GETTING STARTED

PART I

IN THIS PART

CHAPTER 1

WHERE DO I START?

The Web has been around for more than 20 years now, experiencing euphoric early expansion, an economic-driven bust, an innovation-driven rebirth, and constant evolution along the way. One thing is certain: the Web as a communication and commercial medium is here to stay. Not only that, it has found its way onto devices such as smartphones, tablets, TVs, and more. There have never been more opportunities to put web design know-how to use.

Through my experience teaching web design courses and workshops, I've had the opportunity to meet people of all backgrounds who are interested in learning how to build web pages. Allow me to introduce you to just a few:

> *"I've been a print designer for 17 years, and now I am feeling pressure to provide web design services."*
>
> *"I work as a secretary in a small office. My boss has asked me to put together a small internal website to share company information among employees."*
>
> *"I've been a programmer for years, but I want to try my hand at design. I feel like the Web is a good opportunity to explore new skills."*
>
> *"I am an artist and I want to know how to get samples of my paintings and sculpture online."*
>
> *"I tinkered with web pages in high school and I think it might be something I'd like to do for a living."*

Whatever the motivation, the first question is always the same: "Where do I start?" It may seem like there is a mountain of stuff to learn, and it's not easy to know where to jump in. But you have to start somewhere.

This chapter attempts to put the learning curve in perspective by answering the most common questions I get asked by people ready to make the leap. It provides an introduction to the disciplines, technologies, and tools associated with web design.

IN THIS CHAPTER

Where Do I Start?

Your particular starting point will no doubt depend on your background and goals. However, a good first step for everyone is to get a basic understanding of how the Web and web pages work. This book will give you that foundation. Once you learn the fundamentals, there are plenty of resources on the Web and in bookstores for you to further your learning in specific areas.

There are many levels of involvement in web design, from building a small site for yourself to making it a full-blown career. You may enjoy being a full-service website developer or just specializing in one skill. There are a lot of ways you can go.

If your involvement in web design is purely at the hobbyist level, or if you have just one or two web projects you'd like to publish, you may find that a combination of personal research (like reading this book), taking advantage of available templates, and perhaps even investing in a visual web design tool such as Adobe Dreamweaver may be all you need to accomplish the task at hand. Many Continuing Education programs offer introductory courses to web design and production.

If you are interested in pursuing web design or production as a career, you'll need to bring your skills up to a professional level. Employers may not require a web design degree, but they will expect to see working sample sites that demonstrate your skills and experience. These sites can be the result of class assignments, personal projects, or a simple site for a small business or organization. What's important is that they look professional and have well-written, clean HTML, style sheets, and possibly scripts behind the scenes. Getting an entry-level job and working as part of a team is a great way to learn how larger sites are constructed and can help you decide which aspects of web design you would like to pursue.

I Just Want a Blog!

You don't necessarily need to become a web designer to start publishing your words and pictures on the Web. You can start your own "blog" or personal journal site using one of the free or inexpensive blog hosting services. These services provide templates that generally spare you the need to learn HTML (although it still doesn't hurt). These are some of the most popular as of this writing:

- WordPress (*www.wordpress.com*)
- Blogger (*www.blogger.com*)
- Tumblr (*www.tumblr.com*)

Another drag-n-drop site design and hosting service that goes beyond the blog is Squarespace (*www.squarespace.com*).

What Does a Web Designer Do?

Over the years, the term "web design" has become a catchall for a process that encompasses a number of different disciplines, from user experience design, to document markup, to serious programming. This section describes some of the most common roles.

If you are designing a small website on your own, you will need to wear many hats. The good news is that you probably won't notice. Consider that the day-to-day upkeep of your household requires you to be part-time chef, housecleaner, accountant, diplomat, gardener, and construction worker—but to you it's just the stuff you do around the house. In the same way, as a solo web designer, you may be a part-time graphic designer, writer, HTML author, and information architect, but to you, it'll just feel like "making web pages." Nothing to worry about.

AT A GLANCE

The term "web design" has come to encompass a number of disciplines, including:

- Visual (graphic) design
- User interface and experience design
- Web document and style sheet production
- Scripting and programming
- Content strategy
- Multimedia

There are also specialists out there whom you can hire to fill in the skills you don't have. For example, I have been creating websites since 1993 and I still hire programmers and multimedia developers when my clients require interactive features. That allows me to focus on the parts I do well (in my case, it's the content organization, interface, and visual design).

If you are not interested in becoming a jack-of-all-trades solo web designer, you may choose to specialize and work as part of a team or as a freelance contractor.

Large-scale websites are almost always created by a team of people, numbering from a handful to hundreds. In this scenario, each member of the team focuses on one facet of the site-building process. If that is the case, you may be able to simply adapt your current set of skills (writing, Photoshop, programming, etc.) and interests to the new medium.

I've divided the myriad roles and responsibilities typically covered under the umbrella term "web design" into four very broad categories: design, development, content strategy, and multimedia.

Design

Ah, design! It sounds fairly straightforward, but even this simple requirement has been divided into a number of specializations when it comes to creating sites. Here are a few of the new job descriptions related to designing a site, but bear in mind that the disciplines often overlap and that the person calling herself the "Designer" often is responsible for more than one (if not all) of these responsibilities.

User Experience, Interaction, and User Interface design

Often, when we think of design, we think about how something looks. On the Web, the first matter of business is designing how the site *works*. Before picking colors and fonts, it is important to identify the site's goals, how it will be used, and how visitors move through it. These tasks fall under the disciplines of Interaction Design (IxD), User Interface (UI) design, and User Experience (UX) design. There is a lot of overlap between these responsibilities, and it is not uncommon for one person or team to handle all three.

The goal of the Interaction Designer is to make the site as easy, efficient, and delightful to use as possible. Closely related to interaction design is User Interface design, which tends to be more narrowly focused on the functional organization of the page as well as the specific tools (buttons, links, menus, and so on) that users use to navigate content or accomplish tasks.

A more recent job title in the web design realm is the User Experience Designer. The UX designer takes a more holistic view—ensuring the entire experience with the site is favorable. UX design is based on a solid understanding of users and their needs based on observations and interviews. According to Donald Norman (who coined the term), user experience design includes "all aspects of the user's interaction with the product: how it is perceived, learned, and used." For a website or application, that includes

the visual design, the user interface, the quality and message of the content, and even overall site performance. The experience must be in line with the organization's brand and business goals in order to be successful.

Some of the documents an IxD, UI, or UX designer might produce include:

User research and testing reports

Understanding the needs, desires, and limitations of users is central to the success of the design of the site or web application. This approach of designing around the user's needs is referred to as User Centered Design (UCD), and it is central to contemporary design. Site designs often start with user research, including interviews and observations, in order to gain a better understanding of how the site can solve problems or how it will be used. It is typical for designers to do a round of user testing at each phase of the design process to ensure the usability of their designs. If users are having a hard time figuring out where to find content or how to move to the next step in a process, then it's back to the drawing board.

Wireframe diagrams

A wireframe diagram shows the structure of a web page using only outlines for each content type and widget (Figure 1-1). The purpose of a wireframe diagram is to indicate how the screen real estate is divided and indicate where functionality and content such as navigation, search boxes, form elements, and so on, are placed, without any decoration or graphic design. They are usually annotated with instructions for how things should work so the development team knows what to build.

Site diagram

A site diagram indicates the structure of the site as a whole and how individual pages relate to one another. Figure 1-2 shows a very simple site diagram. Some site diagrams fill entire walls!

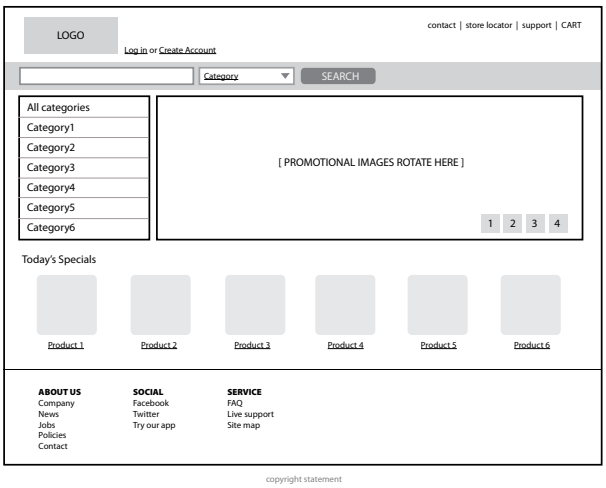


Figure 1-1. Wireframe diagram.

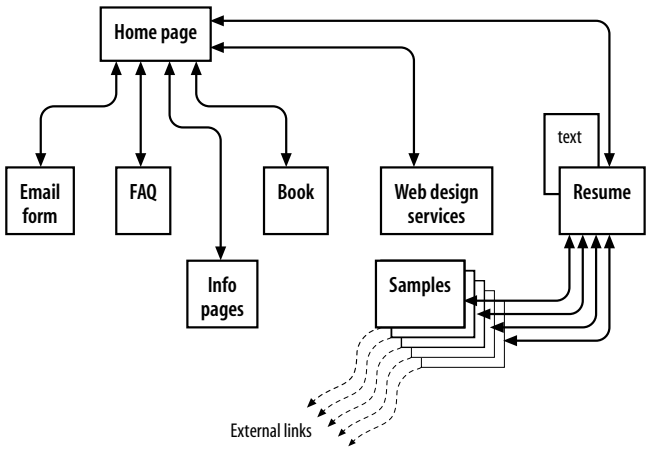


Figure 1-2. A simple site diagram.

