="2">
    <name>John Smith</name>
    <address>
      <suburb>Another City</suburb>
    </address>
    <phone>456 789</phone>
  </contact>
</phoneBook>
```

The text within the <address> element is missing. Excel has also added a namespace to the root element.

Using Excel VBA and XML
You can use VBA to work with XML documents. For example, you could handle the importing of XML documents automatically. Excel 2003 recognizes the XMLMaps collection, and you can use the Import and Export methods to work with XML documents programmatically.

Access

Access 2003 works a little differently than the other Office applications when it comes to XML. The XML documents generated by Access come directly from the structure of your tables and queries. The names of the elements in the resulting XML document come from the Access field names.

This section gives you an overview of the XML functionality available within Access 2003. It isn't a complete reference or tutorial. I'll cover the topic in more detail in Chapter 7.

Exporting XML data

Getting data out of Access and into XML is easy—you just export it in XML format. You need to follow these steps:

1. Display the table or query objects.
2. Right-click a table or query and select Export.
3. Select XML as the file type and choose a destination and file name.
4. Optionally select options for export.

Figure 3-24 shows how to export a table.

Facebook's Exhibit No. 1005
Page 0095

 GOOGLE EXHIBIT 1005

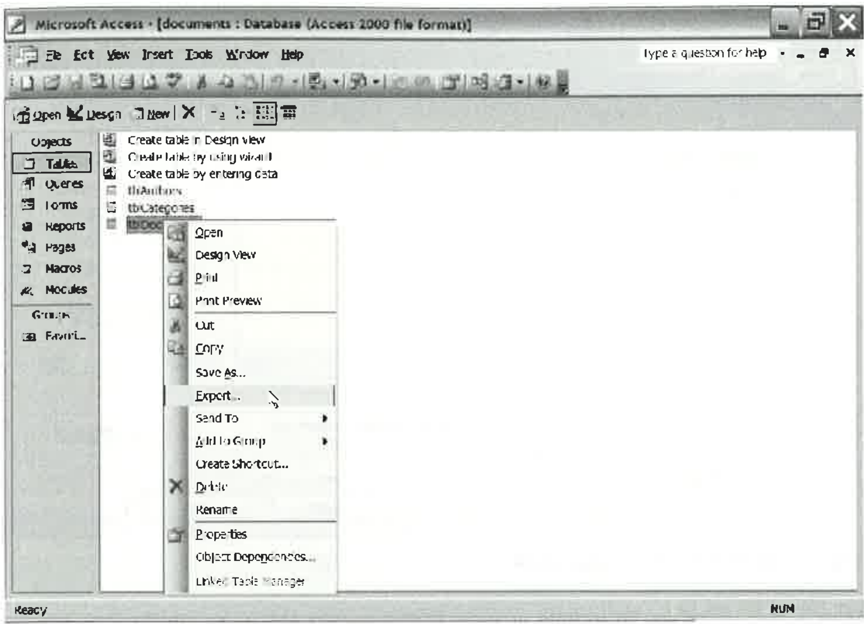


Figure 3-24. Exporting a table in Access 2003

After you chose the Export option, you'll have to enter a file name and choose a destination for the XML file. Don't forget to select XML from the Save as type drop-down list. When you click Export, you'll be asked to choose between exporting the data (XML), a schema (XSD), and presentation of the data (XSL). See Figure 3-25 for a view of the Export XML dialog box.

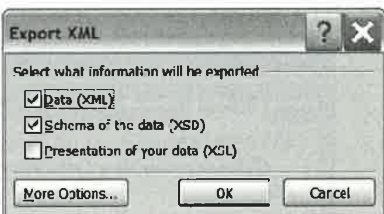


Figure 3-25. Export options in Access 2003

Setting export options

You have some extra options that you can view by clicking the More Options button. Figure 3-26 shows these options. You can also include related records from other tables and apply an XSL transformation to the data.

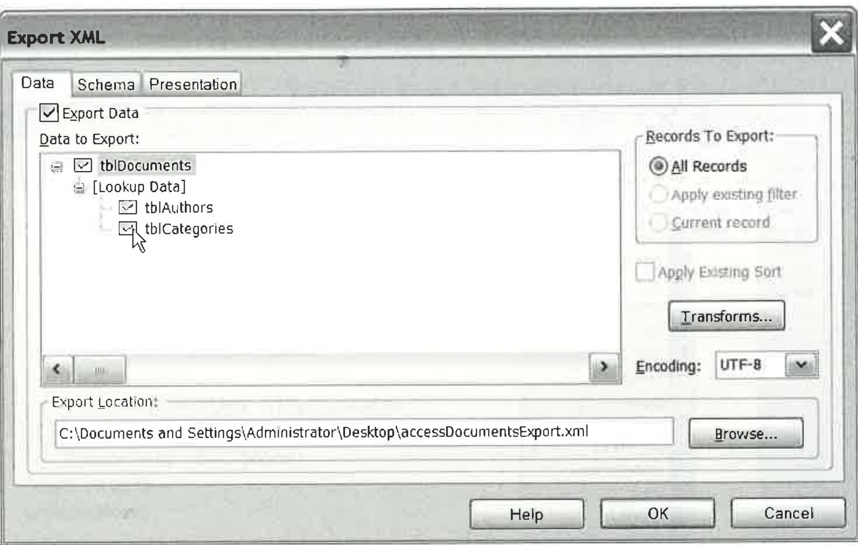


Figure 3-26. Export options in Access 2003

The Schema tab allows you to include or exclude primary key and index information. You can also embed the schema in your XML document or create an external schema. Figure 3-27 shows these options.

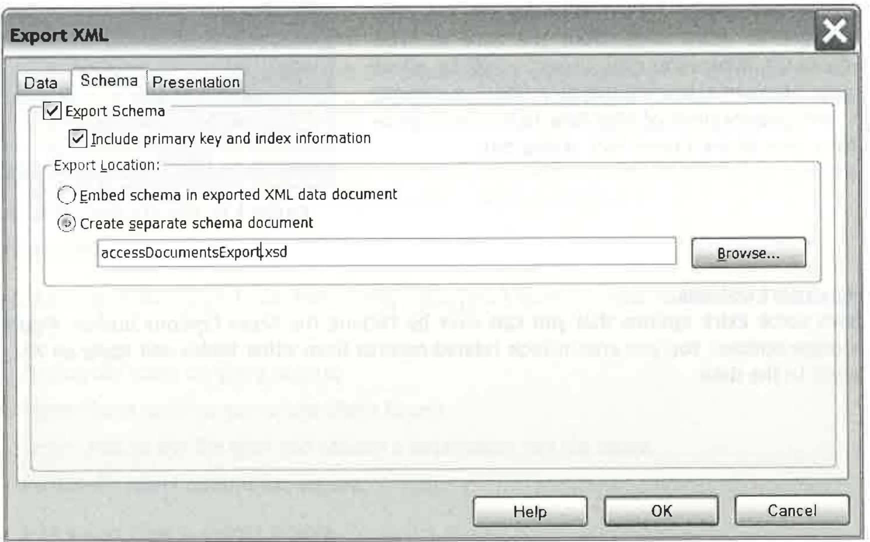


Figure 3-27. Schema export options in Access 2003

Facebook's Exhibit No. 1005
Page 0097

Page 97 of 140 GOOGLE EXHIBIT 1005

The Presentation tab, shown in Figure 3-28, allows you to generate HTML or ASP and an associated style sheet.

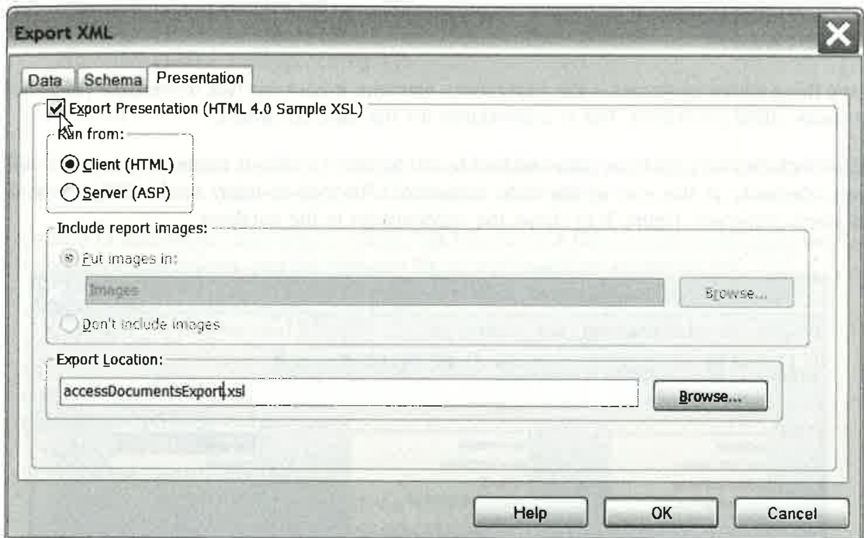


Figure 3-28. Presentation export options in Access 2003

I used the Access database documents.mdb and exported the records from tblDocuments. I included the related records from other tables and created both an XML and an XSD file. The resulting XML documents are called accessDocumentsExport.xml and accessDocumentsExport.xsd, respectively. If you have Access 2003, you can use the documents.mdb database to create the XML files yourself.

This listing shows a section of the sample XML document created by the export:

```
<?xml version="1.0" encoding="UTF-8"?>
<dataroot xmlns:od="urn:schemas-microsoft-com:officedata"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="accessDocumentsExport.xsd"
generated="2005-03-04T18:10:06">
  <tblDocuments>
    <documentID>1</documentID>
    <documentName>Shopping for profit and pleasure</documentName>
    <authorID>1</authorID>
    <documentPublishYear>2002</documentPublishYear>
    <categoryID>4</categoryID>
  </tblDocuments>
  <tblAuthors>
    <authorID>1</authorID>
    <AuthorFirstName>Alison</AuthorFirstName>
    <AuthorLastName>Ambrose</AuthorLastName>
    <AuthorOrganization>Organization A</AuthorOrganization>
```

```
    </tblAuthors>
    <categoryID>4</categoryID>
      <category>Shopping</category>
    </tblCategories>
  </dataroot>
```

The only thing added by Access is the <dataroot> element. It contains two namespace references and an attribute called generated. This is a timestamp for the XML document.

Because I included records from tables related to tblDocuments, Access added the table references as separate elements at the end of the XML document. The one-to-many relationships between the tables aren't preserved. Figure 3-29 shows the relationships in the database.

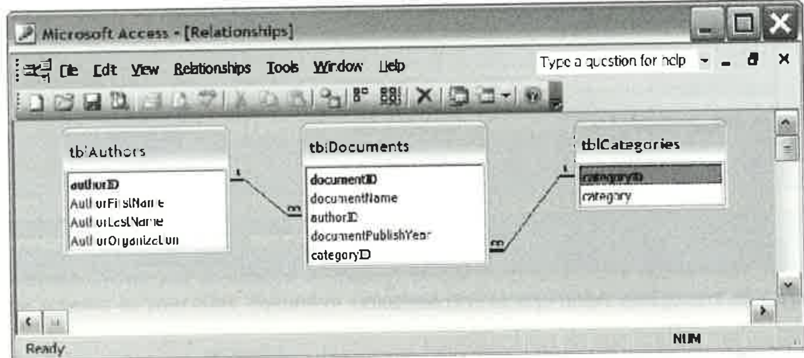


Figure 3-29. The relationships between tables in the documents.mdb database

Controlling the structure of XML documents

XML documents exported from Access are shorter than their Word and Excel equivalents. The elements in the XML document take their names from the field names in the table or query. Access replaces the spaces in field names with an underscore (_) character.

If you don't want to use the default field names in the table, an alternative is to create a query first that joins all the data and then export that to an XML document. Access won't give you the option to export data in linked tables, but the rest of the process is much the same as for exporting tables.

The following listing shows a trimmed-down version of the XML document, accessQryBookDetailsExport.xml. You can also look at the schema file, accessQryBookDetailsExport.xsd.

