IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent of:	John Albert Kembel, et al.		
U.S. Patent No.:	8,510,407	Attorney Docket No.:	39843-0149IP1
Issue Date:	August 13, 2013		
Appl. Serial No.:	11/932,553		
Filing Date:	October 31, 2007		
Title:	DISPLAYING TIME-VARYING INTERNET BASED DATA USING APPLICATION MEDIA PACKAGES		

DECLARATION OF JUNE ANN MUNFORD

1. My name is June Ann Munford. I am over the age of 18, have personal knowledge of the facts set forth herein, and am competent to testify to the same.

2. I earned a Master of Library and Information Science (MLIS) from the University of Wisconsin-Milwaukee in 2009. I have over ten years of experience in the library/information science field. Beginning in 2004, I have served in various positions in the public library sector including Assistant Librarian, Youth Services Librarian and Library Director. I have attached my Curriculum Vitae as Appendix CV.

3. During my career in the library profession, I have been responsible for materials acquisition for multiple libraries. In that position, I have cataloged, purchased and processed incoming library works. That includes purchasing materials directly from vendors, recording publishing data from the material in question, creating detailed material records for library catalogs and physically preparing that material for circulation. In addition to my experience in acquisitions, I was also responsible for analyzing large collections of library materials, tailoring library records for optimal catalog

search performance and creating lending agreements between libraries during my time as a Library Director.

4. I am fully familiar with the catalog record creation process in the library sector. In preparing a material for public availability, a library catalog record describing that material would be created. These records are typically written in Machine Readable Catalog (herein referred to as "MARC") code and contain information such as a physical description of the material, metadata from the material's publisher, and date of library acquisition. In particular, the 008 field of the MARC record is reserved for denoting the date of creation of the library record itself. As this typically occurs during the process of preparing materials for public access, it is my experience that an item's MARC record indicates the date of an item's public availability.

5. Typically, in creating a MARC record, a librarian would gather various bits of metadata such as book title, publisher and subject headings among others and assign each value to a relevant numerical field. For example, a book's physical description is tracked in field 300 while title/attribution is tracked in field 245. The 008 field of the MARC record is reserved for denoting the creation of the library record itself. As this is the only date reflecting the inclusion of said materials within the library's collection, it is my experience

that an item's 008 field accurately indicates the date of an item's public availability.

6. I have reviewed Exhibit 1011 *HTML Unleashed* by Rick Darnell, 1st ed.

7. Attached hereto as Appendix DARNELL01 is a true and correct copy of the MARC record for *HTML Unleashed* as held by the University of California San Diego's library. I secured this record myself from the library's public catalog. The MARC record contained within Appendix DARNELL01 accurately describes the title, author, publisher, and ISBN number of *HTML Unleashed.*

8. Attached hereto as Appendix DARNELL02 is a true and correct copy of selections from *HTML Unleashed* as held by the University of California San Diego's library. These scans were secured from the library collection in person. In comparing Exhibit 1011 to Appendix DARNELL02, it is my determination that Exhibit 1011 is a true and correct copy of *HTML Unleashed* by Rick Darnell.

9. The 008 field of the MARC record in Appendix DARNELL01 indicates the date of record creation. The 008 field of Appendix DARNELL01 indicates University of California San Diego's library first acquired this book as of

August 12, 1997. Considering this information, it is my determination that *HTML Unleashed* was made available to the public shortly after its initial acquisition in August 1997.

10. I have been retained on behalf of the Petitioner to provide assistance in the above-illustrated matter in establishing the authenticity and public availability of the documents discussed in this declaration. I am being compensated for my services in this matter at the rate of $100.00 per hour plus reasonable expenses. My statements are objective, and my compensation does not depend on the outcome of this matter.

11. I declare under penalty of perjury that the foregoing is true and correct. I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code.

Dated: 02/08/2023

[illegible]

June Ann Munford

APPENDIX CV

J. Munford
Curriculum Vitae

Education

University of Wisconsin-Milwaukee - MS, Library & Information Science, 2009
Milwaukee, WI

● Coursework included cataloging, metadata, data analysis, library systems, management strategies and collection development.
● Specialized in library advocacy, cataloging and public administration.

Grand Valley State University - BA, English Language & Literature, 2008
Allendale, MI

● Coursework included linguistics, documentation and literary analysis.
● Minor in political science with a focus in local-level economics and government.

Professional Experience

Researcher / Expert Witness, October 2017 – present
Freelance ● Pittsburgh, Pennsylvania & Grand Rapids, Michigan

● Material authentication and public accessibility determination. Declarations of authenticity and/or public accessibility provided upon research completion. Experienced with appeals and deposition process.

● Research provided on topics of public library operations, material publication history, digital database services and legacy web resources.

● Past clients include Alston & Bird, Arnold & Porter, Baker Botts, Fish & Richardson, Erise IP, Irell & Manella, O'Melveny & Myers, Perkins-Coie, Pillsbury Winthrop Shaw Pittman and Slayden Grubert Beard.

Library Director, February 2013 - March 2015
Dowagiac District Library ● Dowagiac, Michigan

● Executive administrator of the Dowagiac District Library. Located in

Southwest Michigan, this library has a service area of 13,000, an annual operating budget of over $400,000 and total assets of approximately $1,300,000.

● Developed careful budgeting guidelines to produce a 15% surplus during the 2013-2014 & 2014-2015 fiscal years while being audited.

● Using this budget surplus, oversaw significant library investments including the purchase of property for a future building site, demolition of existing buildings and building renovation projects on the current facility.

● Led the organization and digitization of the library's archival records.

● Served as the public representative for the library, developing business relationships with local school, museum and tribal government entities.

● Developed an objective-based analysis system for measuring library services - including a full collection analysis of the library's 50,000+ circulating items and their records.

November 2010 - January 2013
Librarian & Branch Manager, Anchorage Public Library ● Anchorage, Alaska

● Headed the 2013 Anchorage Reads community reading campaign including event planning, staging public performances and creating marketing materials for mass distribution.

● Co-led the social media department of the library's marketing team, drafting social media guidelines, creating original content and instituting long-term planning via content calendars.

● Developed business relationships with The Boys & Girls Club, Anchorage School District and the US Army to establish summer reading programs for children.

June 2004 - September 2005, September 2006 - October 2013
Library Assistant, Hart Area Public Library
Hart, MI

● Responsible for verifying imported MARC records and original MARC cataloging for the local-level collection as well as the Michigan Electronic Library.

● Handled OCLC Worldcat interlibrary loan requests & fulfillment via ongoing communication with lending libraries.

Professional Involvement

Alaska Library Association - Anchorage Chapter
● Treasurer, 2012

Library Of Michigan
● Level VII Certification, 2008
● Level II Certification, 2013

Michigan Library Association Annual Conference 2014
● New Directors Conference Panel Member

Southwest Michigan Library Cooperative
● Represented the Dowagiac District Library, 2013-2015

Professional Development

Library Of Michigan Beginning Workshop, May 2008
Petoskey, MI
● Received training in cataloging, local history, collection management, children's literacy and reference service.

Public Library Association Intensive Library Management Training, October 2011
Nashville, TN
● Attended a five-day workshop focused on strategic planning, staff management, statistical analysis, collections and cataloging theory.

Alaska Library Association Annual Conference 2012 - Fairbanks, February 2012
Fairbanks, AK
● Attended seminars on EBSCO advanced search methods, budgeting, cataloging, database usage and marketing.

Depositions

2019 ● Fish & Richardson
IPR Petitions of 865 Patent, Apple v. Qualcomm (IPR2018-001281 / 39521-00421IP & IPR2018-01282 / 39521-00421IP2)

2019 ● Erise IP
Implicit, LLC v. Netscout Systems, Inc (Civil Action No. 2:18-cv-53-JRG)

2019 ● Perkins-Coie
Adobe Inc. v. RAH Color Technologies LLC (Cases IPR2019-00627, IPR2019-00628, IPR2019-00629 and IPR2019-00646)

2020 ● O'Melveny & Myers
Maxell, Ltd. v. Apple Inc. (Case 5:19-cv-00036-RWS)

2021 ● Pillsbury Winthrop Shaw Pittman LLP
Intel v. SRC (Case IPR2020-1449)

Limited Case History & Potential Conflicts

Alston & Bird
- Nokia (v. Neptune Subsea, Xtera)

Arnold & Porter
- Ivantis (v. Glaukos)

Erise I.P.
- Apple
  - v. Future Link Systems (IPRs 6317804, 6622108, 6807505, and 7917680)
  - v. INVT
  - v. Navblazer LLC (Case No. IPR2020-01253)

v. Qualcomm (IPR2018-001281, 39521-00421IP, IPR2018-01282, 39521-00421IP2)

v. Quest Nettech Corp, Wynn Technologies (Case No. IPR2019-00XXX, RE. Patent Re38137)

● Fanduel (v CGT)

● Garmin (v. Phillips North America LLC, Case No. 2:19-cv-6301-AB-KS Central District of California)

● Netscout

v. Longhorn HD LLC)

v. Implicit, LLC (Civil Action No. 2:18-cv-53-JRG)

● Sony Interactive Entertainment LLC

v. Bot M8 LLC

v. Infernal Technology LLC

● Unified Patents (v GE Video Compression, Civil Action No. 2:19-cv-248)

Fish & Richardson

● Apple

v. LBS Innovations

v. Masimo (IPR 50095-0012IP1, 50095-0012IP2, 50095-0013IP1, 50095-0013IP2, 50095-0006IP1)

v. Neonode

v. Qualcomm (IPR2018-001281, 39521-00421IP, IPR2018-01282, 39521-00421IP2)

● Dish Network

v. Realtime Adaptive Streaming, Case No 1:17-CV-02097-RBJ)

v. TQ Delta LLC

● Huawei (IPR 76933211)

● Kianxis

● LG Electronics (v. Bell Northern Research LLC, Case No. 3:18-cv-2864-CAB-BLM)

● Metaswitch

● MLC Intellectual Property (v. MicronTech, Case No. 3:14-cv-03657-SI)

● Realtek Semiconductor

● Quectel

● Samsung (v. Bell Northern Research, Civil Action No. 2:19-cv-00286-JRG)

● Texas Instruments

Irell & Manella

● Curium

O'Melveny & Myers

● Apple (v. Maxell, Case 5:19-cv-00036-RWS)

Perkins-Coie

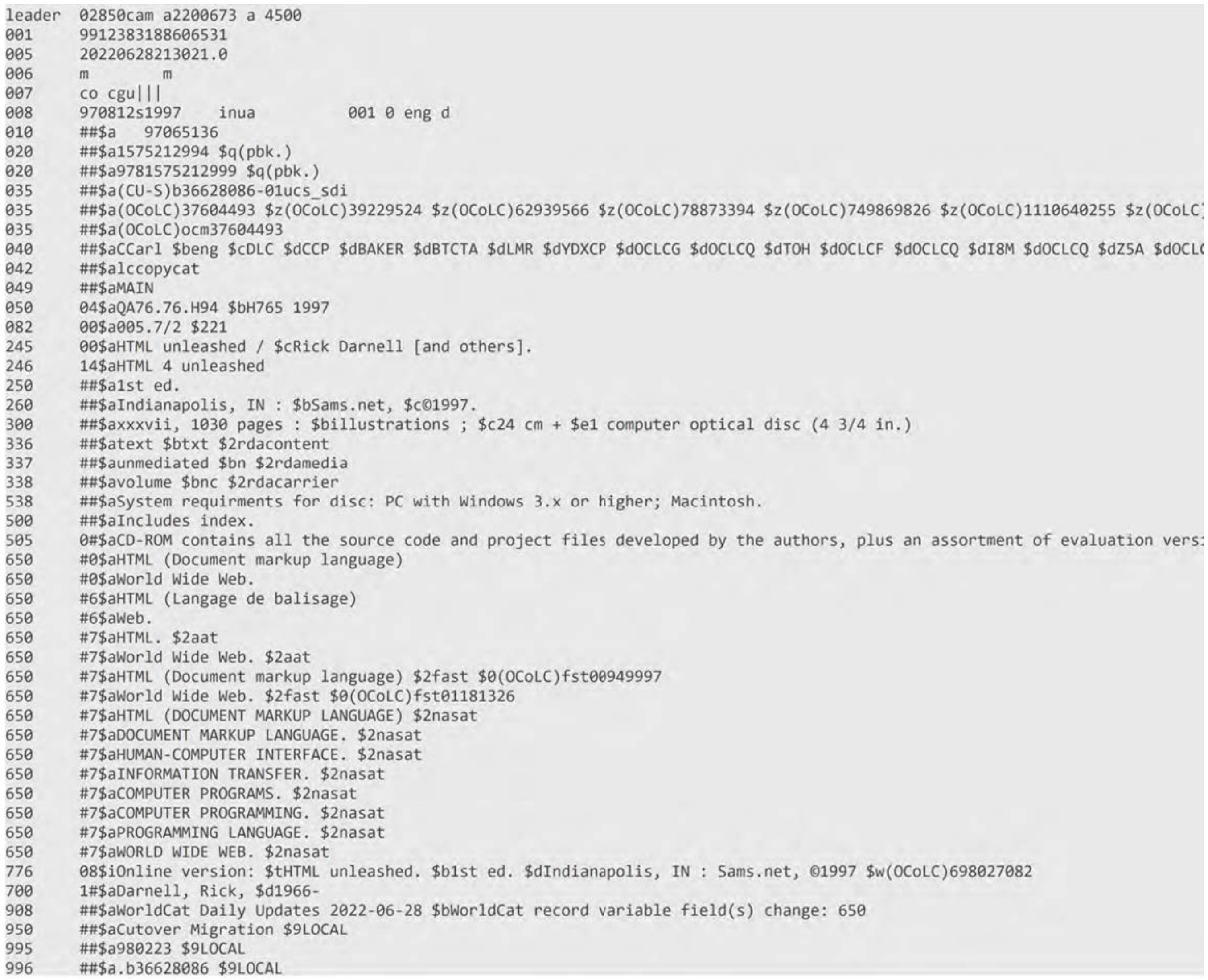
● TCL Industries (v. Koninklijke Philips NV, PTAB Case Nos. IPR2021-00495, IPR2021-00496, and IPR2021-00497)

Pillsbury Winthrop Shaw Pittman

● Intel (v. FG SRC LLC, Case No. 6:20-cv-00315 W.D. Tex)