Storyboards and user flow charts

A storyboard traces the path through a site or application from the point of view of a typical user (a persona in UX lingo). It usually includes a script and "scenes" consisting of screen views or the user interacting with the screen. The storyboard aims to demonstrate the steps it takes to accomplish tasks, possible options, and also introduces some standard page types. Figure 1-3 shows a simple storyboard. A user flow chart is another method for showing how the parts of a site or application are connected that tends to focus on technical details rather than telling a story. For example, when the user does this, it triggers that function on the server. It is common for designers to create a user flow chart for the steps in a process such as member registration or online payments.

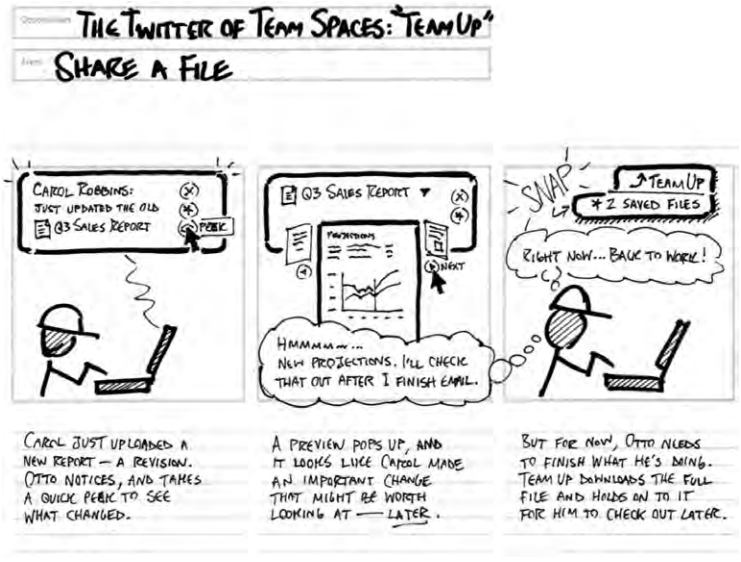


Figure 1-3. A typical storyboard [courtesy of Adaptive Path; drawn by Brandon Schauer].

Visual (graphic) design

Because the Web is a visual medium, web pages require attention to presentation and design. A graphic designer creates the "look and feel" of the site—logos, graphics, type, colors, layout, etc.—to ensure that the site makes a good first impression and is consistent with the brand and message of the organization it represents. Visual designers typically generate sketches of the way the site might look, as shown in Figure 1-4. They may also be responsible for producing the graphic files in a way that is optimized for delivery over the Web (see Chapter 21, Lean and Mean Web Graphics for image optimization techniques).

If you are interested in doing the visual design of commercial sites professionally, I strongly recommend graphic design training as well as a strong proficiency in Adobe Photoshop (the industry standard) or Adobe Fireworks.

Figure 1-4. Look and feel sketches for a simple site.

If you are already a graphic designer, you will be able to adapt your skills to the Web easily, although this will not excuse you from acquiring a solid understanding of HTML, CSS, and other web technologies. Because most sites have at least a few images, even hobbyist web designers will need to know how to create and edit images, at minimum.

Again, I want to note that all of these responsibilities may fall into the hands of one designer who creates both the look and the functionality of a site. But for larger sites with bigger budgets, there is an opportunity to find your own special niche in the design process.

Development

A fair amount of the web design process involves the creation and troubleshooting of the documents, style sheets, scripts, and images that make up a site. At web design firms, the team that handles the creation of the files that make up the website (or templates for pages that get assembled dynamically) is usually called the development or production department.

Web developers may not design the look or structure of the site themselves, but they do need to communicate well with designers and understand the intended site goals so they may suggest solutions that meet those goals.

The broad disciplines that fall under development are authoring, styling, and scripting/programming.

Authoring/markup

Authoring is the term used for the process of preparing content for delivery on the Web, or more specifically, marking up the content with HTML tags that describe its content and function. If you want a job as a web developer, you need to have an *intricate* knowledge of HTML and how it functions on various browsers and devices. The HTML specification is constantly evolving, which means you'll need to keep up with the latest best practices and opportunities as well as bugs and limitations. The good news is, it's not difficult to get started, and from there, you can gradually increase your skills. We'll be dabbling with HTML in Chapter 2, How the Web Works and then discussing it in great detail in Part II of this book.

Styling

In web design, the appearance of the page in the browser is controlled by style rules written in CSS (Cascading Style Sheets). We'll get deep into CSS in Part III of this book (including what "cascading" means!), but for now just know that in contemporary web design, the appearance of the page is handled separately from the HTML markup of the page. Again, if you are interested in working in web development, knowing your way around CSS and how it is supported (or not supported) by browsers is guaranteed to be part of your job description.

Style Tiles

Another approach to capturing the look and feel of a site is to create style tiles, which give examples of color schemes, branding elements, content and UI treatments, and mood boards without applying them to a specific page layout. The idea is to agree upon a consistent visual language for the site. For more on this technique, read the article "Style Tiles and How They Work," by Samantha Warren (*www.alistapart.com/articles/style-tiles-and-how-they-work/*), and visit her excellent site where you can download a template at *styletil.es*.

NOTE

Many visual designers translate their designs into HTML and CSS documents themselves. In fact, there is a popular argument that in order to call yourself a "web designer," you must be able to build your designs yourself, and nearly everyone agrees that your job prospects will be better if you are able to code as well as design.

Scripting and programming

As the Web has evolved into a platform of applications for getting stuff done, programming has never been more important. JavaScript is the language that makes elements on web pages do things. It adds behaviors and functionality to elements in the page and even to the browser window itself.

There are other web-related programming languages as well, including PHP, Ruby, Python, and ASP.NET, that run on the server and process data and information before it is sent to the user's browser. See the sidebar "Frontend Versus Backend" for more information on what happens where.

Web scripting and programming definitely requires some traditional computer programming prowess. While many web programmers have degrees in computer science, it is also common for developers to be self-taught. A few developers I know started by copying and adapting existing scripts, then gradually added to their programming skills with each new project. Still, if you have no experience with programming languages, the initial learning curve may be a bit steep.

Teaching web programming is beyond the scope of this book. JavaScript is introduced in Chapter 19, Introduction to JavaScript (teaching JavaScript could fill a whole book itself). It is possible to turn out content-rich, well-designed sites without the need for programming, so hobbyist web designers should not be discouraged. However, once you get into collecting information via forms or serving information on demand, it is usually necessary to have a programmer on the team. You may also ask your hosting company if they offer the functionality you are looking for in an easy-to-use, canned service.

Frontend Versus Backend

You may hear web designers and developers say that they specialize in either the frontend or backend of website creation.

Frontend design

"Frontend" refers to any aspect of the design process that appears in or relates directly to the browser. This book focuses primarily on frontend web design.

The following tasks are commonly considered to be frontend tasks:

- Graphic design and image production
- Interface design
- Information design as it pertains to the user's experience of the site
- HTML document and style sheet development
- JavaScript

Backend development

"Backend" refers to the programs and scripts that work on the server behind the scenes to make web pages dynamic and interactive. In general, backend web development falls in the hands of experienced programmers, but it is good for all web designers to be familiar with backend functionality.

The following tasks take place on the backend:

- Information design as it pertains to how the information is organized on the server
- Forms processing
- Database programming
- Content management systems
- Other server-side web applications using PHP, JSP, Ruby, ASP.NET, Java, and other programming languages

Content strategy and creation

Third on our list, though ideally first in the actual website creation process, is the critical matter of the site's content itself. Anyone who uses the title "web designer" needs to be aware that everything we do supports the process of getting the content, message, or functionality to our users. Furthermore, good writing can help the user interfaces we create be more effective.

Of course, someone needs to create the content and maintain it—don't underestimate the resources required to do this successfully. In addition, I want to call your attention to two content-related specialists on the modern web development team: the Content Strategist and Information Architect (IA).

When the content isn't written right, the site can't be fully effective. A Content Strategist makes sure that every bit of text on a site, from long explanatory text down to the labels on buttons, supports the brand identity and marketing goals of the company. Content strategy may also extend to data modeling and content management on a large and ongoing scale, such as planning for content reuse and update schedules.

An Information Architect (also called an Information Designer) organizes the content logically and for ease of findability. She may be responsible for search functionality, site diagrams, and how the content and data is organized on the server. Information architecture is inevitably entwined with UX and UI design, and it is not uncommon for a single person or team to perform all roles.

Multimedia

One of the cool things about the Web is that you can add multimedia elements to a site, including sound, video, animation, and even interactive games. You may decide to add multimedia skills, such as audio and video editing or Flash development (see the "A Little More About Flash" sidebar), to your web design tool belt, or you may decide to go all in and become a multimedia specialist. If you are not interested in becoming a multimedia developer, you can always hire one. Web development companies usually look for people who have mastered the standard multimedia tools, and have a good visual sensibility and an instinct for intuitive and creative multimedia design.

A Little More About Flash

Adobe Flash (previously Macromedia Flash, previously FutureSplash) is a multimedia format created especially for the Web. Flash is used for create full-screen animation, interactive graphics, integrated audio and video clips, and even scriptable games and applications, all at remarkably small file sizes. However, recently Flash use has been on the decline due to a number of developments, including:

- Apple's decision not to support Flash on its iPhones and iPads in favor of non-proprietary HTML5 methods.
- Adobe's decision to stop supporting Flash (its own product) for mobile browsers.
- The new programmable **canvas** element in HTML5 that offers some of the same functionality as Flash.
- Criticism that Flash sometimes gets in the way of user goals. For example, who wants to sit through a movie and soundtrack on a restaurant site when all you really want to know is whether they are open on Sunday?
- The fact that a plug-in is required to play Flash media makes some developers squeamish.