```
<?xml version="1.0" encoding="UTF-8"?>
<dataroot xmlns:od="urn:schemas-microsoft-com:officedata"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="accessQryBookDetailsExport.xsd"
generated="2005-03-04T18:50:47">
  <qryBookDetails>
    <documentID>2</documentID>
    <documentName>Bike riding for non-bike riders</documentName>
    <authorID>4</authorID>
```


Facebook's Exhibit No. 1005
Page 0099

 GOOGLE EXHIBIT 1005

```
        <AuthorFirstName>Saul</AuthorFirstName>
        <AuthorLastName>Sorenson</AuthorLastName>
        <AuthorOrganization>Organization D</AuthorOrganization>
        <documentPublishYear>2004</documentPublishYear>
        <categoryID>5</categoryID>
        <category>Bike riding</category>
    </qryBookDetails>
</dataroot>
```

This XML document organizes the data by document and shows the relationships between the related tables. You could also have organized the data by author or category.

For an example of documents organized by author, see the resource file `accessQryAuthorDocuments.xml`, shown in the listing that follows, and the resource file `accessQryAuthorDocuments.xsd`.

```
<?xml version="1.0" encoding="UTF-8"?>
<dataroot xmlns:od="urn:schemas-microsoft-com:officedata"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="accessQryAuthorDocuments.xsd"
generated="2005-03-04T18:53:15">
    <qryAuthorDocuments>
        <authorID>1</authorID>
        <AuthorFirstName>Alison</AuthorFirstName>
        <AuthorLastName>Ambrose</AuthorLastName>
        <AuthorOrganization>Organization A</AuthorOrganization>
        <documentID>4</documentID>
        <documentName>Fishing tips</documentName>
        <documentPublishYear>1999</documentPublishYear>
    </qryAuthorDocuments>
    <qryAuthorDocuments>
      <authorID>1</authorID>
      <AuthorFirstName>Alison</AuthorFirstName>
        <AuthorLastName>Ambrose</AuthorLastName>
        <AuthorOrganization>Organization A</AuthorOrganization>
        <documentID>1</documentID>
        <documentName>Shopping for profit and pleasure</documentName>
        <documentPublishYear>2002</documentPublishYear>
    </qryAuthorDocuments>
</dataroot>
```

Writing queries still doesn't quite solve our problem. A better structure for the XML file from Access would have been to group the documents within each <authorID> element. Access doesn't do this automatically.

Using Access VBA and XML

You can automate XML importing and exporting with Access 2003 VBA. Access recognizes the `Application.ImportXML` and `Application.ExportXML` methods. You can trigger them from buttons on a form. It's important to note that VBA can't transform an XML document during the import process.

Facebook's Exhibit No. 1005
Page 00100

GOOGLE EXHIBIT 1005

InfoPath

Office 2003 for PCs includes a new product called InfoPath that allows people to create and edit XML documents by filling in forms. The forms allow you to collect XML information and use it with your other business systems.

InfoPath is included in Microsoft Office Professional Enterprise Edition 2003, or you can buy it separately. There is no equivalent product for Macintosh Office 2004 users.

Office 2003 and data structure

Office 2003 can generate XML documents for use by other applications, including Flash movies. If you set up Word, Excel, or Access properly, your users can maintain their own XML documents using Office 2003. Most people are familiar with these software packages, so it's not terribly demanding for them to use them as tools for maintaining their data.

As you can see from the previous sections, each of the Office applications works with particular data structures. Word 2003 works best with nonrepeating information, a bit like filling in a form to generate the XML elements. Excel 2003 is best with grid-like data structures that don't include mixed elements. Access 2003 works with relational data, and you can write queries to specify which data to export. Using a schema in Word and Excel greatly simplifies the XML documents that they produce. We'll look at creating schemas a little later in this chapter.

Consuming a web service

You've probably heard the term *web services* mentioned a lot. The official definition from the W3C at `www.w3.org/TR/ws-gloss/#defs` is

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

In simpler terms, a web service is a way for you to access data on another system using an XML format. Web services operate over the Internet and are platform independent. In order to use a web service, you request information and receive a response in an XML document.

You can use web services to look up a variety of information, including television guides, movie reviews, and weather updates. As an author, I can use Amazon's web service to find out the sales ranking and database details for any books that I've written.

When you start reading about web services, you'll see the terms UDDI, WSDL, SOAP, and REST. A glossary for the main terms associated with web services is at `www.w3.org/TR/2004/NOTE-ws-gloss-20040211/`.

You can find out what web services are available through a company's Universal Description, Discovery, and Integration (UDDI) registry. The UDDI contains a description of the web services that are available and the way that you can access them.

Web Services Description Language (WSDL) describes web services in a standard XML format. In case you're interested, most people pronounce this as *whizdle*. At the time of writing, the working draft for WSDL version 2 was available at `www.w3.org/TR/2004/WD-wsdl20-20040803/`.

Facebook's Exhibit No. 1005
Page 00101

GOOGLE EXHIBIT 1005

The WSDL definition explains what is available through the web service, where it is located, and how you should make a request. It lists the parameters you need to include when requesting information, such as the fields and datatypes that the web service expects.

You can request information from a web service using a number of different protocols. The SOAP protocol is probably the most commonly used and has support within Flash. You can also use Representational State Transfer (REST), but Flash doesn't support this format natively.

SOAP, which stands for Simple Object Access Protocol, is a format for sending messages to web services. A SOAP message is an XML document with a specific structure. The request is contained within a part of the document called a SOAP Envelope.

You can find more about SOAP by viewing the note submitted to the W3C at www.w3.org/TR/2000/NOTE-SOAP-20000508/. This document isn't a W3C recommendation. At the time of writing, a working draft of SOAP version 1.2 was available at www.w3.org/TR/2002/WD-soap12-part1-20020626/.

REST is another way to work with web services. It is not a W3C standard; rather, REST is a style for interacting with web services. REST allows you to make requests through a URL rather than by sending an XML document request. Flash doesn't support REST requests, but you'll see a little later on that they can be very useful if you need to add data from a web service to a Flash movie.

Using web services to interact with Amazon

Amazon jumped into web services relatively early on. At the time of writing, the latest version of the Amazon E-Commerce Service (ECS) was version 4.0, which was released on October 4, 2004. You can find comprehensive information about ECS at www.amazon.com/gp/aws/landing.html. It's free to use, but you have to register with Amazon first to get a subscription ID before you can start making requests.

The ECS provides access to information about products, customer content, sellers, marketplace listings, and shopping carts. You could use ECS to build an Amazon search and purchase application on your own website.

The WSDL for the U.S. service can be found at webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl. You can open the file in a web browser if you want to see what it contains. The schema for the U.S. service is at webservices.amazon.com/AWSECommerceService/AWSECommerceService.xsd. Again, you can view this file in a web browser. The other Amazon locations supported are the UK, Germany, Japan, France, and Canada.

The Application Programming Interface (API) for Amazon web services describes all the operations you can perform. This includes functions like ItemLookup and ItemSearch. You can also work with wish lists and shopping carts.

To make a REST query to search for an item at Amazon, you could use the following URL format:

```
http://webservices.amazon.com/onca/xml?Service=AWSECommerceService
&SubscriptionId=[YourSubscription ID Here]&Operation=ItemSearch
&SearchIndex=[A Search Index String]&Keywords=[A Keywords String]
&Sort=[A Sort String]
```

Facebook's Exhibit No. 1005
Page 00102

Page 102 of 140 GOOGLE EXHIBIT 1005

The request can include other optional parameters, and you can find out more in the online documentation. You can also get help by using the Help operation.

In the sample request that follows, I'm using my own name to search for books in the U.S. Amazon database. I have replaced my subscriptionID with XXXX; you'll need to use your own ID if you want to run the query.

```
http://webservices.amazon.com/onca/xml?Service=AWSECommerceService
&SubscriptionId=XXXX&Operation=ItemSearch&SearchIndex=Books
&Author=Sas%20Jacobs
```

If I enter the URL into the address line of a web browser, the request will run and the results will display in the browser window. All Amazon responses have the same structure, as shown in this listing:

```
<?xml version="1.0" encoding="UTF-8">
<rootTag xmlns="http://webservices.amazon.com/AWSECommerceService/ ➥
2004-03-19">
  <OperationRequest>
    ... XML header and HTTP request information
  </OperationRequest>
  <Items>
    ... XML data here
  </Items>
</rootTag>
```

The name of the root element will vary depending on the type of request that you made. For example, an ItemSearch request will use the root element name <ItemSearchResponse>. If there are any errors in your request, they'll be contained inside an <Errors> element.

When I made the preceding REST request, I received the response shown in the following listing. Note that I've removed the sections containing the subscriptionID from the listing. I've saved the results in the file AmazonQueryResults.xml.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ItemSearchResponse xmlns="http://webservices.amazon.com/ ➥
AWSECommerceService/2005-02-23">
  <OperationRequest>
    <HTTPHeaders>
      <Header Name="UserAgent" Value="Mozilla/4.0 (compatible;
      MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)" />
    </HTTPHeaders>
    <RequestId>05BXE60PQPM6P687J1PA</RequestId>
    <Arguments>
      <Argument Name="Service" Value="AWSECommerceService" />
      <Argument Name="SearchIndex" Value="Books" />
      <Argument Name="Author" Value="Sas Jacobs" />
      <Argument Name="Operation" Value="ItemSearch" />
    </Arguments>
    <RequestProcessingTime>0.0390307903289795</RequestProcessingTime>
  </OperationRequest>
```


Facebook's Exhibit No. 1005
Page 00103

 GOOGLE EXHIBIT 1005

```
    <Items>
      <Request>
        <IsValid>True</IsValid>
        <ItemSearchRequest>
          <Author>Sas Jacobs</Author>
          <SearchIndex>Books</SearchIndex>
        </ItemSearchRequest>
      </Request>
      <TotalResults>2</TotalResults>
      <TotalPages>1</TotalPages>
        <Item>
          <ASIN>8931435061</ASIN>
          <DetailPageURL>http://www.amazon.com/exec/obidos/redirect?
          tag=ws%26link_code=xm2%26camp=2025%26creative=165953%26path=
          http://www.amazon.com/gp/redirect.html%253fASIN=8931435061%252
          location=/o/ASIN/8931435061%25253F
          </DetailPageURL>
          <ItemAttributes>
            <Author>Sas Jacobs</Author>
            <Author>YoungJin.com</Author>
            <Author>Sybex</Author>
            <ProductGroup>Book</ProductGroup>
            <Title>Flash MX 2004 Accelerated: A Full-Color Guide</Title>
        </ItemAttributes>
      </Item>
    </Items>
  </ItemSearchResponse>
```

It's common to query web services using a SOAP request. Usually some kind of server-side script generates the request for you. The WebServiceConnector data component in Flash can also generate SOAP requests.

Google provides an example of a web service that you can query with SOAP. At the time of writing, Google provided three different operations: doGetCachedpage, doSpellingSuggestion, and doGoogleSearch. You can see the WSDL at http://api.google.com/GoogleSearch.wsdl.

