- Enhanced photo gallery support
- Hooks for including external pages and code
- Incremental publishing capability
- Flexible meta tag management
- Powerful, full-site management tools

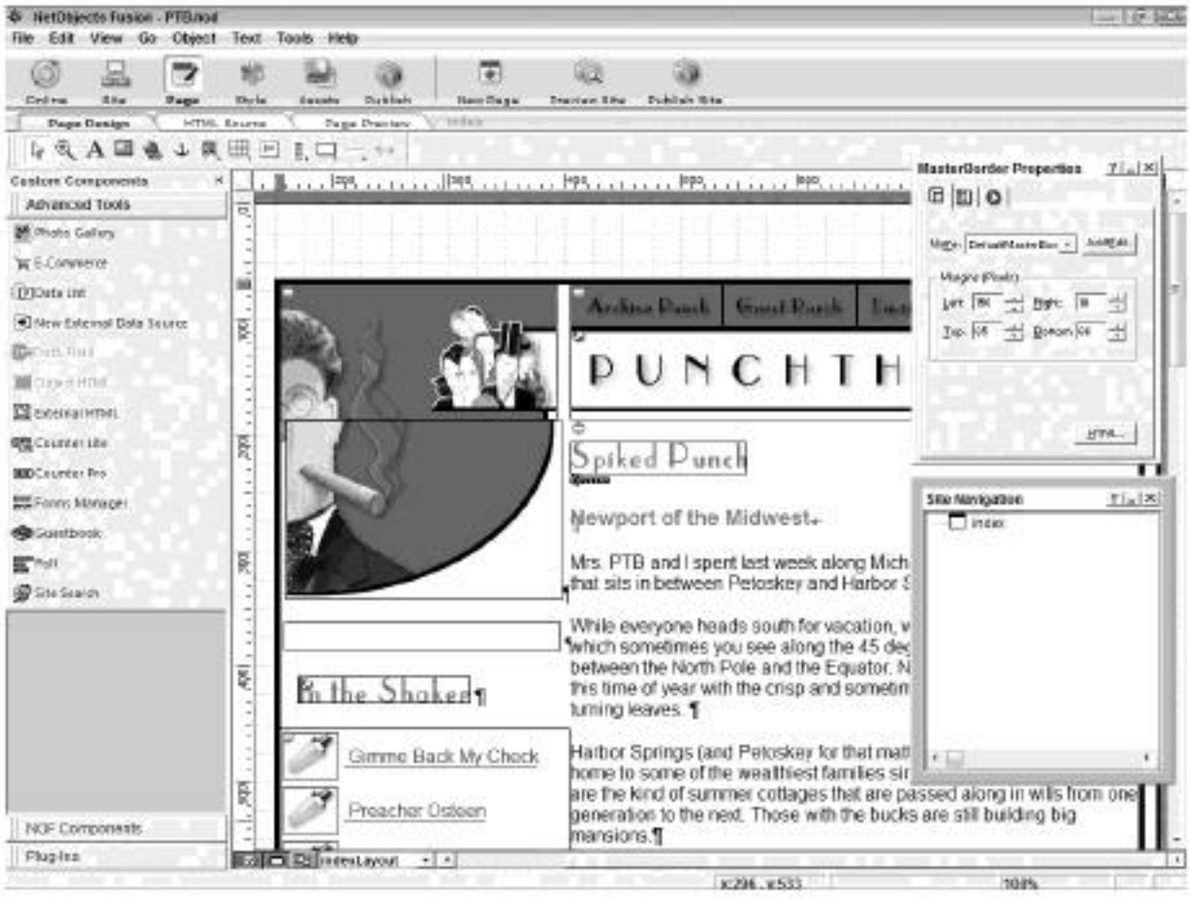
Note

NetObjects Fusion should not be confused with Macromedia's ColdFusion product. The former is owned by Website Pros and is a WYSIWYG Web editor. The latter is owned by Macromedia and is a database integration tool for the Web. ■

Figure 19-4 shows the page design view for NetObjects, and Figure 19-5 shows the site layout view. In the latter, you can easily create, delete, and move pages around your site — NetObjects Fusion will automatically adjust all links, navigation bars, and other references between the pages.

FIGURE 19-4

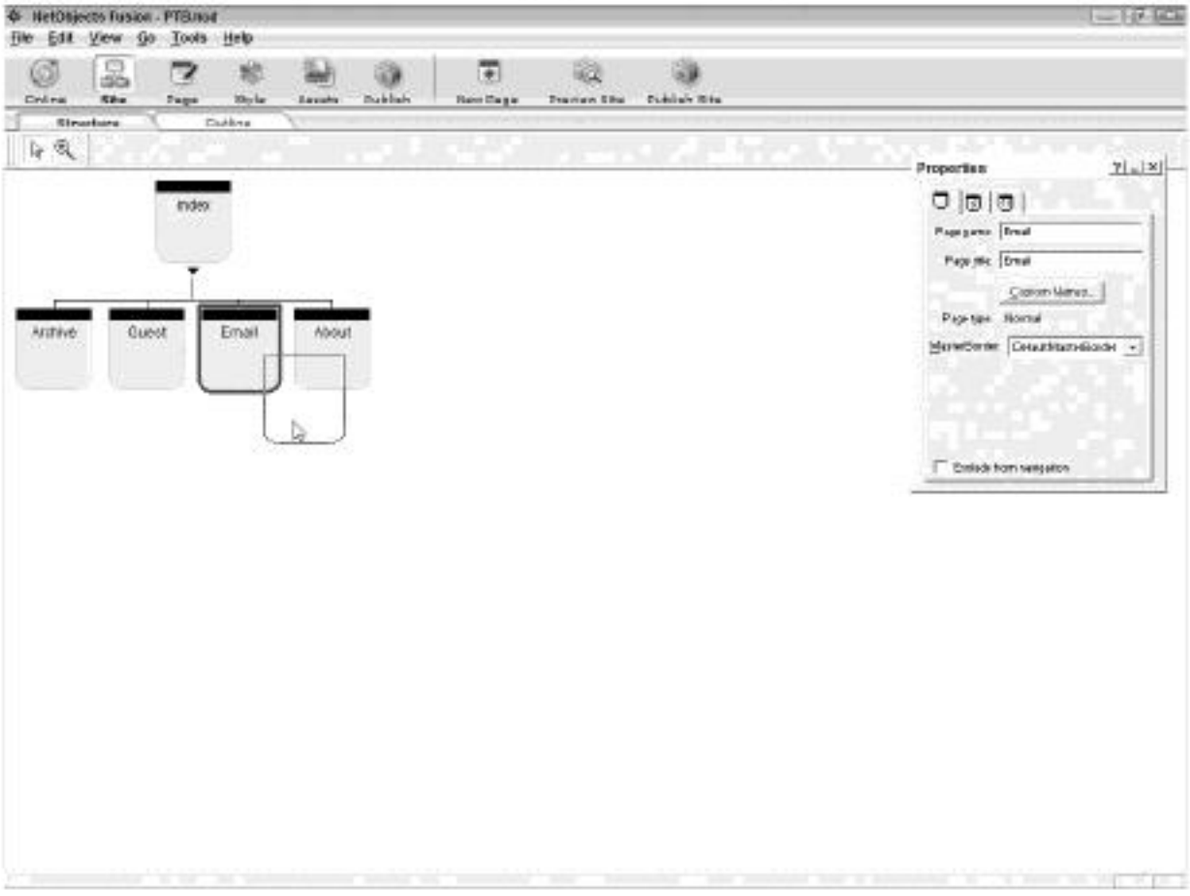
NetObjects Fusion provides a good framework for designing pages visually.



In addition to the visual tools, NetObjects Fusion provides many ways to customize the actual code behind your documents as well. You can learn more about NetObjects Fusion on the Web at www.netobjects.com.

FIGURE 19-5

At the site level, NetObjects Fusion gives you complete control over your site's organization; behind the scenes, it adjusts links between pages automatically.



Dreamweaver

The king of all Web document editing programs is currently Adobe Dreamweaver. Combining the best visual and nonvisual editing tools with several development features, Dreamweaver is the most feature-rich program covered here.

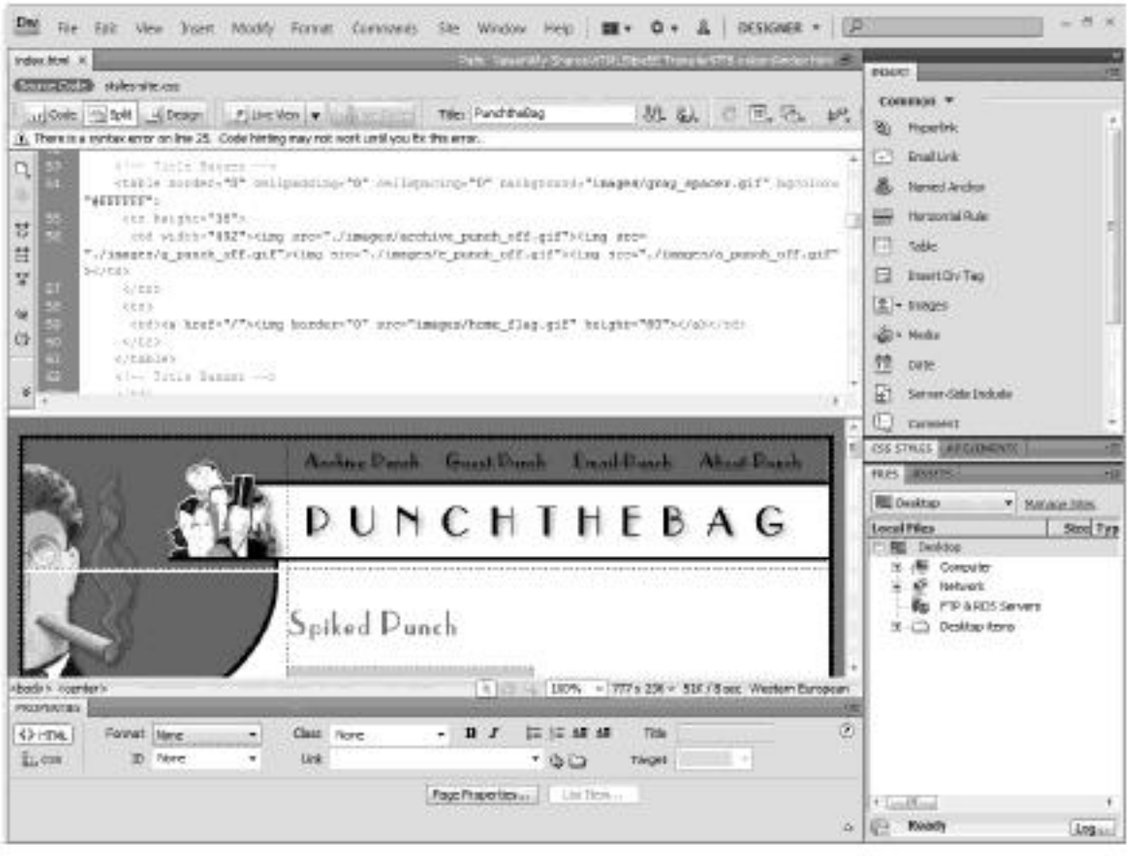
Dreamweaver provides as much or as little automation during creation of new documents as you would like. You can create the entire site in text mode, editing HTML code directly. Alternatively, you can use the WYSIWYG design editor to create your documents visually. Figure 19-6 shows Dreamweaver's main editing window, displaying both the code and visual design windows. Figure 19-7 shows the Check Browser Compatibility feature, which enables you to test your code against the compatibility of specific browsers.

The feature-rich nature of Dreamweaver does come at a price — it is easily the most complicated program covered in this chapter. The learning curve for Dreamweaver can be quite steep, even to create simple sites. However, once you get used to Dreamweaver, it is easy to appreciate its powerful features.

You can learn more about Dreamweaver at `www.adobe.com/products/dreamweaver/`.

FIGURE 19-6

Dreamweaver's main editing window can show the code view, the design (visual) view, or both.



Firefox Add-ons

Firefox is a favorite browser for Web developers for one reason: add-ons. Using Firefox's robust application programming interface, developers can create simple widgets or extensive programs to add to the Firefox interface.

Several add-ons, Firebug in particular, enable you to view and edit your documents in unique ways. Firebug, for example, enables you to inspect individual elements to see their styles, where the styles are applied from (given the cascade), and even edit the HTML and styles on the current *live* document to help tweak your formatting. Figure 19-8 shows Firebug in action.

Find more information on Firefox, Firebug, and other Firefox add-ons at `www.mozilla.org`.

FIGURE 19-7

The Check Browser Compatibility feature checks your code against the compatibility of specific browsers.