In fact, it is not uncommon to hear web professionals cite that "Flash is dead," but despite suddenly becoming the underdog, Flash still has some advantages if used the right way:

- Because it uses vector graphics, Flash files are small and can be resized without loss of detail.
- It is a streaming format, so movies start playing quickly and continue to play as they download.
- You can use ActionScript to add behaviors and advanced interactivity, allowing Flash to be used as the frontend for dynamically generated content or ecommerce functions.
- The Flash plug-in is well-distributed on PCs, so support on desktop browsers is reliable.
- Although HTML5 is promising and rapidly evolving, as of this writing, it cannot match the features and performance of Flash.

Flash is not likely to disappear overnight, but even Adobe is putting its muscle behind HTML5 alternatives.

What Languages Do I Need to Learn?

If you are a visual designer who spends time in Photoshop and Illustrator, you may be put off by needing to learn how to create your designs with text, but I assure you, it's pretty simple to get started. There are also authoring tools that speed up the production process, as we'll discuss later in this chapter.

The following is a list of technologies associated with web development. Which languages and technologies you learn will depend on the role you see yourself in within the web design process. However, I advise *everyone* involved in building websites to know their way around HTML and Cascading Style Sheets, and if you want to do frontend web development for a living, JavaScript know-how is pretty much a job requirement. More technically inclined web professionals may take on server configurations, databases, and site performance, but these are generally not frontend developer tasks (although a basic familiarity with the backend issues never hurts).

AT A GLANCE

Web-related technologies:

- Hypertext Markup Language (HTML)
- Cascading Style Sheets (CSS)
- JavaScript and DOM scripting
- Server-side programming and database management

The World Wide Web Consortium

The World Wide Web Consortium (called the W3C for short) is the organization that oversees the development of web technologies. The group was founded in 1994 by Tim Berners-Lee, the inventor of the Web, at the Massachusetts Institute of Technology (MIT).

In the beginning, the W3C concerned itself mainly with the HTTP protocol and the development of the HTML. Now, the W3C is laying a foundation for the future of the Web by developing dozens of technologies and protocols that must work together in a solid infrastructure.

For the definitive answer on any web technology question, the W3C site is the place to go:

www.w3.org

For more information on the W3C and what they do, see this useful page:

www.w3.org/Consortium/

Hypertext Markup Language (HTML)

You may see HTML and XHTML referred to collectively as (X)HTML.

HTML (HyperText Markup Language) is the language used to create web page documents. There are a few versions of HTML in use today: HTML 4.01 is the most firmly established and the newer, more robust HTML5 is gaining steam and browser support. Both versions have a stricter implementation called XHTML (eXtensible HTML), which is essentially the same language with much stricter syntax rules. We'll get to the particulars of what makes the various versions different in Chapter 10, What's Up, HTML5?.

HTML is not a programming language; it is a markup language, which means it is a system for identifying and describing the various components of a document such as headings, paragraphs, and lists. The markup indicates the document's underlying structure (you can think of it as a detailed, machine-readable outline). You don't need programming skills—only patience and common sense—to write HTML.

The best way to learn HTML is to write out some pages by hand, as we will be doing in the exercises in this book. If you end up working in web production, you'll live and breathe HTML. But even hobbyists will benefit from knowing what is going on under the hood. The good news is that it's simple to learn the basics.

Cascading Style Sheets (CSS)

While HTML is used to describe the content in a web page, it is Cascading Style Sheets (CSS) that describe how that content should *look*. In the web design biz, the way the page looks is known as its presentation. That means fonts, colors, background images, line spacing, page layout, and so on... all controlled with CSS. With the newest version (CSS3), you can even add special effects and basic animation to your page.

CSS also provides methods for controlling how documents will be presented in contexts other than the traditional desktop browser, such as in print and or on devices with small screen widths. It also has rules for specifying the non-visual presentation of documents, such as how they will sound when read by a screen reader (although those are not well supported).

Style sheets are also a great tool for automating production because you can change the way an element looks across all the pages in your site by editing a single style sheet document. Style sheets are supported to some degree by all modern browsers.

NOTE

When this book says "style sheets" it is always referring to Cascading Style Sheets, the standard style sheet language for the World Wide Web.

Although it is possible to publish web pages using HTML alone, you'll probably want to take on style sheets so you're not stuck with the browser's default styles. If you're looking into designing websites professionally, proficiency at style sheets is mandatory.

Style sheets are discussed further in Part III.

JavaScript/DOM scripting

JavaScript is a scripting language that is used to add interactivity and behaviors to web pages, including these (just to name a few):

- Checking form entries for valid entries
- Swapping out styles for an element or an entire site
- Making the browser remember information about the user for the next time she visits
- Building interface widgets, such as expanding menus

JavaScript is used to manipulate the elements on the web page, the styles applied to them, or even the browser itself. There are other web scripting languages, but JavaScript (also called ECMAScript) is the standard and most ubiquitous.

You may also hear the term DOM scripting used in relation to JavaScript. DOM stands for Document Object Model, and it refers to the standardized list of web page elements that can be accessed and manipulated using JavaScript (or another scripting language). DOM scripting is an updated term for what used to be referred to as DHTML (Dynamic HTML), now considered an obsolete approach.

Writing JavaScript is a type of programming, so it may be time-consuming to learn if you have no prior programming experience. Many people teach themselves JavaScript by reading books and following and modifying existing examples. Most web-authoring tools come with standard scripts that you can use right out of the box for common functions.

Professional web developers are required to know JavaScript, however, plenty of visual designers rely on developers to add behaviors to their designs. So while JavaScript is useful, learning to write it may not be mandatory for *all* web designers. Teaching JavaScript is outside the scope of this book; I recommend *Learning JavaScript* by Shelley Powers (O'Reilly, 2006) as a good starting place if you want to learn more.

The Web Design Layer Cake

Contemporary web design is commonly visualized as being made up of three separate "layers."

The content of the document with its (X)HTML markup makes up the Structure Layer. It forms the foundation upon which the other layers may be applied.

Once the structure of the document is in place, you can add style sheets to control how the content should appear. This is called the Presentation Layer.

Finally, the Behavior Layer includes the scripts that make the page an interactive experience.

Server-side programming

Some simple websites are collections of static HTML documents and image files, but most commercial sites have more advanced functionality such as forms handling, dynamically generated pages, shopping carts, content management systems, databases, and so on. These functions are handled by web applications running on the server. There are a number of programming languages and frameworks (listed in parentheses) that are used to create web applications, including:

- PHP (CakePHP, CodeIngniter, Drupal)
- Python (Django, TurboGears)

- Ruby (Ruby on Rails, Sinatra)
- JavaScript (Node.js, Rhino, SpiderMonkey)
- Java (Grails, Google Web Toolkit, JavaServer Faces)
- ASP.Net (DotNetNuke, ASP.Net MVC)

Developing web applications is programmer territory and is not expected of all web designers. However, that doesn't mean you can't offer such functionality to your clients. It is possible to get shopping carts, content management systems, mailing lists, and blogs as prepackaged solutions, without the need to program them from scratch.

What Do I Need to Buy?

It should come as no surprise that professional web designers require a fair amount of gear, both hardware and software. One of the most common questions I'm asked by my students is, "What should I get?" I can't tell you specifically what to buy, but I will provide an overview of the typical tools of the trade.

Bear in mind that while I've listed the most popular commercial software tools available, many of them have freeware or shareware equivalents that you can download if you're on a budget (try CNET's *Download.com*). With a little extra effort, you can get a full website up and running without big cash.

A Quick Introduction to XML

If you hang around the web design world at all, you're sure to hear the acronym XML (which stands for eXtensible Markup Language). XML is not a specific language in itself, but rather a robust set of rules for creating other markup languages.

To use a simplified example, if you were publishing recipes, you might use XML to create a custom markup language that includes the elements `<ingredient>`, `<instructions>`, and `<servings>` that accurately describe the types of information in your recipe documents. Once labeled correctly, that information can be treated as data. In fact, XML has proven to be a powerful tool for sharing data between applications. Despite the fact that XML was developed with the Web in mind, it has actually had a larger impact outside the web environment because of its data-handling capabilities. There are XML files working behind the scenes in an increasing number of software applications, such as Microsoft Office, Adobe Flash, and Apple iTunes.

Still, there are a number of XML languages that are used on the Web. The most prevalent is XHTML, which is HTML rewritten according to the stricter rules of XML (we'll talk more about XHTML in Chapter 10, What's Up, HTML5?). There is also RSS (Really Simple Syndication or RDF Site Summary), which allows your content to be shared as data and read with RSS feed readers; SVG (Scalable Vector Graphics), which uses tags to describe geometric shapes; and MathML, which is used to describe mathematical notation.

As a web designer, your direct experience with XML is likely to be limited to authoring documents in XHTML or perhaps adding an RSS feed or SVG images to a website. Developing new XML languages would be the responsibility of programmers or XML specialists.

Equipment

For a comfortable web development environment, I recommend the following equipment:

A solid, up-to-date computer. Macintosh, Windows, or Linux, is fine. Creative departments in professional web development companies tend to be Mac-based. Although it is nice to have a super-fast machine, the files that make up web pages are very small and tend not to be too taxing on computers. Unless you're getting into sound and video editing, don't worry if your current setup is not the very latest and greatest.

Extra memory. Because you'll tend to bounce between a number of applications, it's a good idea to have enough RAM installed on your computer that allows you to leave several memory-intensive programs running at the same time.

A large monitor. Although not a requirement, a large monitor makes life easier, particularly for a visual designer. (I've seen code-based developers get by just fine on an 11" MacBook Air.) The more monitor real estate you have, the more windows and control panels you can have open at the same time. You can also see more of your page to make design decisions.