The W3C provides a sample SOAP message for Google at www.w3.org/2004/06/03-google-soap-wsdl.html. This listing shows an example based on the W3C sample. It does a search for the term Flash XML books:

```
<?xml version='1.0' encoding='UTF-8'?>
<soap11:Envelope xmlns="urn:GoogleSearch"
xmlns:soap11="http://schemas.xmlsoap.org/soap/envelope/">
  <soap11:Body>
    <doGoogleSearch>
      <key>00000000000000000000000000000000</key>
      <q>Flash XML books</q>
      <start>0</start>
      <maxResults>10</maxResults>
      <filter>true</filter>
```

```
        <restrict></restrict>
        <safeSearch>false</safeSearch>
        <lr></lr>
        <ie>latin1</ie>
        <oe>latin1</oe>
      </doGoogleSearch>
    </soap11:Body>
  </soap11:Envelope>
```

This listing shows a sample result XML document from the W3C site. I've shown the first result only to simplify the output:

```
<?xml version='1.0' encoding='UTF-8'?>
<soap11:Envelope
  xmlns="urn:GoogleSearch"
  xmlns:google="urn:GoogleSearch"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:soap11="http://schemas.xmlsoap.org/soap/envelope/">
  <soap11:Body>
    <doGoogleSearchResponse>
      <return>
        <documentFiltering>false</documentFiltering>
        <estimatedTotalResultsCount>3</estimatedTotalResultsCount>
        <directoryCategories soapenc:arrayType=
        "google:DirectoryCategory[0]">
        </directoryCategories>
        <searchTime>0.194871</searchTime>
        <resultElements soapenc:arrayType="google:ResultElement[3]">
          <item>
            <cachedSize>12k</cachedSize>
            <hostName></hostName>
            <snippet>Snippet for the first result would appear here
            </snippet>
            <directoryCategory>
              <specialEncoding></specialEncoding>
              <fullViewableName></fullViewableName>
            </directoryCategory>
            <relatedInformationPresent>true</relatedInformationPresent>
            <directoryTitle></directoryTitle>
            <summary></summary>
            <URL>http://hci.stanford.edu/cs147/examples/shrdlu/</URL>
            <title><b>SHRDLU</b></title>
          </item>
        </resultElements>
        <endIndex>3</endIndex>
        <searchTips></searchTips>
        <searchComments></searchComments>
        <startIndex>1</startIndex>
        <estimateIsExact>true</estimateIsExact>
```


Facebook's Exhibit No. 1005
Page 00105

GOOGLE EXHIBIT 1005

```
      <searchQuery>shrdlu winograd maclisp teletype</searchQuery>
    </return>
  </doGoogleSearchResponse>
 </soap11:Body>
</soap11:Envelope>
```

We'll look more closely at using Flash to generate SOAP requests later in the book.

You can use the data-binding capabilities in Flash to display the results from the web service. The downside to creating a SOAP request using Flash is that you have to include your key as a parameter in the movie. This is not really a very secure option.

For security reasons, you often can't query a web service using a REST request within Flash. You need some kind of server-side interaction to make the request and pass the results into Flash. You can also use Flash Remoting to work with web services.

REST is a useful tool for Flash developers. As part of its security restrictions, recent Flash players will only let you run SOAP requests on a web service that contains a cross-domain policy file specifying your address. You can imagine that Amazon isn't going to do this for every Flash developer in the world! REST requests are a good workaround; you can use a server-side language to work with the information locally or *proxy* the information. Again, we'll cover this in more detail in Chapter 9.

Transforming XML content

Keeping XML content separate from its presentation allows you to apply many different looks to the same information. It lets you present the XML data on different devices. The requirements for displaying an XML document on a website are likely to be very different from those for printing it out or displaying it on a mobile telephone, even though the data will be the same.

Transformations are a powerful way to change the presentation of your data. Transforming means displaying, sorting, filtering, and printing the information contained within an XML document. You can use Cascading Style Sheets (CSS) to change the way your XML elements display in a web browser. XSL transformations allow you to change the display as well as include more advanced options such as sorting and filtering.

CSS

An easy way to transform the visual appearance of XML documents is by using CSS. CSS is a recommendation from the W3C. You can find out more at `www.w3c.org/Style/CSS/`.

CSS and XML work together in much the same way as CSS and HTML. You can use CSS to redefine the way XML tags display in a web browser. You include a reference to an external style sheet by adding a processing instruction below your XML declaration:

```
<?xml version="1.0"?>
<?xml-stylesheet href="styles.css" type="text/css"?>
```


Facebook's Exhibit No. 1005
Page 00106

GOOGLE EXHIBIT 1005

This is much the same as the HTML instruction:

```
<link href="styles.css" rel="style sheet" type="text/css">
```

As with HTML pages, you can include multiple style sheet links:

```
<?xml-stylesheet href="globalstyles.css" type="text/css"?>
<?xml-stylesheet href="newsstyles.css" type="text/css"?>
<?xml-stylesheet href="homestyles.css" type="text/css"?>
```

The style declarations contained with the style sheets change the appearance of the XML elements. Each style declaration refers to a different element in the XML document. They are the same CSS declarations that you would use in HTML pages, so you can change font characteristics, borders, and colors for each element.

In addition to the standard style declarations, you need to consider whether the element is a block-level or inline element. In HTML, tables and headings are block-level elements while <span> is an inline element. Block-level elements automatically display with white space. In XML, all elements are inline by default. You'll need to declare block-level elements explicitly using the style declaration `display: block`.

This listing shows style declarations from the `addressCSS.xml` file:

```
contact {
  display: block;
  margin: 5px;
}
name {
  display: block;
  font-weight: bold;
  font-size: 16px;
  color: #0033CC;
  font-family: Verdana, Arial, sans-serif;
}
address {
  font-weight: normal;
  font-size: 12px;
  font-family: Verdana, Arial, sans-serif;
}
phone {
  display: block;
  font-weight: normal;
  font-size: 12px;
  color: #0033CC;
  font-family: Verdana, Arial, sans-serif;
}
```

The style sheet is saved as `styles.css`. Note that the file contains a style declaration for every element in the XML document.

Facebook's Exhibit No. 1005
Page 00107

Page 107 of 140
GOOGLE EXHIBIT 1005

Figure 3-30 shows the file `addressCSS.xml` file opened in a web browser. It looks very different from the raw XML document.

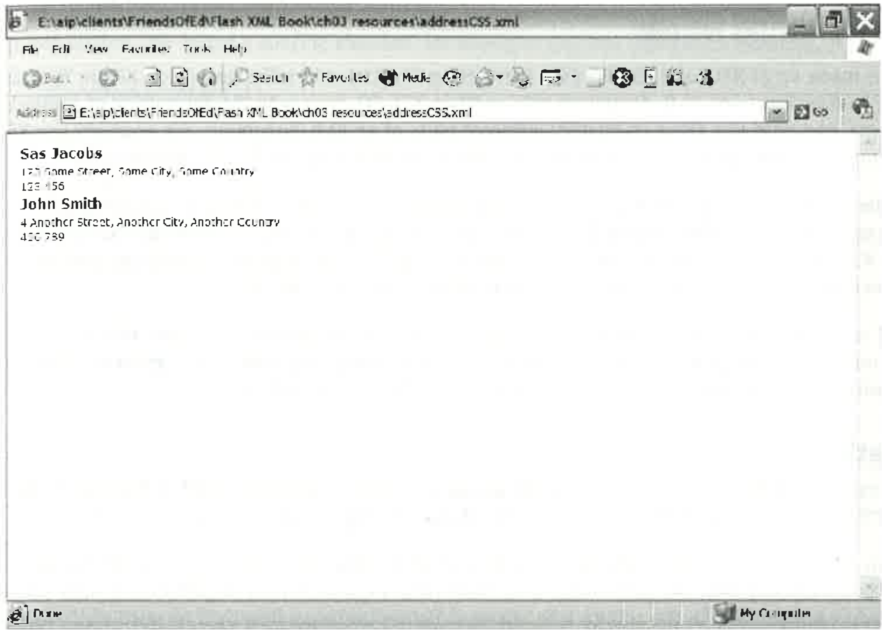


Figure 3-30. Internet Explorer showing an XML document transformed with CSS

As you can see in this example, I had to specify a style for each of my XML elements. In a large XML file, this is likely to be time consuming. CSS displays the XML elements in the same order that they appear in the XML document. I can't use CSS to change this order of the elements.

The W3C has released a recommendation titled "Associating Style Sheets with XML Documents" at `www.w3.org/TR/xml-stylesheet/`.

CSS changes the way that an XML file renders in the web browser. It doesn't offer any of the more advanced transformations that are available through XSL Transformations (XSLT). CSS may be of limited value because, unlike HTML pages, XML documents aren't always designed to be displayed in a web browser.

Bear in mind also that search engines and screen readers are likely to have difficulty when working with XML pages displayed with CSS. Search engines use the `<title>` tag, which isn't likely to be present in the same way in most XML documents. Screen readers normally require a system of headings—`<h1>`, `<h2>`, and so on—to make sense of content. It will be difficult for people using a screen reader to make sense of a document that uses nonstandard tag names.

Facebook's Exhibit No. 1005
Page 00108

GOOGLE EXHIBIT 1005

XSL

Extensible Stylesheet Language (XSL) is another way to change the display of XML documents. XSL transforms one XML document into another. As XHTML is a type of XML, you can use XSL to transform XML into an XHTML web page.

XSL is made up of XSLT and XSL-FO (XSL Formatting Objects), and relies heavily on XPath. You can use XSLT to transform one XML document into another. XSL-FO deals with the formatting of printed documents, and both use XPath to identify different parts of an XML document. We normally use XSL-FO for more complex types of printed transformations, so we'll focus on XSLT in this section.

At the time of writing, the XSL version 1 recommendation was available at www.w3.org/TR/2001/REC-xsl-20011015/. The version 1.1 working draft is at www.w3.org/TR/2004/WD-xsl11-20041216/. The XSLT version 1 recommendation is at www.w3.org/TR/1999/REC-xslt-19991116, and you'll find the version 2 working draft at www.w3.org/TR/2005/WD-xslt20-20050211/.

XSLT is much more powerful than CSS; it can convert XML documents into valid XHTML documents for use by search engines and screen readers. XSLT also filters, sorts, and rearranges data. When working with XSLT, XPath expressions identify which part of the document to transform.

XPath

You can see the XPath 1.0 recommendation at www.w3.org/TR/1999/REC-xpath-19991116. At the time of writing, the working draft for version 2 was at www.w3.org/TR/2005/WD-xpath20-20050211/.

XPath expressions provide a path to a specific part of an XML document. In a way, XPath expressions are similar to file paths. The path to the document root is specified by a single forward slash (/). As we dig into each element in the source tree, element names are separated by a forward slash, for example, /phoneBook or /phoneBook/contact.

This listing shows our simple XML phone book example:

```
<phoneBook>
  <contact id="1">
    <name>Sas Jacobs</name>
    <address>123 Some Street, Some City, Some Country</address>
    <phone>123 456</phone>
  </contact>
  <contact id="2">
    <name>John Smith</name>
    <address>4 Another Street, Another City, Another Country</address>
    <phone>456 789</phone>
  </contact>
</phoneBook>
```

To refer to the <address> elements, use the following path:

```
/phoneBook/contact/address
```

In other words, start at <phoneBook>, move to <contact>, and finish at the <address> element. You can also use references relative to the current location.

Facebook's Exhibit No. 1005
Page 00109

GOOGLE EXHIBIT 1005

Two slashes allow you to start the path anywhere in the XML document. The following code snippet specifies all <contact> elements, wherever they are located:

```
//contact
```

XPath expressions can target a specific element, for example, the first <contact> element.

```
/phoneBook/contact[1]
```

You can use the text() function to refer to the text inside an element:

```
/phoneBook/contact/address/text()
```

XPath recognizes wildcards, so we can specify **all** elements in an XML document by using the asterisk (*) character. This example shows all child elements of <phoneBook>:

```
/phoneBook/*
```

You can refer to attributes with the @ symbol, for example, the id attribute of <contact>:

```
/phoneBook/contact/@id
```

There is a lot more to the XPath specification than we've covered here, but this will provide a good starting point for the examples that will follow.

XSLT

Many people use the terms *XSL* and *XSLT* interchangeably. An XSLT stylesheet is an XML document that contains transformation rules to apply to an XML source document. We call the original XML document the *source tree*. The transformed document is the *result tree*. A style sheet is an XML document so you use the same rules for well-formedness.