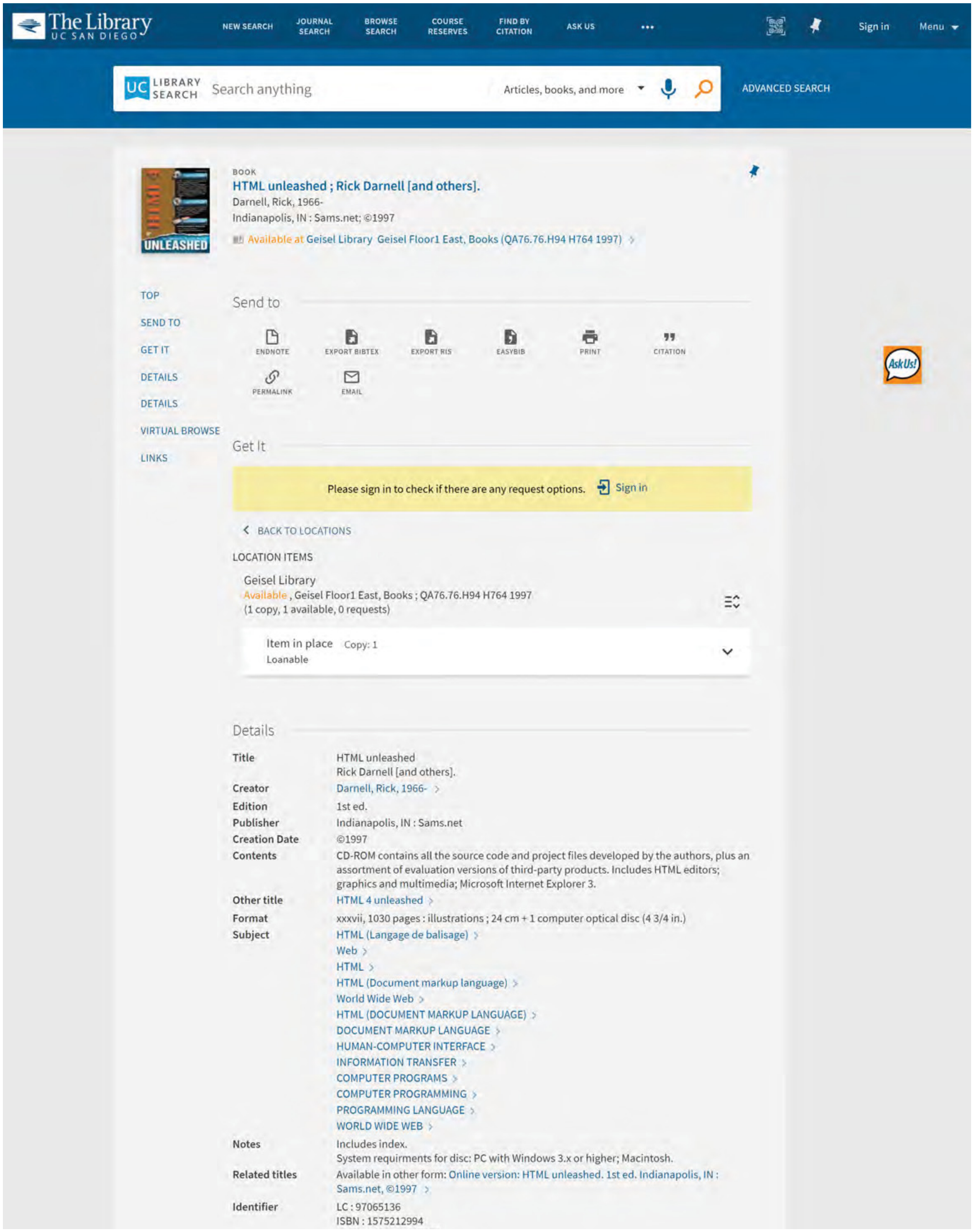
APPENDIX DARNELL01

screenshot-search-library.ucsd.edu-2022.11.21-07_50_42
https://search-library.ucsd.edu/discovery/sourceRecord?vid=01UCS_SDI:UCSD&docId=alma991014528779706535&recordOwner=01UCS_NETWORK
21.11.2022



```
leader  02850cam a2200673 a 4500
001     9912383188606531
005     20220628213021.0
006     m        m
007     co cgu|||
008     970812s1997    inua          001 0 eng d
010     ##$a   97065136
020     ##$a1575212994 $q(pbk.)
020     ##$a9781575212999 $q(pbk.)
035     ##$a(CU-S)b36628086-01ucs_sdi
035     ##$a(OCoLC)37604493 $z(OCoLC)39229524 $z(OCoLC)62939566 $z(OCoLC)78873394 $z(OCoLC)749869826 $z(OCoLC)1110640255 $z(OCoLC
035     ##$a(OCoLC)ocm37604493
040     ##$aCCarl $beng $cDLC $dCCP $dBAKER $dBTCTA $dLMR $dYDXCP $dOCLCG $dOCLCQ $dTOH $dOCLCF $dOCLCQ $dI8M $dOCLCQ $dZ5A $dOCLC
042     ##$alccopycat
049     ##$aMAIN
050     04$aQA76.76.H94 $bH765 1997
082     00$a005.7/2 $221
245     00$aHTML unleashed / $cRick Darnell [and others].
246     14$aHTML 4 unleashed
250     ##$a1st ed.
260     ##$aIndianapolis, IN : $bSams.net, $c©1997.
300     ##$axxxvii, 1030 pages : $billustrations ; $c24 cm + $e1 computer optical disc (4 3/4 in.)
336     ##$atext $btxt $2rdacontent
337     ##$aunmediated $bn $2rdamedia
338     ##$avolume $bnc $2rdacarrier
538     ##$aSystem requirments for disc: PC with Windows 3.x or higher; Macintosh.
500     ##$aIncludes index.
505     0#$aCD-ROM contains all the source code and project files developed by the authors, plus an assortment of evaluation vers
650     #0$aHTML (Document markup language)
650     #0$aWorld Wide Web.
650     #6$aHTML (Langage de balisage)
650     #6$aWeb.
650     #7$aHTML. $2aat
650     #7$aWorld Wide Web. $2aat
650     #7$aHTML (Document markup language) $2fast $0(OCoLC)fst00949997
650     #7$aWorld Wide Web. $2fast $0(OCoLC)fst01181326
650     #7$aHTML (DOCUMENT MARKUP LANGUAGE) $2nasat
650     #7$aDOCUMENT MARKUP LANGUAGE. $2nasat
650     #7$aHUMAN-COMPUTER INTERFACE. $2nasat
650     #7$aINFORMATION TRANSFER. $2nasat
650     #7$aCOMPUTER PROGRAMS. $2nasat
650     #7$aCOMPUTER PROGRAMMING. $2nasat
650     #7$aPROGRAMMING LANGUAGE. $2nasat
650     #7$aWORLD WIDE WEB. $2nasat
776     08$iOnline version: $tHTML unleashed. $b1st ed. $dIndianapolis, IN : Sams.net, ©1997 $w(OCoLC)698027082
700     1#$aDarnell, Rick, $d1966-
908     ##$aWorldCat Daily Updates 2022-06-28 $bWorldCat record variable field(s) change: 650
950     ##$aCutover Migration $9LOCAL
995     ##$a980223 $9LOCAL
996     ##$a.b36628086 $9LOCAL
```

screenshot-search-library.ucsd.edu-2022.11.21-07_51_27
https://search-library.ucsd.edu/discovery/fulldisplay?context=L&vid=01UCS_SDI:UCSD&search_scope=ArticlesBooksEtc&docid=alma991014528779706535
21.11.2022



BOOK
HTML unleashed ; Rick Darnell [and others].
Darnell, Rick, 1966-
Indianapolis, IN : Sams.net; ©1997
Available at Geisel Library Geisel Floor1 East, Books (QA76.76.H94 H764 1997)

Get It

Please sign in to check if there are any request options. Sign in

BACK TO LOCATIONS

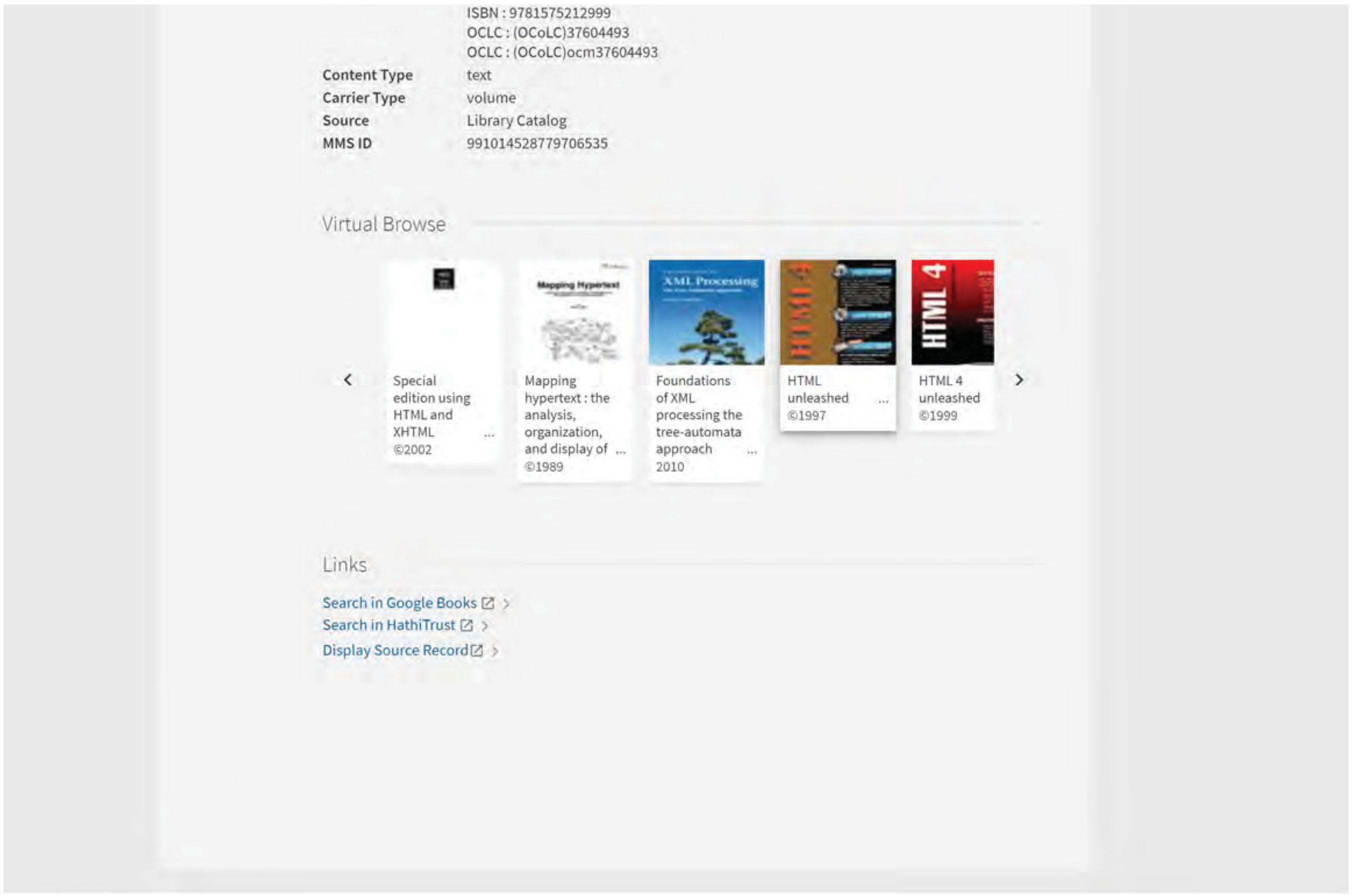
LOCATION ITEMS

Geisel Library
Available , Geisel Floor1 East, Books ; QA76.76.H94 H764 1997
(1 copy, 1 available, 0 requests)

Item in place Copy: 1
Loanable

Details

Title	HTML unleashed Rick Darnell [and others].
Creator	Darnell, Rick, 1966-
Edition	1st ed.
Publisher	Indianapolis, IN : Sams.net
Creation Date	©1997
Contents	CD-ROM contains all the source code and project files developed by the authors, plus an assortment of evaluation versions of third-party products. Includes HTML editors; graphics and multimedia; Microsoft Internet Explorer 3.
Other title	HTML 4 unleashed
Format	xxxvii, 1030 pages : illustrations ; 24 cm + 1 computer optical disc (4 3/4 in.)
Subject	HTML (Langage de balisage) Web HTML HTML (Document markup language) World Wide Web HTML (DOCUMENT MARKUP LANGUAGE) DOCUMENT MARKUP LANGUAGE HUMAN-COMPUTER INTERFACE INFORMATION TRANSFER COMPUTER PROGRAMS COMPUTER PROGRAMMING PROGRAMMING LANGUAGE WORLD WIDE WEB
Notes	Includes index. System requirements for disc: PC with Windows 3.x or higher; Macintosh.
Related titles	Available in other form: Online version: HTML unleashed. 1st ed. Indianapolis, IN : Sams.net, ©1997
Identifier	LC : 97065136 ISBN : 1575212994



	ISBN : 9781575212999 OCLC : (OCoLC)37604493 OCLC : (OCoLC)ocm37604493
Content Type	text
Carrier Type	volume
Source	Library Catalog
MMS ID	991014528779706535

Virtual Browse

Special edition using HTML and XHTML ... ©2002

Mapping hypertext : the analysis, organization, and display of ... ©1989

Foundations of XML processing the tree-automata approach ... 2010

HTML unleashed ... ©1997

HTML 4 unleashed ©1999

Links

Search in Google Books

Search in HathiTrust

Display Source Record

APPENDIX DARNELL02

HTML 4 UNLEASHED

Rick Darnell, et al.

TIMELY SOLUTIONS

■ HTML 4 ■ Dynamic HTML ■ Cascading Style Sheets ■ Netscape Communicator 4 ■ Internet Explorer 4 ■ JavaScript Style Sheets ■ HTML 3.2 Syntax ■ Frames ■ Tables ■ Forms ■ Hyperlinking ■ Layers ■ Internationalization ■ Extensible Markup Language (XML) ■ Image Maps ■ Interface and Navigation Design ■ CGI ■ Java and JavaScript ■ ActiveX and VBScript ■ Databases ■ Web Servers

EXPERT SOFTWARE

NetObjects™ Fusion 2 demo ■ Microsoft® Internet Explorer™ 3 ■ HotDog ■ BBEdit ■ HomeSite free ■ Cel Assembler ■ ActiveX Control Pad and HTML Layout Control ■ Web graphics ■ Examples and source code

BONUS!

ELECTRONIC LIBRARY

Two books fast-linked in HTML format:

- *Java 1.1 Unleashed, Third Edition*
- *Laura Lemay's Web Workshop: ActiveX and VBScript*

sams.net

THE COMPREHENSIVE SOLUTIONS PACKAGE!

HTML 4

Rick Darnell, et al.

sams.net

201 West 103rd Street
Indianapolis, IN 46290

UNLEASHED

International Standard Book Number: 1-57521-299-4

Library of Congress Catalog Card Number: 97-65136

2000 99 98 97 4 3 2

Interpretation of the printing code: the rightmost double-digit number is the year of the book's printing; the rightmost single-digit, the number of the book's printing. For example, a printing code of 96-1 shows that the first printing of the book occurred in 1996.

Composed in AGaramond and MCPdigital by Macmillan Computer Publishing

Printed in the United States of America

President, Sams Publishing	*Richard K. Swadley*
Publishing Manager	*Mark Taber*
Acquisitions Manager	*Beverly M. Eppink*
Director of Editorial Services	*Cindy Morrow*
Managing Editor	*Brice Gosnell*
Director of Marketing	*Kelli Spencer*
Product Marketing Manager	*Wendy Gilbride*
Assistant Marketing Managers	*Jen Pock* *Rachel Wolfe*

Acquisitions Editor
David B. Mayhew

Development Editor
Bob Correll

Software Development Specialist
Bob Correll

Production Editor
Ryan Rader

Copy Editors
Kristen Ivanetich
Carolyn Linn

Indexers
Christine Nelsen
Erika Millen
Bruce Clingaman

Technical Reviewers
Karen Clere
Blake Hall

Editorial Coordinators
Mandi Rowell
Katie Wise

Technical Edit Coordinator
Lorraine Schaffer

Resource Coordinator
Deborah Frisby

Editorial Assistants
Carol Ackerman
Andi Richter
Rhonda Tinch-Mize

Cover Designer
Jason Grisham

Cover Production
Aren Howell

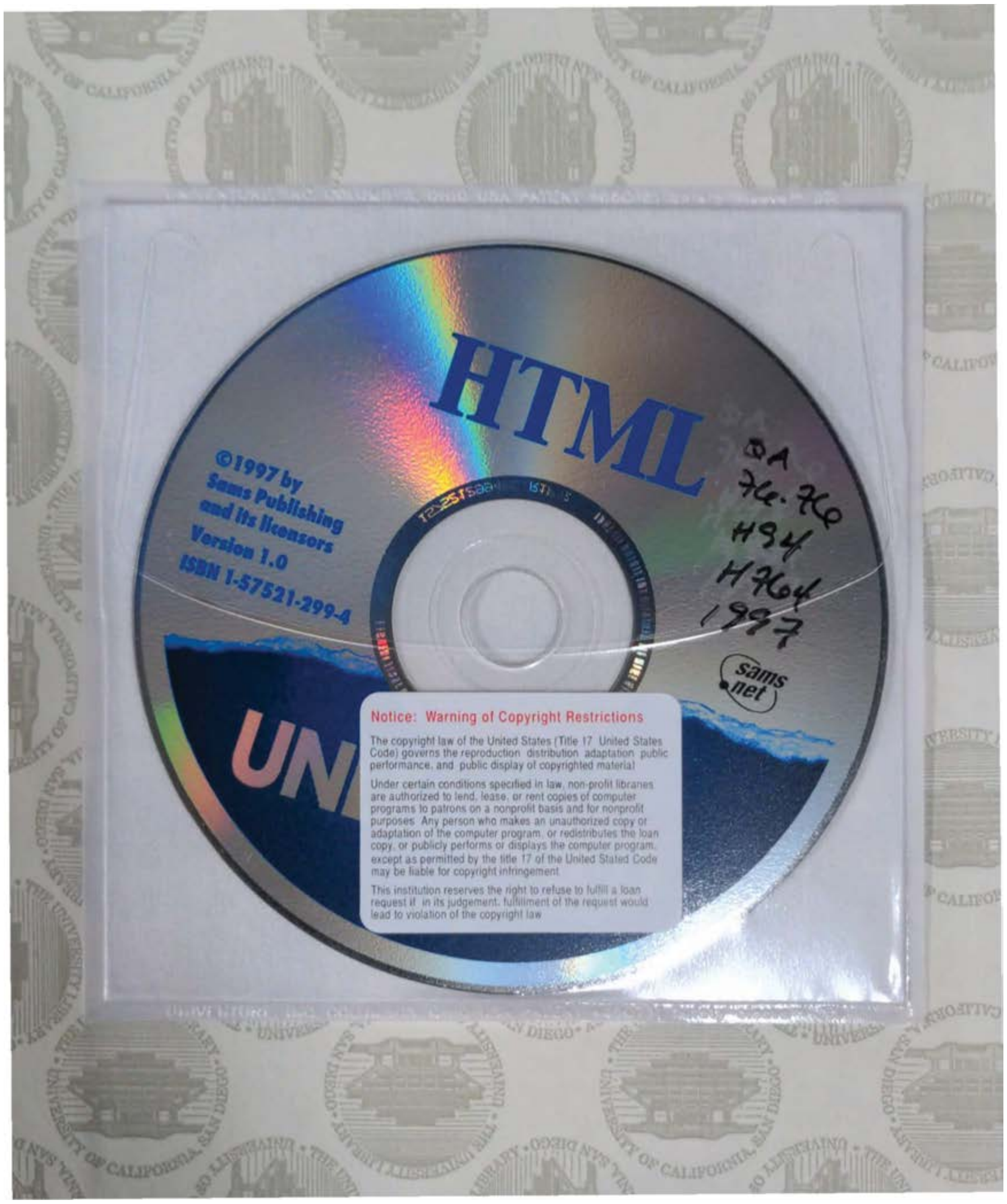
Book Designer
Gary Adair

Copy Writer
David Reichwein

Production Team Supervisor
Brad Chinn

Production
Mona Brown
Jennifer Dierdorff
Shawn Ring
Janet Seib

Overview


HTML
©1997 by Sams Publishing and its licensors
Version 1.0
ISBN 1-57521-299-4
QA 76.76 H94 H764 1997
sams.net
UN