If you're using large monitor, just make sure you design for users with smaller monitors and devices in mind.

A scanner and/or digital camera. If you anticipate making your own images and textures, you'll need some tools for creating them. I know a designer who has two scanners: one is the "good" scanner, and the other he uses to scan things like dead fish and rusty pans.

A second computer. Many web designers find it useful to have a test computer running a different platform than the computer they use for development (i.e., if you design on a Mac, test on a PC). Because browsers work differently on Macs than on Windows machines, it's critical to test your pages in as many environments as possible, and particularly on the current Windows operating system. If you are a hobbyist web designer working at home, check your pages on a friend's machine. Mac users should check out the "Run Windows on Your Mac" sidebar.

Mobile devices. The Web has gone mobile! That means it is absolutely critical that you test the appearance and performance of your site on a mobile browser on a smartphone or tablet device. You may already have a smartphone yourself. If you don't have a budget for devices with multiple platforms, ask your friends if you can spend a few minutes looking at your site on theirs. I have one web developer friend who checks out his designs on the phones at his local mobile carrier store (although you might quickly wear out your welcome).

Run Windows on Your Mac

If you have a Macintosh computer with an Intel chip running OS X (Leopard or later), you don't need a separate computer to test in a Windows environment. It is now possible to run Windows right on your Mac using the free Boot Camp application, which allows you to switch to Windows on reboot.

There are several other VM (Virtual Machine) products for Mac OS that allow you to toggle between Mac and Windows, including:

- VMFusion (*www.vmware.com/fusion*) is a commercial product with a free trial you can download.
- Parallels Desktop for Mac (*www.parallels.com*) is also a commercial product with a free trial.
- Oracle VirtualBox (*virtualbox.org*) is a free program that allows you to run a number of guest operating systems, including Windows and several flavors of Unix.

All VM products require that you purchase a copy of Microsoft Windows, but it sure beats buying a whole machine.

Software

There's no shortage of software available for creating web pages. In the early days, we just made do with tools originally designed for print. Today, there are wonderful tools created specifically with web design in mind that make the process more efficient. Although I can't list every available software release, I'd like to introduce you to the most common and proven web design tools. Note that you can download trial versions of many of these programs from the company websites, as listed in the "Popular Web Design Software Links" sidebar later in this chapter.

Web page authoring

Web-authoring tools are similar to desktop publishing tools, but the end product is a web page (an HTML file and its supporting files). These tools provide a visual "WYSIWYG" (What You See Is What You Get, pronounced "whizzy-wig") interface and shortcuts that save you from typing repetitive HTML and CSS. These tools won't excuse you from learning HTML. Even the most sophisticated tools won't generate HTML as clean or well-considered as a professional writing by hand, but they can speed up the process once you know what you're doing.

NOTE

To do the exercises in this book, all you'll need is the text editor that came with your operating system. No special programs are required.

The following are some popular web-authoring programs:

Adobe Dreamweaver. This is the hands-down industry standard due to its relatively clean code and advanced features.

Microsoft Expression Web (Windows only). Part of Microsoft's suite of professional design tools, MS Expression Web boasts standards-compliant code and CSS-based layouts.

Nvu (Linux, Windows, and Mac OS X). Don't want to pay for a WYSIWYG editor? Nvu (pronounced N-view, for "new view") is an open source tool that matches many of the features in Dreamweaver, and you can download it for free at *nvu.com*.

HTML editors

HTML editors (as opposed to WYSIWYG authoring tools) are designed to speed up the process of writing HTML by hand. They do not allow you edit the page visually, so you need to check your work in a browser. Many professional web designers actually prefer to author HTML documents by hand, and they tend to recommend the following:

TextPad (Windows only). TextPad is a simple and inexpensive plain-text code editor for Windows.

Sublime Text (Window, Mac, Linux). This inexpensive and up-and-coming text editor looks stripped down but has a lot of functionality (like color coding and full code overviews) that developers love.

Coda by Panic (Macintosh only). Coda users like its visual workflow, file management tools, and built-in terminal access.

TextMate by MacroMates (Macintosh only). This advanced text editor features project management tools and an interface that is integrated with the Mac operating system. It is growing in popularity because it is customizable, feature-rich, and inexpensive.

BBEdit by Bare Bones Software (Macintosh only). Lots of great shortcut features have made this the leading editor for Mac-based web developers.

Image editing and drawing software

You'll probably want to add images to your pages, so you will need an image-editing program. We'll look at some of the more popular programs in greater detail in Part IV. In the meantime, you may want to look into the following popular web-graphics-creation tools:

Adobe Photoshop. Photoshop is undeniably the industry standard for image creation in both the print and web worlds.

Adobe Photoshop Elements. This lighter version of Photoshop is designed for photo editing and management, but some hobbyists may find that it has all the tools necessary for putting images on web pages.

Adobe Illustrator. Because designers need to create logos, icons, and illustrations at a variety of sizes and resolutions, many start with a vector image in Illustrator for maximum flexibility. You can output web graphics directly from Illustrator, or bring them into Photoshop for additional fine-tuning.

Adobe Fireworks. This web graphics program combines an image editor with tools for creating vector-based illustrations. It also features advanced tools for outputting web graphics.

Corel Paint Shop Pro Photo (Windows only). This full-featured image editor is popular with the Windows crowd, primarily due to its low price.

GIMP, "GNU Image Manipulation Program" (Unix, Windows, Mac). This free image-editing program is similar to Photoshop.

Internet tools

Because you will be dealing with the Internet, you need to have some tools specifically for viewing and moving files over the network:

A variety of browsers. Because browsers render pages differently, you'll want to test your pages on as many browsers as possible, both on the desktop and on mobile devices. The following lists the desktop browsers most commonly used on Windows and Macintosh operating systems:

Windows:	Macintosh OS X:
Internet Explorer (the current version and at least two prior versions)	Safari
Chrome	Chrome
Firefox	Firefox
Safari	Opera
Opera	

And don't ignore the mobile browsers! The following list is an overview of the most commonly used mobile web browsers as of this writing (although who knows what mobile browsers will be important by the time you read this):

- Mobile Safari (iOS)
- Android Browser (Android)
- BlackBerry Browser (RIM)
- Nokia Series 40 and Nokia Browser for Symbian
- Opera Mobile and Mini (installed on any device)
- Internet Explorer Mobile (Windows Phone)
- Silk (Kindle Fire)

A file-transfer program (FTP). An FTP program enables you to upload and download files between your computer and the computer that will serve your pages to the web. The web authoring tools listed earlier all have FTP programs built right in. There are also dedicated FTP programs, as listed here:

Windows	Macintosh OS X:
WS_FTP	Transmit
CuteFTP	Cyberduck
AceFTP	Fetch
Filezilla	

Terminal application. If you know your way around the Unix operating system, you may find it useful to have a terminal (command-line) application that allows you to type Unix commands on the server. This may be useful for setting file permissions, moving or copying files and directories, or managing the server software.

Windows users can install a Linux emulator called Cygwin for command-line access. There is also PuTTY, a free Telnet/SSH client. Mac OS X includes an application called Terminal that is a full-fledged terminal application, giving you access to the underlying Unix system and the ability to use SSH to access other command-line systems over the Internet.

AT A GLANCE

Popular Web Design Software Links

Web page authoring

Adobe Dreamweaver *www.adobe.com*

Microsoft Expression Web *www.microsoft.com/products/expression*

Nvu (open source web page editor) *www.nvu.com*

HTML editing

TextMate by MacroMates for Mac OS *www.macromates.com*

Sublime Text *www.sublimetext.com*

TextPad for Windows *www.textpad.com*

Coda by Panic Software *www.panic.com/coda/*

BBEdit by Bare Bones Software *www.barebones.com*

Image editing and drawing

Adobe Photoshop *www.adobe.com*

Adobe Photoshop Elements *www.adobe.com*

Adobe Illustrator *www.adobe.com*

Adobe Fireworks *www.adobe.com*

Corel Paint Shop Pro Photo *www.corel.com/paintshoppro*

GIMP *gimp.org*

Browsers

Microsoft Internet Explorer (Windows only) *www.microsoft.com/windows/internet-explorer/*

Firefox *www.firefox.com*

Google Chrome *www.google.com/chrome*

Opera *www.opera.com*

Safari *www.apple.com/safari*

Networking

WS_FTP, CuteFTP, AceFTP, and others for Windows available at: *www.download.com*

Transmit (for Macintosh OSX) *www.panic.com/transmit*

Cyberduck (for Macintosh OSX) *cyberduck.ch*

Fetch (for Macintosh OSX) *fetchsoftworks.com*

Cygwin (Linux emulator for Windows) *www.cygwin.com*

PuTTY (telnet/SSH terminal emulator) *www.chiark.greenend.org.uk/~sgtatham/putty/*

What You've Learned

The lesson to take away from this chapter is: "You don't have to learn everything." And even if you want to learn everything eventually, you don't need to learn it all at once. So relax, and don't worry. The other good news is that, while many professional tools exist, it is possible to create a basic website and get it up and running without spending much money by using freely available or inexpensive tools and your existing computer setup.

As you'll soon see, it's easy to get started making web pages—you will be able to create simple pages by the time you're done reading this book. From there, you can continue adding to your bag of tricks and find your particular niche in web design.

exercise 1-1 | Taking stock

Now that you're taking that first step in learning web design, it might be a good time to take stock of your assets and goals. Using the lists in this chapter as a general guide, try jotting down answers to the following questions:

- What are your web design goals? To become a professional web designer? To make personal websites only?
- Which aspects of web design interest you the most?
- What current skills do you have that will be useful in creating web pages?
- Which skills will you need to brush up on?
- Which hardware and software tools do you already have for web design?
- Which tools do you need to buy? Which tools would you like to buy eventually?