XSLT style sheets start with a declaration followed by a document root. XSLT documents have a root element of either <stylesheet> or <transform>. We also need to include a reference to the namespace.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

or

```
<?xml version="1.0"?>
<xsl:transform version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

It's more common to use <stylesheet> than <transform>. The closing tag in the style sheet will need to match this declaration.

In the previous examples, the namespace uses the xsl prefix so the elements are written <xsl:stylesheet> or <xsl:transform>. The code that follows also uses the xsl prefix as we're working in the same namespace.

Facebook's Exhibit No. 1005
Page 00110

Page 110 of 140 GOOGLE EXHIBIT 1005

Transforming content

XSLT documents can include an `<template>` and an `<output>` element. The `<template>` element shows how to transform the XML elements:

```
<xsl:template match = "Xpath expression">
```

The attribute `match` specifies which elements the template should affect.

The `<output>` element defines the format for the output document:

```
<xsl:output method="html" version="4.0" indent="yes"/>
```

We use an XPath expression to target each element or group of elements to be transformed. This listing shows a transformation of the phone book XML document into an HTML document:

```
<xsl:template match="/">
  <html>
  <body>
  <h1>Phone Book</h1>
  <ul>
  <xsl:for-each select="/phoneBook/contact">
    <li><xsl:value-of select="name" /></li>
  </xsl:for-each>
  </ul>
  </body>
  </html>
</xsl:template>
```

In the example, the document root is identified and transformed to create the `<html>`, `<body>`, and `<ul>` elements. For simplicity, no DTD or <head> section has been included in HTML in this example.

Each contact creates a `<li>` element that contains the value of the <name> element.

We can use a `for-each` statement to work with elements that appear more than once in the XML source document. It's a way to loop through a collection of elements. The example loops through the `<contact>` elements using the XPath expression /phoneBook/contact as the value of the select attribute.

The `value-of` statement returns the value of an element so that it can be included with the transformed HTML. In our example, `value-of` retrieves the value of the <name> element and includes it as a list item. Because the statement is inside a `for-each` loop, the <name> element uses a relative reference. It is a child of the `<contact>` element.

I've saved the complete style sheet as the resource file `listStyle.xsl`. It's a valid XML document, so you can display it in a web browser as shown in Figure 3-31.

Facebook's Exhibit No. 1005
Page 00111

 GOOGLE EXHIBIT 1005

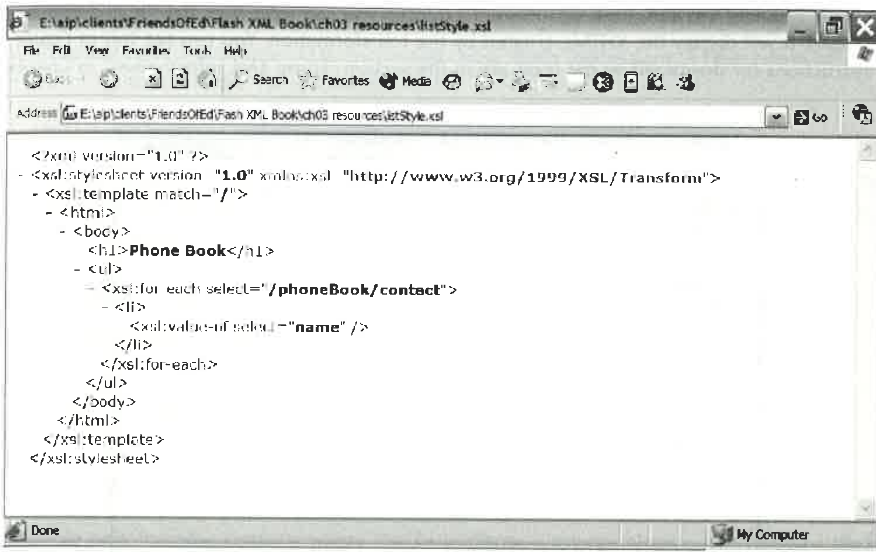


Figure 3-31. Internet Explorer showing the XSLT file

The following line applies the XSLT style sheet to the source XML document `addressXSL.xml`. It is included below the XML declaration:

```
<?xml-stylesheet type="text/xsl" href="listStyle.xsl"?>
```

Figure 3-32 shows the source document displayed in Internet Explorer after the transformation.

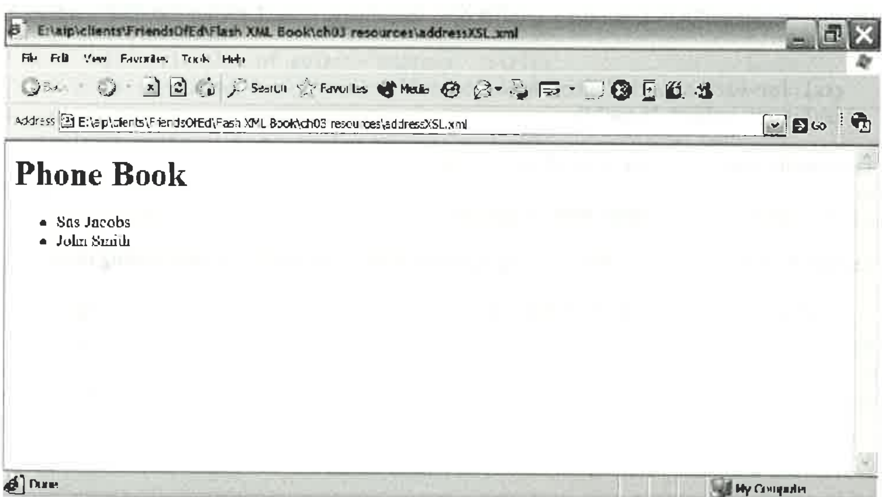


Figure 3-32. The transformed XML file in Internet Explorer

If you have installed the XML tools for Internet Explorer, you can right click the file and choose View XSL Output. It will display the XHTML created by the transformation. You can see this in Figure 3-33. The instructions for downloading the tools are in the section "Using XML information" in Chapter 2.

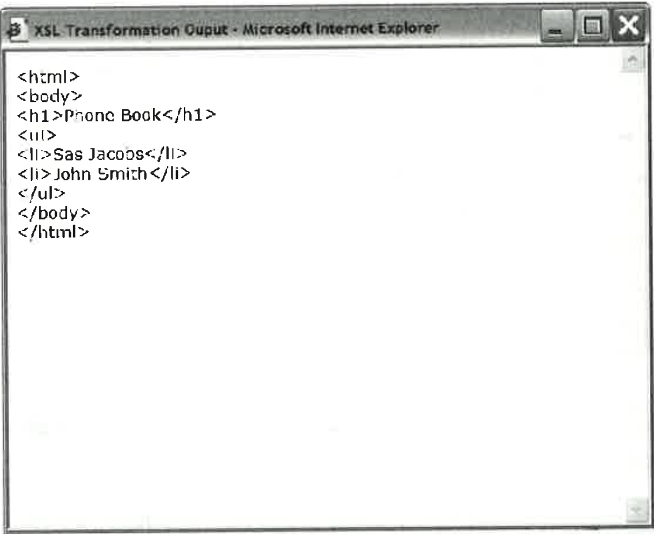


Figure 3-33. The XSL output in Internet Explorer.

Sorting content

You can sort XML documents at the same time that they are transformed. Sorting is only relevant where you have an element that repeats in the XML document. A sort element is added below a `for-each` element:

```
<xsl:for-each select="/phoneBook/contact">
<xsl:sort select="name"/>
```

You can specify more than one level of sorting with

```
<xsl:sort select="name,address,phone"/>
```

The `<sort>` element allows for other sorting options, such as ascending or descending order:

```
<xsl:sort select="name" order="descending"/>
```


Facebook's Exhibit No. 1005
Page 00113

GOOGLE EXHIBIT 1005

Filtering content

XSLT transformations can also filter XML documents using XPath expressions. A filter criterion is added to the select attributes in the for-each statement:

```
<xsl:for-each select="/phoneBook/contact[name='Sas Jacobs']">
```

This XPath expression uses [name='Sas Jacobs'] to specify which contact should be selected. This is called a *predicate*. This criterion would only display the contact with a <name> value of Sas Jacobs. You can also use != (not equal to), < (less than), and > (greater than) in filter criteria.

Conditional content

The <if> element is used to conditionally include content in the result tree:

```
<xsl:for-each select="/phoneBook/contact">
  <xsl:if test="@id&gt;1">
    <li><xsl:value-of select="name" /></li>
  </xsl:if>
</xsl:for-each>
```

This example only includes contacts where the id attribute is greater than 1. Notice that I used the entity > to replace the > sign. This is necessary because using the > sign would indicate that the element should be closed.

You can specify alternative treatment for elements using <choose>, as shown here:

```
<xsl:for-each select="/phoneBook/contact">
  <xsl:choose>
    <xsl:when test="@id&gt;1">
      <li><xsl:value-of select="name" /></li>
    </xsl:when>
    <xsl:otherwise>
      <li><xsl:value-of select="address" /></li>
    </xsl:otherwise>
  </xsl:choose>
</xsl:for-each>
```

This listing uses <when> to test whether the contact attribute id is greater than 1. If so, the value of the <name> element displays. If not, the <otherwise> element specifies that the value of the <address> element should display. It's a nonsensical example, but I think you'll get the idea.

An example

XSLT will probably become clearer when we look at another example. I'll use XSLT to transform the phone book XML document into a table layout. The example will sort the XML document into name order. Each contact will display in a new row and the name, address, and phone details in a different cell.

Figure 3-34 shows how each XML element maps to XHTML content.

Facebook's Exhibit No. 1005
Page 00114

GOOGLE EXHIBIT 1005

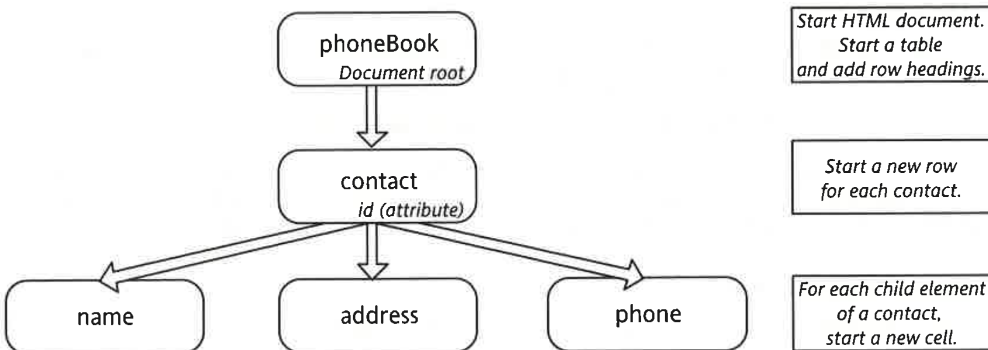


Figure 3-34. Mapping elements from the source tree to the result tree

In addition, the transformed content will be styled with a CSS style sheet.

I've called the completed XSLT file `tableStyle.xsl` (which you'll find in the resource files). The following listing shows the style sheet. Note that I've simplified the structure of the final HTML document.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <head>
      <title>Phone Book</title>
      <link href="tablestyles.css" type="text/css" rel="stylesheet"/>
    </head>
    <body>
    <h1>Phone Book</h1>
    <table>
    <tr>
      <th>Name</th>
      <th>Address</th>
      <th>Phone</th>
    </tr>
    <xsl:for-each select="/phoneBook/contact">
    <xsl:sort select="name" />
    <tr>
      <td><xsl:value-of select="name" /></td>
      <td class="shading"><xsl:value-of select="address"/></td>
      <td><xsl:value-of select="phone" /></td>
    </tr>
    </xsl:for-each>
    </table>
    </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```


Facebook's Exhibit No. 1005
Page 00115

GOOGLE EXHIBIT 1005

The style sheet starts with an XML declaration and a style sheet processing instruction. It indicates a template starting at the root element. The transformation creates the HTML document, a page head, and a title. It also creates a reference to a CSS style sheet called `tablestyles.css`. Within the body, a table with a row of headings is created.

The transformation sorts each `<contact>` element by name and displays it in a table row. Table cells are created for the `<name>`, `<address>`, and `<phone>` elements.

The XML document `address_tableXSL.xml` uses this style sheet. Figure 3-35 shows the transformed file in Internet Explorer.

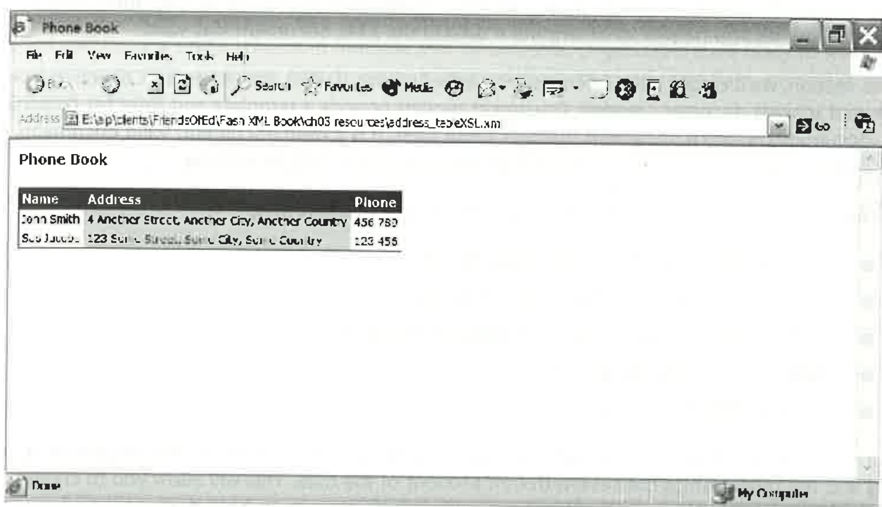


Figure 3-35. The transformed XML file in Internet Explorer

Other methods of applying transformations

You can use JavaScript to apply an XSLT transformation. This is an alternative if you don't want to include a reference to an XSLT file in your source XML document. For example, you might consider this if you want the transformations to be browser specific after detecting the browser version.

JavaScript can use the XML DOM of a web browser to transform the source tree. For example, in Internet Explorer, the `Microsoft.XMLDOM` includes the `transformNode` method. The following listing loads the XML source document and XSLT style sheet, and uses JavaScript to apply the transformation:

```
<script type="text/javascript">
  var xmlSourceDoc = new ActiveXObject("Microsoft.XMLDOM");
  xmlSourceDoc.async = false;
  xmlSourceDoc.load("address_tableXSL.xml");
  var xslTransformDoc = new ActiveXObject("Microsoft.XMLDOM");
  xslTransformDoc.async = false;
  xslTransformDoc.load("tableStyle.xsl");
  document.write(xmlSourceDoc.transformNode(xslTransformDoc));
</script>
```

Facebook's Exhibit No. 1005
Page 00116

Page 116 of 140

GOOGLE EXHIBIT 1005

Open the resource file `transform.htm` in a web browser to see the transformation in action. It looks just like the previous example, but we achieved the look in a different way.

You can also use languages like PHP, ASP.NET, and ColdFusion to apply a transformation server-side and deliver formatted XHTML to the web browser.

Determining valid XML

Earlier in this chapter, we worked with Office 2003 documents and converted them to an XML format. Using a schema allowed us to specify the element names and structures for the documents that we created. Schemas also helped to determine if data in the XML document was valid.

In this section, we'll create both Document Type Definitions (DTDs) and schemas. Collectively, we call DTDs and schema *document models*. Document models provide a template and rules for constructing XML documents. When a document matches these rules, it is a *valid* document. Valid documents must start by being well formed. Then they have to conform to the DTD or schema.

The rules contained in DTDs and schemas usually involve the following:

- Specifying the name of elements and attributes
- Identifying the type of content that can be stored
- Specifying hierarchical relationships between elements
- Stating the order for the elements
- Indicating default values for attributes

Before you create either a DTD or schema, you should be familiar with the information that you're using and the relationships between different sections of the data. This will allow you to create a useful XML representation. I find it best to work with sample data in an XML document and create the DTD or schema once I'm sure that the structure of the document meets my needs.

It's good practice to create a DTD or schema when you create multiple XML documents with the same structure. Document models are also useful where more than one author has to create the same or similar XML documents. Finally, if you need to use XML documents with other software, there may be a requirement to produce a DTD or schema so that the data translates correctly.

If you're writing a one-off XML document with element structures that you'll never use again, it's probably overkill to create a document model. It will certainly be quicker for you to create the elements as you need them and make changes as required without worrying about documentation.

Comparing DTDs and schemas

The DTD specification is older than XML schemas. In fact, DTDs predate XML documents and have their roots in Standard Generalized Markup Language (SGML). Because the specification is much older than XML, it doesn't use an XML structure.

On the other hand, schemas use XML to provide descriptions of the document rules. This means that it's possible to use an XML editor to check whether a schema is a well-formed document. You don't have this kind of checking ability with DTDs.

Facebook's Exhibit No. 1005
Page 00117

GOOGLE EXHIBIT 1005

Schemas provide many more options for specifying the type of data for elements and attributes than DTDs. You can choose from 44 built-in datatypes so, for example, you can specify whether an element contains a string, datetime, or Boolean value. You can also add restrictions to specify a range of values, for example, numbers greater than 500. If the built-in types don't meet your needs, you can create your own datatypes and inherit details from existing datatypes.

The datatype support within XML schemas gives you the ability to be very specific in your specifications. You can include much more detail about elements and attributes than is possible in a DTD. Schemas can apply more rigorous error checking than DTDs.

Schemas also support namespaces. Namespaces allow you to identify elements from different sources by providing a unique identifier. This means that you can include multiple schemas in an XML document and reuse a single schema in multiple XML documents. Organizations are likely to work with the same kinds of data, so being able to reuse schema definitions is an important advantage when working with schemas.

One common criticism of XML documents is that they are verbose. As XML documents, the same criticism could be leveled at schemas. When compared with DTDs, XML schemas tend to be much longer. It often takes several lines to achieve something that you could declare in a single line within a DTD.

Table 3-1 shows the main differences between DTDs and schemas.

Table 3-1. The main differences between DTDs and schemas

DTDs	XML Schema
Non-XML syntax.	XML syntax.
DTD can't be parsed.	XSD document can be parsed.
No support for data typing.	Datatypes can be specified and custom datatypes created.
DTDs can't inherit from one another.	Schemas support inheritance.
No support for namespaces.	Support for namespaces.
One DTD for each XML document.	Multiple schema documents can be used.
Less content.	More content.

Document Type Definitions

A DTD defines an XML document by providing a list of elements that are legal within that document. It also specifies where the elements must appear in the document as well as the number of times the element should appear.

You create or reference a DTD with a DOCTYPE declaration; you've probably seen these at the top of XHTML and HTML documents. A DTD can either be stored within an XML document or in an external DTD file.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.0//EN"
"http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd">
```

Facebook's Exhibit No. 1005
Page 00118

GOOGLE EXHIBIT 1005

The simplest DOCTYPE declaration includes only a reference to the root element of the document:

```
<!DOCTYPE phoneBook>
```

This declaration can also include other declarations, a reference to an external file, or both. DTD declarations are listed under the XML declaration:

```
<?xml version="1.0"?>
<!DOCTYPE documentRoot [element declarations]>
```

All internal declarations are contained in a DOCTYPE declaration at the top of the XML document. This includes information about the elements and attributes in the document. The element declarations can be on different lines:

```
<!DOCTYPE documentRoot [
<!ELEMENT declaration 1>
<!ELEMENT declaration 2>
]>
```

External file references point to declarations saved in files with the extension `.dtd`. They are useful if you are working with multiple documents that have the same rules. External DTD references are included in an XML document with

```
<!DOCTYPE documentRoot SYSTEM "file.dtd">
```

DTDs contain declarations for elements, attributes, and entities.

Elements

You declare an element in the following way:

```
<!ELEMENT elementName (elementContents)>
```

Make sure that you use the same case for the element name in both the declaration and XML document.

Elements that are empty—that is, that don't have any content—use the word `EMPTY`:

```
<!ELEMENT elementName (EMPTY)>
```

Child elements appear in a list after the parent element name. The order within the DTD indicates the order for the elements in the XML document:

```
<!ELEMENT elementName (child1, child2, child3)>
```

Elements can also include modifiers to indicate how often they should appear in the XML document. Children that appear once or more use a plus + sign as a modifier:

```
<!ELEMENT elementName (childName+)>
```

The pipe character (|) indicates a choice of elements. It's like including the word *or*.

```
<!ELEMENT elementName (child1|child2)>
```


Facebook's Exhibit No. 1005
Page 00119

GOOGLE EXHIBIT 1005

You can combine a choice with other elements by using brackets to group elements together:

```
<!ELEMENT elementName ((child1|child2),child3)>
<!ELEMENT elementName (child1, child2|(child3,child4))>
```

Optional child elements are shown with an asterisk. This means they can appear any number of times or not at all.

```
<!ELEMENT elementName (childName*)>
```

A question mark (?) indicates child elements that are optional but that can appear a maximum of once:

```
<!ELEMENT elementName (childName?)>
```

Elements that contain character data include CDATA as content:

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

You can also use the word ANY to indicate that any type of data is acceptable:

```
<!ELEMENT elementName (ANY)>
```

The element declarations can be quite complicated. For example:

```
<!ELEMENT elementName ((child1|child2|child3),child4+,child5*,#CDATA)>
```

This declaration means that the element called elementName contains character data. It includes a choice between the child1, child2, or child3 elements, followed by child4, which can appear once or more. The element child5 is optional.

Table 3-2 provides an overview of the symbols used in element declarations.

Table 3-2. An explanation of the symbols used in element declarations within DTDs

Symbol	Explanation
,	Specifies the order of child elements.
+	Signifies that an element has to appear at least once, i.e., one or more times.
|	Allows a choice between elements.
( )	Marks content as a group.
*	Specifies that the element is optional and can appear any number of times, i.e., 0 or more times.
?	Specifies that the element is optional, but if it is present, it can only appear once, i.e., 0 or 1 times.
	No symbol indicates that element must appear exactly once.

Attributes

Attributes declarations come after the elements. Their declarations are a little more complicated:

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

The elementName is the element that includes this attribute. Table 3-3 shows the main values for attributeType.

Table 3-3. The main attributeType values

Attribute Type	Comments
CDATA	Character data
ID	A unique identifier
IDREF	The id of another element
IDREFS	A list of ids from other elements
NMTOKEN	A valid XML name, i.e., doesn't start with a number and has no spaces
NMTOKENS	A list of valid XML names
ENTITY	An entity name
ENTITIES	A list of entity names
LIST	A list of specific values, e.g., (red | blue | green)

Most commonly, attributes are of the type CDATA.

The defaultValue indicates a default value for the element. In the following example, the XML element <address> will have an <addressType> attribute with a default value of home. In other words, if the attribute isn't included in the XML document, a value of home will be assumed.

```
<!ATTLIST address addressType CDATA "home">
```

Using #REQUIRED will force a value to be set for the attribute in the XML document:

```
<!ATTLIST address addressType CDATA #REQUIRED>
```

You can use #IMPLIED if the attribute is optional:

```
<!ATTLIST address addressType CDATA #IMPLIED>
```

If you always want to use the same value for an attribute and don't want it to be overridden, use #FIXED:

```
<!ATTLIST address addressType CDATA #FIXED "home">
```


Facebook's Exhibit No. 1005
Page 00121

GOOGLE EXHIBIT 1005

You can also specify a range of acceptable values separated by a pipe character |:

```
<!ATTLIST address addressType (home|work|mailing) "home">
```

You can declare all attributes of a single element at the same time within the same ATTLIST declaration:

```
<!ATTLIST address
  addressType (home|postal|work) #REQUIRED
  addressID CDATA #IMPLIED
  addressDefault (true|false) "true">
```

The declaration lists a required addressType attribute, which has to have a value of home, postal, or work. The addressID is a CDATA type and is optional. The final attribute, addressDefault, can have a value of either true or false with the default value being true.

You can also declare attributes separately:

```
<!ATTLIST address addressType (home|postal|work) #REQUIRED>
<!ATTLIST address addressID CDATA #IMPLIED >
<!ATTLIST address addressDefault (true|false) "true">
```

Entities

Entities are a shorthand way to refer to something that you want to use in more than one place or in more than one XML document. You also use them for specific characters on a keyboard. If you've worked with HTML, you've probably used entities for nonbreaking spaces ( ) and the copyright symbol (©).

You declare an entity as follows:

```
<!ENTITY entityName "entityValue">
```

Whenever you want to use the value of the entity in an XML document, you can use &entityName;.

In the following example, I've declared two entities, email and author:

```
<!ENTITY email "sas@aip.net.au">
<!ENTITY author "Sas Jacobs, AIP">
```

I could refer to these entities in my XML document using &email; or &author;. The entities mean sas@aip.net.au and Sas Jacobs, AIP.