TIMELY SOLUTIONS

- Learn the new features of the HTML 4 specification (Chapter 17)
- Add interactivity to your Web pages with Dynamic HTML (Chapter 22)
- Apply your own styles with Cascading Style Sheets (Chapters 19-20)
- Create sophisticated page layouts with frames and layers (Chapter 18)
- Enliven your pages with images and multimedia (Chapters 12 and 13)
- Build and use HTML forms to enable user input (Chapter 15)
- Learn advanced HTML design techniques (Chapters 23-26)
- Review the most capable Web development and site tools available (Chapters 34-37)
- Integrate Java™, JavaScript™, VBScript, ActiveX™ and CGI into HTML (Chapters 27-33)
- Discover Extensible Markup Language (XML) (Chapter 38)
- Learn the pros and cons of using pure HTML (Chapter 40)
- Learn how to interpret the HTML Document Type Definition (Chapter 3)

CATEGORY	➤ Internet/Web Publishing
COVERS	➤ HTML 3.2 and 4.0

EXPERT SOFTWARE

CD-ROM includes:

- **HTML Editors:** NetObjects Fusion™ 2 demo for Windows® and Fusion 1 demo for Macintosh®, Evaluation versions of Spider and Web Analyzer from InContext, HotDog32, Microsoft® ActiveX Control Pad and HTML Layout Control, HTMLed for Windows®, BBEdit 4.0 demo and BBEdit Light 4 for Macintosh®
- **Graphics and Multimedia:** PaintShop Pro, MapThis, Snagit32, Thumbs Plus, and Goldwave sound editor for Windows; WebMap, GIFConverter, Fast Player, Sparkle, and Sound App for Macintosh
- **Microsoft Internet Explorer™ 3** for Windows 3.1, 95, NT 3.51, and NT 4 and Macintosh; 164 Web graphics from The Rocket Shop, WinZip, ZipIt for Macintosh

BONUS!

ELECTRONIC LIBRARY

Two books fast-linked in HTML format:

- *Java 1.1 Unleashed, Third Edition*
- *Laura Lemay's Web Workshop: ActiveX and VBScript*

$49.99 USA / $70.95 CAN / £46.95 Net UK

7 52063 12994 4

sams net

www.samspublishing.com

APPROVED FOR TECHNICAL ACCURACY
Blake Hall & Karen Clere

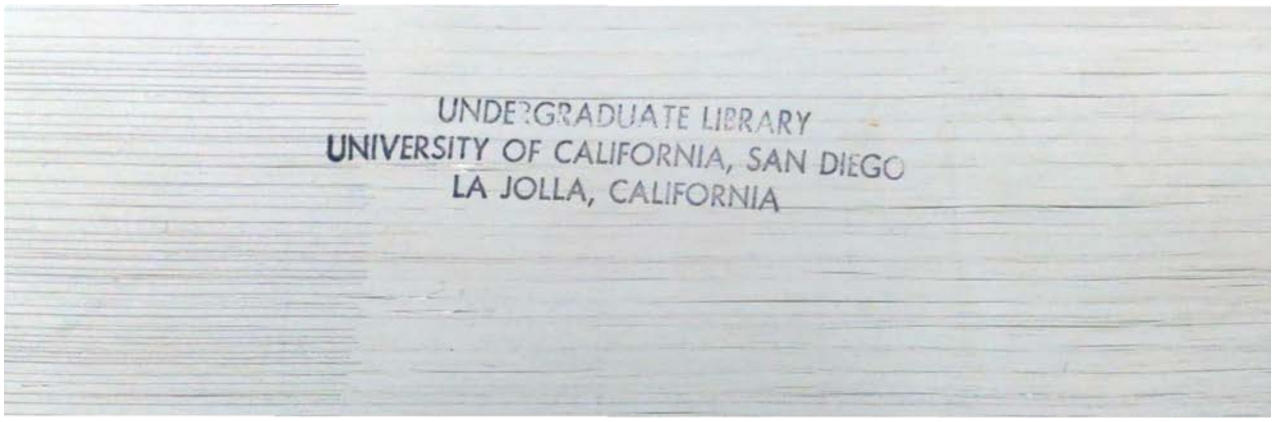
ISBN 1-57521-299-4

94999

9 781575 212999

HTML 4

UNLEASHE



CHAPTER 12

Adding Images to Your Web Page

by Rick Darnell

IN THIS CHAPTER

Once upon a time, before the World Wide Web got its name, there was a network that looked and acted like the Web, but it wasn't available to everyone. It was pretty limited in scope and appearance and was used by a bunch of guys with funny hats and whose hobbies included missile button design. These folks didn't care much for aesthetics and entertainment, so their Web didn't support images. Besides, they were still working in the days of yore when computer screens were green. Their browsers only displayed text, and there was no need or demand for presenting graphics across the network connection.

Some of the people with access to the early incarnations of the Web were at universities and other institutions of higher learning. As more and more people came into contact with the Web, it became more popular and more ideas surfaced about other uses and how it could look much nicer than it looked.

But all of the browsers in those early days were text-based. As the desire to share ideas grew, someone came up with the bright idea of including images as part of a Web page.

TIP

One of the most popular choices at the time was Lynx. It's still around today and in wide use among people who use nongraphics-based computer systems, and those who just prefer not to bother with all of the Web adornments such as images, applets, virtual reality, and so on.

Enter the National Center for Supercomputing Applications at the University of Illinois (NCSA). They put together a little program called Mosaic that supported the display of images right on the page with the rest of the text. Thus, the graphical browser was born, and this first work became the basis for virtually every other browser created since, including Netscape Navigator and Microsoft Internet Explorer.

NCSA added an additional tag called `<IMG>` to the fledgling HTML, which enabled early authors to insert a picture inline with the rest of the text. That tag is still around, and it has some brand new tricks, which I show you in this chapter. This chapter also includes some friends and relations of the tag that also relate to images. This feature has come a long way since its first inception, and you'll learn a multitude of ways to manipulate appearance and placement.

The Basic <IMG>

The `<IMG>` tag is an empty element used to insert inline images. This includes items such as small icons and graphics, in addition to large image maps that occupy most of the browser window. Because the tag is a single resource element (one tag for one image), an ending tag is not supported.

In addition to identifying which image to use, the tag also has various attributes for defining its position relative to the surrounding text and Web content. This includes floating the image in the left or right margin, or placing it on top of, below, or centered on the textline it appears on.

The syntax for an image tag is

```
<IMG src="[URL]filename" [alt="textDescription">
```

where the `src` attribute identifies the image *`filename`* through a physical or relative URL and filename. The `alt` attribute is used to define a brief text description of the image or its use for browsers that don't load the image.

TIP

Browsers don't load images for a variety of reasons. First, the browser could be a nongraphical browser such as Lynx. Second, the user might have image autoloading turned off to speed up download time. Providing a value for `alt` ensures that the user has some idea of what's supposed to be going on with that big blank space.

The attribute is vital for interoperability with speech-based and text-only user agents. For disabled persons, the `alt` value can provide a brief description of what the image is. For text-only browsers, it's the only indication that the user is missing any content.

The `src` attribute is required for every image. It identifies the specific image to use and its type. The two most popular image file types are GIF (Graphics Interchange Format) and JPEG (Joint Photographic Experts Group, also used as JPG), although PNG (Portable Network Graphic) images are starting to gain acceptance and wider usage.

The second attribute in the syntax definition, `alt`, is optional but recommended. It provides a textual description of the image and is the only portion of the tag used by browsers that don't support inline images.

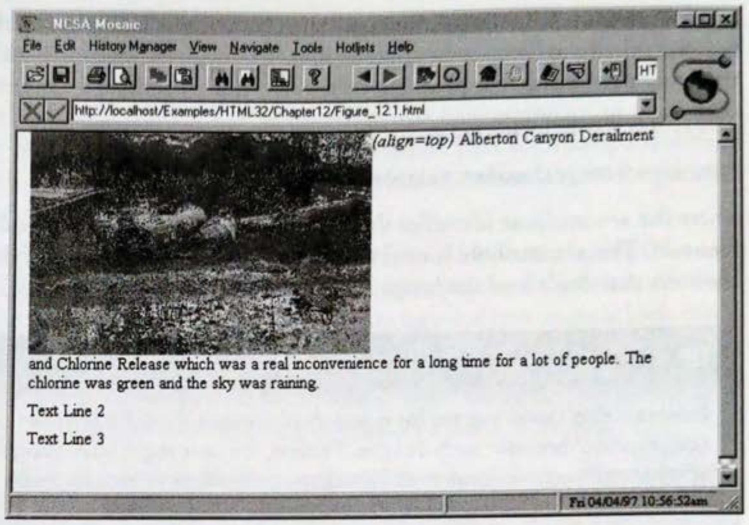
Where to `align` the Image

The `align` attribute controls how the image is positioned in relation to the line of text in which it occurs. Unlike other alignment attributes for items such as tables, `align` controls both the horizontal and vertical placement. Its syntax is

```
<IMG src="URL" align="position">
```

where *`URL`* includes the name of the file, and *`position`* is one of five values: `top`, `middle`, `bottom`, `left`, or `right`. The specific action of each value is as follows:

- `top` positions the top of the graphic with the top of the current line. (See Figure 12.1.) If the text line is formatted with a tag such as `<H1>`, the image will appear to occupy more space within the line itself than if it occurs within a line of standard body text.


NCSA Mosaic
File Edit History Manager View Navigate Tools Hotlists Help
http://localhost/Examples/HTML32/Chapter12/Figure_12.1.html
Fri 04/04/97 10:56:52am

(align=top) Alberton Canyon Derailment

and Chlorine Release which was a real inconvenience for a long time for a lot of people. The chlorine was green and the sky was raining.

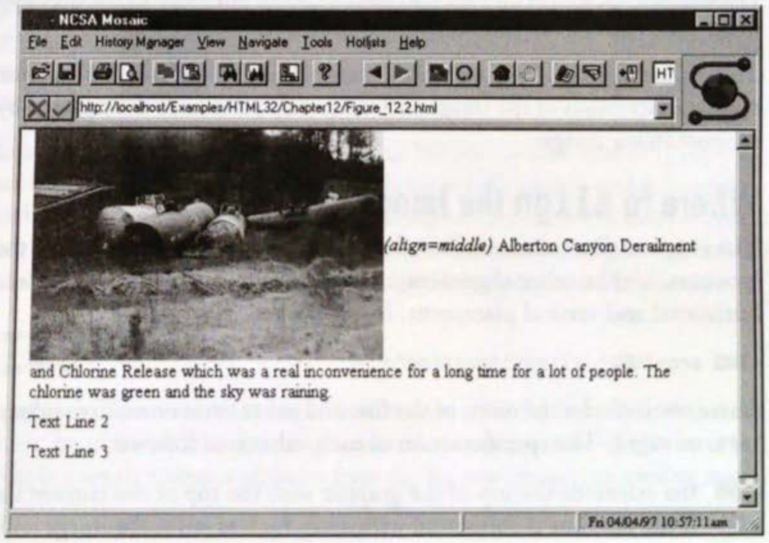
Text Line 2

Text Line 3

FIGURE 12.1. *Top alignment causes the top of the image to line up with the top of the current line. Note the position of the top border of the image in relation to the letters next to it.*

Browsers differ concerning whether the line immediately preceding or following is used to determine alignment.

- `middle` is similar to `top`, but the vertical midline of the image is aligned with the baseline of the current line. (See Figure 12.2.) Its interpretation is consistent across browsers.


NCSA Mosaic
File Edit History Manager View Navigate Tools Hotlists Help
http://localhost/Examples/HTML32/Chapter12/Figure_12.2.html
Fri 04/04/97 10:57:11am

(align=middle) Alberton Canyon Derailment

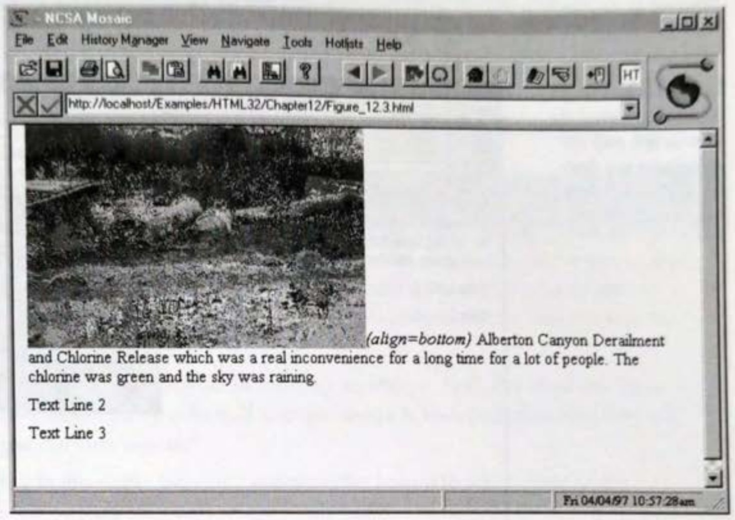
and Chlorine Release which was a real inconvenience for a long time for a lot of people. The chlorine was green and the sky was raining.

Text Line 2

Text Line 3

FIGURE 12.2. *Middle alignment matches the vertical halfway point of the image to the baseline of the current line of text.*

- `bottom` is the default value if `align` is omitted from the `<IMG>` tag. (See Figure 12.3.) In this case, the bottom of the image rests on the baseline for the current line.


- NCSA Mosaic
File Edit History Manager View Navigate Tools Hotlists Help
http://localhost/Examples/HTML32/Chapter12/Figure_12.3.html
Fri 04/04/97 10:57:28am

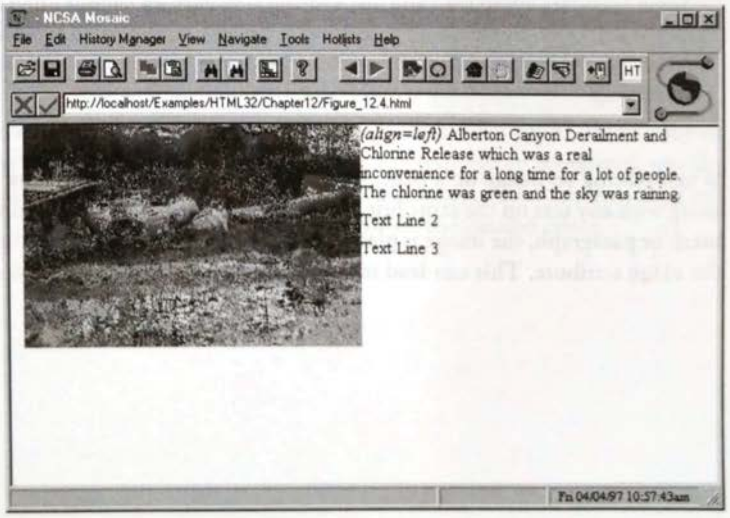
(align=bottom) Alberton Canyon Derailment and Chlorine Release which was a real inconvenience for a long time for a lot of people. The chlorine was green and the sky was raining.

Text Line 2

Text Line 3

FIGURE 12.3.
Bottom alignment is the default placement for images. The bottom of the image rests on the baseline for the current line of text.

- `left` forces the image to the current left margin, and any text following the image flows around the right margin of the image. (See Figure 12.4.) Its interpretation depends on whether any images or other material with `left` alignment appear earlier. Preceding text generally forces the image to wrap to a new line, with the subsequent text continuing on the line preceding the image.