Test Yourself

Each chapter in this book ends with a few questions that you can answer to see if you picked up the important bits of information. Answers appear in Appendix A.

1. Match these web professionals with the final product they might be responsible for producing.

A. Graphic designer	_____ HTML and CSS documents
B. Production department	_____ PHP scripts
C. User experience designer	_____ Photoshop page sketch
D. Web programmer	_____ Storyboards

2. What does the W3C do?

3. Match the web technology with its appropriate task:

A. HTML	_____ Checks a form field for a valid entry
B. CSS	_____ Creates a custom server-side web application
C. JavaScript	_____ Identifies text as a second-level heading
D. PHP	_____ Defines a new markup language for sharing financial information
E. XML	_____ Makes all second-level headings blue

4. What is the difference between frontend and backend web development?

5. What is the difference between a web-authoring program and an HTML-editing tool?

HOW THE WEB WORKS

I got started in web design in early 1993—pretty close to the start of the Web itself. In web time, that makes me an old-timer, but it's not so long ago that I can't remember the first time I looked at a web page. It was difficult to tell where the information was coming from and how it all worked.

This chapter sorts out the pieces and introduces some basic terminology. We'll start with the big picture and work down to specifics.

IN THIS CHAPTER

An explanation of the Web, as it relates to the Internet

The role of the server

The role of the browser

Introduction to URLs and their components

The anatomy of a web page

The Internet Versus the Web

No, it's not a battle to the death, just an opportunity to point out the distinction between these two words that are increasingly being used interchangeably.

The Internet is a network of connected computers. No company owns the Internet; it is a cooperative effort governed by a system of standards and rules. The purpose of connecting computers together, of course, is to share information. There are many ways information can be passed between computers, including email, file transfer (FTP), and many more specialized modes upon which the Internet is built. These standardized methods for transferring data or documents over a network are known as protocols.

The Web (originally called the World Wide Web, thus the "www" in site addresses) is just one of the ways information can be shared over the Internet. It is unique in that it allows documents to be linked to one another using hypertext links—thus forming a huge "web" of connected information. The Web uses a protocol called HTTP (HyperText Transfer Protocol). That acronym should look familiar because it is the first four letters of nearly all website addresses, as we'll discuss in an upcoming section.

The Web is a subset of the Internet. It is just one of many ways information can be transferred over networked computers.

Serving Up Your Information

Let's talk more about the computers that make up the Internet. Because they "serve up" documents upon request, these computers are known as servers. More accurately, the server is the software (not the computer itself) that

A Brief History of the Web

The Web was born in a particle physics laboratory (CERN) in Geneva, Switzerland in 1989. There a computer specialist named Tim Berners-Lee first proposed a system of information management that used a "hypertext" process to link related documents over a network. He and his partner, Robert Cailliau, created a prototype and released it for review. For the first several years, web pages were text-only. It's difficult to believe that in 1992, the world had only about 50 web servers, total.

The real boost to the Web's popularity came in 1992 when the first graphical browser (NCSA Mosaic) was introduced, and the Web broke out of the realm of scientific research into mass media. The ongoing development of web technologies is overseen by the World Wide Web Consortium (W3C).

If you want to dig deeper into the Web's history, check out this site:

W3C's History Archives

www.w3.org/History.html

allows the computer to communicate with other computers; however, it is common to use the word "server" to refer to the computer as well. The role of server software is to wait for a request for information, then retrieve and send that information back as quickly as possible.

There's nothing special about the computers themselves...picture anything from a high-powered Unix machine to a humble personal computer. It's the server software that makes it all happen. In order for a computer to be part of the Web, it must be running special web server software that allows it to handle Hypertext Transfer Protocol transactions. Web servers are also called "HTTP servers."

There are many server software options out there, but the two most popular are Apache (open source software) and Microsoft Internet Information Services (IIS). Apache is freely available for Unix-based computers and comes installed on Macs running Mac OS X. There is a Windows version as well. Microsoft IIS is part of Microsoft's family of server solutions.

Every computer and device (modem, router, smartphone, cars, etc.) connected to the Internet is assigned a unique numeric IP address (IP stands for Internet Protocol). For example, the computer that hosts *oreilly.com* has the IP address 208.201.239.100. All those numbers can be dizzying, so fortunately, the Domain Name System (DNS) was developed to allow us to refer to that server by its domain name, "oreilly.com", as well. The numeric IP address is useful for computer software, while the domain name is more accessible to humans. Matching the text domain names to their respective numeric IP addresses is the job of a separate DNS server.

It is possible to configure your web server so that more than one domain name is mapped to a single IP address, allowing several sites to share a single server.

TERMINOLOGY

Open Source

Open source software is developed as a collaborative effort with the intent to make its source code available to other programmers for use and modification. Open source programs are usually available for free.

No More IP Addresses

The IANA, the organization that assigns IP numbers, handed out its last bundle of IP addresses on February 3, 2011. That's right, no more ###.###.###.###-style IPs. That format of IP address (called IPv4) is able to produce 4.3 billion unique addresses, which seemed like plenty when the Internet "experiment" was first conceived in 1977. There was no way the creators could anticipate that one day every phone, television, and object on store shelves would be clamoring for one.

The solution is a new IP format (IPv6, already in the works) that allows for trillions and trillions of unique IP numbers, with the slight snag that it is incompatible with our current IPv4-based network, so IPv6 will operate as a sort of parallel Internet to the one we have today. Eventually, IPv4 will be phased out, but some say it will take decades.

A Word About Browsers

We now know that the server does the servin', but what about the other half of the equation? The software that does the requesting is called the client. People use desktop browsers, mobile browsers, and other assistive technologies (such as screen readers) as clients to access documents on the Web. The server returns the documents for the browser (also referred to as the user agent in technical circles) to display.

The requests and responses are handled via the HTTP protocol, mentioned earlier. Although we've been talking about "documents," HTTP can be used to transfer images, movies, audio files, data, scripts, and all the other web resources that commonly make up web sites and applications.

It is common to think of a browser as a window on a computer monitor with a web page displayed in it. These are known as graphical browsers or desktop browsers and for a long time, they were the only web-viewing game in town. The most popular desktop browsers as of this writing include Internet Explorer for Windows, Chrome, Firefox, and Safari, with Opera bringing up the rear. These days, however, more and more people are accessing the Web on the go using browsing clients built into mobile phones or tablets.

It is also important to keep alternative web experiences in mind. Users with sight disabilities may be listening to a web page read by a screen reader (or simply make their text extremely large). Users with limited mobility may use assistive devices to access links and to type. The sites we build must be accessible and usable for all users, regardless of their browsing experiences.

Even on the desktop browsers that first introduced us to the wide world of the Web, pages may look and perform differently from browser to browser. This is due to varying support for web technologies and the users' ability to set their own browsing preferences.

TERMINOLOGY

Server-side and Client-side

Often in web design, you'll hear reference to "client-side" or "server-side" applications. These terms are used to indicate which machine is doing the processing. Client-side applications run on the user's machine, while server-side applications and functions use the processing power of the server computer.

Intranets and Extranets

When you think of a website, you generally assume that it is accessible to anyone surfing the Web. However, many companies take advantage of the awesome information sharing and gathering power of websites to exchange information just within their own business. These special web-based networks are called intranets. They are created and function like ordinary websites, but they use special security devices (called firewalls) that prevent the outside world from seeing them. Intranets have lots of uses, such as sharing human resource information or providing access to inventory databases.

An extranet is like an intranet, only it allows access to select users outside of the company. For instance, a manufacturing company may provide its customers with passwords that allow them to check the status of their orders in the company's orders database. Of course, the passwords determine which slice of the company's information is accessible.

Web Page Addresses (URLs)

Every page and resource on the Web has its own special address called a URL, which stands for Uniform Resource Locator. It's nearly impossible to get through a day without seeing a URL (pronounced "U-R-L," not "erl") plastered on the side of a bus, printed on a business card, or broadcast on a television commercial. Web addresses are fully integrated into modern vernacular.

Some URLs are short and sweet. Others may look like crazy strings of characters separated by dots (periods) and slashes, but each part has a specific purpose. Let's pick one apart.

The parts of a URL

A complete URL is generally made up of three components: the protocol, the site name, and the absolute path to the document or resource, as shown in Figure 2-1.

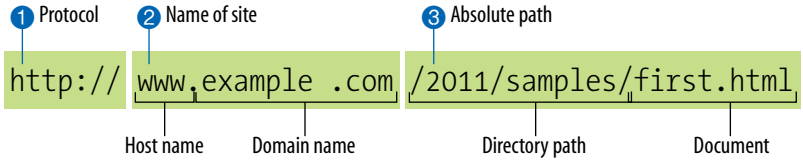


Figure 2-1. The parts of a URL.

❶ `http://`

The first thing the URL does is define the protocol that will be used for that particular transaction. The letters HTTP let the server know to use Hypertext Transfer Protocol, or get into "web mode."

❷ `www.example.com`

The next portion of the URL identifies the website by its domain name. In this example, the domain name is example.com. The "www." part at the beginning is the particular host name at that domain. The host name "www" has become a convention, but is not a rule. In fact, sometimes the host name may be omitted. There can be more than one website at a domain (sometimes called subdomains). For example, there might also be *development.example.com*, *clients.example.com*, and so on.