Entities can also reference external content; we call these *external entities*. They are a little like using a server-side include file in an HTML document.

```
<!ENTITY address SYSTEM "addressBlock.xml">
```

The XML document would use the entity &address; to insert the contents from the addressBlock.xml file. You could also use a URL like http://www.friendsofed.com/addressBlock.xml. The advantage here is that you only have to update the entity in a single location and the value will change throughout the XML document.

Facebook's Exhibit No. 1005
Page 00122

Page 122 of 140

GOOGLE EXHIBIT 1005

A sample DTD

The following listing shows a sample inline DTD. The DTD describes our phone book XML document:

```
<!DOCTYPE phoneBook[
  <!ELEMENT phoneBook (contact+)>
  <!ELEMENT contact (name,address,phone)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT address (#PCDATA)>
  <!ELEMENT phone (#PCDATA)>
  <!ATTLIST contact id CDATA #REQUIRED>
  ]>
```

I've saved the XML document containing these declarations in the resource file `addressDTD.xml`. Figure 3-36 shows this file validated within XMLSpy.

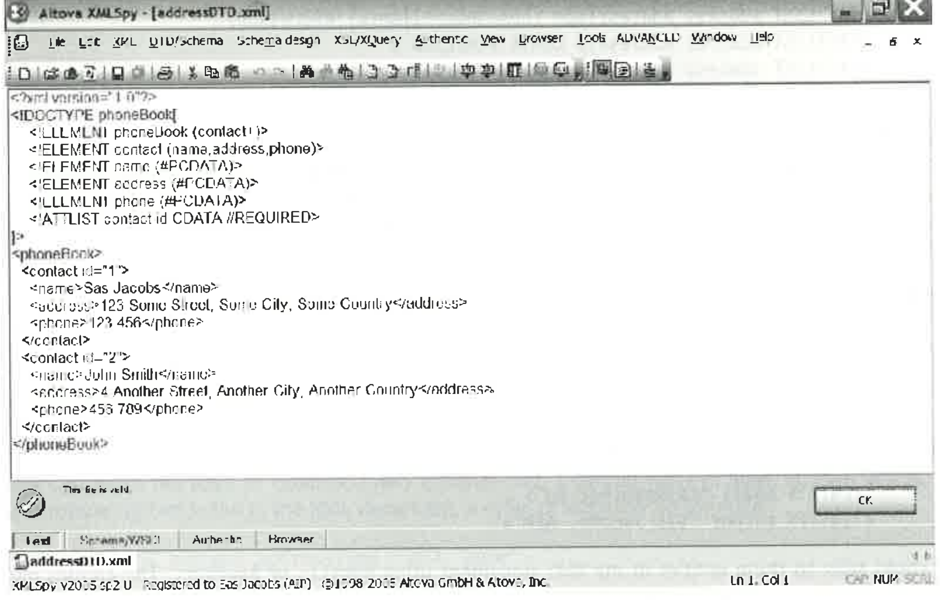


Figure 3-36. The file addressDTD.xml contains an inline DTD, which can be used to validate the contents in XMLSpy.

The file `addressEDTD.xml` refers to the same declarations in the external DTD. If you open the resource file `phoneBook.dtd` you'll see that it doesn't include a DOCTYPE declaration at the top of the file. This listing shows the content:

```
<!ELEMENT phoneBook (contact+)>
<!ELEMENT contact (name,address,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
```


Facebook's Exhibit No. 1005
Page 00123

GOOGLE EXHIBIT 1005

```
<!ELEMENT phone (#PCDATA)>
<!ATTLIST contact id CDATA #REQUIRED>
```

This DTD declares the root element phoneBook. The root element can contain a single element contact, which can appear one or more times. The contact element contains three elements—name, address, and phone—each of which must appear exactly once. The data in these elements is of type PCDATA or parsed character data.

The DTD includes a declaration for the attribute id within the contact element. The type is CDATA, and it is a required attribute.

Designing DTDs can be a tricky process, so you will probably find it easier if you organize your declarations carefully. You can add extra lines and spaces so that the DTD is easy to read.

XML schemas

An XML schema is an XML document that lists the rules for other XML documents. It defines the way elements and attributes are structured, the order of elements, and the datatypes used for elements and attributes.

A schema has the same role as a DTD. It determines the rules for valid XML documents. Unlike DTDs, however, you don't have to learn new syntax to create schemas because they are another example of an XML document. Schemas are popular for this reason. Some people find it strange that DTDs use a non-XML approach to define XML document structure.

At the time of writing, the current recommendation for XML schemas was at www.w3.org/TR/2004/REC-xmlschema-1-20041028/. You'll find the Datatypes section of the recommendation at www.w3.org/TR/2004/REC-xmlschema-2-20041028/. The working drafts for XML Schema version 1.1 are at www.w3.org/TR/2005/WD-xmlschema11-1-20050224/ and www.w3.org/TR/2005/WD-xmlschema11-2-20050224/.

Schemas offer several advantages over DTDs. Because schemas can inherit from each other, you can reuse them with different document groups. It's easier to use XML documents created from databases with schemas because they recognize different datatypes. You write schemas in XML so you can use the same tools that you use for your other XML documents.

You can embed a schema within an XML document or store it within an external XML file saved with an .xsd extension. In most cases, it's better to store the schema information externally so you'll be able to reuse it with other XML documents that follow the same format.

An external schema starts with an optional XML declaration followed by a <schema> element, which is the document root. The <schema> element contains a reference to the default namespace. The xmlns declaration shows that all elements and datatypes come from the namespace http://www.w3.org/2001/XMLSchema. In my declaration, elements from this namespace should use the prefix xsd.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

As with a DTD, a schema describes the document model for an XML document. This can consist of declarations about elements and attributes and about datatypes. The order of the declarations in the XSD document doesn't matter.

You declare elements as either simpleType or complexType. They can also have empty, simple, complex, or mixed content. Elements that have attributes are automatically complexType elements. Elements that only include text are simpleType.

I've included a sample schema document called addressSchema.xsd with your resources to illustrate some of the concepts in this section. You'll probably want to have it open as you refer to the examples that follow. You can see the complete schema at the end of this section.

In the sample schema, you'll notice that the prefix xsd is used in front of all elements. This is because I've referred to the namespace with the xsd prefix, that is, xmlns:xsd=http://www.w3.org/2001/XMLSchema. Everything included from this namespace will be prefixed in the same way.

Simple types

Simple type elements contain text only and have no attributes or child elements. In other words, simple elements contain character data. The text included in a simple element can be of any datatype. You can define simple element as follows:

```
<xsd:element name="elementName" type="elementType"/>
```

In our phone book XML document, the <name>, <address>, and <phone> elements are simple type elements. The definitions in the XSD schema document show this:

```
<xsd:element name="name" type="xsd:string"/>
<xsd:element name="address" type="xsd:string"/>
<xsd:element name="phone" type="xsd:string"/>
```

There are 44 built-in simple types in the W3C Schema Recommendation. You can find out more about these types at www.w3.org/TR/xmlschema-2/. Common simple types include string, integer, float, decimal, date, time, ID, and boolean.

Attributes are also simple type elements and are defined with

```
<xsd:attribute name="attributeName" type="elementType"/>
```

All attributes are optional unless their use attribute is set to required:

```
<xsd:attribute name="attributeName" type="elementType" use="required"/>
```

The id attribute in the <contact> element is an example of a required attribute:

```
<xsd:attribute name="id" type="xsd:integer" use="required"/>
```

A default or fixed value can be set for simple elements by using

```
<xsd:attribute name="attributeName" type="elementType"
default="defaultValue"/>
```

or

```
<xsd:attribute name="attributeName" type="elementType" fixed="fixedValue"/>
```


Facebook's Exhibit No. 1005
Page 00125

 GOOGLE EXHIBIT 1005

You can't change the value of a simple type element that has a fixed value.

Complex types

Complex type elements include attributes and/or child elements. In fact, any time an element has one or more attributes it is automatically a complex type element. The <contact> element is an example of a complex type element.

Complex type elements have different content types, as shown in Table 3-4.

Table 3-4. Complex content types

Content	Explanation	Example
Empty	Element has no content.	`<recipe id="1234"/>`
Simple	Element contains only text.	`<recipe id="1234">Omelette</recipe>`
Complex	Element contains only child elements.	`<recipe><food>Eggs</food></recipe>`
Mixed	Element contains child elements and text.	`<recipe>Omelette<food>Eggs</food></recipe>`

It's a little confusing. An element can have a complex type with simple content, or it can be a complex type element with empty content. I'll go through these alternatives next.

A complex type element with empty content such as

```
<recipe id="1234"/>
```

is defined in a schema with

```
<xsd:element name="recipe">
  <xsd:complexType>
    <xsd:attribute name="id" type="xsd:positiveInteger"/>
  </xsd:complexType>
</xsd:element>
```

The <recipe> element is a complexType but only contains an attribute. In the example, the attribute is declared. We could also use a ref attribute to refer to an attribute that is already declared elsewhere within the schema.

A complex type element with simple content like

```
<recipe id="1234">
  Omelette
</recipe>
```

is declared in the following way:

```
<xsd:element name="recipe">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="id" type="xsd:positiveInteger"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

In other words, the complex element called <recipe> has a complex type but simple content. The content has a base type of string. The element includes an attribute called id that is a positiveInteger.

Complex types have content that is either a sequence, a list, or a choice of elements. You must use either <sequence>, <all>, or <choice> to enclose your child elements. Attributes are defined outside of the <sequence>, <all>, or <choice> elements.

A complex type element with complex content such as

```
<recipe>
  <food>
    Eggs
  </food>
</recipe>
```

is declared as follows:

```
<xsd:element name="recipe">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="food"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

A complex type element with mixed content such as

```
<recipe>
  Omelette
  <food>
    Eggs
  </food>
</recipe>
```

is defined with

```
<xsd:element name="recipe">
  <xsd:complexType mixed="true">
```


Facebook's Exhibit No. 1005
Page 00127

GOOGLE EXHIBIT 1005

```
      <xsd:sequence>
        <xsd:element name="food" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

The mixed attribute is set to true so that the <recipe> element can contain a mixture of both child elements and text or character data.

If an element has children, the declaration needs to specify the names of the child elements, the order in which they appear, and the number of times that they can be included.

Ordering child elements

The sequence element specifies the order of child elements:

```
<xsd:element name="elementName">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="childElement1" type="xsd:string"/>
      <xsd:element name="childElement2" type="xsd:string"/>
      <xsd:element name="childElement3" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

You can replace sequence with all where child elements can be written in any order but each child element must appear only once:

```
<xsd:all>
  <xsd:element name="childElement1" type="xsd:string"/>
  <xsd:element name="childElement2" type="xsd:string"/>
  <xsd:element name="childElement3" type="xsd:string"/>
</xsd:all>
```

The element choice indicates that only one of the child elements should be included from the group:

```
<xsd:choice>
  <xsd:element name="childElement1" type="xsd:string"/>
  <xsd:element name="childElement2" type="xsd:string"/>
</xsd:choice>
```

Element occurrences

The number of times an element appears within another can be set with the minOccurs and maxOccurs attributes:

```
<xsd:element name="food" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
```

In the previous example, the element is optional but if it is present, it must appear only once. You can use the value unbounded to specify an unlimited number of occurrences:

```
<xsd:element name="food" type="xsd:string" minOccurs="0"
maxOccurs="unbounded"/>
```

When neither of these attributes is present, the element must appear exactly once.

Creating undefined content

If you're not sure about the structure of a complex element, you can specify any content:

```
<xsd:element name="elementName">
  <xsd:complexType>
    <xsd:any minOccurs="0" />
  </xsd:complexType>
</xsd:element>
```

The author of an XML document that uses this schema will be able to create an optional child element.

You can also use the element anyAttribute to add attributes to an element:

```
<xsd:element name="elementName">
  <xsd:complexType>
    <xsd:element name="childElement" type="xsd:string"/>
    <xsd:anyAttribute />
  </xsd:complexType>
</xsd:element>
```

Annotations

You can use annotations to describe your schemas. An <annotation> element contains a <documentation> element that encloses the description. You can add annotations anywhere, but it's often helpful to include them underneath an element declaration:

```
<xsd:element name="recipe">
  <xsd:annotation>
    <xsd:documentation>
      A description about the element
    </xsd:documentation>
  </xsd:annotation>
  ... more declarations
</xsd:element>
```

Including a schema

You can include a schema in an XML document by referencing it in the document root. Schemas always include a reference to the XMLSchema namespace. Optionally, they may include a reference to a target namespace:

```
<phoneBook xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="addressSchema.xsd">
```


Facebook's Exhibit No. 1005
Page 00129

 GOOGLE EXHIBIT 1005

The reference uses noNamespaceSchemaLocation because the schema document doesn't have a target namespace.