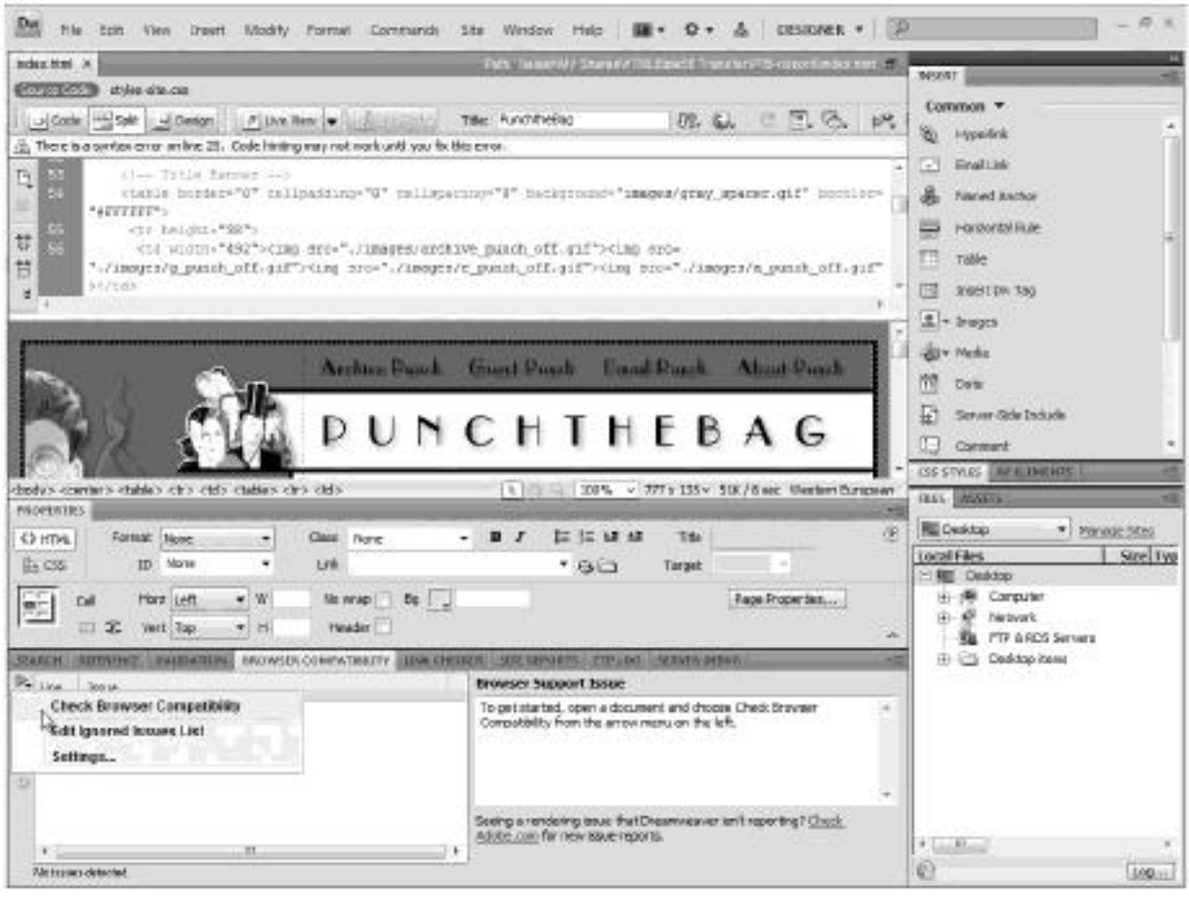


FIGURE 19-8

When inspecting an element, Firebug displays all the information on styles affecting the element (their source and settings), and allows you to tweak the styles on the live document.

Other Tools

Tools to create HTML are only half of the equation when creating online documents. You must also have tools available to do graphics editing and supply any multimedia content you use. This section covers a handful of additional tools necessary to create rich, online content.

Graphics editors

Years ago, text-only Web pages were the norm. However, today's Web is a visual feast, and your documents must incorporate as much imagery as possible in order to be noticed.

Almost every operating system comes with at least one graphics editor, but the capabilities of the included editors are quite limited, and you shouldn't rely on them for much. The same goes for graphics programs bundled with many scanners, printers, and other graphics peripherals.

Ideally, you should consider using both a vector-based and a raster-based editing program. Vector-based editors use shapes and lines to create images, whereas raster-based editors use individual dots (pixels) to create images. Vector-based images are traditionally more exact and clear, but raster-based images allow for more visually striking effects. The best results can be obtained using both — use the vector tools to create solid imagery, and the raster tools for special effects and finishing work.

Note

Only raster-based images (specifically JPEG, GIF, and PNG images) are supported by common user agents. ■

Vector-based editing tools include the following:

- Adobe Illustrator
  `www.adobe.com/products/illustrator/main.html`
- Adobe Freehand
  `www.adobe.com/products/freehand`

Raster-based editing tools include the following:

- Paint Shop Pro Photo X2
  `http://store.corel.com/webapp/wcs/stores/servlet/ProductDisplay?partNumber=OL_PR12`
- Adobe Photoshop
  `www.adobe.com/products/photoshop`
- Adobe Fireworks
  `www.adobe.com/products/fireworks`
- The GIMP
  `www.gimp.org`

Note

Paint Shop Pro Photo X2 actually supports both raster and vector editing. ■

Note that these tools can be quite expensive — the latest version of Photoshop is several hundred dollars. Of course, Photoshop is without equal for raster editing; no other tool provides as much power and extensibility. Paint Shop Pro Photo X2 is quite capable at around $100, and The GIMP provides suitable editing without a price tag (it's open source).

Adobe Flash

Adobe Flash is the staple for most multimedia on the Web. Flash provides an animation platform with plenty of power via ActionScript, a flexible scripting language, and can be used for simple buttons or full-blown product demos.

Although the interface is a bit idiosyncratic, Flash is an indispensable tool for online animation. Figure 19-9 shows a Flash document in development.

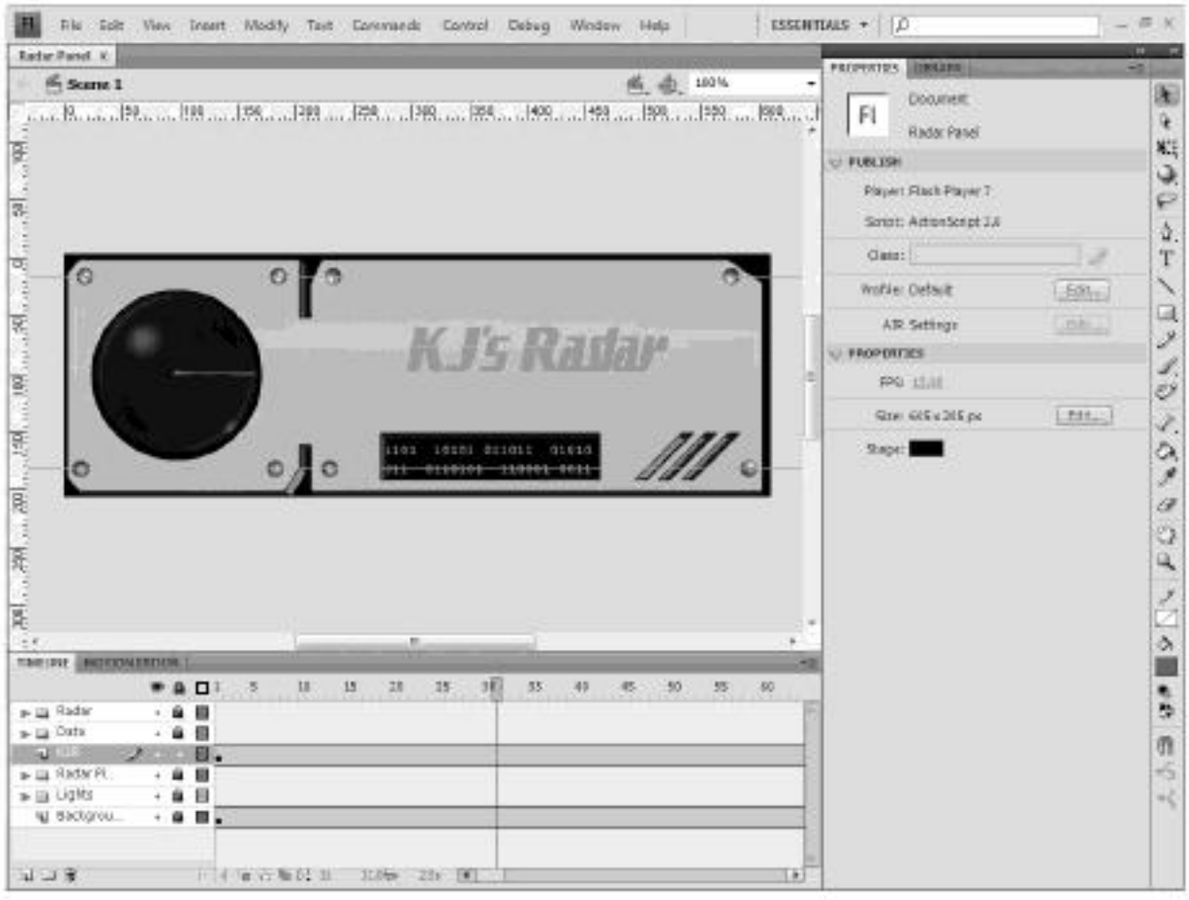
The main draw of Flash is two-fold:

- It has become a standard on the Web that users expect.
- Flash can provide even complex animations in a small package (small file size).

Flash is another tool you should consider adding to your collection. You can learn more about Flash at `www.adobe.com/products/flash`.

FIGURE 19-9

Flash can be used for simple or complex animations.



Summary

This chapter introduced you to a handful of HTML, CSS, and graphics editing tools you can use to make the creation of Web documents easier. Of course, there are many more Web-oriented

tools on the market; the ones presented here only scratch the surface. When evaluating tools for your own use, keep in mind that there is a balance between cost and effectiveness. The tools included on the free disc from your ISP may be very affordable, but implementing them may wind up costing more than a tool you have to buy. The sweet spot lies somewhere in between, in capable but budget-minded tools. Many software manufacturers provide evaluation copies of their software that you can download and try before you purchase.

The next several chapters (20 through 24) continue the coverage of tools and utilities you can employ to develop and deploy your documents.

CHAPTER 20

Publishing Your Site

Now that you have documents to deploy on the Web, how do you actually move the files to the Web server? If you don't have an automated publishing tool (as covered in Chapter 19), you will probably use File Transfer Protocol (FTP). This chapter provides an introduction to FTP and explains how you can use it to deploy your files to a server.

IN THIS CHAPTER

Introducing FTP

File Transfer Protocol was created to easily move files between computers on the Internet. Dating back to the very early days of the Internet, FTP hasn't evolved much during the years it has been in service. FTP encapsulates several functions to transfer files, view files on both sides of the connection, and more.

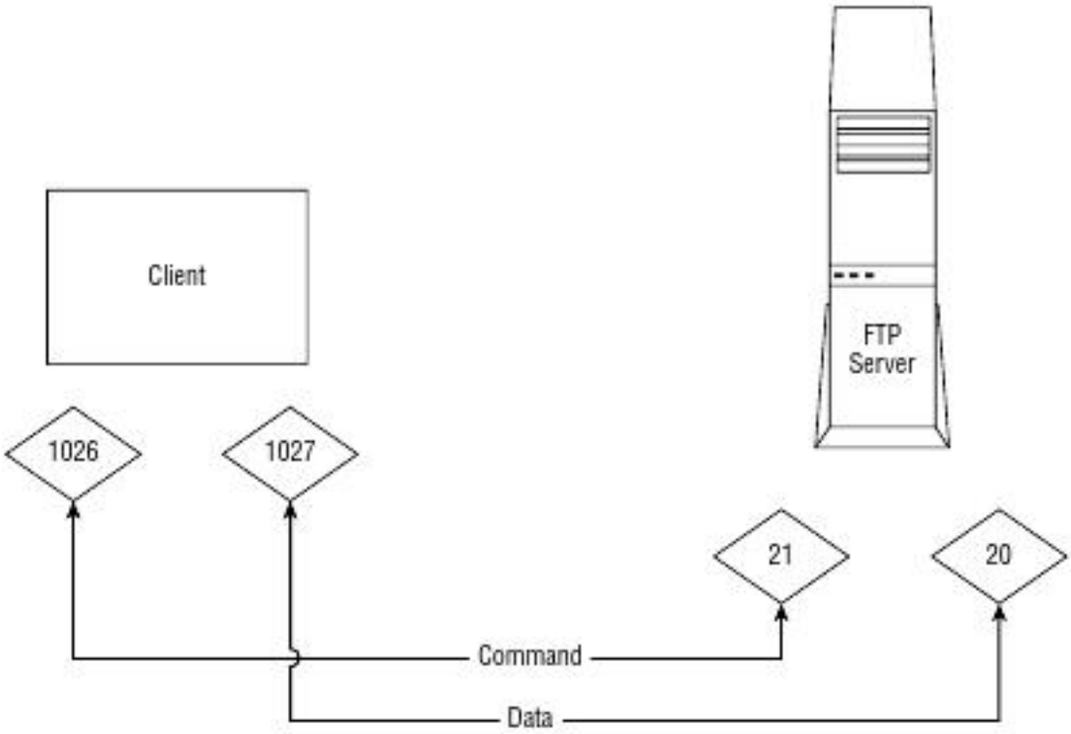
FTP servers use the same protocol as the rest of the Internet: TCP/IP. TCP/IP is a packet-switching protocol that enables computers all over the world to communicate with one another via the Internet. The protocol uses well-defined ports — data doorways reserved for particular applications — to segregate the types of information traveling over the network. FTP uses TCP/IP ports 20 and 21. These ports are unique to the FTP service, allowing a computer to run a Web server (port 80), an FTP server (ports 20 and 21), as well as other services at the same time.

The FTP server sits patiently waiting for a client to request a connection on port 21. The client opens a port greater than port 1024 and requests a connection from the server. After the connection is authenticated — that is, a user logs in — the client can initiate commands to transfer files, and so on. When data is transferred between the client and the server, the server initiates the connection using port 20 — the client uses one port higher than

the port used for commands. Figure 20-1 shows a graphical representation of the connection and port arrangement.