❸ `/2012/samples/first.html`

This is the absolute path through directories on the server to the requested HTML document, *first.html*. The words separated by slashes are the directory names, starting with the root directory of the host (as indicated by the initial `/`). Because the Internet originally comprised computers running the Unix operating system, our current way of doing things still

Hey, There's No http:// on That URL!

Because nearly all web pages use the Hypertext Transfer Protocol, the **`http://`** part is often just implied. This is the case when site names are advertised in print or on TV, as a way to keep the URL easy to remember.

Additionally, browsers are programmed to add **`http://`** automatically as a convenience to save you some keystrokes. It may seem like you're leaving it out, but it is being sent to the server behind the scenes.

When we begin using URLs to create hyperlinks in HTML documents in Chapter 6, Adding Links, you'll learn that it is necessary to include the protocol when making a link to a web page on another server.

NOTE

Sometimes you'll see a URL that begins with `https://`*. This is an indication that it is a secure server transaction. Secure servers have special encryption devices that hide delicate content, such as credit card numbers, while they are transferred to and from the browser. Look for it the next time you're shopping online.*

follows many Unix rules and conventions, hence the / separating directory names.

To sum it up, the URL in Figure 2-1 says it would like to use the HTTP protocol to connect to a web server on the Internet called *www.example.com* and request the document *first.html* (located in the *samples* directory, which is in the *2012* directory).

Default files

Obviously, not every URL you see is so lengthy. Many addresses do not include a filename, but simply point to a directory, like these:

```
http://www.oreilly.com
http://www.jendesign.com/resume/
```

When a server receives a request for a directory name rather than a specific file, it looks in that directory for a default document, typically named *index.html*. So when someone types the above URLs into their browser, what they'll actually see is this:

```
http://www.oreilly.com/index.html
http://www.jendesign.com/resume/index.html
```

The name of the default file (also referred to as the index file) may vary, and depends on how the server is configured. In these examples, it is named *index.html*, but some servers use the filename *default.htm*. If your site uses server-side programming to generate pages, the index file might be named *index.php* or *index.asp*. Just check with your server administrator or the tech support department at your hosting service to make sure you give your default file the proper name.

Another thing to notice is that in the first example, the original URL did not have a trailing slash to indicate it was a directory. When the slash is omitted, the server simply adds one if it finds a directory with that name.

The index file is also useful for security. Some servers (depending on their configuration) display the contents of the directory if the default file is not found. Figure 2-2 shows how the documents in the *housepics* directory are exposed as the result of a missing default file. One way to prevent people from snooping around in your files is to be sure there is an index file in every directory. Your server administrator may also add other protections to prevent your directories from displaying in the browser.

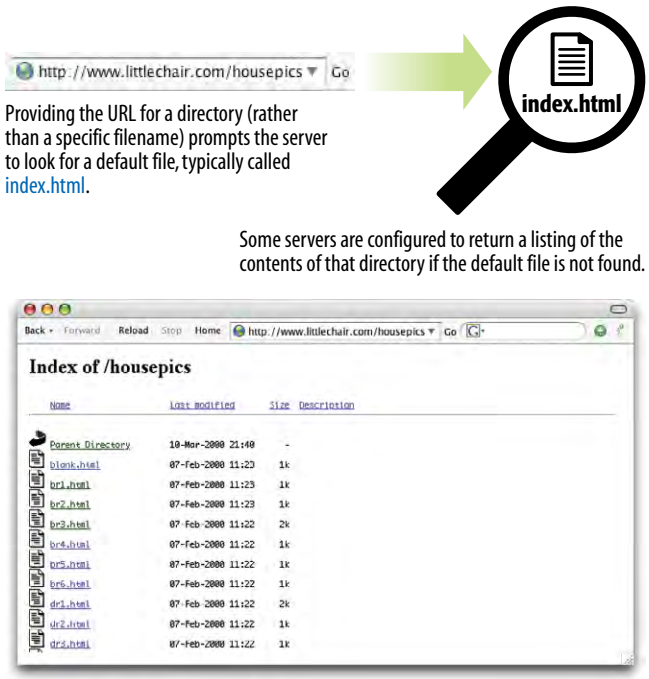


Figure 2-2. *Some servers display the contents of the directory if an index file is not found.*

The Anatomy of a Web Page

We're all familiar with what web pages look like in the browser window, but what's happening "under the hood?"

At the top of Figure 2-3, you see a minimal web page as it appears in a graphical browser. Although you see it as one coherent page, it is actually assembled from four separate files: an HTML document (*index.html*), a style sheet (*kitchen.css*), and two graphics (*foods.gif* and *spoon.gif*). The HTML document is running the show.

HTML documents

You may be as surprised as I was to learn that the graphically rich and interactive pages we see on the Web are generated by simple, text-only documents. This text file is referred to as the source document.

Take a look at *index.html*, the source document for the Jen's Kitchen web page. You can see it contains the text content of the page plus special tags (indicated with angle brackets, < and >) that describe each element on the page.

Adding descriptive tags to a text document is known as "marking up" the document. Web pages use a markup language called HyperText Markup Language, or HTML for short, which was created especially for documents with hypertext links. HTML defines dozens of text elements that make up documents such as headings, paragraphs, emphasized text, and of course, links. There are also elements that add information about the document (such as its title), media such as images and videos, and widgets for form inputs, just to name a few.

It is worth noting briefly that there are actually several versions of HTML in use today. The most firmly established are HTML version 4.01 and its stricter cousin, XHTML 1.0. And you may have heard how all the Web is a-buzz with the emerging HTML5 specification that is designed to better handle web applications and is gradually gaining browser support. I will give you the lowdown on all the various versions and what makes them unique in Chapter 10, What's Up, HTML5?. In the meantime, we have to cover some basics that apply regardless of the HTML flavor you choose.

A quick introduction to HTML markup

You'll be learning the nitty-gritty of markup in Part II, so I don't want to bog you down with too much detail right now, but there are a few things I'd like to point out about how HTML works and how browsers interpret it.

Read through the HTML document in Figure 2-3 and compare it to the browser results. It's easy to see how the elements marked up with HTML tags in the source document correspond to what displays in the browser window.

exercise 2-1 | View source

You can see the HTML file for any web page by choosing View → Page Source or (View → Source) in your browser's menu. Your browser typically opens the source document in a separate window. Let's take a look under the hood of a web page.

1. Enter this URL into your browser:

   www.learningwebdesign.com/4e/materials/chapter02/kitchen.html

   You should see the Jen's Kitchen web page from Figure 2-3.
2. Select View → Page Source (or View → Source) from the browser menu. On Chrome and Opera, View Source is located in the Developer menu. A window opens showing the source document shown in the figure.
3. The source for most sites is considerably more complicated. View the source of *oreilly.com* or the site of your choice. Don't worry if you don't understand what's going on. Much of it will look more familiar by the time you are done with this book.

WARNING

Keep in mind that while learning from others' work is fine, the all-out stealing of other people's code is poor form (or even illegal). If you want to use code as you see it, ask for permission and always give credit to those who did the work.

The web page shown in this browser window consists of four separate files: an HTML text document, a style sheet and two images. Tags in the HTML source document give the browser instructions for how the text is structured and where the images should be placed.

index.html

```
<!DOCTYPE html>
<html>
<head>
<title>Jen's Kitchen</title>
<link rel="stylesheet" href="kitchen.css" type="text/css" >
</head>

<body>
<h1><img src="foods.gif" alt="food illustration"> Jen&rsquo;s Kitchen</h1>

<p>If you love to read about <strong>cooking and eating</strong>, would like to find out about
of some of the best restaurants in the world, or just want a few choice recipes to add to your
collection, <em>this is the site for you!</em></p>

<p><img src="spoon.gif" alt="spoon illustration"> Your pal, Jen at Jen's Kitchen</p>
<hr>
<p><small>Copyright 2011, Jennifer Robbins</small></p>
</body>
</html>
```

kitchen.css

```
body { font: normal 1em Verdana; margin: 1em 10%;}
h1 { font: italic 3em Georgia; color: rgb(23, 109, 109); margin: 1em 0 1em;}
img { margin: 0 20px 0 0; }
h1 img { margin-bottom: -20px; }
small { color: #666666; }
```

foods.gif

spoon.gif

Figure 2-3. The source file and images that make up a simple web page.

First, you'll notice that the text within brackets (for example, **<body>**) does not display in the final page. The browser displays only what's between the tags—the content of the element. The markup is hidden. The tag provides the name of the HTML element—usually an abbreviation such as "h1" for "heading level 1," or "em" for "emphasized text."

Second, you'll see that most of the HTML tags appear in pairs surrounding the content of the element. In our HTML document, **<h1>** indicates that the following text should be a level-1 heading; **</h1>** indicates the end of the heading. Some elements, called empty elements, do not have content. In our sample, the **<hr>** tag indicates an empty element that tells the browser to "insert a thematic divider here" (most browsers indicate the thematic divider with a horizontal rule [line], which is how the **hr** element got its initials).

Because I was unfamiliar with computer programming when I first began writing HTML, it helped me to think of the tags and text as "beads on a string" that the browser interprets one by one, in sequence. For example, when the browser encounters an open bracket (**<**), it assumes all of the following characters are part of the markup until it finds the closing bracket (**>**). Similarly, it assumes all of the content following an opening **<h1>** tag is a heading until it encounters the closing **</h1>** tag. This is the manner in which the browser parses the HTML document. Understanding the browser's method can be helpful when troubleshooting a misbehaving HTML document.

But where are the pictures?

Obviously, there are no pictures in the HTML file itself, so how do they get there when you view the final page?