- NCSA Mosaic
File Edit History Manager View Navigate Tools Hotlists Help
http://localhost/Examples/HTML32/Chapter12/Figure_12.4.html
Fri 04/04/97 10:57:43am

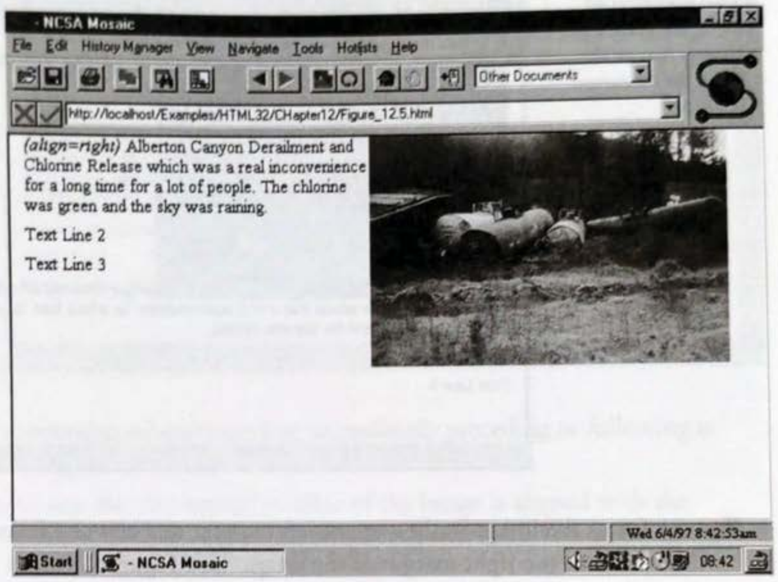
(align=left) Alberton Canyon Derailment and Chlorine Release which was a real inconvenience for a long time for a lot of people. The chlorine was green and the sky was raining.

Text Line 2

Text Line 3

FIGURE 12.4.
Left alignment results in the text following the `<IMG>` tag flowing around the right side of the image.

- `right` is similar to `left`, but the image is forced to the right margin. (See Figure 12.5.) Any following text is wrapped along the image's left side. It exhibits the same behavior as `left` in the opposite direction, depending on the alignment of preceding text and other material.


- NCSA Mosaic
File Edit History Manager View Navigate Tools Hotlists Help
Other Documents
http://localhost/Examples/HTML32/CHapter12/Figure_12.5.html
Wed 6/4/97 8:42:53am
Start
- NCSA Mosaic
08:42

(align=right) Alberton Canyon Derailment and Chlorine Release which was a real inconvenience for a long time for a lot of people. The chlorine was green and the sky was raining.

Text Line 2

Text Line 3

FIGURE 12.5.
Right alignment is the same as left, only the subsequent text flows around the left side of the image.

TIP

Some browsers introduce extra line spacing with multiple images using `left` or `right` alignment. Don't depend on the spacing to be uniform across all browsers and all platforms. For more information on controlling text flow, see Chapter 8, "Text Alignment and Formatting."

When placing an image on a page, remember that it is an inline feature that is displayed right along with any text on the same line. If it's not separated from surrounding material with a line break or paragraph, the image is placed on the same line as the current line of text according to the `align` attribute. This can lead to some rather undesirable appearances for images.

Making Space with `width` and `height`

The two size attributes set the desired horizontal and vertical space for the image in pixels. They are typically used as a pair to reserve space in the browser window before the image is loaded. The syntax is

```
<UMG src="URL" width=pixels height=pixels>
```

where *URL* is the name and *pixels* is the amount of space reserved for the image.

NOTE

For consistent and predictable results, images should be used at actual size; however, that's not always possible for a variety of reasons. In most browsers, the `width` and `height` attributes take precedence over the actual image size, allowing an easy way to force the image into a different-sized hole.

There are some drawbacks to this method of resizing an image. First, if a small image is enlarged, the overall image quality suffers. If a large image is reduced, download time is increased for a larger file than needed.

If the values specified in the `<IMG>` tag don't maintain the height to width ratio of the original, the image will appear distorted on the Web page.

There are a couple of reasons why you'll want to specify a size for your image. The first, already mentioned, is that it can speed display time for the rest of the page. When size information is omitted, some browsers set aside a minimal amount of space and then begin downloading the first bit of the image, which includes its size information. While the browser is working on that, it doesn't work on downloading any more of the body of the page.

Depending on the browser, the rest of the page might not display until it knows how much space each image needs. Or, the display might update and reload as each piece of information is acquired. Using the `width` and `height` attributes removes the guesswork for the browser.

The two attributes also preserve page formatting. If your page's overall appearance is dependent on the size and relation of the images to the text (like a newspaper or magazine), specifying the image size ensures that the proper amount of space is blocked. Although the image still won't appear, the right amount of space is held open to ensure the desired effect.

Would You Care for a border with That Image?

When an image appears as part of a hypertext link, the browser usually responds by drawing a colored border (usually blue) around the image. The width of this border is set using the `border` attribute. The syntax is

```
<IMG src="URL" border=pixels>
```

where *URL* is the path and name of the image file, and *pixels* is the width of the border in pixels. Use a value of `0` to hide the border. The color of the border is controlled by the link color attributes in the `<BODY>` tag. For more information, see Chapter 7, "Structural Elements and Their Usage."

TIP

Most browsers also indicate a hyperlink image by changing the mouse pointer when it passes over the graphic, so that the user doesn't have to guess whether the borderless image is a hyperlink.

Give the Image a Little hspace and vspace

The space attributes set up a *buffer zone* around the perimeter of the image. This is very useful when white space is needed immediately adjacent to the image. The `hspace` and `vspace` attributes set the width of this white space in pixels. The syntax is

```
<IMG src="URL" hspace=pixels vspace=pixels>
```

where *URL* is the image file and *pixels* is the number of pixels added to the appropriate side. By default, both are small nonzero numbers. This provides just enough white space to keep the image from touching adjacent text.

NOTE

White space is a design term meaning "space without anything in it." The space doesn't necessarily have to be white. Depending on the background color of the page, it could also be green, blue, pink, or any other color.

The space is added to both sides of the attribute. Therefore, if you include a value of `40` for `hspace`, 40 pixels of space will be added to the right and left sides of the image. This will make the image appear not in alignment with the other margins on the page.

ismap for Server-Side Maps

The `ismap` attribute identifies the image as an image map. Image maps enable you to associate specific areas of an image with hyperlinks to other documents.

In order to have validity, an `<IMG>` element with `ismap` is encased with a hypertext anchor tag. When the user clicks the image, the `ismap` attribute passes the x and y location of the click to the server. The syntax is

```
<a href="URL/file.map"><img src="imageURL" ismap></a>
```

where `URL/file.map` is the name of the MAP file with the coordinate information and `imageURL` is the name of the actual image. This is called a *server-side* image map because all of the hyperlink information is processed by the Web server.

TIP

The `ismap` attribute is used only with server-side image maps. In order to function, the server must have the appropriate CGI software to process the image map information and the file that defines each of the hot spots.

The location on the image where the user clicks is passed to the server by creating a new URL from the URL specified in the anchor tag. A question mark is added to the end, followed by the value of the x and y coordinates separated by a comma. The link is then followed using the new URL. For example, if the user clicks the location `x=10` and `y=27`, the URL sent to the server is `/cgibin/navbar.map?10,27`.

TIP

It is a good idea to use `border=0` with the image and use graphical clues to let the user know that the image is a clickable map. Otherwise, it's easy to confuse an image map with a one-shot hyperlinked image. Graphical clues include techniques such as bold shapes with descriptive text, or a long horizontal shape in the traditional navigation bar layout.

A recent feature added to browsers is the capability to interpret an image map without any help from the server. These are called client-side image maps, and they include all the necessary information to process their hyperlinks without further communication from the server. This type of image map is covered with the next attribute—`usemap`. More information on image maps is provided in Chapter 14, "Creating Image Maps."

usemap for Client-Side Image Maps

The `usemap` attribute is used to mark an image as a client-side image map that is used with a `<MAP>` element. In most cases, a client-side map is preferable to a server-side map because it requires no communication across network lines and no additional support from the server to operate. The syntax is

```
<IMG src="URL" usemap="[mapURL.html]#mapName">
```

where *URL* is the image file and *[mapURL.html]#mapName* is an incomplete URL specifying the location of the `<MAP>` element that has the coordinate and hyperlink information, and is identified similar to a target anchor. If the filename is omitted, the `<MAP>` element is assumed to be in the current HTML file.

A MAP FOR A FILE

Note the difference between the extension of the file specified for the hyperlink anchor on `ismap` and the source of the coordinates for `usemap`. The former requires a MAP file, which consists of a collection of coordinates and hyperlinks. The latter is a block of HTML tags within an HTML file. The file can contain other content, or it can exist solely for containing the map information.

The various active regions of the image map are described for the browser using `<MAP>` and `<AREA>` tags. This information is usually placed inside the HTML file with the `<IMG>` tag that requires it, although it can be placed in a separate file if the same map is used on several pages.

To ensure compatibility across browsers and platforms, `ismap` and `usemap` can be used on the same image. This allows the browser to choose which way it wants to interpret the image map, giving the document the maximum chance for compatibility.

Background Images

Although they are not directly related to the `<IMG>` tag, this seems like a good place to mention the use of background images for Web pages. These images are placed on a page through an attribute of the `<BODY>` tag. The syntax is

```
<BODY background="[URL]filename">
```

where *URL* is the location of the image specified with a complete URL including server name and path or a relative URL with or without a path, and *filename* is the name of the image to use. For more information on relative pathnames, see Chapter 11, "Linking Documents and Images."

TIP

For compatibility, limit the file type to GIF or JPEG. Image files in these formats are supported by virtually every graphical browser in use.

The image is placed in the background of the Web page, similar to hanging wallpaper in a bedroom. If the image is smaller than the space occupied by the browser window, it's tiled to fill the entire background. Use care to ensure that the image isn't so bold or distracting as to make the rest of the page unreadable. For more information on using `background` and other color attributes of `<BODY>`, see Chapter 7.

Using Images as Substitute Content

With the wide variety of browsers and capabilities on the Web, it's increasingly hard to make one page work for everyone. This is especially true when including content such as Java applets, ActiveX controls, and Netscape plug-ins. If a browser doesn't recognize your special content, it ignores it and either leaves a big hole or a blank space in your layout.

You can use browser tag incompatibility to your advantage by placing an image tag just before the closing tag of the specialized content. The format is

```
<APPLET attributes height=contentHeight width=contentWidth>
<PARAM attributes>
…other applet-specific lines…
<IMG src="URL" height=imageHeight width=imageWidth>
</APPLET>
```

where the applet tag is any of the specialized content container tags and the parameter tags are any of the tags subordinate to the container. As a matter of style and convention, the `<IMG>` tag is placed immediately preceding the closing container tag, and the content and image size attributes have the same value.

When this set of tags is encountered by a browser that doesn't support applets, the following is what happens:

- The opening container tag is not understood, so the browser ignores it and does nothing.
- Likewise, the subsequent tags that support the container tag are ignored. So far, the browser has done nothing but throw away the material that it doesn't understand.
- When the `<IMG>` tag is encountered, the browser suddenly has something it can work with. It interprets this tag and loads the image into the document in the space that was originally designed for the specialized content.
- The closing container tag is reached. Because it closes something the browser never understood, it is ignored, too.

This technique ensures that everyone who views your page has something to see, even if it's a large graphic that says, "You need a Java-compatible browser for this page." Another common usage is to use a screen capture from a video or VRML for the image. Even though the image is static, the user still gets a taste of what was intended.

Summary

Use of inline images in Web pages is part of what made the World Wide Web such a popular place to spend time. Use of the image tag depends on only one attribute, `src`, to identify the graphics file. Although plug-ins and different browsers support a wide variety of image formats, GIF and JPEG are the most popular and are supported in virtually every setting.

With the advent of HTML 3.2, additional capabilities within the `<IMG>` tag are officially extended to make it easier to control the appearance and behavior of graphics on the Web page. Using alignment and spacing attributes enables you to left- or right-justify an image while causing the surrounding text to wrap around the margins.

Mapping attributes set the graphics as an image map. Graphical image maps are one of the more popular uses for graphics files, including menu bars and full-screen images for site navigation.

Additional attributes of the `<IMG>` tag are covered in the next chapter, including those used to insert virtual reality worlds and aviation clips. Read on for more information on making your graphical Web page really graphical.

18

Creating Sophisticated Layouts with Frames and Layers

by John Jung

IN THIS CHAPTER

There are times when using the basic HTML tags that you've learned simply isn't enough to get what you want. Sometimes you really want your Web page to stand out and truly be different. Many people turn to Java and JavaScript in such situations. Although those are generally adequate solutions, they aren't the only ones. A number of HTML extensions that are being proposed for the next update to HTML are available to you. Two such concepts and extensions are frames and layers.

Frames

Frames are the implementation of a concept currently under discussion at the World Wide Web Consortium (W3C). With frames, you can separate Web pages into distinct sections. These sections can be any size within the browser window and, if allowed, can be resized. Also, the content in one frame can be independent of other frames if the Web author so desires. It's also possible to have the user's actions in one frame affect the content of other frames.

What Are Frames Good For?

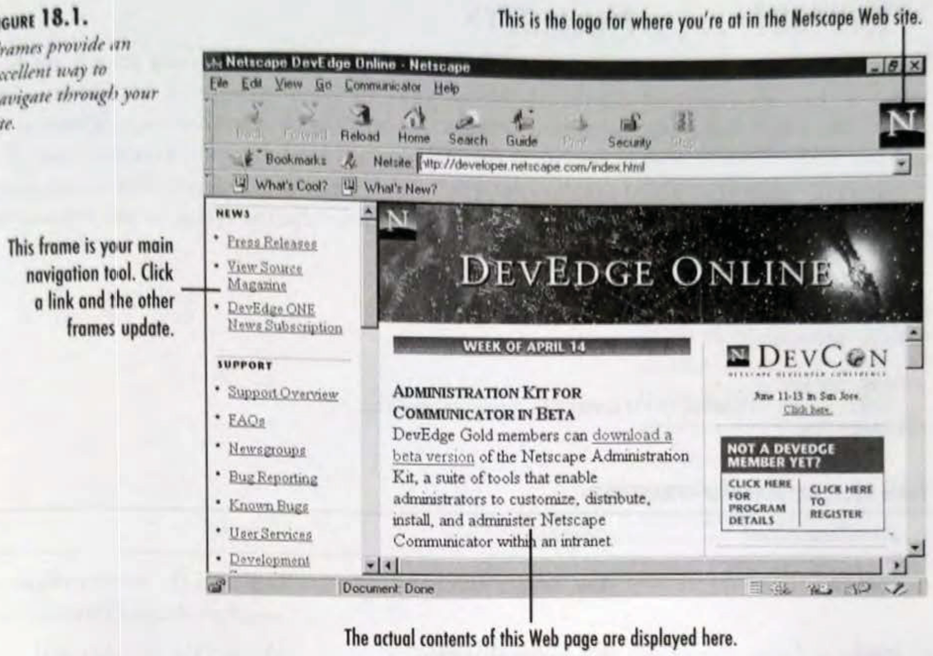
Because the contents in each frame are separate and distinctive of each other, you get better control over the information presented. For example, a commercial Web site could put the logos of its big advertisers in separate frames. Whatever the user does at such a site won't make the logos disappear. This way, you can provide free advertising to your big sponsors and not worry about someone missing it.

But frames give you far more functionality, rather than just control over what is presented. With the frames extension, you can change the content of other frames, which makes it possible for you to use frames as a navigation tool. You could display the main menu for your site in one frame and the contents in another. When the user makes a selection on the main menu, the contents frame will be updated. In fact, many such cool Web sites do precisely this (as shown in Figure 18.1).

Considerations of Frame Use

Frames can be a useful tool, but you shouldn't just rush out and start using them. You need to take some considerations into account before using frames. First, frames are a relatively new idea. As a result, some older browsers don't even support frames. For example, Lynx, the text-only Web browser, provides limited support for frames. As a result, if you use frames, many Lynx users will have problems getting around your site. Additionally, moderately older browsers have a poor user interface for navigating through frames. Netscape Navigator 2.0 was the first Web browser to support frames. Everything was fine, unless you wanted to go back a frame configuration or two.

FIGURE 18.1.
Frames provide an excellent way to navigate through your site.


This is the logo for where you're at in the Netscape Web site.
Netscape DevEdge Online - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Netsite: http://developer.netscape.com/index.html
What's Cool? What's New?
This frame is your main navigation tool. Click a link and the other frames update.
Document: Done
The actual contents of this Web page are displayed here.

NEWS
- Press Releases
- View Source Magazine
- DevEdge ONE News Subscription

SUPPORT
- Support Overview
- FAQs
- Newsgroups
- Bug Reporting
- Known Bugs
- User Services
- Development

DEVEDGE ONLINE

WEEK OF APRIL 14

ADMINISTRATION KIT FOR COMMUNICATOR IN BETA
DevEdge Gold members can download a beta version of the Netscape Administration Kit, a suite of tools that enable administrators to customize, distribute, install, and administer Netscape Communicator within an intranet.

DEVCON
June 11-13 in San Jose. Click here.

NOT A DEVEDGE MEMBER YET?
CLICK HERE FOR PROGRAM DETAILS | CLICK HERE TO REGISTER

Another consideration in using frames is whether they're really necessary. Although frames are a good tool, they aren't appropriate for all Web sites. If you're creating your own personal Web page, you can probably get away with using frames. After all, you want the Web page to impress only yourself and maybe your friends. You're not trying to ply any goods, so you're not really losing out on anything. If your company has a history of being a "hip" and "with-it" type of company, you could also use frames. If you're selling airplanes or power tools, however, you probably don't need them. So when should you use frames? Look at the following criteria. If you meet any of them, you should consider using frames:

- It's a personal Web page.
- Your company has a maverick/independent image.
- Your target audience will most likely be using the latest browser.