FIGURE 20-1

A typical FTP connection



One problem with the traditional FTP process is that the server must initiate the data connection. This requires that the server be able to access the requisite port on the client to initiate the connection. If the client is using a firewall, the firewall might prevent the server from accessing the correct port. Because the client port isn't consistent, configuring the firewall to allow access is problematic.

To solve this problem, a new mode of FTP was created. Passive mode (typically referred to as *PASV*) allows the client to initiate both connections.

Note
If you are behind a firewall, you should always try to use passive mode. ■

FTP Clients

The first FTP clients were text-only applications, meaning the connection is initiated and data is transferred using textual commands. The latest FTP clients employ the same graphical interface as most modern operating systems, using standard file manager-like interfaces to accomplish FTP operations.

Note

Graphical FTP clients use the same methods and commands to communicate with the FTP server, but typically hide the communication from the user. The term "client" comes from the fact that the software — "application" by any other name — is connecting to a "server." As such, the application inherits the client moniker because of its role in the connection relationship. ■

The following code example shows a typical dialogue using a textual FTP client. The client initiates a connection, and the user logs in, gets a directory listing on the server, and then transfers a file. For clarity, the commands entered by the user are in boldface:

```
$ ftp ftp.example.com
Connected to ftp.example.com.
220 ftp.example.com FTP server ready.
Name: sschafer
331 Password required for sschafer.
Password: ******
230 User sschafer logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd www
250 CWD command successful.
ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for file list.
drwxr-xr-x  2 sschafer sschafer  4096 Jun 20 16:45 Products
drwxr-xr-x  2 sschafer sschafer  4096 Jun 16 18:41 About
drwxr-xr-x  2 sschafer sschafer  4096 Jun 6 15:16 Images
-rwxr-xr-x  1 sschafer sschafer  1571 Jun 12 17:58 index.html
drwxr-xr-x  2 sschafer sschafer  4096 Jun 15 04:16 Scripts
226-Transfer complete.
226 Quotas off
ftp> put index.html
local: index.html remote: index.html
200 PORT command successful.
150 Opening BINARY mode data connection for index.html.
226 Transfer complete.
2095 bytes sent in 0.3 secs (3.6 kB/s)
ftp> close
221 Goodbye.
ftp> quit
$
```

Figure 20-2 shows a graphical FTP application accessing the same site. The application shows the file listing of the remote server. To transfer a file, the user simply drags the file into or out of the application window to a local window or a destination pane within the same FTP application. Notice the underlying FTP commands and output in the lower-right corner of the application. Some graphical FTP client applications allow you to take manual control, entering various commands as required.

FIGURE 20-2

Graphical FTP clients use graphical user interface methods to transfer files.

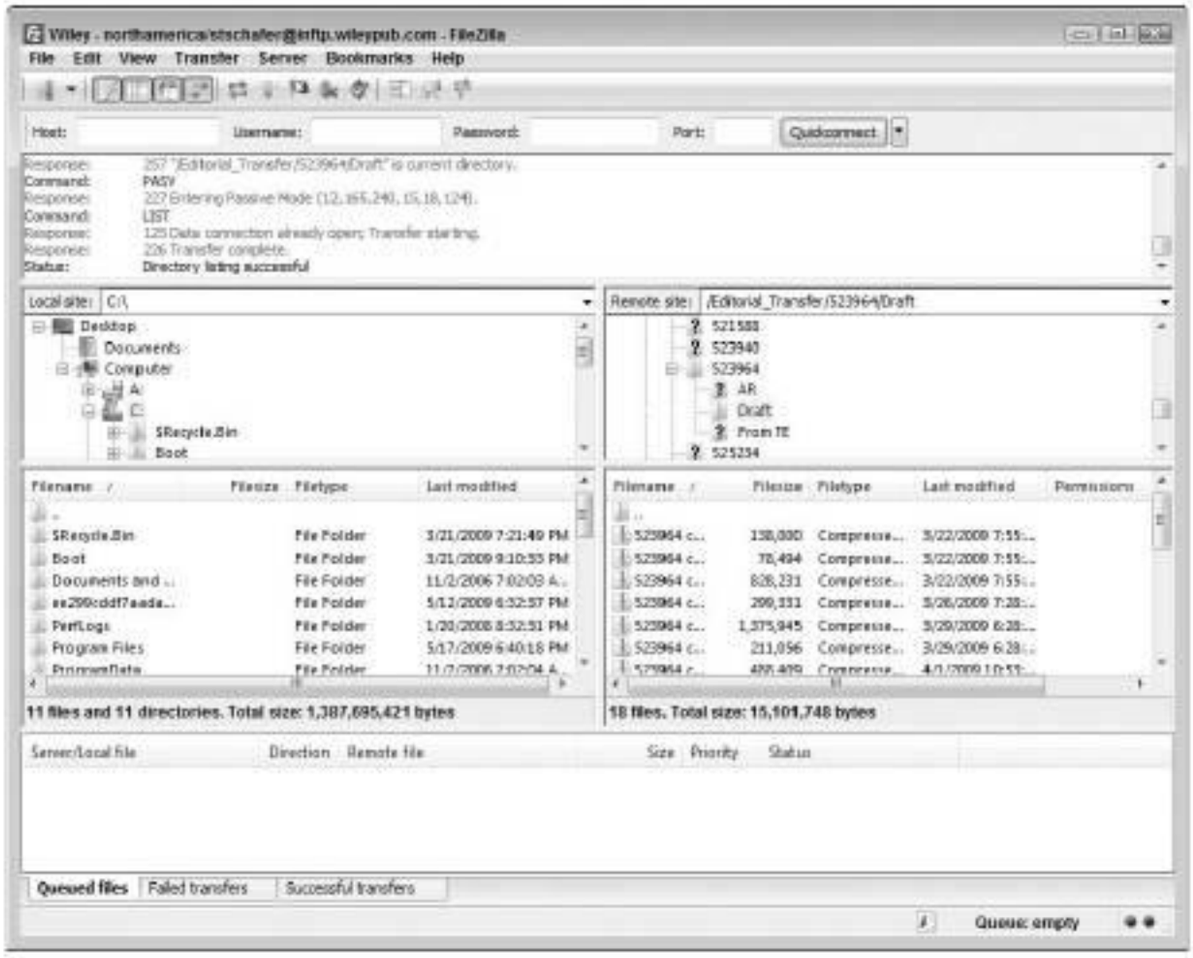


Table 20-1 shows a list of common FTP commands.

TABLE 20-1

Common FTP Commands

Command	Syntax	Use
`ascii`	`ascii` or `asc`	Switch to ASCII mode for file transfers.
`binary`	`binary` or `bin`	Switch to binary mode for file transfers.
`cd`	`cd` *`directory_name`*	Change the remote directory.
`close`	`close`	Close the current connection to the server (log off).
`get`	`get` *`filename`*	Download a file from the server.

Command	Syntax	Use
lcd	lcd *directory_name*	Change the directory on the local machine.
ls	ls *[file_spec]*	List files on the server (in the current directory).
mget	mget *file_spec*	Download multiple files from the server.
mkdir	mkdir *directory_name*	Create a new directory on the server.
mput	mput *file_spec*	Upload multiple files to the server.
user	user *username*	Initiate login as username (prompt for password).
pasv	pasv	Enter passive mode.
put	put *filename*	Upload a file to the server.
quit	quit	Exit the client.
rmdir	rmdir *directory_name*	Remove a directory on the server.
open	open *server_address*	Open a new connection to the server.

Notable FTP Clients

Most operating systems include a textual FTP client, aptly named FTP. To use the client, type **ftp** at a command prompt. For example, on Windows XP, you would click the Start button, choose Run, type **command**, and press Enter. When the system prompt appears, type **ftp** and press Enter. Once the FTP program loads, an `ftp>` prompt appears. Other operating systems utilize different means to access their command prompt, but the concept is similar.

However, not all textual clients use the same commands or have the same options. Most clients support a help command: Type **help** followed by the name of the command for which you need help. Unfortunately, the standard help output simply tells you what the command does, not the syntax or options.

Tip

There are many ways to place files on the Web server. The easiest, of course, is to create and edit the files directly on the server. If you are using a development application, you can use its features to upload your content (typically such programs use FTP to transfer files). ■

Quite a few graphical FTP clients are available, from $100 commercial solutions to open-source and shareware solutions. The following list is a subset of available clients:

- Cross-platform clients:
  - **FileZilla** — This FTP client provides a host of valuable features in an open-source, cross-platform (and free) package. FileZilla is available for Windows, Linux, Mac OS X, and BSD platforms and supports a wide range of languages. Visit the FileZilla project page at `http://filezilla-project.org` for more information.

- **FireFTP** — This FTP client is courtesy of a free add-on for Firefox. It provides the basic functionality to copy files to and from FTP sites as well as some nifty features like file compression. More on FireFTP can be found at `http://fireftp.mozdev.org/`.

- Windows clients:
  - **FTP Voyager** — This client enables you to transfer files between servers, resume aborted downloads, and more. It also has a scheduler that can automatically transfer files at set times.
  - **CoffeeCup FTP client** — This freeware client contains the usual options for graphical clients.
  - **CuteFTP** — This popular client contains a number of features to make FTP transfers easier. It provides a download queue, macro recording, and a scheduler to automate file transfers.
  - **WS-FTP** — This FTP client has the typical features found in other commercial solutions.
- Linux clients:
  - **Desktop-specific clients** — Both K Desktop Environment (KDE) and Gnome include graphical clients specific to the desktop environment.
  - **Additional open-source solutions** — Many graphical FTP clients are available for Linux. Each distribution contains several from which you can choose. Even more are available from various online sources.

Tip

Your Web browser can be used as a graphical client. Simply specify the FTP protocol (`ftp:`) and the server address, as in the following example:

```
ftp://ftp.example.com ■
```

If the server requires authentication, you will be prompted for your login information.

Principles of Web Server File Organization

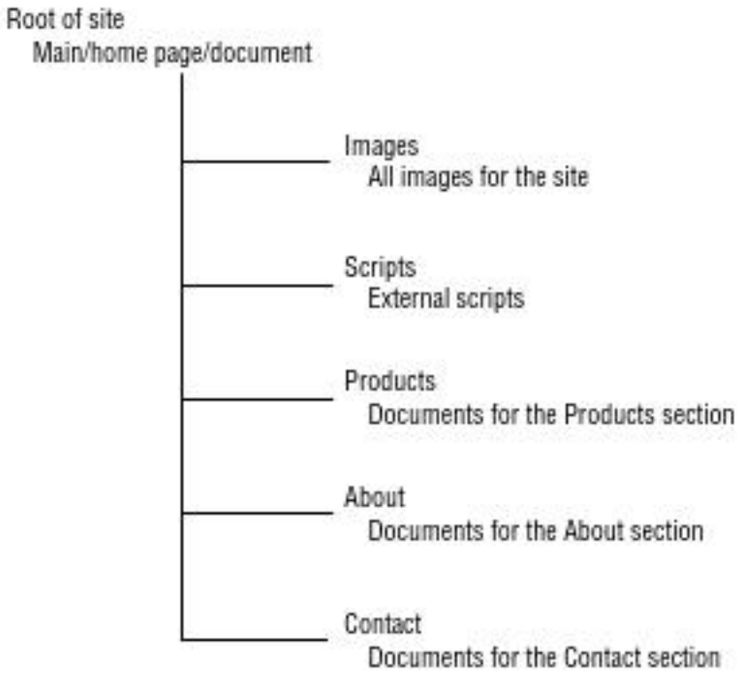
Files on a Web server typically follow a tiered organization, placing subordinate pages in subdirectories. Furthermore, supplemental files — scripts, images, and so on — are typically placed in separate directories. Keeping the same hierarchical structure on both your computer and the server is an advisable tactic. Figure 20-3 shows the organizational structure of a typical website.

Note

There really isn't anything *typical* on the Web. As such, you should use a file and directory structure that suits your needs. The examples in this chapter are just that, examples. The important thing is that you use *some* logical organizational structure in your files and directories, and be consistent. ■

FIGURE 20-3

Typical organization of a website



If your site is small enough, it can be contained in one single directory. A site with many files, however, should be organized within several directories. Use your FTP client's features to create subdirectories, and transfer your files into the directories accordingly.

Summary

Although you can host HTML documents on a local machine, their true potential is realized when you publish your documents on a public server, where the rest of the world can view and interact with them. This chapter taught you the basics of FTP — the File Transfer Protocol of the Internet — which is used to transfer most of the world's HTML documents from local machines to public servers. You learned the basics of using an FTP application or client to connect and transfer files. You also learned that most Web development applications have built-in FTP services to help you get your content online.

The rest of this part covers variants of the version of HTML covered in Part I, including XML (Chapter 21) and XHTML Basic (Chapter 22).

CHAPTER

21

An Introduction To XML

Extensible Markup Language (XML) is a popular scheme for representing data. Although created as a more portable version of Standard Generalized Markup Language (SGML), XML lives mostly on the application side of the computer world. XML is used to store preferences and data from applications, provide a unified data structure for transferring data, encapsulate syndicated feeds from websites, and more. XML standards are being adopted by other data formats such as HTML (creating XHTML).

This chapter presents a primer on XML, including its format, methods, and tools.