You can see in Figure 2-3 that each image is a separate file. The images are placed in the flow of the text with the HTML image element (**img**) that tells the browser where to find the graphic (its URL). When the browser sees the **img** element, it makes another request to the server for the image file, and then places it in the content flow. The browser software brings the separate pieces together into the final page. Videos and other embedded media files are added in much the same way.

The assembly of the page generally happens in an instant, so it appears as though the whole page loads all at once. Over slow connections or if the page includes huge graphics or media files, the assembly process may be more apparent as images lag behind the text. The page may even need to be redrawn as new images arrive (although you can construct your pages in a way to prevent that from happening).

Adding a little style

I want to direct your attention to one last key ingredient of our minimal page. Near the top of the HTML document there is a `link` element that points to the style sheet document *kitchen.css*. That style sheet includes a few lines of instructions for how the page should look in the browser. These are style instructions written according to the rules of Cascading Style Sheets (CSS). CSS allows designers to add visual style instructions (known as the document's presentation) to the marked-up text (the document's structure, in web design terminology). In Part III, you'll really get to know the power of Cascading Style Sheets.

Figure 2-4 shows the Jen's Kitchen page with and without the style instructions. Browsers come equipped with default styles for every HTML element they support, so if an HTML document lacks its own custom style instructions, the browser will use its own (that's what you see in the screen shot on the right). Even just a few style rules can make big improvements to the appearance of a page.

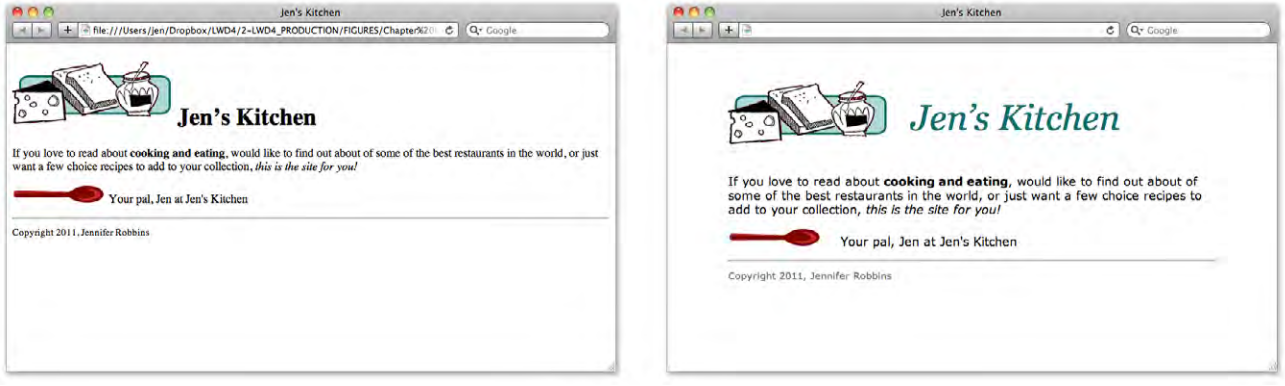


Figure 2-4. The Jen's Kitchen page before (left) and after (right) style rules.

Adding Behaviors with JavaScript

In addition to a document's structure and presentation, there is also a behavior component that defines how things *work*. On the Web, behaviors are defined by a scripting language called JavaScript. We'll touch on it lightly in this book in Part IV, but learning JavaScript from scratch is more than we can take on here. Many designers (myself included) rely on people with scripting experience to add functionality to sites. However, knowing how to write JavaScript is becoming more essential to the "web designer" job description.

Putting It All Together

To wrap up our introduction to how the web works, let's trace a typical stream of events that occurs with every web page that appears on your screen (Figure 2-5).

1. You request a web page by either typing its URL (for example, *http://jenskitchensite.com*) directly in the browser or by clicking on a link on a page. The URL contains all the information needed to target a specific document on a specific web server on the Internet.
2. Your browser sends an HTTP Request to the server named in the URL and asks for the specific file. If the URL specifies a directory (not a file), it is the same as requesting the default file in that directory.
3. The server looks for the requested file and issues an HTTP response.
   a. If the page cannot be found, the server returns an error message. The message typically says "404 Not Found," although more hospitable error messages may be provided.
   b. If the document *is* found, the server retrieves the requested file and returns it to the browser.
4. The browser parses the HTML document. If the page contains images (indicated by the HTML `img` element) or other external resources like scripts, the browser contacts the server again to request each resource specified in the markup.
5. The browser inserts each image in the document flow where indicated by the `img` element. And *voila*! The assembled web page is displayed for your viewing pleasure.

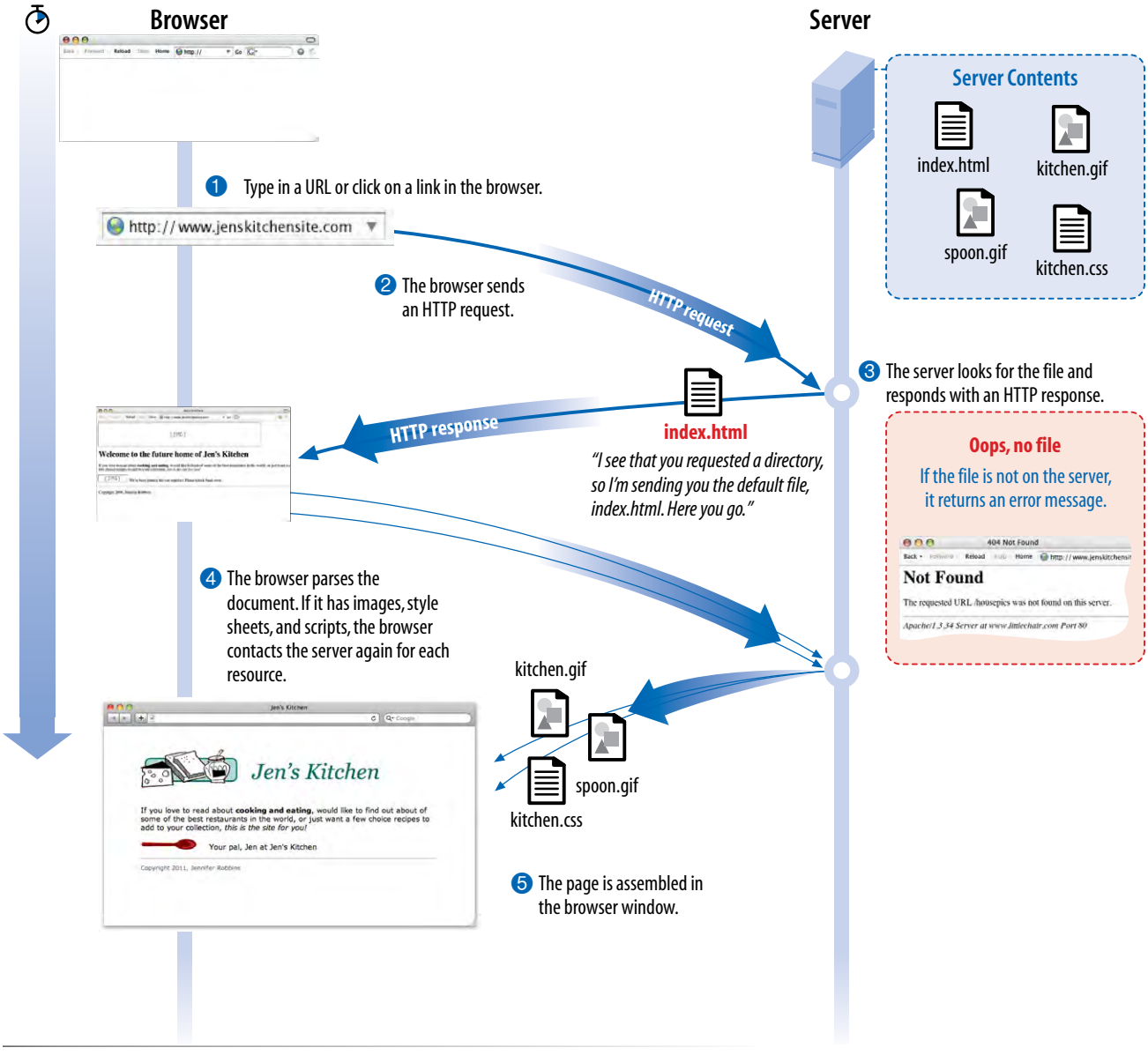


Figure 2-5. How browsers display web pages.

Test Yourself

Let's play a round of "Identify that Acronym!" The following are a few basic web terms mentioned in this chapter. Answers are in Appendix A.

1) HTML	_____	a) Home of Mosaic, the first graphical browser
2) W3C	_____	b) The location of a web document or resource
3) CERN	_____	c) The markup language used to describe web content
4) CSS	_____	d) Matches domain names with numeric IP addresses
5) HTTP	_____	e) A protocol for file transfer
6) IP	_____	f) Protocol for transferring web documents on the Internet
7) URL	_____	g) The language used to instruct how web content looks
8) NCSA	_____	h) Particle physics lab where the Web was born
9) DNS	_____	i) Internet Protocol
10) FTP	_____	j) The organization that monitors web technologies

HTML MARKUP FOR STRUCTURE

PART II

IN THIS PART

CREATING A SIMPLE PAGE
(HTML Overview)

IN THIS CHAPTER

An introduction to elements and attributes

A step-by-step demo of marking up a simple web page

The elements that provide document structure

A simple stylesheet

Troubleshooting broken web pages

Part I provided a general overview of the web design environment. Now that we've covered the big concepts, it's time to roll up our sleeves and start creating a real web page. It will be an extremely simple page, but even the most complicated pages are based on the principles described here.

In this chapter, we'll create a web page step by step so you can get a feel for what it's like to mark up a document with HTML tags. The exercises allow you to work along.