Syntax of Frame Creation

Like working with most other building blocks of HTML, creating frames isn't terribly difficult. In fact, I found tables a bit harder to create than frames. The frame extension is made up of three new HTML tags and one new attribute. The easiest of the three tags to understand is the `<NOFRAMES></NOFRAMES>` container. Basically, only Web browsers that don't support frames will display whatever is between those tags. You can use these tags to tell users what browser you prefer them to use so that they can view your frames.

<FRAMESET>...</FRAMESET>

The `<FRAMESET>...</FRAMESET>` container is the starting point for designing your frame layout. Use this container to specify the relative and absolute size of each frame. You can use only the `<FRAME>` tag and the `<NOFRAMES>` and `<FRAMESET>` containers within these tags. When creating any Web page with frames, you must first define a top-level frameset. You can create this top-level frameset only if you use the `<FRAMESET>` container instead of the `<BODY>` container. Table 18.1 details the attributes for the `<FRAMESET>` tag. Generally, the syntax for the `<FRAMESET>` `</FRAMESET>` group of tags is as follows:

```
<FRAMESET frame_configuration...>
<FRAME SRC=URL1...>
<FRAME SRC=URL2...>
...
<FRAME SRC=URLn...>
...Additional <FRAMESET></FRAMESET> tags as needed…
</FRAMESET>
```

Table 18.1. Attributes for <FRAMESET>.

Attribute Name	*Acceptable Values*	*Purpose*
BORDER	Any integer number	Indicates the border thickness for all child frames.
COLS	A number that may or may not have a percentage sign (%) or asterisk (*)	Defines the number and size of the columns you want to create. A number without a percentage sign indicates the size of the column in pixels. A number with a percentage sign indicates the width of the frame relative to the width of the browser. An asterisk indicates that the frame is sized proportionately to other frames. Specify multiple columns by quoting the size in a comma-separated list.
FRAMEBORDER	1 or 0	Specifies whether frames are displayed with a border. A value of 1 indicates the presence of a frame border and 0 disables the frame border. Individual frames can override this attribute.

Attribute Name	*Acceptable Values*	*Purpose*
ROWS	A number that may or may not have a percentage sign (%) or asterisk (*)	Defines the number and size of the rows you want to create. A number without a percentage sign indicates the size of the row in pixels. A number with a percentage sign indicates the height of the frame relative to the width of the browser. The asterisk indicates that the frame is sized proportionately to other frames. Specify multiple columns by quoting the size in a comma-separated list.

The <FRAME> Tag

After you specify a frameset, you must define the content of those frames, which is where the <FRAME> tag comes in. The <FRAME> tag controls various attributes for a particular frame, such as the content, color, and border width. For every row or column you define in the frameset, you must have a corresponding <FRAME>. The first frame, specified by either COLS or ROWS, will use the first <FRAME> tag (as shown in Listing 18.1). You can create complicated frame layouts using the <FRAMESET> container tags instead of the <FRAME> tag. (See the section "Creating Fancy Frame Layouts," later in this chapter.) Table 18.2 completely lists all the attributes that the <FRAME> tag uses.

Listing 18.1. Code to create a frame layout.

```
<HTML>
<HEAD>
<TITLE>Sample Frameset</TITLE>
</HEAD>
<FRAMESET ROWS="50%,50%">
  <FRAME SRC="agenda.html">
  <FRAME SRC="minutes.html">
</FRAMESET>
</HTML>
```

Table 18.2. Attributes for <FRAME>.

Attribute Name	*Acceptable Values*	*Purpose*
`FRAMEBORDER`	1 or 0	Specifies whether frames are displayed with a border. A value of 1 indicates the presence of a frame border and 0 disables the frame border. Individual frames can override this attribute.
`MARGINHEIGHT`	Any integer number	Indicates the height of the top and bottom margins in pixels.
`MARGINWIDTH`	Any integer number	Indicates the height of the left and right margins in pixels.
`NAME`	Any string beginning with a letter	Identifies the name of the frame for reference. There are four reserved names: `_blank`, `_parent`, `_self`, and `_top`. The `_blank` name opens a new window with the specified URL. The `_parent` name opens the URL in the parent frame of the current frame. If you call a `_self` frame, the URL replaces the frame the link was originally in. The `_top` name displays the URL in the full window.
`NORESIZE`	None	Simply acts as a toggle switch. When present, it prevents the user from resizing the frame.
`SCROLLING`	Yes, No, or Auto	Specifies whether a scrollbar appears in the frame. When set to Auto, the browser determines whether a scrollbar should be created. Auto is the default behavior.
`SRC`	Any URL	Specifies the file to be displayed within the frame. If you do not specify a SRC attribute, the space where the frame would appear will be blank.

The New Attribute

The final addition to HTML that the frames extensions made was to add a new extension. That extension, TARGET, was added to the Anchor element. This attribute takes a single string that specifies the name of the frame to display the contents. The `<FRAME>` tag, using the NAME attribute, defines the name of the frame (as shown in Listing 18.2). This enables you to change the contents of a frame by using hypertext links. For example, Figure 18.2 shows a framed Web page and the names for each. If you click a link in the left frame, named `left`, the right frame, named `right`, is updated appropriately. The HTML source for the left frame is shown in Listing 18.1. Frequently using the TARGET attribute updates the frame. The TARGET attribute isn't part of the `<FRAMESET>` or `<FRAME>` tag definition. Rather, it's an attribute for the `<A>` tag, used for navigation purposes. The TARGET attribute is entirely optional, and not using it would make each frame update independently. The last link in the left frame of Figure 18.2 updates the contents of its own frame.

NOTE

To create a new window, specify an invalid frame name.

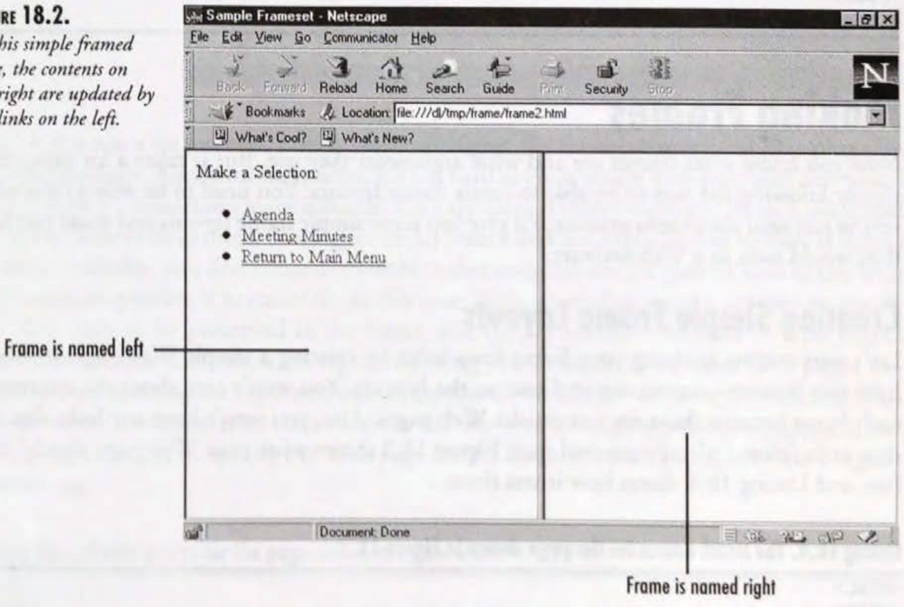

Sample Frameset - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: file:///dl/tmp/frame/frame2.html
What's Cool? What's New?
Make a Selection
Agenda
Meeting Minutes
Return to Main Menu
Document: Done
Frame is named left
Frame is named right

FIGURE 18.2.
In this simple framed page, the contents on the right are updated by the links on the left.

Listing 18.2. The HTML source for the entire Web page shown in Figure 18.2.

```
<HTML>
<HEAD>
<TITLE>Sample Frameset</TITLE>
</HEAD>
<FRAMESET COLS="50%,50%">
  <FRAME SRC="left.html">
  <FRAME SRC="right.html">
</FRAMESET>
</HTML>
```

Listing 18.3. HTML source for the frame `Left` shown in Figure 18.2.

```
<HTML>
<HEAD>
<TITLE>This title doesn't show up in the browser window</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
Make a Selection:
<UL>
<LI><A HREF="agenda.html" TARGET="right">Agenda</A></LI>
<LI><A HREF="minutes.html" TARGET="right">Meeting Minutes</A></LI>
<LI><A HREF="mainmenu.html" TARGET="_self">Return to Main Menu</A></LI>
</UL>
</BODY>
</HTML>
```

Making Frames

Now you know what frames are and what arguments they use. But it takes a lot more than simply knowing the tags to be able to create frame layouts. You need to be able to put what you've just read about into practice. I'll give you some simple frame layouts and show you how they would look in a Web browser.

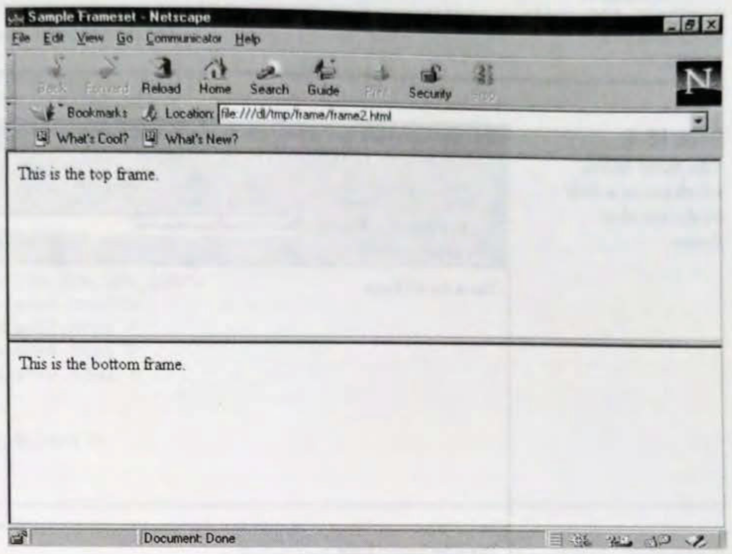
Creating Simple Frame Layouts

Let's start out on applying your frame knowledge by creating a simple Web page. It should have two frames—one on top and one on the bottom. You won't care about the content in each frame because those are just regular Web pages. Also, you won't have any links that update each other; I already covered that. Figure 18.3 shows what your Web page should look like, and Listing 18.4 shows how it was done.

Listing 18.4. The HTML source for the page shown in Figure 18.3.

```
<HTML>
<HEAD>
<TITLE>Two Frame Web Page</TITLE>
</HEAD>
```

```
<FRAMESET ROWS="50%,50%">
  <FRAME SRC="top.html">
  <FRAME SRC="bottom.html">
</FRAMESET>
</HTML>
```


Sample Frameset - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: file:///d/tmp/frame/frame2.html
What's Cool? What's New?
Document: Done

This is the top frame.

This is the bottom frame.

FIGURE 18.3.
This simple frame layout has only two frames.

Maybe that was a bit too easy. I think you could figure out the appropriate values to plug into the ROWS attribute. Also, with two frames, it's not hard to guess which Web page goes into which frame. Let's try something more difficult. Let's try three frames—two next to each other and the third under both of them. (See Figure 18.4.) Take a look at Listing 18.5 to see how it should be done. Basically, you first create a <FRAMESET> that encapsulates the general look of the Web page, such as splitting it horizontally in this case. Next, instead of using a <FRAME> to specify the Web page to be presented in the frame, you specify another <FRAMESET>. This nested <FRAMESET> further subdivides the region, allowing you to display even more Web pages. Because you are splitting the top frame region, the nested <FRAMESET> will be split vertically. This nested <FRAMESET> receives its contents from the two <FRAME> tags between the beginning and ending tags. Finally, to specify the Web page for the bottom frame, you simply use a regular <FRAME> tag.

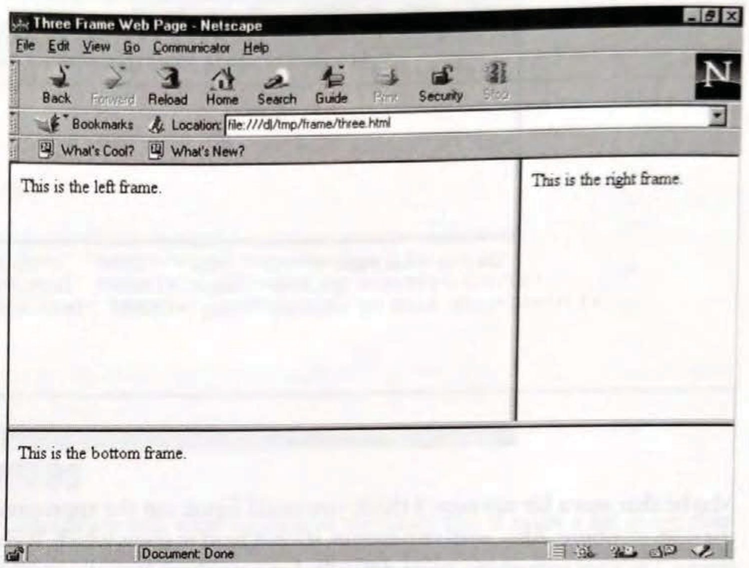
Listing 18.5. HTML source for the page shown in Figure 18.4.

```
<HTML>
<HEAD>
<TITLE>Three Frame Web Page</TITLE>
</HEAD>
```

continues

Listing 18.5. continued

```
<FRAMESET ROWS="70%,30%">
  <FRAMESET COLS="70%,30%">
    <FRAME SRC="left.html">
    <FRAME SRC="right.html">
  </FRAMESET>
  <FRAME SRC="bottom.html">
</FRAMESET>
</HTML>
```


Three Frame Web Page - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: file:///d|/tmp/frame/three.html
What's Cool? What's New?
Document: Done

This is the left frame.

This is the right frame.

This is the bottom frame.

FIGURE 18.4.
This frame layout, which can be a little tricky, uses three frames.

The important part to remember about this example is that you nested <FRAMESET> tags, which is how you were able to have the left and right frames. You now have the basics of creating nice, functional layouts. With these ideas, you can create truly advanced frame layouts if you think about how to lay out a page beforehand. The next section deals with creating sophisticated frame layouts.

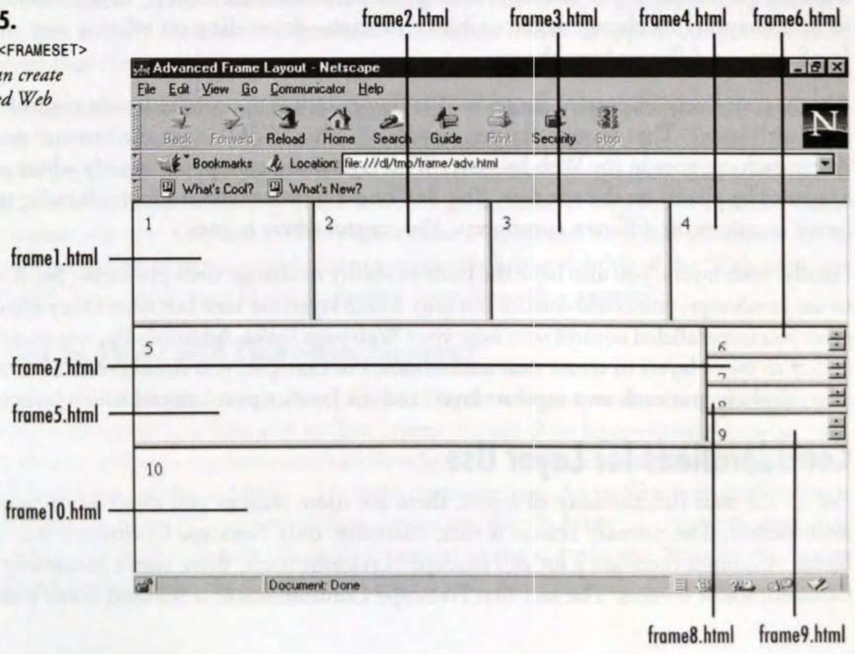
Creating Fancy Frame Layouts

It's quite possible, by using frames, to have really cool-looking frame layouts. Earlier, I briefly mentioned that you could nest <FRAMESET> tags. This procedure is the heart of making a truly spectacular Web page. Because <FRAMESET> uses the currently available window space, nesting enables you to further subdivide the window. For example, consider how Figure 18.5 was created. Or, you can cheat and look at Listing 18.6. Basically, you split the window into three horizontal frames, leaving the bottom frame alone. You subdivide the top frame into four nested frames, all of them next to each other. You then split the middle region into two nested frames, leaving the left frame region alone. Finally, you split the middle left frame region into four frames, one on top of each other.