IN THIS CHAPTER

Note

Full coverage of XML can occupy an entire book on its own, and is therefore outside the scope of this one. In the case of the Web, XML is a bystander technology, useful to know but not entirely critical for publishing on the Web. However, because XHTML is XML-compliant, coverage is mandatory. If you desire more information about XML, you can pick up a book dedicated to the subject, such as *WROX Beginning XML*, Third Edition, *WROX XSLT 2.0 Programmer's Reference*, Third Edition, or Wiley's *XML Weekend Crash Course* or *XML Programming Bible*. ■

XML Basics

XML was created to bring the advantages of the SGML standard to smaller platforms such as Web browsers. XML retains the flexibility of its older sibling but has been redesigned for the Web, enabling it to be easily transmitted via the Internet's architecture and displayed with less overhead.

The XML design strategy attempted to address the following points:

- Form should follow function. In other words, the language should be flexible enough to encapsulate many types of data. Instead of shoehorning multiple forms of data into one structure, the structure should be able to change to adequately fit the data.
- Documents should be easily understood by their content alone. The markup should be constructed in such a way that there is no doubt about the content it frames. XML documents are often referred to as *self-describing* because of this attribute.
- Format should be separated from presentation. The markup language should represent the difference in pieces of data only, and should make no attempt to describe how the data will be presented. For example, elements should be marked with tags such as `<emphasis>` instead of `<b>` (bold), leaving the presentation of the data (which should be emphasized, but not necessarily bold) to the platform using the data.
- The language should be simple and easily parsed, with intrinsic error checking.

These attributes are evident in the goals stated in the W3C's Recommendation for XML 1.0 (found at `www.w3.org/TR/1998/REC-xml-19980210`):

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

As is, XML is ill suited for the World Wide Web. Because XML document elements can be author defined, user agents cannot possibly interpret and display all XML documents in the way the author intended. However, standardized XML structures are excellent for storing application data. For example, consider the following applications of XML:

- The popular RSS syndication format defines particular element tags in XML format to encapsulate syndicated news and blog feeds. This enables many applications to easily disseminate the information contained within the feed.
- Several online statistic sites (computer game stats, and so on) store their information in XML because it can be easily parsed by a variety of applications.
- Many applications store their preferences in XML-formatted files. This format proves to be easily parsed, changed, and rewritten, as necessary.

- Many word-processing and other document-based applications (e.g., spreadsheets) store their documents in XML format.
- Many business-to-business applications use XML to share and transfer data between each other.

Note that while XML provides an ideal data structure for some applications, it should be used only for smaller, sequential collections of data. Data collections that require random access or have thousands of records would benefit from an actual database format, rather than XML.

Note

XHTML was designed to bring HTML into XML compliance (each element being properly closed, and so on), not the other way around (to add extensibility to HTML). In short, XHTML adheres to XML standards but is not itself an extensible markup language. ■

XML Syntax

XML and XHTML follow many of the guidelines already set forth for HTML but are slightly more stringent:

- Element and attribute names are case sensitive.
- All elements must be properly closed.
- Elements must be properly nested, not overlapping.
- All attributes must have values.
- All attribute values must be quoted.

Note

This book espouses the formatting syntax of XHTML, which is more exacting than straight HTML. As such, you already know many of the conventions for XML. ■

Within documents, the structure is similar to that of HTML, where element tags are used to encapsulate content that may itself contain tag-delimited content.

The following sections outline the particular syntax of the various XML elements.

XML Declaration and DOCTYPE

Each XML document must begin with an XML declaration similar to the following:

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

The declaration is `<?xml?>`, with `version` and `encoding` attributes. The `version` attribute specifies the version of XML the document uses, and the `encoding` attribute specifies the character encoding used within the document's content.

As with other markup languages, XML supports Document Type Definitions (DTDs), which specify the rules used for the elements within documents using the DTD. Applications can then use the DTD to check the document's syntax. An XML document's DTD declaration resembles that of an XHTML document, specifying a `SYSTEM` or `PUBLIC` definition. For example, the following DTD is used for OpenOffice documents:

```
<!DOCTYPE office:document-content PUBLIC
   "-//OpenOffice.org//DTD OfficeDocument 1.0//EN" "office.dtd">
```

The following is an example of an XHTML document's DTD:

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

Elements

XML elements resemble HTML elements. However, because XML is extensible, elements are generally not of the HTML variety. For example, consider the following snippet from an RSS feed, presented in XML format:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="/externalflash/NASA_Detail/NASA_Detail.xsl"
   type="text/xsl"?>
<rss version="2.0">
  <channel>
    <title>NASA Breaking News</title>
    <link>http://www.nasa.gov/audience/formedia/features/index.html</link>
    <description>A RSS news feed containing the latest NASA news
      articles and press releases.</description>
    <language>en-us</language>

    <item>
      <title>Atlantis Set for Return to Kennedy Space Center</title>
      <link>./HQ_M07077_Atlantis_ferry_flight.html</link>
      <description>The shuttle's ferry flight aboard a modified 747
         is expected to occur this weekend.</description>
      <pubDate>Fri, 29 Jun 2007 00:00:00 EDT</pubDate>
    </item>
    <item>
      <title>ISS Status Report: SS07-32</title>
      <link>./HQ_SS0732_station_status.html</link>
      <description>Operations and research occupied the crew this
         week.</description>
      <pubDate>Fri, 29 Jun 2007 00:00:00 EDT</pubDate>
    </item>
    <item>
      <title>Satellite Captures First View of 'Night-Shining'
         Clouds</title>
      <link>./HQ_07145_AIM_First_Light.html</link>
      <description>A NASA satellite has captured the first occurrence
         this summer of mysterious iridescent polar clouds that form 50
```

```
        miles above Earth's surface.</description>
      <pubDate>Thu, 28 Jun 2007 00:00:00 EDT</pubDate>
    </item>
    <item>
      <title>NASA Mars Rover Ready for Descent Into Crater</title>
      <link>./HQ_07145_Rover_Victoria_Crater.html</link>
      <description>NASA's Mars rover Opportunity is scheduled to begin a
        descent down a rock-paved slope into the Red Planet's massive
        Victoria Crater.</description>
      <pubDate>Thu, 28 Jun 2007 00:00:00 EDT</pubDate>
    </item>
  </channel>
</rss>
```

In this case, the following elements are used:

- `channel` — The container for the channel, that is, the feed itself. The `channel` container has the following subcontainers:
  - `title` — The title of the channel or feed
  - `link` — The link to the feed on the Web
  - `description` — The description of the feed
  - `language` — The language of the feed's content
- `item` — The feed encapsulates each news item within an `item` element, which has the following sub-elements:
  - `title` — The title of the item
  - `link` — A link to the item on the Web
  - `description` — A short description of the item
  - `pubDate` — The publication date of the item

Note that several elements have multiple contexts. For example, the `channel` and `item` elements both provide context for `title` elements — the placement of each `title` element (usually its parent) determines to what element the `title` refers.

Attributes

XML elements support attributes much like XHTML. Again, the difference is that the attributes can be defined in accordance with the document's purpose. Consider the following code snippet:

```
<employee sex="female">
  <lastName>Moore</lastName>
  <firstName>Terri</firstName>
  <hireDate>2003-02-20</hireDate>
</employee>
<employee sex="male">
  <lastName>Robinson</lastName>
  <firstName>Branden</firstName>
  <hireDate>2000-04-30</hireDate>
</employee>
```

In this example, the sex of the employee is coded as an attribute of the `employee` element.

In most cases, the use of attributes instead of elements is arbitrary. For example, the preceding example could have been coded with `sex` as a child element instead of as an attribute, as in the following:

```
<employee>
  <sex>female</sex>
  <lastName>Moore</lastName>
  <firstName>Terri</firstName>
  <hireDate>2003-02-20</hireDate>
</employee>
<employee>
  <sex>male</sex>
  <lastName>Robinson</lastName>
  <firstName>Branden</firstName>
  <hireDate>2000-04-30</hireDate>
</employee>
```

The mitigating factor in deciding how to code data is whether the content is ever to be used as data, instead of just a modifier. If an application will use the content as data, it's best to code it within an element where it is more easily parsed as such.

Comments

XML supports the same comment tag as HTML:

```
<!-- comment_text /-->
```

You can embed comments anywhere inside an XML document as long as the standard XML conventions and corresponding DTD rules are not violated by doing so.

Non-parsed data

On occasion, you will need to define content that should not be parsed (interpreted by the application reading the data). Such data is defined as character data, or CDATA. Nonparsed data is formatted within a `CDATA` element, which has the following syntax:

```
<!CDATA [non_parsed_data]]>
```

Generally, `CDATA` elements are used to improve the legibility of documents by placing reserved characters within a `CDATA` element instead of using cryptic entities. For example, both of the following paragraph elements result in identical data, but the first is more legible because the `CDATA` elements are used instead of entities:

```
The table element should be used instead of the pre element
whenever possible.
```

```
The <!CDATA [table]]> element should be used instead of the
<!CDATA
[pre]]> element whenever possible.
```

Entities

XML also allows for user-defined entities. Entities are content mapped to mnemonics — the mnemonics can then be used as shorthand for the content within the rest of the document. Entities are defined using the following syntax:

```
<!ENTITY entity_name "entity_value">
```

Entities are defined within a document's DTD. For example, the following document prologue defines "Acme, Inc." as the entity customer:

```
<?xml version="1.0"?>
<!DOCTYPE report SYSTEM "/xml/dtds/reports.dtd" [
  <!ENTITY customer "Acme, Inc.">
]>
```

Elsewhere in the document the entity (referenced by &entityname;) can be used to insert the customer name:

```
<report>
  <title>TPS Report</title>
  <date>2005-01-25</date>
  <summary>The latest run of the regression test have yielded
  perfect results. The engagement for &customer; can now be
  completed and the final code delivered.</summary>
...
```

Entities can also be declared as external resources. Such external resources are generally larger than a few words or a phrase, such as complete documents. A system entity, used for declaring external resources, is defined using the following syntax:

```
<!ENTITY entity_name SYSTEM "URL">
```

For example, the following code defines a chapter01 entity that references a local document named chapter01.xml:

```
<!ENTITY chapter01 SYSTEM "chapter01.xml">
```

The chapter01 entity can then be used to insert the contents of chapter01.xml in the current document.

Namespaces

The concept of namespaces is relatively new to XML. Namespaces enable you to group elements together by their purpose using a unique name. Such groupings can serve a variety of purposes, but are commonly used to distinguish elements from one another.

For example, an element named `table` can refer to a data construct or a physical object, such as a dining room table:

```
<!-- Data construct in one document-->
<table>
  <tr><th>Date</th><th>Customer</th><th>Amount</th></tr>
  <tr><td>2005-01-25</td><td>Acme, Inc</td><td>125.61</td></tr>
...
</table>

<!-- Home furnishing definition in another document /-->
<table>
  <type>Dining</type>
  <width>4</width>
  <length>8</width>
  <color>Cherry</color>
</table>
```

If both elements are used in the same document there will be a conflict because the two refer to two completely different things. This is a perfect place to specify namespaces. Namespace designations are added as prefixes to element names. For example, you could use a furniture namespace to identify the table elements that refer to furnishings:

```
<furniture:table>
  <type>Dining</type>
  <width>4</width>
  <length>8</width>
  <color>Cherry</color>
</furniture:table>
```

The prefix should be uniquely tied to a namespace using a namespace declaration with an appropriate `xmlns` attribute. The namespace declaration has the following form:

```
<prefix:tag xmlns:tag="url">
```

For example, using our furniture prefix with the table tag, we would have something similar to the following:

```
<furniture:table xmlns:table="http://www.w3.org/XML/">
```

Note that the URL in the declaration serves only as a unique identifier — it is not perceived by the XML parser as an actual Uniform Resource Identifier (URI) of any type and might not even exist. It does, however, need to be unique within its sphere of influence.

Stylesheets

XML also offers support for stylesheets. Stylesheets are linked to XML documents using the `xml-stylesheet` tag, which has the following syntax:

```
<?xml-stylesheet type="mime_type" href="url_to_stylesheet"?>
```

For example, to link a stylesheet to an XML document, you could use a tag similar to the following:

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

Working with Document Type Definitions

As previously mentioned, an XML document that follows the syntax rules of XML is called a well-formed document. You can also have, or not have, a *valid* document. A document is valid if it validates against a DTD. Just as with HTML, an XML DTD is a document containing a list of rules that define the structure of the XML document. For example, a DTD can dictate whether all contact elements contain a phone element, as in the following code:

```
<contact>
   <name>Jill Hennessy</name>
   <address>111 East Main St.</address>
   <phone>1-303-555-4444</phone>
</contact>
```

The preceding code fragment is well formed as it stands. However, you may wish to define rules that more clearly delineate the purpose of each element and the position of each element within the framework or document as a whole. You might also want to define how many times each element can (or should) appear in the document.

A DTD can exist either outside the XML document that validates against it or within that same document. If the DTD exists outside of the document, you must declare it within the XML document so the XML parser knows you're referring to an *external* DTD, similar to the following:

```
<!DOCTYPE root SYSTEM "filename">
```

For example, for our contact XML document, an external `DOCTYPE` declaration might look like this (the `DOCTYPE` declaration is in bold):

```
<?xml version="1.0"?>
<!DOCTYPE contact SYSTEM "contact.dtd">
<contact>
   <name>Jill Hennessy</name>
   <address>111 East Main St.</address>
   <phone>1-303-555-4444</phone>
</contact>
```

The definitions would then be placed in a separate file, `contact.dtd`, accessible by the document.

You can also place the DOCTYPE rules within the XML document itself, as in the following example:

```
<?xml version="1.0"?>
<!DOCTYPE contact [
 <!ELEMENT contact (name, address, phone)>
  <!ELEMENT name        (#PCDATA)>
  <!ELEMENT address     (#PCDATA)>
  <!ELEMENT phone       (#PCDATA)>
]>
<contact>
   <name>Jill Hennessy</name>
   <address>111 East Main St.</address>
   <phone>1-303-555-4444</phone>
</contact>
```

Although DTDs are not XML documents, DTD and XML structure alike is defined using the following core components of XML:

- Elements
- Attributes
- Entities
- PCDATA
- CDATA

Each of these is described in the sections that follow.

Using elements in DTDs

Elements are the main data-containing components of XML. They are used to structure a document. You've seen them in HTML, and the core principles are the same as that of HTML. Elements can contain data or be empty. If they are empty they normally include an attribute, but it isn't a requirement. The HTML `br` and `img` elements are good examples of empty elements, as they don't encapsulate any data.

XML elements are declared with an element declaration using the following syntax:

```
<!ELEMENT name datatype>
```

The first part of the declaration (`!ELEMENT`) says that you are defining an element. The next part (`name`) is where you declare the name of your element. The last part (*`datatype`*) declares the type of data that an element can contain. An element can contain the following types of data when defined by DTDs:

- `EMPTY` data, which means there is no data within the element
- `PCDATA`, or parsed character data
- One or more child elements

Using element declaration syntax for empty elements

Empty elements are declared by using the keyword EMPTY:

```
<!ELEMENT name EMPTY>
```

For example, to declare an empty rug element, you would write the following:

```
<!ELEMENT rug EMPTY>
```

This element would appear as follows in an XML document:

```
<rug />
```

Using element declaration syntax for elements with PCDATA

Elements that do not contain any child elements and contain only character data are declared with the keyword #PCDATA inside parentheses, like this:

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

A typical example of such an element follows:

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

An XML parser might then encounter an actual note element that looks like this:

```
<note>The saunas will be closed for maintenance all of next
week. Please be sure to let your clients know.</note>
```

As you can see, the note element contains only text (PCDATA), and no child elements.

Using element declaration syntax for elements with child elements

Elements can contain sequences of one or more children, and are defined with the name of the children elements inside parentheses:

```
<!ELEMENT name (child_name)>
```

If you have more than one child element, separate each element with a comma:

```
<!ELEMENT name (child_name_1, child_name_2)>
```

An example, using the code you saw earlier for the contact document, might look like this:

```
<!ELEMENT contact (name, address, phone)>
```

Declaring the number of occurrences for elements

You can also declare how often an element can appear within another element by using an *occurrence operator* in your element declaration. The plus sign (+) indicates that an element *must*

occur one or more times within an element. Therefore, if you create the following declaration, the phone element must appear at least once within the contact element:

```
<!ELEMENT contact (phone+)>
```

You can declare that a group of elements must appear at least one or more times:

```
<!ELEMENT contact (name, address, phone)+>
```

To declare that an element can appear zero or more times (in other words, it's an optional element), use an asterisk instead of a plus sign, as in the following:

```
<!ELEMENT contact (phone*)>
```

If you want to limit an element to zero or one occurrence (meaning it can't appear more than once), use a question mark (?) operator instead:

```
<!ELEMENT contact (phone?)>
```

The following XML would not be valid when the declaration uses a ? operator for the phone element:

```
<contact>
   <phone>303-555-4444</phone>
   <phone>303-555-4447</phone>
</contact>
```

You can also use a pipe operator (|) to indicate that one element *or* another element can be contained within an element:

```
<!ELEMENT contact (name,address,phone,(email | fax))>
```

In the preceding declaration, the sequence of name, address, and phone elements must all appear in the order shown, followed by either the email or fax elements. This means the following XML is valid:

```
<contact>
   <name>Jill Hennessy</name>
   <address>111 East Main St.</address>
   <phone>1-303-555-4444</phone>
   <email>jill@oasisoftranquility.com</email>
</contact>
```

However, the following XML would not be valid if validating against the same DTD:

```
<contact>
   <name>Jill Hennessy</name>
   <address>111 East Main St.</address>
   <phone>1-303-555-4444</phone>
   <email>jill@oasisoftranquility.com</email>
   <fax>303-555-4447</fax>
</contact>
```

Using attributes in DTDs

Attributes define the properties of an element. For example, in HTML, the `img` element has an `src` property, or attribute, that describes where an image can be found.

To define attributes for elements, you use an `ATTLIST` declaration. The `ATTLIST` declaration has the following format:

```
<!ATTLIST  element_name  attribute_name
  attribute_type  default_value>
```

The `element_name` and `attribute_name` parameters are what you would expect — the element to which the attribute applies and the name of the actual attribute. The `attribute_type` and `default_value` parameters are more complex, as they must handle several different values.

Table 21-1 shows the various values possible for the `attribute_type` parameter.

TABLE 21-1

Attribute Types

Value	Definition
`CDATA`	Character data
`(value|value|...)`	Enumerated data
`ID`	Unique ID
`IDREF`	ID of another element
`IDREFS`	List of IDs of other elements
`NMTOKEN`	An XML name
`ENTITY`	An entity
`ENTITIES`	A list of entities
`NOTATION`	Name of a notation
`xml:`	A predefined value

Table 21-2 shows the list of acceptable values for the `default_value` of the attribute.

For example, the following DTD declaration defines a `phonenumber` element with a default `type` attribute of `home`:

```
<!ELEMENT phonenumber  (#PCDATA)>
  <!ATTLIST phonenumber   type  CDATA "home">
```

TABLE 21-2

Default Value Settings

Value	Definition
`value`	The attribute has a default value of `value`.
`#REQUIRED`	The attribute is always required in the element.
`#IMPLIED`	The attribute does not need to be included in the element.
`#FIXED value`	The attribute has a default value of `value` and that value is fixed — it cannot be changed by the author.

To limit the values of the `type` attribute to `home`, `work`, `cell`, or `fax` — with a default of `home` — you could change the declaration as follows:

```
<!ATTLIST phonenumber type (home|work|cell|fax)>
```

Using entities in DTDs

You saw how to create entities in XML in the "Entities" section of this chapter. As a reminder, you use an `ENTITY` declaration of the following syntax:

```
<!ENTITY entity_name "entity_value">
```

Entities are defined within a document's DTD. For example, the following document prologue defines `"Acme, Inc."` as the entity `customer`:

```
<?xml version="1.0"?>
<!DOCTYPE report SYSTEM "/xml/dtds/reports.dtd" [
  <!ENTITY customer "Acme, Inc.">
]>
```

Elsewhere in the document the entity (referenced by `&customer;`) can be used to insert the customer name.

Using PCDATA and CDATA in DTDs

PCDATA is parsed character data, which means that all character data is parsed as XML; any starting or closing tags are recognized, and entities are expanded. Elements contain PCDATA.

CDATA is data that is not parsed by the processor. This means that tags are not recognized, and entities are not expanded. Attributes do not contain PCDATA; they contain CDATA.

Introducing XML Schemas

However important, DTDs can be somewhat limiting. Consider, for example, the following XML document:

```
<datatypes>
  <Boolean>true</Boolean>
  <integer>1</integer>
  <double>563.34</double>
  <date>06-01-2007</date>
</datatypes>
```

As far as DTD rules might be concerned, every element contains character data. The value for the `integer` element is not actually an integer, and the `date` isn't a date. This is because DTDs don't have mathematical, Boolean, or date types of data.

The W3C introduced another rules development methodology called *XML Schema* to handle richer data typing and more granular sets of rules that allow for much greater specificity than DTDs. In addition to the types of rules DTDs manage, schemas manage the data types allowed in an element, such as Booleans and integers.

The use of datatyping is especially important because it facilitates working with traditional databases and application program interfaces (APIs) based on Java, C++, and other languages, such as JavaScript.

Working with Schemas

Now that you're familiar with DTDs, it should be fairly easy to see how their concepts can extend to a greater range of datatypes. XML Schema uses XML syntax to develop rule sets, so it is a bit more intuitive than the DTD syntax shown earlier in the chapter.

Recall that an example earlier created a simple XML document for contacts derived from `contact.dtd`. The following listing shows the same principles at work in a schema. Pay particular attention to the `xs:sequence` `xs:element` children (in bold) that live in the `xs:complexType` element:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.tumeric.net/schemas"
xmlns="http://www.tumeric.net/schemas"
elementFormDefault="qualified">
<xs:element name="contact">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="address" type="xs:string"/>
        <xs:element name="city" type="xs:string"/>
```

```
            <xs:element name="state" type="xs:string"/>
            <xs:element name="postalcode" type="xs:string"/>
            <xs:element name="age" type="xs:integer" />
          </xs:sequence>
        </xs:complexType>
    </xs:element>
    </xs:schema>
```

Note

The `contact` element is a complex type of element because it contains other elements. If an element isn't defined to contain child elements, it's a *simple* type of element. ■

In a DTD, the sequence of elements that should appear in the document is defined by placing a comma-delimited list of the elements in an element definition. In XML Schema, a sequence is defined by creating a sequence of elements in a specific order within an `xs:sequence` element. This is part of the larger definition of the XML document's root element, which is the `contact` element. Note the use of the `type` attribute in the `xs:element` elements, which defines the datatype of each element.

Numerous datatypes are available for XML elements using schema. If you're familiar with the Java programming language, it might help you to know that most of the XML Schema datatypes are similar to Java datatypes. If you're not familiar with Java, there are four basic datatypes:

- Numerical (such as integer and double)
- Date
- String
- Booleans

Tip

You can find out the specifics of various datatypes available in XML Schema at `www.w3.org/TR/xmlschema-2`. ■

You can also place your schema in an external document and reference it from within your XML document. To reference an external schema in an XML document, use the following syntax:

```
<?xml version="1.0"?>
<contact xmlns="http://www.tumeric.net/schemas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=
"http://www.on-target-games.com/schemas/contact.xsd" >
   <name>Johhny Rude</name>
   <address>111 East Onion Ave.</address>
   <city>Big City</city>
   <state>CA</state>
   <postalcode>96777</postalcode>
   <phone>1-323-456-4444</phone>
```

```
    <fax>test</fax>
    <email>rude@rude.com</email>
</contact>
```

The schema is referenced through the namespace for the document. The specific syntax for the namespace declaration looks like this:

```
xmlns="http://www.tumeric.net/schemas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="
http://www.on-target-games.com/schemas/contact.xsd"
```

The other two lines of code are additional namespaces, which serve as identifiers. They tell a parser that elements associated with them are unique and may have specially developed definitions. The important part of the namespace is the Uniform Resource Identifier (URI), which is what gives a namespace its unique identity. When elements live within a specific namespace governed by a schema, they must adhere to the rules of that schema.

The first namespace in the preceding code fragment refers to a namespace established in the schema that uniquely binds the schema to a specified resource, in this case a website. You don't have to refer to a website, and the reference is not actually a physical pointer. Instead, the URI is simply an easy way to establish identity because a website should be unique. While it isn't guaranteed to be unique, because anyone can hijack your website address name and use it for their own schema, using a website address has become fairly standard practice. Instead of a website name, you could use a long mash of characters, as in the following example:

```
xmlns="hk45kskds-scld456ksaldkttsslae697hg"
```

The second namespace refers to the W3C's schema location so that XML processors will validate the XML document against the schema. This is necessary because you then need to call the resource you're using — in this case, a schema that can be found on the path named in the `xsi:SchemaLocation` attribute. When the processor finds the schema, it attempts to validate the XML document as the document loads. If the XML document doesn't conform to the rules you set forth in the schema definition, an error will result (assuming your parser can work with XML Schema).

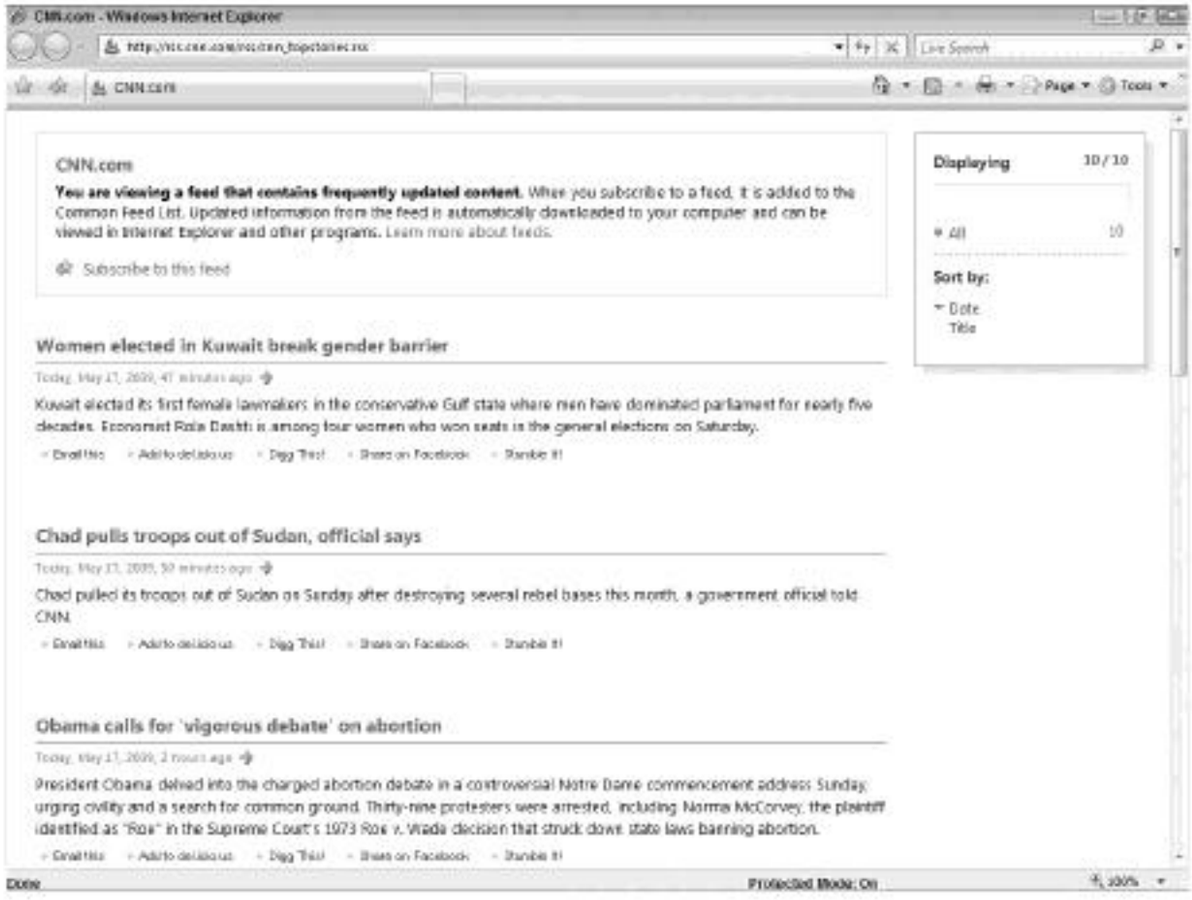
Using XML

Actual use of an XML document requires that the document be transformed into a usable format. There are many means and formats to translate XML — the limits are governed only by your imagination and the tools at hand.