An example

The topic of schemas is very complicated. There are other areas that I haven't discussed in this chapter. An example that relates to the phone book XML document should make things a little clearer.

This listing shows the complete schema from the resource file addressSchema.xsd:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="phoneBook">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element ref="contact" minOccurs="1"
       maxOccurs="unbounded"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>
 <xsd:element name="contact">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element name="name" type="xsd:string"/>
       <xsd:element name="address" type="xsd:string"/>
       <xsd:element name="phone" type="xsd:string"/>
     </xsd:sequence>
   <xsd:attribute name="id" type="xsd:integer" use="required"/>
   </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

The schema starts by declaring itself as an XML document and referring to the http://www.w3.org/2001/XMLSchema namespace. The first element defined is <phoneBook>. This is a complexType element that contains one or more <contact> elements. The attribute ref indicates that I've defined <contact> elsewhere in the document.

The <contact> element contains the simple elements <name>, <address>, and <phone> in that order. Each child element of <contact> can appear only once and is of type string. The <contact> element also contains a required attribute called id that is an integer type.

The schema is saved as resource file addressSchema.xsd. The XML file that references this schema is addressSchema.xml. You can open the XML file in XMLSpy or another validating XML editor and validate it against the schema.

We haven't covered everything there is to know about XML schemas in this section, but there should be enough to get you started.

Facebook's Exhibit No. 1005
Page 00130

Page 130 of 140

GOOGLE EXHIBIT 1005

XML documents and Flash

Flash can use XML documents from any source providing that they are well formed. The most straightforward method is to use a file saved with an `.xml` extension. This document can be something that you've written in NotePad, SimpleText, or XMLSpy. It can also be an Office 2003 document, perhaps from a Word template or Excel spreadsheet.

Flash can also consume XML documents provided by web services. You can do this either by using data components or by writing ActionScript. Your Flash movie can display parts of the XML document, perhaps by binding it to a UI component such as the DataGrid.

You can also use a server-side file to consume a web service using REST. The server-side file accesses a URL and receives the XML content. The file can then provide the XML document to Flash.

In Flash version 5, a measurable speed difference was caused by different XML document structures. Information in attributes parsed more quickly than information contained in elements. As a result, early XML documents created for Flash used attributes quite a bit. I'm not sure if there is still a noticeable speed difference with later Flash players.

One useful piece of advice that I can give you is that if you're going to write ActionScript to work with XML, it will really benefit you to keep the element structures in your XML document as simple as possible. Try to avoid deeply nested elements as the document will be much harder to process than if you use flatter structures.

Creating an XML document

I'd like to finish this chapter by creating an XML document from scratch. We'll also create a schema for the document so that we can update it in Office 2003 later.

I'll work through some of the decisions that we'll need to consider when creating our XML document. For this example, you can use either a text editor or an XML editor like XMLSpy. I've used XMLSpy as you'll see from my screenshots.

We'll be creating an XML document to describe photographs for an XML photo gallery that we'll create in Chapter 4. The photos that we're going to use are stored with the resource files in the `photos` folder. To make things easier, they are all landscape photos that are exactly the same width and height. In case you're interested, they're all photos that I've taken during my travels.

Our task is to design an XML document that will store information about these photos. We'll need to store the file name of the photo, a caption, and a description. If you look at the photo names, you'll see that they all have a two-letter prefix indicating where they were taken. There are photos from Australia, Europe, the United Kingdom, the United States, and South Africa.

You can see a working example of this photo gallery at `www.sasjacobs.com`. Click the photo gallery link on the home page. The online example uses transitions between each photo.

Before we start typing, let's consider the relationships between the pieces of data that we're going to store. The photo gallery contains many photos. Each photo comes from an area or location, and more than one photo can be associated with an area.

Figure 3-37 shows the relationships that we'll need to capture in our XML document.

Facebook's Exhibit No. 1005
Page 00131

 GOOGLE EXHIBIT 1005

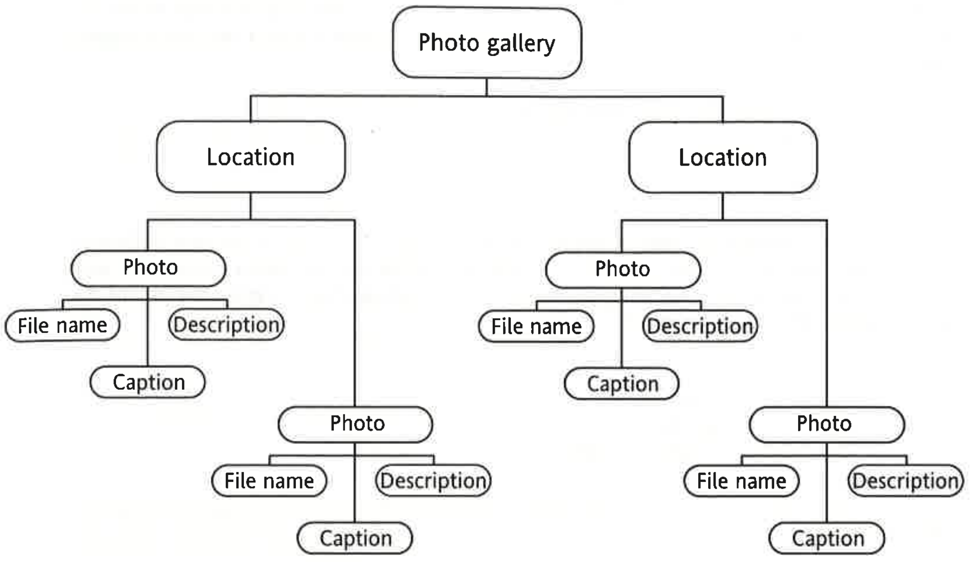


Figure 3-37. The data structure for the photo gallery

Let's start at the top and work down. We'll need to begin with an XML declaration. Actually, Flash doesn't need this, but it's a good habit for you to get into.

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

All of the information in this XML document is contained inside the photo gallery, so we'll use that for the document root:

```
<photoGallery></photoGallery>
```

The photo gallery has multiple locations for the photos. Each location has its own name. I'll create an element with an attribute for the location name. This listing shows the XML document with the list of locations—Australia, Europe, the United Kingdom, the United States, and South Africa:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<photoGallery>
  <location locationName="Australia"></location>
  <location locationName = "Europe"></location>
  <location locationName = "South Africa"></location>
  <location locationName = "UK"></location>
  <location locationName = "US"></location>
</photoGallery>
```

You'll notice that I called the attribute locationName instead of just name. There's nothing wrong with using the word name; it's just not very specific and could easily refer to other elements. More importantly, name is often a reserved word in programming languages. Even though Flash will probably let us use the word name, it may color it incorrectly in the Actions panel.

Facebook's Exhibit No. 1005
Page 00132

GOOGLE EXHIBIT 1005

Each <location> contains one or more photos, so we'll include child <photo> elements in each <location>. Using one of the <location> elements as an example, the XML document fragment looks like this:

```
<location locationName="Australia">
  <photo></photo>
  <photo></photo>
</location>
```

Each photo has a single file name, caption, and description. Here's where we have a choice to make. Photos have two characteristics as well as a text description. We can either enter these as attributes within the <photo> element, as child elements, or as a mixture of both. Here are some of the choices for structuring the XML document:

```
<photo>
  <filename></filename>
  <caption></caption
  <description></description>
</photo>
```

or

```
<photo filename="xxx" caption="yyy" description="zzz"/>
```

or

```
<photo filename="xxx" caption="yyy">
  Text displayed inside the photo element
</photo>
```

All of these choices are valid structures for the XML documents; however, the implications of each will be different.

The first choice, where all elements are child elements of <photo>, creates a clearly defined hierarchy in our elements. In the schema, we can specify the datatypes for each element as well as the order in which the elements are to appear. Actually, the order probably doesn't really matter. What is important is that there is only one occurrence of each element.

However, the first option creates a structure that nests more deeply than either of the other two examples. This means we'll need a little extra code to display the data within Flash.

The second option isn't too bad, but the description could be a problem. We'll probably want to enter quite a long description for some photos, and this might make the attribute difficult to read. We may also want to add some basic HTML tags for display within Flash, and we can't do that inside an attribute.

I favor the third option. Logically, the file name and caption are attributes of a <photo> element. Placing the text description inside the <photo> element allows us to enclose it within a CDATA declaration to preserve any HTML tags. There are no child elements within the <photo> element, which means it will be easier to process this document within Flash.

Facebook's Exhibit No. 1005
Page 00133

Page 133 of 140 GOOGLE EXHIBIT 1005

This listing shows the completed <location> element for the photographs of the United States:

```
<location locationName = "US">
  <photo filename="us-grandcanyon.jpg" caption="The Grand Canyon">
    <![CDATA[Flying through the <b>Grand Canyon</b> in a helicopter
    was an amazing experience.]]>
  </photo>
  <photo filename="us-timessquare.jpg" caption="Times Square">
    <![CDATA[There is <b>no</b> place in the world like Manhattan.]]>
  </photo>
</location>
```

Notice that I've included tags in my description text. I'll be able to display these words as bold in Flash. I had to include the text as CDATA so that the tags don't get parsed when the XML document is loaded.

I've saved the completed XML document as resource file `photoGallery.xml`. Figure 3-38 shows this document open in XMLSpy. I checked to see that the document was well formed.

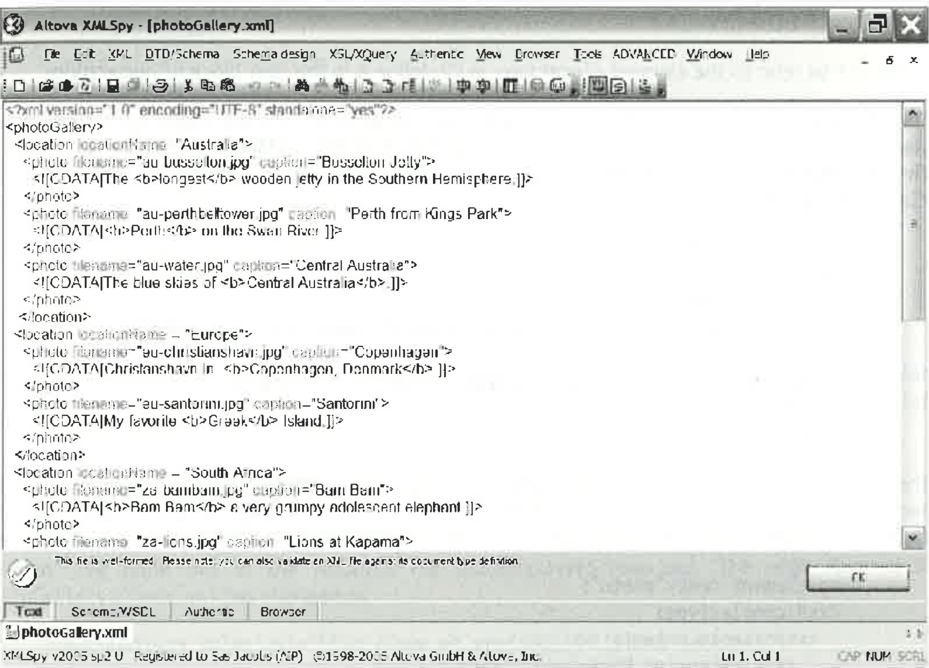


Figure 3-38. The complete file photoGallery.xml displayed in XMLSpy

Creating a schema

Now it's time to write a schema for this XML document. To start with, we'll need a new file containing an XML declaration and a root node that refers to the appropriate namespace:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
</xsd:schema>
```

I'll need to add declarations, starting with <photoGallery>, the document root of photoGallery.xml. This is a complexType element because it contains <location> elements. Each <location> element has to occur at least once and there is no upper limit for the number of repeats.

```
<xsd:element name="photoGallery">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="location" minOccurs="1"
      maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

I used ref to refer to the element <location> as I'll define it in the next block of declarations.

The <location> element contains an attribute locationName so it is automatically a complexType element. The attribute is of string type. The <location> element contains the child element <photo>. This element must occur at least once but can appear an unlimited number of times inside the <location> element.