Listing 18.6. HTML source for the frame layout in Figure 18.5.

```
<HTML>
<HEAD>
<TITLE>Advanced Frame Layout</TITLE>
</HEAD>
<FRAMESET ROWS="33%,33%,33%">
  <FRAMESET COLS="25%,25%,25%,25%">
    <FRAME SRC="blank1.html">
    <FRAME SRC="blank2.html">
    <FRAME SRC="blank3.html">
    <FRAME SRC="blank4.html">
  </FRAMESET>
  <FRAMESET COLS="80%,20%">
    <FRAME SRC="blank5.html">
    <FRAMESET ROWS="25%,25%,25%,25%">
      <FRAME SRC="blank6.html">
      <FRAME SRC="blank7.html">
      <FRAME SRC="blank8.html">
      <FRAME SRC="blank9.html">
    </FRAMESET>
  </FRAMESET>
  <FRAME SRC="blank10.html">
</FRAMESET>
</HTML>
```

FIGURE 18.5.
By nesting <FRAMESET> *tags, you can create sophisticated Web pages.*


frame2.html
frame3.html
frame4.html
frame6.html
Advanced Frame Layout - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: file:///d|/tmp/frame/adv.html
What's Cool? What's New?
1
2
3
4
frame1.html
5
6
7
8
9
frame7.html
frame5.html
10
frame10.html
Document: Done
frame8.html
frame9.html

Layers

Layers are a very new set of tools that help make neat-looking Web pages. Layers, created by Netscape, enable Web authors to define any number of *sub-Web pages*—pages whose content is independent of each other and the main Web page. They can show any URL, just like a Web page, or a frame, but also have much flexibility. Frames can be positioned, sized, and moved around with various JavaScript code. Further, you can put layers on top of each other, and have only certain ones showing their content. Hence, they're called "layers." When you use layers correctly, you can create some outstanding Web pages. Currently, only Netscape Navigator 4.0 will support the layers, although Internet Explorer 4.0 will probably do so as well.

Differences Between Frames and Layers

"Layers sound a lot like frames," I hear you saying. Not at all. Layers and frames are different in numerous respects. First, you can't simply have one frame; you must have at least two. That means that the frame layout takes up the entire Web browser window. Layers, on the other hand, are separate entities that take up only a certain amount of space. If the Web author desires, he or she can use a layer that takes up the entire Web browser window space. But perhaps more importantly, layers have two functions that frames simply lack.

First, you can control the visibility of a layer. In other words, you can show or hide any and all layers. This is quite different from frames, which simply must take up the entire Web browser window space. You cannot create a frame that takes up only a certain portion of the browser window. Furthermore, you can use JavaScript (as discussed in the section "Layers and JavaScript" in this chapter) to change layer visibility so that—depending on what a user does with JavaScript—a different layer shows.

A second and very important function that frames lack is the precise layout control that you have with layers. That is, with frames, as with all other HTML tags, the browser determines where each tag goes in the Web browser. With layers, you can specify exactly where you want a layer to be placed on the window. You don't have to worry about a layer showing up in different locations on different computers. *You* control where it goes.

Finally, with layers, you also have the built-in ability to change their positions. So, if you were to use JavaScript, you could control not only which layers are seen but where they appear. This gives you unparalleled control over how your Web page looks. Additionally, you could use this ability to move layers to create neat animations. For example, you could create a set of images of an airplane, put each on a separate layer, and use JavaScript to control which layer is visible.

Considerations for Layer Use

For all the neat functionality of layers, there are some reasons you don't want to use them immediately. The primary reason is that, currently, only Netscape Communicator supports layers. Although there are a lot of Netscape Navigator users, there aren't necessarily a lot of Communicator owners. The fact that Netscape Communicator is released doesn't mean that all Navigator users will rush out and use it. As a result, by using layers, you could be aiming for a rather small market. Almost all corporate sites should avoid layers until other browsers start to support them. You can probably get away with using layers in personal Web pages, but don't be surprised if you get some complaints. The primary reasons you would want to use layers are the same as those for wanting to use frames. If you meet any of the following criteria, you should probably use layers:

- You are creating a personal Web page.
- Your company has a maverick or independent image.
- Your target audience will most likely be using the latest browser.

Another consideration in using layers is that JavaScript handles most of its functionality. That means that along with further alienating some users, you should watch the content. It's too easy for developers to overload their JavaScript Web pages with graphics and sound. Layers makes it all the more tempting to put up really cool Web pages and not consider the user. Far too many Web pages today, without layers, have large Java applets and large images. It can take a considerable amount of time, even with intranet-level connections, to retrieve some of these pages. This situation can only get worse if people don't think about the consequences of using layers.

An additional consideration for the usage of layers is that the W3C hasn't agreed on its specifications. Although this isn't surprising for cutting-edge HTML code, the standards committee has also shelved discussion on layers, so it's possible that the W3C has completely lost interest in the idea of layers. Therefore, it's possible that the W3C will simply drop the proposed layer tags entirely. You have no guarantee that layers *will* be dropped, but you also have no guarantee that they won't.

Yet another consideration in using layers is how much work you want for yourself. If you intend to use layers extensively, you must remember that it will create a bit of work. Because layers can be anywhere on the screen, they are out-flow layers, which means that any HTML code after a layer will appear exactly where it would go if the layer tags weren't used. For example, suppose you give a title to a Web page, define a layer, and then add a bulleted list of items. The bulleted list of items would come immediately below the title of the Web page, not after the layer. You might want to use an in-flow layer for such a purpose.

What Are In-Flow and Out-Flow Layers?

In this section, I talk about how to create layers. The new layer specifications allow for two different types of layers: out-flow and in-flow layers. An out-flow layer—probably what you, like most people, will use—is positioned exactly where the Web author wants it to be. To create an out-flow layer, use the `<LAYER>...</LAYER>` container tags. An in-flow layer is offset from the last HTML tag or text. To create in-flow layers, use the `<ILAYER>...</ILAYER>` container tags. The `<ILAYER>` tag has exactly the same argument as the `<LAYER>` tag. Because the syntax for using both are the same, for simplicity's sake I'll talk about only out-flow layers.

Attributes and Methods of Layers

Layers are of a similar nature to frames in that they divide the Web browser window into distinct objects. You can't change frames dynamically. Therefore, when a user selects a particular link, you should have a frame change its size. Layers are simply the next progression from frames, allowing you to modify their attributes based on user events. In creating layers, you can set a number of possible attributes. In addition, each layer you create has a number of predefined methods associated with it. *Methods* is just another name for *functions* or *subroutines*, if you prefer. Neither layer attributes nor methods were created for frames.

Layer Attributes

You can set a certain number of attributes for each layer you create. These attributes are just like regular HTML tag attributes, with one big difference: For almost every attribute available, a corresponding layer property is available. I'll talk more about layer properties in the next section, "Layer Properties and Methods." Table 18.3 lists all the attributes for the `<LAYER>...</LAYER>` container tags. The general `<LAYER>` tag can be broken down as follows:

```
<LAYER layer_attributes>
...HTML code or plain text...
</LAYER>
```

Table 18.3. Attributes for <LAYER> and <ILAYER> container tags.

Attribute Name	*Acceptable Values*	*Purpose*
`ABOVE`	Any layer name	Indicates that the current layer is above the specified layer name.
`BACKGROUND`	Any URL	Specifies the background image to be displayed in the layer.
`BELOW`	Any layer name	Indicates that the current layer is below the specified layer name.
`BGCOLOR`	`#RRGGBB`	Identifies the background color for the layer. `RR`, `GG`, and `BB` are hexadecimal numbers that give the intensity of the red, green, and blue values of a color. You can specify only two digits for the hexadecimal numbers.
`CLIP`	`Integer,Integer,` `Integer,Integer`	Specifies the left, top, right, and bottom coordinates by four-integer numbers for the layer in pixels. Whatever is in the layer contained in the `CLIP` rectangle is displayed.

Attribute Name	*Acceptable Values*	*Purpose*
`LEFT`	Any integer value	Specifies the left edge of the layer in pixels from the left side of the browser window. For `<ILAYER>`, this attribute specifies the offset from the previous HTML tag or text.
`NAME`	Any string beginning with a letter	Identifies the layer for Java or JavaScript referential use.
`TOP`	Any integer value	Specifies the top edge of the layer in pixels from the top of the browser window. For `<ILAYER>`, this attribute specifies the offset from the previous HTML tag or text.
`VISIBILITY`	`SHOW`, `HIDE`, or `INHERIT`	Determines whether the layer is visible. The `INHERIT` value causes the layer to take the visibility attribute of its parent.
`WIDTH`	Any integer value	Indicates the width of the layer in pixels.
`ZINDEX`	Any integer value	Indicates the index value for the layer for stacking and displaying.

Layer Properties and Methods

Because Netscape intended layers to enable you to create dynamic Web pages, layers have a few extra features. All of these features provide hooks for scripting languages into accessing aspects of the layer. The script can change values of some properties or call some layer methods. Table 18.4 shows all the properties for all layers. Table 18.5 gives you a list of methods available to all layers.

Table 18.4. The <LAYER> tag properties.

Attribute Name	*Can Be Changed?*	*Acceptable Values*	*Description*
`above`	No	Any layer name	Specifies the parent layer of the current layer for nested layers.
`background`	Yes	Any URL	Specifies the background image to be displayed in the layer.

continues

Table 18.4. continued

Attribute Name	*Can Be Changed?*	*Acceptable Values*	*Description*
`below`	No	Any layer name	Specifies the child layer of the current layer for nested layers.
`clip.top`, `clip.left`, `clip.right`, `clip.bottom`, `clip.width`, `clip.height`	Yes	All properties accept any integer value	Control the various aspects of the clipping rectangle.
`bgColor`	Yes	`#RRGGBB`	Identifies the background color for the layer. `RR`, `GG`, and `BB` are hexadecimal numbers that give the intensity of the red, green, and blue values of a color. You can specify only two digits for the hexadecimal numbers.
`height`	No	Any integer value	Indicates the height of the layer in pixels.
`layers`	No	An array	Stores the index and name for each defined layer.
`left`	Yes	Any integer value	Specifies the number of pixels, from the left edge of the browser window, where the layer should start.
`name`	No	Any string beginning with a letter	Identifies the layer for Java or JavaScript referential use.
`sibling.Above`	No	Any layer name	Indicates that the current layer is above the specified layer name. A null value indicates the top-level layer.

Attribute Name	*Can Be Changed?*	*Acceptable Values*	*Description*
`sibling.Below`	No	Any layer name	Indicates that the current layer is below the specified layer name. A null value indicates the bottom-most layer.
`top`	Yes	Any integer value	Specifies the number of pixels from the top of the browser window, where the layer starts.
`visibility`	Yes	`show`, `hide`, or `inherit`	Determines whether the layer is visible. The `INHERIT` value causes the layer to take the visibility attribute of its parent.
`width`	No	Any integer value	Indicates the width of the layer in pixels.
`zINDEX`	Yes	Any integer value	Indicates the index value for the layer for stacking and displaying. Layers with a higher `zINDEX` value will be stacked above lower `zINDEX` numbered layers.

Table 18.5. The <LAYER> tag methods.

Method Name	*Parameters*	*Parameter Data Type*	*Description*
`offset`	`(x,y)`	`integer, integer`	Causes the layer to be offset on the left by the specified x value. The top is similarly offset by the specified y value.
`moveAbove`	`(layer_name)`	`string`	Adjusts the stacking array to place the current layer above the specified layer. Both layers will have the same parent.

continues

Table 18.5. continued

Method Name	*Parameters*	*Parameter Data Type*	*Description*
`moveBelow`	`(layer_name)`	`string`	Adjusts the stacking array to place the current layer below the specified layer.
`moveTo`	`(x,y)`	`integer, integer`	Places a layer at a precise screen location. You specify the exact position at which you want the upper-left corner of the layer to be placed.
`resize`	`(width, height)`	`integer, integer`	Resizes the current layer. The first value is the desired width of the layer and the second is the desired height.

Layers and Languages

One nicer aspect of layers is that it has built-in support for languages. You can use just about any scripting language, such as JavaScript, to expand the capabilities of layers. You can dynamically turn layers on or off when the user moves his or her mouse. You can exploit the features of both the language and layers. The first part in accessing the features of layers is to be able to look at and change its properties. You can do this by using the following syntax:

```
document.layername.property
```

Here, `document` is the required string, but `layername` and `property` are variables. The `layername` represents the name that you've given to the layer, and `property` is the property of the layer that you want to access. (See Table 18.4.) Similarly, you can access the methods of a particular layer with the following syntax:

```
document.layername.method(params)
```

Once again, `document` is a required string, and `layername` is the name of the layer. This time, `method` is the name of the method you want to access, and `params` are the parameters you're passing. Table 18.5 completely lists all the methods and their parameters built into all layers. I'll talk more about using properties and methods later in this chapter.

Creating Layers

So now that you know the technical syntax of creating layers, it should be a snap to create them, right? Well, not always. Sometimes, it helps to try out what you know before you put it into your Web page. As a result, I'll go through a couple of examples of using layers. I'll show you a Web page with some layers and then show you how it was done.

Basic Layer Use

In some ways, creating layers is much easier than creating frames. With layers, you can specify the exact size and position you want the layer to be. Also, all HTML code between the `<LAYER>...</LAYER>` container is processed as usual. So, let's start by trying to create three layers, each next to each other. Figure 18.6 shows you the look to strive for, and it was created with the code in Listing 18.7. In this example, you use three out-flow layers and simply specify their background color. Out-flow layers are used because they give you very precise control over where the layers are placed. In this example, the layers are simply positioned next to each other, so that it is easier to see all of them.

Listing 18.7. HTML source for the page shown in Figure 18.6.

```
<HTML>
<HEAD><TITLE>Sample of Three Layers</TITLE></HEAD>
<BODY BGCOLOR="#FFFFFF">
<H1><CENTER>Three Layers Next to Each Other</CENTER></H1>
<LAYER NAME="red" LEFT=50 TOP=50 WIDTH=50 HEIGHT=50 VISIBILITY=SHOW
  BGCOLOR="#FF0000">
</LAYER>
<LAYER NAME="green" LEFT=100 TOP=50 WIDTH=50 HEIGHT=50 VISIBILITY=SHOW
  BGCOLOR="#00FF00">
</LAYER>
<LAYER NAME="blue" LEFT=150 TOP=50 WIDTH=50 HEIGHT=50 VISIBILITY=SHOW
  BGCOLOR="#0000FF">
</LAYER>
</BODY>
</HTML>
```

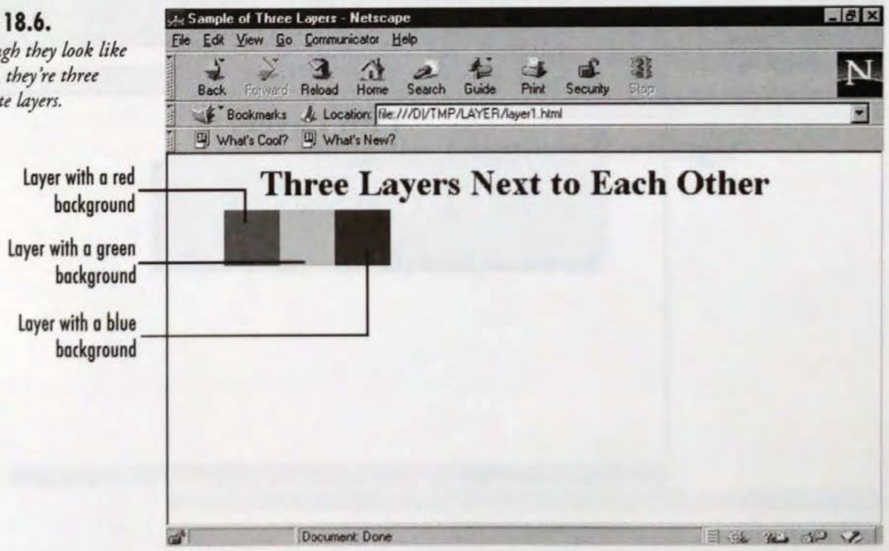

Sample of Three Layers - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: file:///D|/TMP/LAYER/layer1.html
What's Cool? What's New?
Three Layers Next to Each Other
Document: Done
Layer with a red background
Layer with a green background
Layer with a blue background

FIGURE 18.6.
Although they look like blocks, they're three separate layers.

That wasn't terribly hard, was it? Let's go on to something a little bit trickier—stacking the order of the layers. This is an important exercise to understand because you can use it to script. You could easily have multiple layers on top of each other and simply choose which one you want on top. Figure 18.7 shows two layers, one slightly offset from the other, which create a drop-shadow effect. Listing 18.8 shows you how it was done. In this example, you first create a layer that will act as the shadow of the second layer. Because it's a shadow, use a gray color as the background, although black would work as well. Next, create an out-flow layer for the layer that's getting the shadow, and position it up and to the left from its shadow. This makes only a small part of the shadow layer appear from underneath the green layer.