Viewing XML documents doesn't require special tools. Many of the modern user agents assemble various functionality to view XML documents and even add capabilities such as tag highlighting and the ability to collapse portions of the document, as shown in Figure 21-1, where Internet Explorer is displaying an RSS document.

FIGURE 21-1

Internet Explorer is able to render XML documents in a fairly robust manner.



Extensible Stylesheet Language Transformations

Extensible Stylesheet Language Transformations (XSLT) change XML documents into formatted documents and can rearrange document contents to generate new document elements. XSLT takes two items as its input: the XML document (sometimes referred to as the *source tree*) and a stylesheet to determine the transformation. The output document (sometimes referred to as the *result tree*) is in the desired format, ready for output to the desired device.

Many tools are available to help you manage XML documents and perform XSLT, including many open-source solutions (search for "XSLT" on `www.sourceforge.org`).

XML editing

You have many options for editing XML files. Because XML is a text-only format, you can use any text editor (Emacs, vi, Notepad, and so on) to create and edit XML documents. However, dedicated XML editors make the editing job easier by adding syntax highlighting, syntax checking, validation, auto-code completion, and more. Following are some XML text editing options:

- Many open-source XML editors are available (search "XML editor" on http://sourceforge.org).
- Lennart Staflin has developed a major mode for Emacs called PSGML (www.lysator.liu.se/projects/about_psgml.html).
- XMetal — formerly owned by Corel, now owned by Blast Radius — is a well-known, capable (albeit commercial and expensive) XML editor (www.xmetal.com).
- XMLSpy, by Altova, is another capable XML editor in the same price range as XMetal, although the personal edition is free (www.altova.com).
- <oXygen/>, by SyncRO Soft Ltd., is a lower-cost, multiplatform XML editor and XSLT debugger (www.oxygenxml.com).

XML parsing

Many XML parsing applications are available, including many open-source applications (search for "XML parsing" on http://sourceforge.org). In addition, there are XML parsing modules for most programming languages:

- James Clark's XML parser, expat, is well known as the standard for XML parsing (http://expat.sourceforge.net and www.jclark.com/xml/expat.html).
- Many XML modules are available for Perl via CPAN (www.cpan.org).
- Several XML tools are available for Python, including the many found on the Python website (http://pyxml.sourceforge.net/topics).
- PHP has a handful of XML functions built in as extensions to support expat (www.php.net/manual/en/ref.xml.php).
- The PHP Extension and Application Repository has several additional extensions for XML maintenance and manipulation (http://pear.php.net).

Summary

This chapter covered the basics of XML, a fairly robust and extensible markup language that can be used to represent a wide range of data. You learned how similar XML and HTML were in structure, but also how different XML can be to suit its purposes. You learned how an XML document is defined, its elements declared, and how the language can be extended.

The rest of this part of the book covers XHTML Basic, as well as how to clean up and validate your documents, and presents a handful of HTML tips and tricks. The next part of the book covers CSS, the other half of the Web document equation.

CHAPTER

22

Creating Mobile Documents

IN THIS CHAPTER

Understanding the Evolution of the Mobile Web

XHTML Basic 1.1

Mobile Web Development Tools

As I've repeatedly pointed out, the Web and its technologies have grown up. Starting as a simple, text-only medium, the Web is now capable of delivering almost any kind of media. The underlying technologies, HTML and HTTP, have also evolved to better support this growth.

One inevitable side-effect of a popular delivery technology is its rapid adoption by other media devices. Today, it isn't just Web browsers on PCs that access Web content and use Web-related technologies — devices such as mobile phones, mall kiosks, and even ATMs utilize the technology. However, many of these devices have limited resources and cannot make full use of HTML or display the same rich content that a dedicated PC browser can. As a result, if you intend to deliver content via one of these resource-constrained devices, you must limit the scope of your code appropriately.

This chapter covers XHTML Basic, a specification designed for smaller devices. It also covers some ancillary technologies that help deploy content to these devices.

Tip

Before taking the time to create content for mobile devices, ask yourself these questions:

- **Does my audience need my content on their mobile devices?**
- **Does my content lend itself to mobile devices?**
- **Can I spend the time and other resources to keep both my mobile and traditional content up-to-date and in sync?**

If you answer even one of these questions with a "no," then creating and maintaining mobile content is not appropriate for you. ■

Understanding the Evolution of the Mobile Web

If you choose to develop Web content for mobile devices, it is important for you to understand the evolution of those devices and their relationship to the Web. This history is important because the mobile landscape is very different from that of the normal Web. The capabilities of mobile devices tend to be quite limited, and in many cases devices are quite different from one another — from the markup language they understand to their individual capabilities and their connectivity to the Internet (or lack thereof).

The following sections provide a short introduction to these topics.

The first, dark years of mobility

Web-enabled mobile devices have been around for many years. In the late 1990s, several cellular phones were launched with Web features. In the U.S., the technology was backed by a popular phone manufacturer (Nokia) and a mobile connectivity company (Openwave). The two companies created a protocol for mobile data connectivity called Wireless Access Protocol, or WAP. They also created a new minimalistic markup language, Wireless Markup Language (WML). WAP protocol used special gateways for mobile devices to connect to and receive their content, and WML language was very different from normal Web markup specifications like HTML. However, mobile devices were able to receive Web-like content.

Around the same time, the Japanese mobile communications company NTT DoCoMo launched its i-Mode service in Japan, bringing Web-like content to mobile devices. NTT DoCoMo created another HTML variant, Compact HTML, to support its content.

Note

Throughout this section, I use the phrase "Web-like content" because the early mobile content was not delivered via Web standard markup (HTML) or the standard Internet gateways. The content was coded in WML or Compact HTML, and as such could not be as rich. In addition, because it was delivered via proprietary gateways, it was prone to being filtered, and most of the content was created and delivered by the service provider — it was unusual to be able to reach a site of the user's choice. Therefore, the content was only "Web-like." ■

As mobile connectivity became more popular, and even expected, new devices appeared with even more capable user agents. However, more players in the market meant more proprietary solutions. Consumers found that what worked on one phone would not work on another, and the expectation of being able to browse the actual Web was an unrealized one. Thankfully, these consumer issues did not go unnoticed for long.

The Open Mobile Alliance and other standards

Several companies realized the shortcomings of mobile connectivity and the divergence of technologies taking place, and formed the Open Mobile Alliance. This alliance sought to help create better, more globally adopted standards and generally improve the mobile connectivity experience. Several new, exceedingly capable user agents began appearing on devices, while proprietary gateways began disappearing.

Around the same time as the formation of the Alliance, the W3C put together a mobile markup specification designed to bring more order to the mobility market. The new standard, known as *XHTML Basic*, was developed as a minimal set of XHTML tags for mobile user agent support. The Open Mobile Alliance embraced the standard and expanded it to create the XHTML Mobile Profile standard, designed to be adopted by future mobile user agents to enable a more capable and rich browsing experience.

As with everything else in the mobile environment, the new standards were met with spotty acceptance and adoption. Most mobile user agent vendors chose to support XHMTL Basic, but not the expanded Mobile Profile specification. A few vendors with more capable browsers (who were members in the Open Mobile Alliance) chose to support the expanded specification, enabling their users to enjoy a more rich experience while still being backward compatible with XHTML Basic.

However, as you can probably guess, most content developers chose to develop for XHMTL Basic, ensuring that the widest possible audience could use their content.

Note

Older devices (i.e., those two years or older as of this writing) support only WML or one of the older WAP variants. Even user agents that render almost perfect XHTML display their results on smaller screens, have less memory to utilize, and so on. Keep these points in mind as you develop your content — especially if owners of older devices are part of your target audience. ■

The bottom line

The bottom line of this retelling of mobile Web history is this: Although standards have evolved and browser manufacturers are adopting them, you can never be completely certain what browsing capabilities your audience will possess.

Coding to XHTML Basic is a fairly safe bet, but when possible it is best to test several devices in your target audience for compliance with your code.

Note

In the last few years, Web technology on mobile devices has leaped ahead in capabilities. A few user agents, running on select handheld devices, can interpret and display standard HTML content. In rare cases, the

user agents can even support more advanced technologies like Flash. However, when developing for handheld devices, never assume that such capabilities will be available to your entire audience; endeavor to create content for the most basic of devices. ■

XHTML Basic 1.1

XHTML Basic was developed as a subset of XHTML and is defined using a method known as XHTML Modularization. XHTML Modularization is a methodology for creating a markup language by first defining smaller components and then defining how those components fit together to create the entire language.

Note

The current XHTML Basic 1.1 specification can be found at `www.w3.org/TR/xhtml-basic/`. ■

The XHTML Basic 1.1 doctype

As with any other Web document, XHTML Basic documents must start with a proper doctype. In the case of XHTML Basic 1.1, the document header should be as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.1//EN"
    "http://www.w3.org/TR/xhtml-basic/xhtml-basic11.dtd">
```

Also, to ensure that your documents' file type is interpreted correctly from their name alone, they should be saved with a `.xhtml` extension, not `.htm` or `.html`.

XHTML Basic 1.1 elements

Because XHTML Basic was patterned after XHTML, you will find most of the elements familiar in structure, scope, and usage. Table 22-1 lists the modules created for XHMTL Basic, and the elements present in each.

TABLE 22-1

XHTML Basic 1.1 Modules and Related Elements

Module	Elements
Structure Module	`body, head, html, title`
Text Module	`abbr, acronym, address, blockquote, br, cite, code, dfn, div, em, h1, h2, h3, h4, h5, h6, kbd, p, pre, q, samp, span, strong, var`
Hypertext Module	`a`
List Module	`dl, dt, dd, ol, ul, li`

Module	Elements
Basic Tables Module	caption, table, td, th, tr
Image Module	img
Object Module	object, param
Presentation Module	b, big, hr, i, small, sub, sup, tt
Meta Information Module	meta
Link Module	link
Base Module	base
Intrinsic Events Module	event attributes
Scripting Module	script, noscript
Stylesheet Module	style element
Style Attribute Module (Deprecated)	style attribute
Target Module	target attribute

Unless otherwise indicated in Table 22-1, the items in the Elements column are tag elements —
, <h5>, <ol>, and so forth. Attributes are properly noted.

It is interesting to note that XHTML Basic retains all of the text formatting elements (including a few deprecated in HTML 4.01), but deprecates the style attribute, disallowing style definitions within tag elements.

Note

Most mobile devices are not JavaScript enabled, so you should not use JavaScript in any of your documents meant for mobile users. Instead, consider the use of server scripting technologies — PHP, Perl, Python, and so on — to do data processing on the back end to present dynamic, but compliant, XHTML Basic documents. ■

Special considerations

Although the XHTML Basic specification allows for many HTML constructs, a handful of considerations should be taken into account when using XHTML Basic.

Tip

For some excellent guidelines on mobile development, check out the "Mobile Web Best Practices" document by the W3C, found at www.w3.org/TR/mobile-bp/. ■

Screen size

It has often been mentioned that mobile devices have limited screen real estate. However, to fully appreciate the lack of display space on some devices, you should navigate to your favorite website and size your PC's browser window to fewer than 200 pixels. Figure 22-1 shows a similarly sized browser window trying to display the Yahoo! main page.

FIGURE 22-1

Most Web pages look entirely different when viewed through a tiny viewport.

Balancing content for bandwidth and cost

It is easy to get carried away with content when developing for the Web and to assume that most of your audience has a fast computer connected to the Internet via a fast broadband connection. However, that is not the case with most mobile devices. In fact, some users pay a premium to have their device connected to the Internet.

When developing for mobile devices, keep a healthy balance between your content and what users might end up paying for it. This means self-censoring your content and not adding any fluff or out-of-context material. It also means keeping your content lean and mean, coding the bare minimum content, and realizing that mobile content will not be glitzy and flashy at this point in time.

Input restrictions

It is tempting to solicit various pieces of input from mobile users — location data for looking up local services, names for registering in databases, and so forth. However, keep in mind that most mobile devices lack a real keyboard, making a chore out of entering even the most trivial of data. Therefore, it becomes important to limit the amount of data entry required, relying more on alternative data entry schemes, such as select lists, option buttons, links, and so forth.

Easy URLs

Although it is advisable to put your mobile documents in a separate directory on your Web server, you want to keep that directory (and full URL) as easy to "type" as possible. For example, consider these guidelines:

- Keep your directory names short.
- Do not place content further than one level down from the root of the server.
- Avoid any special characters in the URL.
- Use abbreviations instead of long words in the URL (for example, `dev` instead of `developer`).
- Consider creating shorter URLs, decoded by the server.

Another alternative is to place your mobile content in a specific location and have the Web server redirect user agents to that location based upon their capabilities. Most user agents advertise their capabilities — whether they can accept HTML, XHTML MP, WML, and so on — when they request a document. Web servers can read this information and act accordingly. For example, the rewrite module for the Apache server can use the following code to detect a user agent that accepts XHTML Mobile Profile and WML content and deliver an `index.xhtml` document instead of the default `index.html` document:

```
# Test for acceptance of xhtml+xml (MP) and WML
RewriteCond %{HTTP_ACCEPT} application/xhtml+xml
RewriteCond %{HTTP_ACCEPT} text/vnd\.wap\.wml
# If user agent accepts both, it's MP enabled-give
# it the xhtml file instead
RewriteRule index.html$ index.xhtml [L]
```

Note

Full coverage of the Apache rewrite module and how to use it to redirect mobile user agents is outside the scope of this book. You can find more information on Apache's `mod_rewrite` module at `http://httpd.apache.org/docs/2.2/misc/rewriteguide.html`. An excellent primer on how to use several server-based methods of redirection can be found at `www.oreillynet.com/pub/a/wireless/2004/02/20/mobile_browsing.html`. ■

There are also several online scripting services to help you parse the agent trying to access your documents:

- Handset Detection provides a service that your documents can call to determine exactly which handset is accessing them. Visit `http://handsetdetection.com/pages/home` for more information.
- Studio Hyperset is a handful of scripts that can be run on or with your documents to determine the features of the accessing agent. Visit `http://studiohyperset.wordpress.com/2006/11/12/mobile-redirect-update/` for more information.

Small images

For practicality's sake, your images have to be small in terms of dimension, but you should ensure that they are as small as possible in terms of file size too. Run every image through a palette optimizer and consider using black-and-white images wherever possible.

Descriptive alt attributes and link text

When developing mobile content, it is also important to ensure that all images have short but highly descriptive `alt` tags. This ensures that devices that have images disabled or are on a slow network (slow to load images) can display *something* to alert the user of the actual content. In addition, provide descriptive text for all links to ensure that users know where each will take them.

Reliable navigation schemes

When display and usability are limited, reliable navigation schemes become much more significant. Logical access keys and logically structured tab order are two easy methods to improve usability. Placing frequently used navigation toward the top of the page where it can be easily found is another way to improve navigation.

Limit complex display structures

Tables were grudgingly included in XHTML Basic 1.1 by the W3C. Their inclusion was to help ensure that tabular data could be represented in mobile documents. However, tables *should not* be used to format entire documents, as discussed in Chapter 34. Stick to textual data in your tables to help keep mobile users happy.

Mobile Web Development Tools

Many development tools are available to aid your Web document efforts. Almost every phone manufacturer has a tool or two available to help developers create content for delivery on their devices. Table 22-2 lists a handful of the more prolific developer sites.

Each of these sites offers several resources, which are available only after you sign up for the respective developer program.

Tip

Several of the toolsets available from the mobile vendors contain full IDEs and debugging tools that can help you write compliant XHTML mobile code for a variety of devices. ■

TABLE 22-2

Popular Sites for Mobile Content Development Tools

Company	URL
Ericsson Mobility World Developer Program	www.ericsson.com/mobilityworld
ForumNokia	http://forum.nokia.com/
MOTODEV, the Motorola Developer Network	http://developer.motorola.com/

Summary

This chapter covered mobile HTML, from how Web mobility started to how it evolved through the components of XHTML Basic. You learned how easy it can be to create pages to be displayed on a variety of mobile devices, but also how difficult it can be to create content for resource-constrained devices. The next chapter covers how to clean and validate your code. Then you will learn a few HTML tips and tricks.

CHAPTER

23

Tidying and Validating Your Documents

Most of your documents will endure multiple rounds of editing before and after they are published. It is important to keep your code as tidy as possible so that you can easily read and change them in the future. Also, after creating your documents, it is important to test them to ensure that visitors to your site will not encounter any unforeseen problems. This chapter covers the basics of testing your code, including what tools are at your disposal.

IN THIS CHAPTER

Tidying Your HTML Code

One important step while developing HTML documents is to keep the code *tidy*. Tidy code is code that is kept orderly with logical line breaks and intelligent indentation. Although this may not seem like a crucial issue, it takes only small snippets of code to see the difference, as illustrated in Listings 23-1 and 23-2.

LISTING 23-1

Untidy code

```
<table border="1" rules="all">
<tr><td><img src="./images/JillHennessy.jpg" width="100px"
height="150px" alt="Jill" /><br />Jill Hennessy</td><td>Jill is
the founder of the Oasis of Tranquility and its executive
officer. She oversees the business side of the spa and operates
as its chief recruiting officer. Jill has degrees in business
and cosmetology.</td></tr><tr><td><img
src="./images/SandraBrown.jpg" width="100px" height="150px"
alt="Sandra" /><br />Sandra Brown</td><td>Sandra is the
co-founder of the Oasis and its operating officer. She is
```

continued

LISTING 23-1 *(continued)*

```
responsible for the day-to-day operation of the spa and
management of its employees. Sandra has journalism and
management degrees and several accreditations from various
cosmetology schools.</td></tr><tr><td><img
src="./images/DamienSanders.jpg" width="100px" height="150px"
alt="Damien" /><br />Damien Sanders</td><td>Damien is the lead
stylist at the Oasis. He has worked at several of the most
esteemed salons in Hollywood and has several degrees and
accreditations in hair design from schools across the nation.
Damien was one of two stylists invited to travel with a popular
modern burlesque group on their last tour.</td></tr><tr><td><img
src="./images/MartyTowers.jpg" width="100px" height="150px"
alt="Marty" /><br />Marty Towers</td><td>Marty is the Oasis
makeover specialist. She has several degrees and accreditations
to her credit and is known as one of the top makeup and nail
color specialists in the country. She was recently invited on a
nationally syndicated talk show to discuss women's image and
skin and nail techniques.</td></tr><tr><td><img
src="./images/TaliaOwens.jpg" width="100px" height="150px"
alt="Talia" /><br />Talia Owens</td><td>Talia is the lead
masseuse for the Oasis. She has over 2000 hours of instruction
in various massage techniques--from deep tissue to
neuromuscular--and holds certifications in each. Talia is also
an accredited acupuncturist and aromatherapy
specialist.</td></tr><tr><td><img src="./images/ThomasBaker.jpg"
width="100px" height="150px" alt="Thomas" /><br />Thomas
Baker</td><td>Thomas rounds out the massage team here at the
Oasis. He has over 1500 hours of instruction in several massage techniques
and is accredited in several. Thomas also holds a degree
in sports therapy and rehabilitation.</td></tr>
</table>
```

LISTING 23-2

Tidy code

```
<table border="1" rules="all">
<tr>
  <td><img src="./images/JillHennessy.jpg" width="100px"
        height="150px" alt="Jill" />
  <br />Jill Hennessy</td>
  <td>Jill is the founder of the Oasis of Tranquility and its
executive officer. She oversees the business side of the spa and
operates as its chief recruiting officer. Jill has degrees in
business and cosmetology.</td>
</tr>
```

```
<tr>
  <td><img src="./images/SandraBrown.jpg" width="100px"
        height="150px" alt="Sandra" />
  <br />Sandra Brown</td>
  <td>Sandra is the co-founder of the Oasis and its operating
officer. She is responsible for the day-to-day operation of the
spa and management of its employees. Sandra has journalism and
management degrees and several accreditations from various
cosmetology schools.</td>
</tr>
<tr>
  <td><img src="./images/DamienSanders.jpg" width="100px"
        height="150px" alt="Damien" />
  <br />Damien Sanders</td>
  <td>Damien is the lead stylist at the Oasis. He has worked at
several of the most esteemed salons in Hollywood and has several
degrees and accreditations in hair design from schools across
the nation. Damien was one of two stylists invited to travel
with a popular modern burlesque group on their last tour.</td>
</tr>
<tr>
  <td><img src="./images/MartyTowers.jpg" width="100px"
        height="150px" alt="Marty" />
  <br />Marty Towers</td>
  <td>Marty is the Oasis makeover specialist. She has several
degrees and accreditations to her credit and is known as one of
the top makeup and nail color specialists in the country. She
was recently invited on a nationally syndicated talk show to
discuss women's image and skin and nail techniques.</td>
</tr>
<tr>
  <td><img src="./images/TaliaOwens.jpg" width="100px"
        height="150px" alt="Talia" />
  <br />Talia Owens</td>
  <td>Talia is the lead masseuse for the Oasis. She has over
2000 hours of instruction in various massage techniques--from
deep tissue to neuromuscular--and holds certifications in each.
Talia is also an accredited acupuncturist and aromatherapy
specialist.</td>
</tr>
<tr>
  <td><img src="./images/ThomasBaker.jpg" width="100px"
        height="150px" alt="Thomas" />
  <br />Thomas Baker</td>
  <td>Thomas rounds out the massage team here at the Oasis. He
has over 1500 hours of instruction in several massage techniques
and is accredited in several. Thomas also holds a degree in
sports therapy and rehabilitation.</td>
</tr>
</table>
```

As you can see, with the table elements isolated by line breaks and indentations, the table is easier to read and troubleshoot. Although tables are some of the more complex and problematic elements, many HTML elements can become exponentially harder to read and troubleshoot without liberal formatting.

The act of tidying your code is simply adding the additional spacing, indentation, and other formatting to your HTML code in order to make it readable. Ideally, you would add this formatting as you write your code, but should you need to do so after the fact, there are a few ways, and one powerful tool, to help add the formatting later.

Note

When writing tidy code it is important to be careful of where you insert blank lines and spaces. Although important for readability, both of these entities can change the format of your final document, usually for the worse. As a general rule, always insert white space between closing and opening tags. For example, when adding space to table cells, place your white space between the ending tag of one cell (`</td>`) and the beginning tag of the next cell (`<td>`). Any white space in this area will simply be ignored by a user agent, and as such will not change the formatting of your document.

Consistency is also important while formatting your documents so you know exactly where to expect elements and when. ■

HTML Tidy

The HTML Tidy tool was created several years ago by Dave Raggett and maintained for several years by Dave and various entities at the W3C. HTML Tidy was originally created for two reasons:

- To clean up HTML code, adding liberal white space and indentation to help increase the readability of the code (also making it easier to troubleshoot)
- To check the HTML for basic errors — missing tags, inappropriate attributes, and so on

Today, the program has been taken over by a group of "enthusiastic volunteers" at Source Forge, where it is now actively maintained.

Note

The HTML Tidy home page is now at `http://tidy.sourceforge.net`. ■

Getting HTML Tidy

HTML Tidy exists mostly as an executable available for Linux/UNIX platforms and is downloadable from the HTML Tidy home page. Additional HTML Tidy projects have included the following:

- Java version of Tidy
- Perl XS version of Tidy
- Python wrapper for TidyLib
- HTMLTrim, a highly customizable X(HTML)/XML pretty-printer and fixer for Windows

- Jase, a simple editor with TidyLib integration
- mod_tidy for Apache 2

There are also ad hoc versions compiled for Windows, OS/2, and MAC OS (classic and OS X).

Running HTML Tidy

Running HTML Tidy is straightforward; you simply run the executable with an HTML file as an argument. For example, the following command line could be used to run our earlier non-tidy table example (refer to Listing 23-1) through Tidy:

```
tidy tableexample.html
```

Tidy, in return, provides a tidied example of the file, along with a few hints for our document, as shown in Listing 23-3.

LISTING 23-3

Tidy output

```
line 8 column 1 - Warning: <table> lacks "summary" attribute
Info: Doctype given is "-//W3C//DTD HTML 4.01//EN"
Info: Document content looks like HTML 4.01 Strict
1 warning, 0 errors were found!
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta name="generator" content=
"HTML Tidy for Linux/x86 (vers 12 April 2005), see www.w3.org">
<title>Using Tidy</title>
</head>
<body>
<table border="1" rules="all">
<tr>
<td><img src="./images/JillHennessy.jpg" width="100" height="150"
alt="Jill" /><br>
Jill Hennessy</td>
<td>Jill is the founder of the Oasis of Tranquility and its
executive officer. She oversees the business side of the spa and
operates as its chief recruiting officer. Jill has degrees in
business and cosmetology.</td>
</tr>
<tr>
<td><img src="./images/SandraBrown.jpg" width="100" height="150"
alt="Sandra" /><br>
Sandra Brown</td>
<td>Sandra is the co-founder of the Oasis and its operating
officer. She is responsible for the day-to-day operation of the spa
```

continued

LISTING 23-3 *(continued)*