```
<xsd:element name="location">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="photo" minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="locationName" type="xsd:string"/>
  </xsd:complexType>
  </xsd:element>
```

The final block will deal with the <photo> element. Each <photo> element has two attributes: filename and caption—and contains only text. An element with an attribute is automatically a complexType element, but because it only has text content, it is a simpleContent element. The text is string information.

```
<xsd:element name="photo">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="filename" type="xsd:string"/>
        <xsd:attribute name="caption" type="xsd:string"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```


Facebook's Exhibit No. 1005
Page 00135

 GOOGLE EXHIBIT 1005

The following listing shows the complete schema. You can also see it in the resource file photoGallerySchema.xsd.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="photoGallery">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="location" minOccurs="1"
        maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="location">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="photo" minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="locationName" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="photo">
    <xsd:complexType>
      <xsd:simpleContent>
        <xsd:extension base="xsd:string">
          <xsd:attribute name="filename" type="xsd:string"/>
          <xsd:attribute name="caption" type="xsd:string"/>
        </xsd:extension>
      </xsd:simpleContent>
     </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

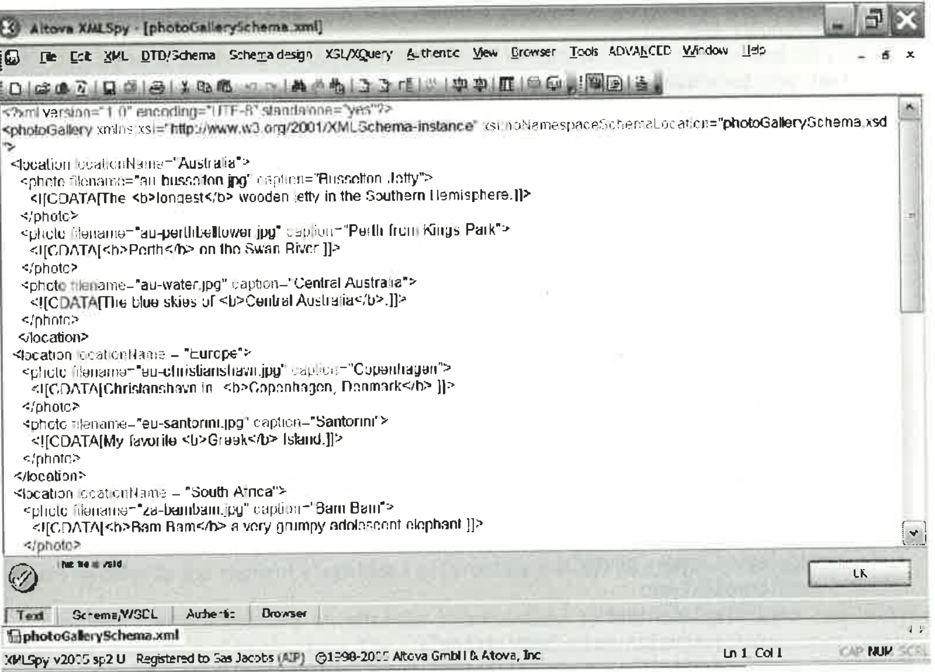
There are other ways that I could have arranged the declarations in the schema. For example, instead of using ref, I could have nested the element declarations within their parent elements. That's often referred to as a *Russian Doll* arrangement.

Linking the schema with an XML document

The final job is to link the schema with the photoGallery.xml file by adding a reference in the root element. I've done this in the resource file photoGallerySchema.xml. The root element in photoGallerySchema.xml has changed to

```
<photoGallery xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="photoGallerySchema.xsd">
```

Figure 3-39 shows the completed file in XMLSpy, validated against the schema.



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<photoGallery xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="photoGallerySchema.xsd
">
 <location locationName="Australia">
  <photo filename="au_busselton.jpg" caption="Busselton Jetty">
   <![CDATA[The <b>longest</b> wooden jetty in the Southern Hemisphere.]]>
  </photo>
  <photo filename="au-perthbelltower.jpg" caption="Perth from Kings Park">
   <![CDATA[<b>Perth</b> on the Swan River ]]>
  </photo>
  <photo filename="au-water.jpg" caption="Central Australia">
   <![CDATA[The blue skies of <b>Central Australia</b>.]]>
  </photo>
 </location>
 <location locationName = "Europe">
  <photo filename="eu-christianshavn.jpg" caption="Copenhagen">
   <![CDATA[Christianshavn in  <b>Copenhagen, Denmark</b> ]]>
  </photo>
  <photo filename="eu-santorini.jpg" caption="Santorini">
   <![CDATA[My favorite <b>Greek</b> Island.]]>
  </photo>
 </location>
 <location locationName = "South Africa">
  <photo filename="za-bambam.jpg" caption="Bam Bam">
   <![CDATA[<b>Bam Bam</b> a very grumpy adolescent elephant ]]>
  </photo>
```

Figure 3-39. The complete file photoGallerySchema.xml displayed in XMLSpy

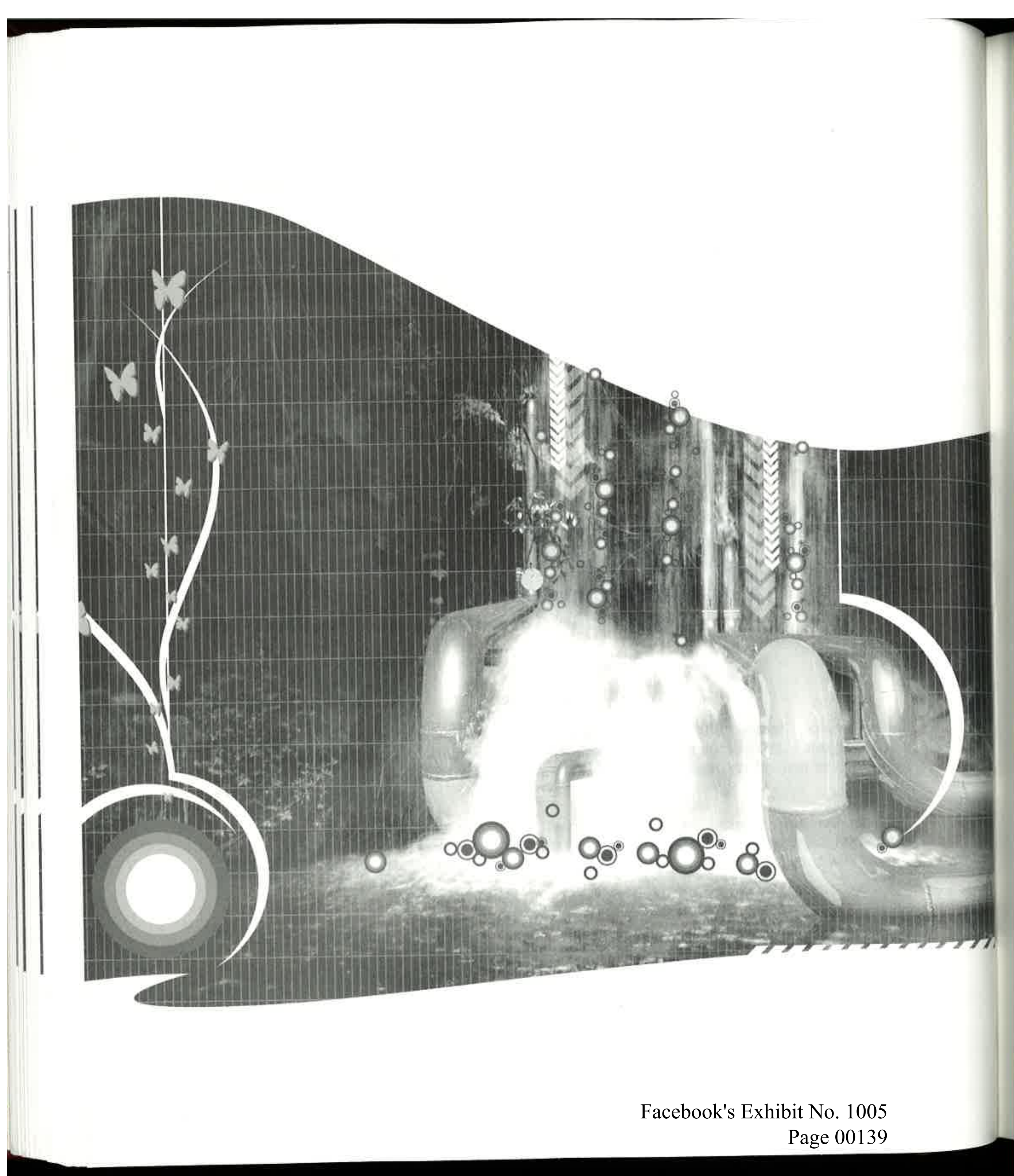
Summary

In this chapter, you looked at the different ways that you could create XML content. I covered the use of XMLSpy, Office 2003, and the role of server-side documents in generating XML. I also gave you a brief introduction to XPath, XSL transformations, DTDs, and XML schemas. We finished by creating an XML document and schema.

That's it for the theory behind XML documents. The rest of the book will focus on using Flash with XML documents. In Chapter 4, I'll look at the XML class and we'll create a photo gallery and MP3 player driven by XML data. Later in the book, we'll cover the data components that are included with Flash.

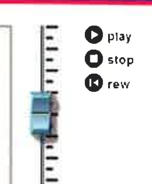
Facebook's Exhibit No. 1005
Page 00137

Page 137 of 140 GOOGLE EXHIBIT 1005



Facebook's Exhibit No. 1005
Page 00139

GOOGLE EXHIBIT 1005



LIBRARY OF CONGRESS
0 013 842 449 2

In this book, you'll learn how to:

- Add XML content to Flash applications with the XML class
- Use data components to import XML documents and bind data to user interface components
- Work with web services and display their results within Flash
- Generate XML content from Word, Excel, and Access 2003
- Build simple Flash XML applications.

Foundation XML for Flash

XML is an important technology that allows people and applications to share data in self-describing documents. Many software packages support XML data exchange, and most major databases can share information using XML. Therefore, an understanding of XML is essential for anyone working in web development.

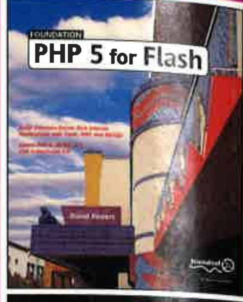
And this includes Flash! You can load external XML data and include it within your Flash movies, and also send XML content from Flash to other applications. You can harness the multimedia capabilities of Flash in a flexible XML framework to create visually appealing and usable rich Internet applications.

In this book, Sas Jacobs shows you how. She first introduces XML and related technologies, and then covers the different ways that Flash can work with XML content, including the XML class, data components, XML sockets, and web services.

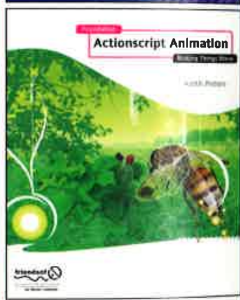
Throughout the book, you will build many different fully functional examples, including an MP3 player, an XML photo gallery, an address book manager, and an XML-driven chat application. Some of these examples utilize Office 2003 for the PC and ASP.NET or PHP.

You should read this book if you're a Flash designer or developer and you have a basic understanding of ActionScript. The book supports Flash 8, but much of the content is suitable for any version of Flash from 5 up. Whether you're new to XML or ActionScript, or an experienced Flash developer, Foundation XML for Flash is essential reading.

Also Available



Extending Flash MX 2004



SHELVING CATEGORY
1. FLASH
2. XML

Mac/PC compatible

US $39.99
www.friendsofed.com

ISBN 1-59059-543-2
53999
9 781590 595435

Facebook's Exhibit No. 1005
Page 00140