Listing 18.8. HTML source for the page shown in Figure 18.7.

```
<HTML>
<HEAD><TITLE>Sample of Two Layers</TITLE></HEAD>
<BODY BGCOLOR="#FFFFFF">
<H1><CENTER>Two Layers</CENTER></H1>
<LAYER NAME="grey" LEFT=60 TOP=60 WIDTH=400 HEIGHT=100 VISIBILITY=SHOW
  BGCOLOR="#888888" zINDEX=3>
</LAYER>
<LAYER NAME="green" LEFT=50 TOP=50 WIDTH=400 HEIGHT=100 VISIBILITY=SHOW
  BGCOLOR="#00FF00" zINDEX=1>
  <CENTER>This is the green layer</CENTER>
</LAYER>
</BODY>
</HTML>
```

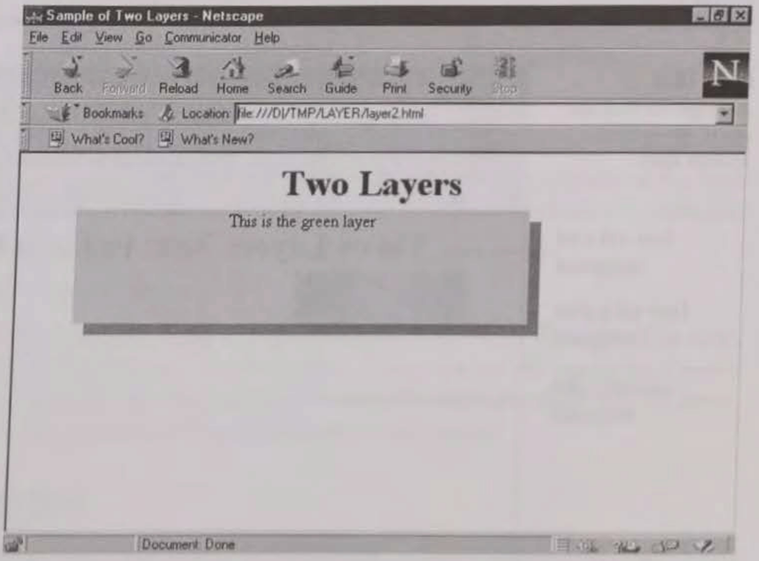

Sample of Two Layers - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: file:///D|/TMP/LAYER/layer2.html
What's Cool? What's New?
Two Layers
This is the green layer
Document: Done

Figure 18.7.
By controlling the stacking order of the layers, you can create nice special effects.

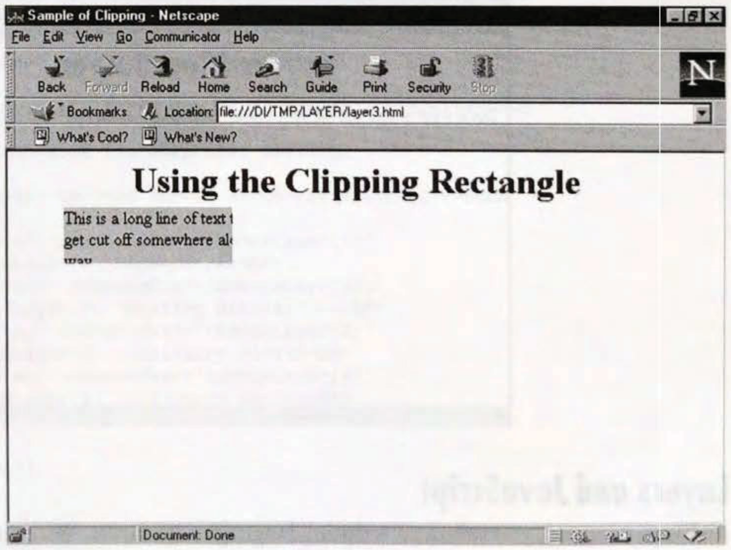
This might seem rather simple and straightforward—because it is. Layers aren't that hard to understand or to use. Their syntax is similar in nature to that of frames. As a result, if you know how to manage frames well, you should be able to manage layers pretty well, too.

Layers and Clipping Rectangles

Another useful aspect of layers is the ability to define clipping rectangles. A clipping rectangle is a specified region in which only contents in that region are displayed. You could have a layer with lots of HTML tags but only display part of it. When the user makes a particular selection, you could simply move the clipping rectangle. Take a look at Figure 18.8 and notice that some of the text is cut off. That's the clipping rectangle in action, even though you don't see a border around it. You can see how I created that figure by looking at Listing 18.9.

Listing 18.9. HTML source for the page shown in Figure 18.8.

```
<HTML>
<HEAD><TITLE>Sample of Clipping</TITLE></HEAD>
<BODY BGCOLOR="#FFFFFF">
<H1><CENTER>Using the Clipping Rectangle</CENTER></H1>
<LAYER NAME="green" LEFT=50 TOP=50 WIDTH=200 HEIGHT=200 VISIBILITY=SHOW
  CLIP="0,0,150,50" BGCOLOR="#00FF00">
This is a long line of text that <B>will</B> get cut off somewhere along the
 way.
</LAYER>
</BODY>
</HTML>
```


Sample of Clipping - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: file:///D|/TMP/LAYER/layer3.html
What's Cool? What's New?
Document: Done

Using the Clipping Rectangle

This is a long line of text t
get cut off somewhere al
way

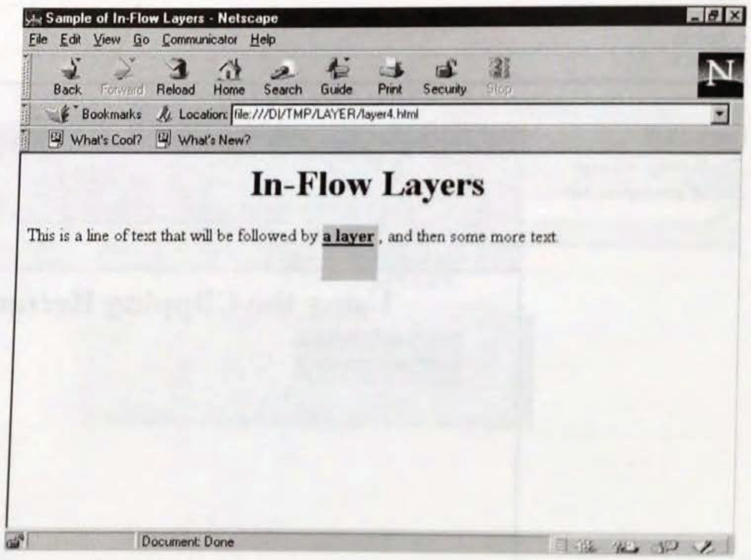
FIGURE 18.8.
The clipping rectangle cuts off some of the text.

In-Flow Layers

So far, I've shown you how to use only out-flow layers. What about the layers that simply start off where the last bit of HTML code ends? These in-flow layers are equally easy to manage and create. Look at Figure 18.9 and you'll notice that some of the text appears to be on a colored block. That green block is actually an in-flow layer. Notice that the text before and after the layer continues as usual, with no break-up. Listing 18.10 shows you how the in-flow layer was created.

Listing 18.10. HTML source for the page shown in Figure 18.9.

```
<HTML>
<HEAD><TITLE>Sample of In-Flow Layers</TITLE></HEAD>
<BODY BGCOLOR="#FFFFFF">
<H1><CENTER>In-Flow Layers</CENTER></H1>
This is a line of text that will be followed by
<ILAYER VISIBILITY=SHOW WIDTH=50 HEIGHT=50 BGCOLOR="#00FF00">
<B><U>a layer</U></B>
</ILAYER>
, and then some more text.
</BODY>
</HTML>
```


Sample of In-Flow Layers - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: file:///D|/TMP/LAYER/layer4.html
What's Cool? What's New?
Document: Done

In-Flow Layers

This is a line of text that will be followed by **a layer** , and then some more text.

FIGURE 18.9.
Some of this text is actually part of an in-flow layer.

Layers and JavaScript

As I've mentioned, you can use a scripting language with layers. With this combination, you can create an extremely impressive looking Web page. It makes it possible to have floating help that describes what each link does. Also, you can provide a great deal more interaction with the user than regular HTML allows.

Figure 18.10 gives an example of a Web page with links on the left side. When the user moves his or her mouse over each link, the help layer on the right changes. The right layer gives specific information for the purpose of the link on the left. Listing 18.11 shows you the entire Web page that I used to accomplish this feat. This trick is accomplished by using a separate layer for each link. When the cursor is over a particular link, all layers are hidden except for the appropriate one. Also, when the mouse is moved off the link, the generic description layer is displayed.

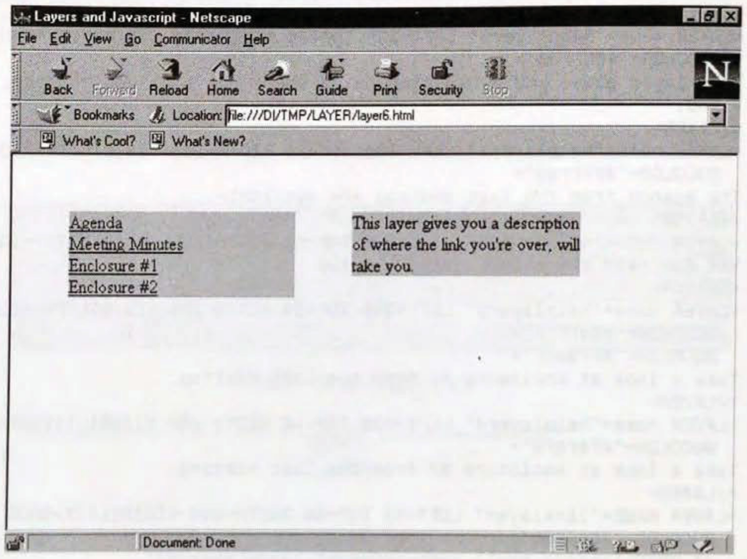
Listing 18.11. HTML source for the page shown in Figure 18.10.

```
<HTML>
<HEAD><TITLE>Layers and JavaScript</TITLE></HEAD>
<BODY BGCOLOR="#FFFFFF">
<LAYER name="helplayer0" LEFT=300 TOP=50 WIDTH=200 VISIBILITY=SHOW
  BGCOLOR="#00FF00">
This layer gives you a description of where the link you're over, will take
  you.
</LAYER>
<LAYER name="helplayer1" LEFT=300 TOP=50 WIDTH=200 VISIBILITY=HIDE
  BGCOLOR="#FFFF00">
The agenda from the last meeting are available.
</LAYER>
<LAYER name="helplayer2" LEFT=300 TOP=50 WIDTH=200 VISIBILITY=HIDE
You can read the last meeting minutes.
</LAYER>
<LAYER name="helplayer3" LEFT=300 TOP=50 WIDTH=200 VISIBILITY=HIDE
  BGCOLOR="#00FFFF">
  BGCOLOR="#FF00FF">
Take a look at enclosure #1 from the last meeting.
</LAYER>
<LAYER name="helplayer4" LEFT=300 TOP=50 WIDTH=200 VISIBILITY=HIDE
  BGCOLOR="#F0F0F0">
Take a look at enclosure #2 from the last meeting.
</LAYER>
<LAYER NAME="linklayer" LEFT=50 TOP=50 WIDTH=200 VISIBILITY=SHOW
  BGCOLOR="#00FF00">
  <A HREF="agenda.html" onMouseOver="changeLayer(1)"
  onMouseOut="changeLayer(0)">Agenda</A><BR>
  <A HREF="minutes.html" onMouseOver="changeLayer(2)"
  onMouseOut="changeLayer(0)">Meeting Minutes</A><BR>
  <A HREF="encl01.html" onMouseOver="changeLayer(3)"
  onMouseOut="changeLayer(0)">Enclosure #1</A><BR>
  <A HREF="encl02.html" onMouseOver="changeLayer(4)"
  onMouseOut="changeLayer(0)">Enclosure #2</A><BR>
</LAYER>
<SCRIPT>
function hideLayers ()
{
```

continues

Listing 18.11. continued

```
  document.layers["helplayer0"].visibility = "hide";
  document.layers["helplayer1"].visibility = "hide";
  document.layers["helplayer2"].visibility = "hide";
  document.layers["helplayer3"].visibility = "hide";
  document.layers["helplayer4"].visibility = "hide";
}
function changeLayer (n)
{
  hideLayers();
  document.layers["helplayer" + n].visibility = "inherit";
}
</SCRIPT>
</BODY>
</HTML>
```


Layers and Javascript - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: file:///D|/TMP/LAYER/layer6.html
What's Cool? What's New?
Agenda
Meeting Minutes
Enclosure #1
Enclosure #2
Document: Done

This layer gives you a description of where the link you're over, will take you.

FIGURE 18.10.
The text on the left is contained within its own layer.

The key parts of this listing are the `onMouseOver` and `onMouseOut` attributes in the anchor elements. Netscape proposed these extensions to provide a better mechanism for event handling. You'll also notice that all I'm really doing is hiding all the layers and then revealing the correct layer. Therefore, when the user positions the mouse over a particular link in the left layer, the corresponding right layer is exposed. Furthermore, a "default" generic layer explains how to get more help on each link. So, whenever the user moves off of any link in the left layer, the generic layer is displayed. With this approach, you don't need to keep track of which layer is currently visible.

Obviously, this is just the tip of the iceberg for the functionality of layers and JavaScript. You really modified only one attribute—the `visibility` attribute—and used JavaScript code to modify the values. It's possible to expand this Web page to do much more. You could simulate nested menus with floating help simply by adding the `moveTo()` or `offset()` methods. Or, you could have a layer on the screen and, depending on the user input, resize it to show more information. You could even create a simple game just by messing around with layers, their visibility, and their positions.

Summary

In this chapter, I covered some of the cooler HTML extensions that Netscape proposed. I talked about frames and layers and the pluses and minuses of each. Also, I gave you a comprehensive list of the attributes and syntax for creating such elements. I provided you with some examples on how to create sample Web pages with these elements. You can take these examples and expand on them, based on what you learned here.

What's Next?

The next chapter focuses on a new development created for HTML: cascading style sheets. Style sheets are a way of providing a uniform look and feel to all your Web pages. For corporate Web sites, this takes a lot of the work needed to make a site look consistent. But cascading style sheets also give you some extra functionality that you might not have thought about.

CHAPTER 38

The Emergence of Extensible Markup Language

by Dmitry Kirsanov

IN THIS CHAPTER

Everyone interested in the future of the Web must be curious—and pretty uncertain—about what may be the outcome of the HTML case. Browser wars and incompatible extensions tearing apart the language are not only bad by themselves, they're a sure sign that something's going wrong with HTML in the first place. It may sound like heresy that the tongue spoken by millions of Web pages is approaching the end of its useful life, but many serious observers cannot suppress exactly that feeling.

If we strip away for a moment the innumerable struts, crutches, and sophisticated gizmos that make the HTML golem walk and speak and look alive, what we'll see will be a pretty simple (not to say primitive) markup language designed for basic documents of a quite predictable structure. Just headers, paragraphs, block quotations, and the good old ADDRESS at the end. Does *this* sound like a model structure for the whole world of information out there?

Of course, now HTML has tables and fonts and so on. Indeed, visual HTML extensions (or inline images, as the last resort) enable you to emulate any document structure—that is, make the document *look* as if it is properly structured. But as a result, the internal structure of the text will inevitably become illogical, cumbersome, presentation-oriented (and with images, the text may cease to be text at all). This is very likely to prevent reusing the document in the future; it becomes difficult even to convert it into another visually oriented format, not to mention isolate its logical elements.

Fortunately, there's a new important language designed to address this issue. XML (eXtensible Markup Language) is a simple and compact subset of SGML designed specifically for use on the Internet in the way that HTML is currently used. This new project of W3C is gaining momentum at a surprising rate, and everybody seriously concerned with HTML may want to check it out. Maybe someday, you'll find yourself saying, "back in those days when everyone was using HTML..."

The Premises of XML

As with every new and promising technology, it is probably more important to explain what it isn't rather than what it is. XML, just like SGML, is not a page layout or graphics language. By itself, XML provides even fewer presentation tools than you have with HTML. Strictly speaking, it's not even a markup language, but rather a system making it possible to build such languages to match any conceivable document type.

Chapter 3, "SGML and the HTML DTD," explains the HTML document type definition (DTD), the specification of HTML tags and document structure written in SGML. Similarly, with XML you can build a DTD that exactly matches the structure of your document and introduces a set of self-explanatory, logically organized tags and attributes fine-tuned for your markup needs.

By attaching the DTD with the document sent over the network, you can ensure that the XML software reading the document can parse it correctly and thereby guarantee its correct formatting, conversion, adding to a database, or whatever the receiver will choose to do with the document. In short, with XML you can create your own HTML, or XYZML, or Whatever-You-Like-ML! (It's no surprise such a language was called "extensible" in the first place.)

It is important to understand that XML isn't better than HTML because it makes it easier to change fonts or position images. In the visual presentation realm, XML is not better than HTML (and some might say it's worse because it lacks all those neat Netscape enhancements—unless you've defined them in your DTD). It was the intention of the creators of the language that the visual presentation of an XML document can be (optionally) specified by an attached style sheet, which is an *external* mechanism for XML just as it is for HTML.

XML's visualization power is thus completely determined by the style sheet language you use—for example, Cascading Style Sheets (CSS) or Document Style Semantics and Specification Language (DSSSL)—and if you don't care about logical markup you can achieve exactly the same *visible* results by using this chosen style sheet mechanism with HTML. (Remember that you can use the neutral `SPAN` tag in HTML to apply any attributes, style names included, to arbitrary fragments of text.) It is when the proper internal structure of your data really matters that XML easily outshines HTML.