```
and management of its employees. Sandra has journalism and
management degrees and several accreditations from various
cosmetology schools.</td>
</tr>
<tr>
<td><img src="./images/DamienSanders.jpg" width="100" height="150"
alt="Damien" /><br>
Damien Sanders</td>
<td>Damien is the lead stylist at the Oasis. He has worked at
several of the most esteemed salons in Hollywood and has several
degrees and accreditations in hair design from schools across the
nation. Damien was one of two stylists invited to travel with a
popular modern burlesque group on their last tour.</td>
</tr>
<tr>
<td><img src="./images/MartyTowers.jpg" width="100" height="150"
alt="Marty" /><br>
Marty Towers</td>
<td>Marty is the Oasis makeover specialist. She has several
degrees and accreditations to her credit and is known as one of the
top makeup and nail color specialists in the country. She was
recently invited on a nationally syndicated talk show to discuss
women's image and skin and nail techniques.</td>
</tr>
<tr>
<td><img src="./images/TaliaOwens.jpg" width="100" height="150"
alt="Talia" /><br>
Talia Owens</td>
<td>Talia is the lead masseuse for the Oasis. She has over 2000
hours of instruction in various massage techniques--from deep
tissue to neuromuscular--and holds certifications in each. Talia
is also an accredited acupuncturist and aromatherapy
specialist.</td>
</tr>
<tr>
<td><img src="./images/ThomasBaker.jpg" width="100" height="150"
alt="Thomas" /><br>
Thomas Baker</td>
<td>Thomas rounds out the massage team here at the Oasis. He has
over 1500 hours of instruction in several massage techniques and is
accredited in several. Thomas also holds a degree in sports therapy
and rehabilitation.</td>
</tr>
</table>
</body>
</html>
```

Note the various lines at the beginning of the output: The table summary attribute should be used to describe the table structure. It is very helpful for people using nonvisual browsers. The scope and headers attributes for table cells are useful for specifying which headers apply to each table cell, enabling nonvisual browsers to provide a meaningful context for each cell.

For further advice on how to make your pages accessible, see `www.w3.org/WAI/GL`. You may also want to try `www.cast.org/bobby`, which is a free Web-based service that checks URLs for accessibility.

To learn more about HTML Tidy, see `http://tidy.sourceforge.net`. Please send bug reports to html-tidy@w3.org. HTML and CSS specifications are available from `www.w3.org`.

Lobby your company to join W3C — see `www.w3.org/Consortium`.

As you can see from Listing 23-3 and the suggestion of the `<table>` tag's `summary` attribute, Tidy is a bit aggressive in its diagnostics of documents. Thankfully, with several dozen options at your disposal, you can tailor Tiny's behavior to your liking.

To get a list of Tidy's options, either read the documentation or run Tidy with the help parameter:

```
tiny -h
```

Note

As with validation tools (covered in the next section), it is important to include a valid `DOCTYPE` in all the documents being run through Tidy. ■

Validating Your Code

Validating your document code is a very good idea. It helps double-check your document for simple errors — typos, unclosed tags, and so on — and verifies that your code meets expected standards.

Specifying the correct document type declaration

There are many ways to validate your documents, but they all rely on your documents containing a correct document type declaration that references a specific document type definition (DTD). For example, if you want to base your documents on Strict HTML 4.01, you would include the following document type declaration at the top of your document:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
```

The `DOCTYPE` declaration informs any user agent reading the document on what standard the document is based. The information is primarily used by validation clients in validating the code within the document, but it might also be used by a display agent to determine what features it must support.

Tip

You can find a list of valid DTDs at `www.w3.org/QA/2002/04/valid-dtd-list.html`. ■

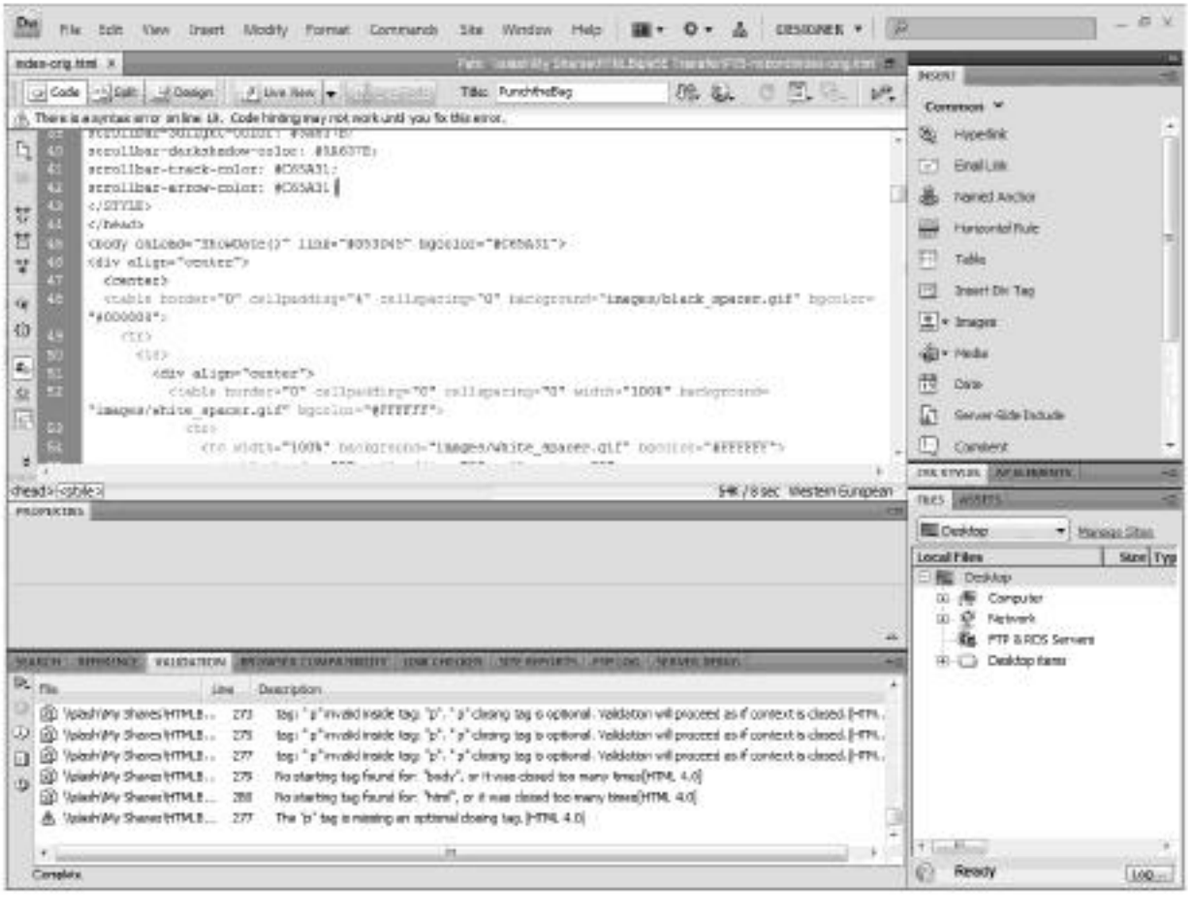
Validation tools

You can use several tools to validate your documents. Tools at your disposal include the following:

- The online W3C HTML validation tool, found at `http://validator.w3.org/`
- The online Web Design Group (WDG) validation tool, found at `http://htmlhelp.com/tools/validator/`
- Validation utilities built into Web development tools such as Adobe's Dreamweaver, shown in Figure 23-1
- Any of the various separate applications that can be run locally. A comprehensive list is maintained on the WDG site at `www.htmlhelp.com/links/validators.htm`

FIGURE 23-1

Adobe's Dreamweaver includes a comprehensive code validation feature.



Understanding validation output

Consider the following HTML document:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <title>Validation Test</title>
</head>
<body>
  <form action="" method="POST">
    <input name="text" type="text" />
    <br>
    <input name="submit" type="submit" />
  </form>
</body>
</html>
```

When this code is passed through the W3C Markup Validation Service, the following first error is returned:

```
Line 9, column 30: document type does not allow element
"INPUT" here; missing one of "P", "H1", "H2", "H3", "H4",
"H5", "H6", "PRE", "DIV", "ADDRESS" start-tag
  <input name="text" type="text">
```

Although the document looks to be conforming HTML, the validation service thinks otherwise. But what exactly does the error mean?

In short, it means that the `input` element must be contained within a block element other than the `form` tag. Typically, the paragraph tag (`<p>`) is used, but you can also use `<div>`, a heading, `<pre>`, and so on.

Note

The W3C also has an online CSS validation tool, accessible at `http://validator.w3.org/`. Similar to the HTML validation tool, this tool will ensure that your CSS is free from typos and that all the attributes are paired with their matching styles. ■

Adding a paragraph container solves the problem and makes the document valid:

```
...
  <form action="" method="POST">
    <p>
    <input name="text" type="text" />
    <br>
    <input name="submit" type="submit" />
    </p>
  </form>
...
```

Tip

When working on making a document validate, always handle the errors in order. The example in this section actually results in four separate errors, each relating to the missing block elements. Adding the preceding elements solves all four problems. ■

Additional Testing and Validation

Besides using prefab Tidy and validation services, you can run other quick tests on your code to ensure that it runs well in the real world.

Testing with a variety of user agents

Despite being built on standards, no two user agents support HTML and CSS to the same degree. Some user agents don't implement certain features, while others implement them differently.

Note

Contrary to popular belief, Microsoft's Internet Explorer is no worse than other user agents regarding supporting standards. Even though Microsoft has created many proprietary technologies for its user agent, it does only a fair job of supporting the actual standards. ■

When coding your documents, it is important to understand your expected audience and what user agents they may be using. Although Microsoft Internet Explorer has market share on its side, many people use other user agents, such as Firefox, Opera, Konqueror, Safari, and so forth. As such, it is doubtful that everyone will be able to view your documents the way you originally intended, especially if you use some of the more esoteric features and technologies.

Make sure you test your pages on all target platforms to ensure that no show-stopping errors exist on any of the platforms. At a bare minimum, you should test on a current Microsoft (Internet Explorer) browser and a Firefox/Mozilla browser because most user agents incorporate one of these two technology bases.

Also, don't forget the non-computer user agents used by cell phones, PDAs, and other mobile devices. If your site will appeal to mobile device users, at least obtain the Software Development Kit (SDK) or emulator for each likely platform and preview your documents accordingly.

Testing for a variety of displays

Many Web designers make the mistake of designing their documents for specific screen resolutions. When the document is displayed at a smaller resolution, the page elements tend to jam together or break across unexpected lines.

Your documents should be suitable for many resolutions. Although most users will be running at resolutions of at least 800 × 600 pixels, you may have the occasional user running lower resolutions.

Always test your documents at various resolutions and color depths to look for any shortcomings.

Summary

This chapter taught you the importance of keeping your code tidy and consistent, and how to validate it using tools and a lot of testing. The next chapter wraps up the HTML coverage by providing a handful of tips and tricks you can employ in your documents. The next part of this book covers CSS in topical chapters.

CHAPTER 24

HTML Tips and Tricks

Throughout this book, you have read about the ins and outs of the various HTML tags and entities. This chapter covers a few tips and tricks you can use to supplement your HTML knowledge to achieve real-world results.

IN THIS CHAPTER

Preloading Images

One of the things that can really slow down the display of Web pages is an abundance of images, each of which can contain the equivalent of 17,000 to 20,000 characters.

A trick that was developed to help overcome the delays experienced while image-rich documents load is *image preloading*. Through the use of JavaScript, image files are loaded into image objects. The net result is that the graphics are not displayed but are loaded into the browser's cache for later use. When it is time for the browser to actually display the image(s), they are taken from the local cache instead of having to make the trip across the Internet.

The script embedded in the following document is an example of an image preload script:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <title>Preloading Images</title>
  <script type="text/JavaScript">
  // Assign path of images to be preloaded to
  //   array, one image per index
```

```
  var imagenames = [];
  imagenames[0]  = "images/header.gif";
  imagenames[1]  = "images/logo.jpg";
  imagenames[2]  = "images/picture1.gif";
  imagenames[3]  = "images/picture2.gif";
  imagenames[4]  = "images/picture3.gif";
  imagenames[5]  = "images/rule.gif";
  imagenames[6]  = "images/button01.gif";
  imagenames[7]  = "images/button02.gif";
  imagenames[8]  = "images/button03.gif";
  imagenames[9]  = "images/footer.gif";
  imagenames[10] = "images/gradient.gif";
  imagenames[11] = "images/sphere.gif";
  // Create new image object for each image
  //   and then assign a path to the src, preloading
  //   the actual image
  function preload(imagenames) {
        var images = [];
        for (var i=0; i < imagenames.length; i++) {
              images[i] = new Image();
              images[i].src = imagenames[i];
        }
  }
  // Run script, preloading images, before document loads
  //   (alternately, place call in an onLoad in body tag to
  //   preload images after document loads
  preload(imagenames);
  </script>
</head>
<body>
  <p>Document body goes here.</p>
</body>
</html>
```

The script builds an array of image paths and then iterates through them, creating an image object and assigning an `src` property to cause the image to actually be preloaded. The script can be run via two different means: by a function call in the `head section`, which causes the script to run before the document loads, or by an `onLoad` handler in the `<body>` tag, which causes the script to run after the document loads.

Note

Image preloaders aren't useful for individual documents; they are most useful for sites of multiple documents that reuse the same images repeatedly (buttons, rules, backgrounds, and so on). When seeding the loader, don't forget to include images from all documents your audience may see. ■

The former, before the document loads, is handy when the document itself contains many images — running the preloader first can speed the display of the initial document. The latter, after the document loads, is a better choice when subsequent documents contain the majority of images. This enables the initial document to load more quickly because it doesn't have to wait