XML specification (found at `http://www.textuality.com/sgml-erb/WD-xml-lang.html`) defines the language in the terms of behavior of a *parser*, which is a piece of software whose sole purpose is to understand the element structure of your document and break it down into nested elements in accordance with the DTD. Another program (termed simply "application" in the XML specification) is supposed to obtain the document thus dissected from the parser and process it further. Exactly what the application performs on the document is outside the scope of XML; for instance, it may be a browser that displays the document using an appropriate style sheet.

Because XML is a subset of SGML, an XML document is almost always a valid SGML document; small discrepancies between these two languages are likely to be eliminated soon with the acceptance of certain amendments to the SGML standard. The relation between XML and HTML is more complex. With the capability to define new tags, XML documents are not likely to count as valid HTML very often; on the other hand, an HTML file is relatively easy to make XML-conformant on one of the two levels of conformance (described later in this chapter), depending on whether or not you provide a DTD for your document.

I don't attempt a real tutorial of XML in this chapter for two reasons. First, one chapter's space is surely insufficient to cover even the basics of the language, and second, the language itself is so young and unstable that it is probably untimely to start teaching it in a serious fashion. (A quote from the language specification: "Please be advised that the draft you are now reading is unusually volatile.") Instead, this chapter presents a couple of small examples that will help you to quickly grasp the "look and feel" of the language.

In a sense, XML is positioned somewhere in between SGML and HTML, with the intent of its creators being to combine the best features of these two languages. However, XML is much closer to SGML than to HTML, and although knowledge of HTML will help you understand the most obvious XML features, an acquaintance with SGML syntax and ideology would be of much more help. So I recommend that you brush up what you remember from reading Chapter 3 before proceeding to subsequent sections of this chapter.

Well-Formed XML Documents

If you're scared by the prospect of learning the art of writing document type definitions, there's good news for you: With XML, you can create a document even without DTD. If the lack of a DTD is the sole violation of XML requirements, such a document is called *well-formed,* as opposed to a *valid* document that has a DTD provided (or referred to). Thus, well-formedness is the lower of the two levels of XML conformance, but although it is inferior to validity, it is still very useful.

The permission to omit the DTD means, in essence, that you are free to use *any* tags that seem necessary for your document. You may just go wild and write in plain English what each part of your text is supposed to represent. For example, if you're a grammarian, you could try the following:

```
<SENTENCE>
  <SUBJECT TYPE="COMPLEX">
    <ARTICLE TYPE="INDEFINITE">A</ARTICLE>
    <ADJECTIVE>quick</ADJECTIVE>
    <ADJECTIVE>brown</ADJECTIVE>
    <NOUN>fox</NOUN>
  </SUBJECT>
  <VERB TYPE="INTRANSITIVE">jumps</VERB>
  <PREPOSITION>over</PREPOSITION>
  <ARTICLE TYPE="INDEFINITE">a</ARTICLE>
  <OBJECT>
    <ADJECTIVE>lazy</ADJECTIVE>
    <NOUN>dog</NOUN>
  </OBJECT>.
</SENTENCE>
```

Given this well-formed document, an XML parser will be able to break the text into the elements that you deemed essential in this case and pass each element to the application along with its name (derived from the tag name) and attribute information you provided. It is the application, not the XML parser, that must be programmed to perform some useful tasks with this structured information. Most often, you'll need to provide the application with a style sheet that associates certain formatting parameters with each element. (In our example, the style sheet might specify displaying different parts of speech in different colors.)

In fact, even "plain English" is not an obligatory requirement for your tags. Creators of XML intended that the language must be international from the very beginning. They painstakingly identified all Unicode characters that may be called "letters" in some sense or in some language and included these characters in the set of characters allowed in element and attribute names. This means that you can write your tags in Russian or Chinese instead of English.

There are, of course, numerous restrictions that are imposed even on well-formed documents. Some of these requirements are even stricter than those of HTML and thus deserve special attention:

- In XML, *every* start tag must have a corresponding end tag (unless it is an empty tag that takes special form, as described in the next item). Tags should be properly nested; that is, you can't have an open tag within the scope of some other tag and the corresponding end tag outside that scope.
- If a tag is empty by its nature (in other words, the corresponding element can never contain any text), it must have a forward slash (/) before the closing greater than symbol (>), as in the following example:

  ```
  <IMG alt="XML logo" src="http://www.ucc.ie/xml/xml.gif"/>
  ```

  Such tags are the only tags that do not have corresponding end tags.
- All attribute values without exception must be enclosed in quotes—either single quotes (' ') or double quotes (" ").

In fact, the preceding requirements are the *only* ones that you must satisfy to make your HTML files well-formed XML. It doesn't matter which browser's HTML extensions you use or whether you "abuse" HTML tags or not. XML is a truly liberal language; it makes you a creator of your own universe whose rules you're unlikely to break simply because you established them.

Those familiar with the material in Chapter 3 may be wondering about another essential part of any SGML application—the SGML declaration that always accompanies a DTD. Because XML is pretty simple compared to SGML, its authors decided to omit this part of the language; XML parsers (or SGML software processing XML documents) should behave as if all XML documents were assigned the same generic SGML declaration that is listed in an appendix to the XML standard.

XML DTDs and Valid XML Documents

Although in many cases well-formed XML documents are sufficient for practical purposes, designing a DTD for your document has a number of advantages:

- First and foremost, a DTD allows an XML parser to *validate* your document (which is why such documents are called valid). When validating, the parser checks for misspelled tags or attributes, for errors in types of attribute values and in elements' content models (covered in Chapter 3), and so on. For HTML, similar validation services exist that will check your file against one of the existing HTML DTDs.

- For a human reader, a DTD is a convenient way to quickly learn the structure of the particular type of documents. Compared to SGML, the simplified DTD syntax of XML is very straightforward and unambiguous.
- With DTD, you can define not only elements and their attributes, but also *entities*. (See "Entity Declarations," later in this chapter.) Similarly to macros in word processors or `#define` preprocessor instructions in C, entities can be used to abbreviate text strings and markup instructions in an obvious and easy-to-modify manner. Also, you can use *external entities* to refer to other XML documents, DTDs, or binary data located in separate files.

Accessing the DTD

Let's examine an example of a valid XML document, namely a play by Shakespeare (*The Tempest*) marked up by Jon Bosak, one of the authors of XML. The package (`http://sunsite.unc.edu/pub/sun-info/standards/dsssl/egs/21_shaks/`) includes, besides the XML document and its DTD, a DSSSL style sheet that contains formatting instructions for each element and a Postscript output of a DSSSL processor that formatted the play. (See Chapter 21, "JavaScript Style Sheets and Other Alternatives to CSS.")

Here's the very beginning of the XML document `play.xml`:

```
<?XML version="1.0"?>
<!DOCTYPE play PUBLIC "-//Free Text Project//DTD Play//EN">
```

The first line here is an *XML declaration*, a special instruction that is XML-specific and would not be recognized by an SGML parser because of the `<?` delimiter (which is not used in SGML). Here, the XML declaration provides information about the version of XML standard that the document conforms to.

Next comes the `DOCTYPE` statement that, like its namesake in SGML, provides the DTD for the document to be parsed. In XML, a DTD may be in two parts: *internal* is contained in the document file itself, and *external* is referenced by its URL or *public identifier* (covered in Chapter 3), with the internal part taking precedence over the external one in a case of conflict.

In our example, only the external part of DTD is present, which is referred to by the public identifier preceded by the keyword `PUBLIC`. An XML parser is supposed to be able to retrieve the text of the DTD using its public identifier (that is, to translate the identifier into a URL or some other sort of physical address). If the DTD you're using is not assigned a well-known public identifier, you should provide a URL instead of it, with the `SYSTEM` keyword instead of `PUBLIC`. Here is an example:

```
<!DOCTYPE HTML SYSTEM "http://www.foo.com/myfiles/html3x.dtd">
```

Finally, to provide an internal part for a DTD, you must put it in brackets within the DOCTYPE declaration. Such a declaration may also contain a SYSTEM or PUBLIC external reference, as in the following example:

```
<!DOCTYPE HTML SYSTEM "http://www.foo.com/myfiles/html3x.dtd"
[
  <!-- your DTD goes here -->
]>
```

Element Declarations

The name right after the DOCTYPE keyword in the preceding statements is the name of the *root element* of your document type, the top-level element that encloses all other elements. In HTML, this element is named HTML, and in our Shakespearean example it is named PLAY. Here's how the PLAY element is defined in play.dtd:

```
<!ELEMENT play (title, fm, personae, scndescr, playsubt, induct?,
                                        prologue?, act+, epilogue?)>
```

You can see that the *content model* (discussed in Chapter 3) for this element is quite simple and immediately translatable into human talk: "A PLAY is formed by its TITLE, followed by the front matter (FM), followed by the list of dramatis PERSONAE, and so on." The question mark indicates optional elements, and the plus sign indicates the elements that may occur once or more. Note that the XML spec prescribes to drop the SGML minimization parameters that are useless in XML, which doesn't permit tag omission anyway.

One more excerpt from play.dtd shows a hierarchical set of related tags to mark a personage's speech:

```
<!ELEMENT speech   (speaker+, (line ¦ stagedir ¦ subhead)+)>
<!ELEMENT speaker  (#PCDATA)>
<!ELEMENT line     (stagedir ¦ #PCDATA)+>
<!ELEMENT stagedir (#PCDATA)>
<!ELEMENT subhead  (#PCDATA)>
```

Thus a SPEECH is constituted by one or more SPEAKER elements followed by at least one of the LINE, STAGEDIR (stage direction), or SUBHEAD elements, in no particular order. (The ¦ sign means that any one of the connected particles may occur.) The #PCDATA keyword has the meaning of "any character data without tags"; thus, the SPEAKER, STAGEDIR, and SUBHEAD elements are allowed to contain only text characters, while a LINE may have STAGEDIRs intermingled with text.

Note that nothing in the definition of LINE (except the name) suggests that what the element contains is really a line of verse. It is just implied to be so by the person who did markup, and it may be formatted as a line if an appropriate style sheet is used. However, XML only serves as an intermediator between the author and the formatter, and it is not intended to describe the nature of data elements that are marked up with it.

Here's a SPEECH element exemplifying these DTD provisions:

```
<SPEECH>
<SPEAKER>PROSPERO</SPEAKER>
<LINE><STAGEDIR>Aside</STAGEDIR> The Duke of Milan</LINE>
<LINE>And his more braver daughter could control thee,</LINE>
<LINE>If now 'twere fit to do't. At the first sight</LINE>
<LINE>They have changed eyes. Delicate Ariel,</LINE>
<LINE>I'll set thee free for this.</LINE>
<STAGEDIR>To FERDINAND</STAGEDIR>
<LINE>A word, good sir;</LINE>
<LINE>I fear you have done yourself some wrong: a word.</LINE>
</SPEECH>
```

Entity Declarations

Entities can be declared in a DTD as follows:

```
<!ENTITY me "Dmitry Kirsanov, St.Petersburg, Russia">
```

In the document, such an entity can be used similarly to mnemonic character entities of HTML:

```
This document was created by &me; on April 21, 1997
```

Another syntax is used to define entities that refer to external files or documents, as in the following example:

```
<!ENTITY mypage SYSTEM "http://www.symbol.ru/dk/index.xml">
<!ENTITY xml-logo SYSTEM "http://www.ucc.ie/xml/xml.gif" NDATA gif>
```

In the second declaration, `gif` is the name of a *notation* (similar to a data type), which must be declared somewhere in the DTD along with information on where an XML processor can access a helper software capable of handling data in this notation.

Now, `&mypage;` and `&xml-logo;` entities can be used in documents using this DTD. However, XML specification does not prescribe the exact behavior of the XML application on encountering such an entity. For example, it might incorporate it into the text of the current document, or it might present it as a link that the user can activate.

Linking Capabilities of XML

The first part of the XML specification referenced earlier in this chapter (`http://www.textuality.com/sgml-erb/WD-xml-lang.html`) fully describes the syntax of the language. However, a second part of the specification at `http://www.textuality.com/sgml-erb/WD-xml-link.html` is concerned with *linking capabilities* of XML documents similar to the hypertext features of HTML.

In some sense, this is beyond the scope of an SGML-like language, because the *semantics* of elements (including their capability to link) should be left to creators of SGML or XML applications. On the other hand, links are difficult to implement using common style sheet languages and therefore need support from a deeper level. For this reason (as well as, apparently, to make XML a better competitor to HTML), authors of the language chose to make linking provisions a part of the main body of the XML standard.

The linking model of XML (which can be used with SGML as well) is much more versatile than that of HTML. It is largely based on the efforts of other SGML projects such as HyTime and Text Encoding Initiative (TEI). I don't describe the detailed syntax here, because it is likely to change in subsequent drafts of the specification. It is worth noting, however, that all linking constructs are implemented on the level of reserved attributes and their values, not elements. This means that you can easily turn any element into a link by expanding its list of attributes.

A simple link intended to connect the XML document you're reading to some other Internet resource may be assigned a number of parameters, including the following:

- Strings describing the *role* of the link in the document and its associated *label* (for example, this information may be displayed by the browser in its status line)
- A parameter defining whether the linked resource should, upon activating the link, replace the current document, be opened in a new context (for example, in a new window), or be embedded into the current document in place of the link
- A parameter indicating whether the user can activate the link or it is activated automatically when encountered by the processing application

Very important, although perfectly backwards-compatible, enhancements are proposed for the syntax of URLs that are used in XML to address resources. Using search parts (separated by `?`) or fragment identifiers (separated by `#`) of special form, you can access any part of another XML document without the necessity of attaching a label to it beforehand (such as `<A NAME="...">` in HTML).

The syntax of such *extended locators* in XML is capable of expressing formally such requests as "everything from the third paragraph of the first chapter to the end of the chapter" or "the whole section to which the last sidebar belongs." Of course "chapters" or "sidebars" are to be declared as corresponding elements in the DTD of the linked document.

Moreover, you may prescribe whether the necessary fragment is extracted right on the server where the linked document is stored (which may result in considerable bandwidth savings) or whether the server sends you the whole document and your application needs to winkle out the part you requested.

XML links can be not only inline but also *out-of-line*, which means that they do not need to reside in one of the documents connected by the link. Links can be grouped so that all the links in a group are activated at once. Finally, you can create collections of interlinked documents and inform the processing application about what other documents contain links to the current one.

The Prospects of XML

XML is really a quite recent development. Its basic principles were all worked out in 1996, and the first draft of the specification was presented to the public at the SGML 96 conference in November. A revised draft (still not final specification) was released in March 1997. As of this writing, software created for parsing XML files is all experimental and can fit on one 3.5-inch disk.

Nevertheless, this new development is likely to seriously impact the Web industry in the near future, and in the more distant future, it could completely change the landscape we're accustomed to. Here are several points that were chosen as the most important goals by the designers of the language. These could become the key advantages in the competition of XML with other technologies:

- **XML shall be straightforwardly usable over the Internet.** Web servers in use today require minimal configuration changes to be able to serve XML documents. The standard way to link and bind together XML documents and DTDs is via URLs that are understood by the majority of Internet software.
- **It shall be easy to write programs that process XML documents.** The experimental XML software mentioned earlier is all written in Java, with some of the experimental XML parsers being contained in class files of a few kilobytes in size.
- **XML documents should be human-legible and reasonably clear.** With the users of XML being able to create their own tags and attributes with self-explanatory names, an XML file is likely to be nearly as readable as (and in some cases, even more readable than) plain text.
- **The XML standard should be prepared quickly.** It is not yet finalized, but the amount of work done in such a short term is impressive.
- **The design of XML shall be formal and concise.** Syntax descriptions in XML specification use a formal grammar that is concise, easy to understand, and easy to translate into programming code.
- **XML documents shall be easy to create.** As you've seen in this chapter, the concept of well-formedness enables you to quickly mark up any document or translate it from HTML to XML.
- **Terseness in XML markup is of minimal importance.** Clear and unambiguous syntax was always given preference over the possibility of saving a few keystrokes.

The XML technology is in an embryonic state, so any attempts to augur its future are almost sure to not come true. However, the growing interest in XML shown by many people concerned with Web development is a clear indication that the Web is eager to try out something more powerful and elegant than the HTML of today.

Summary

In this chapter, you've become acquainted with an important new development, the extensible markup language, that, combined with a style sheet language, has a potential to eventually replace the heavily overused HTML of nowadays. Because it is a simplified subset of SGML, XML allows you to create customized markup tags and specify the corresponding document strucure for any type of documents you might need to store or disseminate. In particular, you learned about the following topics:

- What XML is and isn't, in what aspects it is better than HTML, and why it is important to mark up the logical structure of a document in the first place.
- What are well-formed XML documents, and how they can be easily created from HTML documents.
- What are the advantages of providing an XML document with a DTD, and how to refer to the DTD file from within an XML document.
- How elements and entities are declared in an XML DTD and then used in the documents.
- What linking capabilities are provided by the XML standard, and how they extend the HTML hypertext model.
- What were the main goals of the creators of XML, and in what ways its design principles help it compete with HTML and other technologies.