This is what I want you to get out of this chapter:

- Get a feel for how markup works, including an understanding of elements and attributes.
- See how browsers interpret HTML documents.
- Learn the basic structure of an HTML document.
- Get a first glimpse of a style sheet in action.

Don't worry about learning the specific text elements or style sheet rules at this point; we'll get to those in the following chapters. For now, just pay attention to the process, the overall structure of the document, and the new terminology.

A Web Page, Step by Step

You got a look at an HTML document in Chapter 2, How the Web Works, but now you'll get to create one yourself and play around with it in the browser. The demonstration in this chapter has five steps that cover the basics of page production.

Step 1: Start with content. As a starting point, we'll write up raw text content and see what browsers do with it.

Step 2: Give the document structure. You'll learn about HTML element syntax and the elements that give a document its structure.

HTML the Hard Way

I stand by my method of teaching HTML the old-fashioned way—*by hand*. There's no better way to truly understand how markup works than typing it out, one tag at a time, then opening your page in a browser. It doesn't take long to develop a feel for marking up documents properly.

Although you may choose to use a web-authoring tool down the line, understanding HTML will make using your tools easier and more efficient. In addition, you will be glad that you can look at a source file and understand what you're seeing. It is also crucial for troubleshooting broken pages or fine-tuning the default formatting that web tools produce.

And for what it's worth, professional web developers tend to mark up content manually because it gives them better control over the code and allows them to make deliberate decisions about what elements are used.

Step 3: Identify text elements. You'll describe the content using the appropriate text elements and learn about the proper way to use HTML.

Step 4: Add an image. By adding an image to the page, you'll learn about attributes and empty elements.

Step 5: Change the page appearance with a style sheet. This exercise gives you a taste of formatting content with Cascading Style Sheets.

By the time we're finished, you will have written the source document for the page shown in Figure 4-1. It's not very fancy, but you have to start somewhere.

We'll be checking our work in a browser frequently throughout this demonstration—probably more than you would in real life. But because this is an introduction to HTML, it is helpful to see the cause and effect of each small change to the source file along the way.

Before We Begin, Launch a Text Editor

In this chapter and throughout the book, we'll be writing out HTML documents by hand, so the first thing we need to do is launch a text editor. The text editor that is provided with your operating system, such as Notepad (Windows) or TextEdit (Macintosh), will do for these purposes. Other text editors are fine as long as you can save plain text files with the *.html* extension. If you have a WYSIWYG web-authoring tool such as Dreamweaver, set it aside for now. I want you to get a feel for marking up a document manually (see the sidebar "HTML the Hard Way").

This section shows how to open new documents in Notepad and TextEdit. Even if you've used these programs before, skim through for some special settings that will make the exercises go more smoothly. We'll start with Notepad; Mac users can jump ahead.

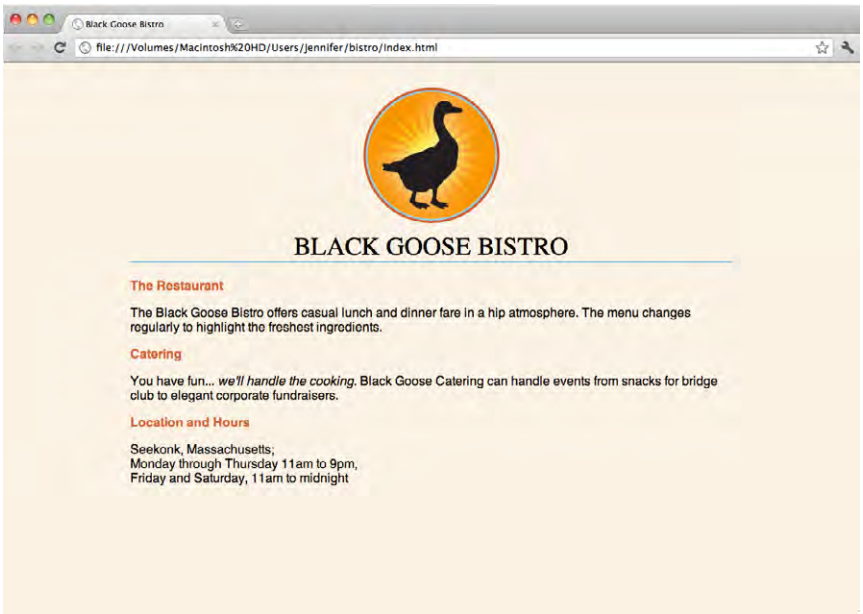


Figure 4-1. *In this chapter, we'll write the source document for this page step by step.*

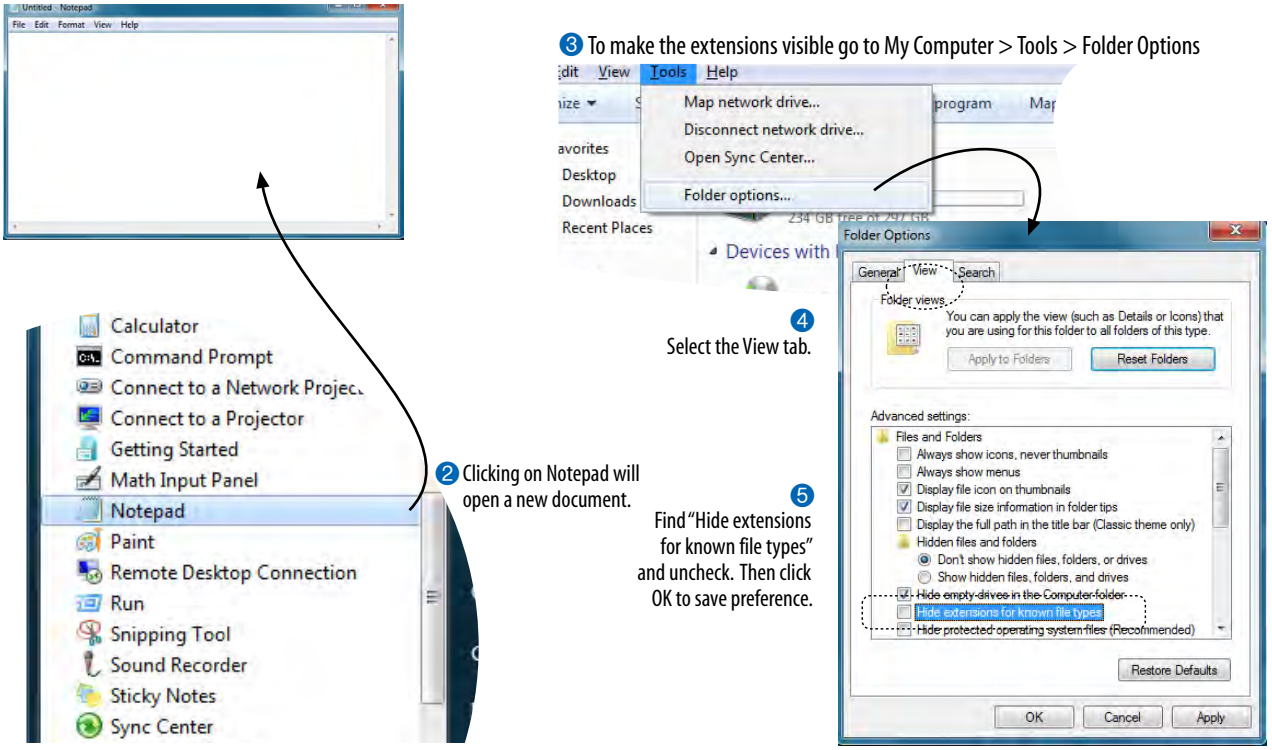
Creating a new document in Notepad (Windows)

These are the steps to creating a new document in Notepad on Windows 7 (Figure 4-2):

1. Open the Start menu and navigate to Notepad (in Accessories). ❶
2. Click on Notepad to open a new document window, and you're ready to start typing. ❷
3. Next, we'll make the extensions visible. This step is not required to make HTML documents, but it will help make the file types clearer at a glance. Select "Folder Options..." from the Tools menu ❸ and select the View tab ❹. Find "Hide extensions for known file types" and uncheck that option. ❺ Click OK to save the preference, and the file extensions will now be visible.

NOTE

In Windows 7, hit the ALT key to reveal the menu to access Tools and Folder Options. In Windows Vista, it is labeled "Folder and Search Options."



❶ Open the Start menu and navigate to Notepad *(All Programs > Accessories > Notepad)*

Figure 4-2. Creating a new document in Notepad.

Creating a new document in TextEdit (Mac OS X)

By default, TextEdit creates "rich text" documents, that is, documents that have hidden style formatting instructions for making text bold, setting font size, and so on. You can tell that TextEdit is in rich text mode when it has a formatting toolbar at the top of the window (plain text mode does not). HTML documents need to be plain text documents, so we'll need to change the Format, as shown in this example (Figure 4-3).

1. Use the Finder to look in the Applications folder for TextEdit. When you've found it, double-click the name or icon to launch the application.
2. TextEdit opens a new document. The text-formatting menu at the top shows that you are in Rich Text mode. Here's how you change it.
3. Open the Preferences dialog box from the TextEdit menu.
4. There are three settings you need to adjust:

   On the "New Document" tab, select "Plain text".

   On the "Open and Save" tab, select "Ignore rich text commands in HTML files" and turn off "Append '.txt' extensions to plain text files".
5. When you are done, click the red button in the top-left corner.
6. When you create a new document, the formatting menu will no longer be there and you can save your text as an HTML document. You can always convert a document back to rich text by selecting Format → Make Rich Text when you are not using TextEdit for HTML.

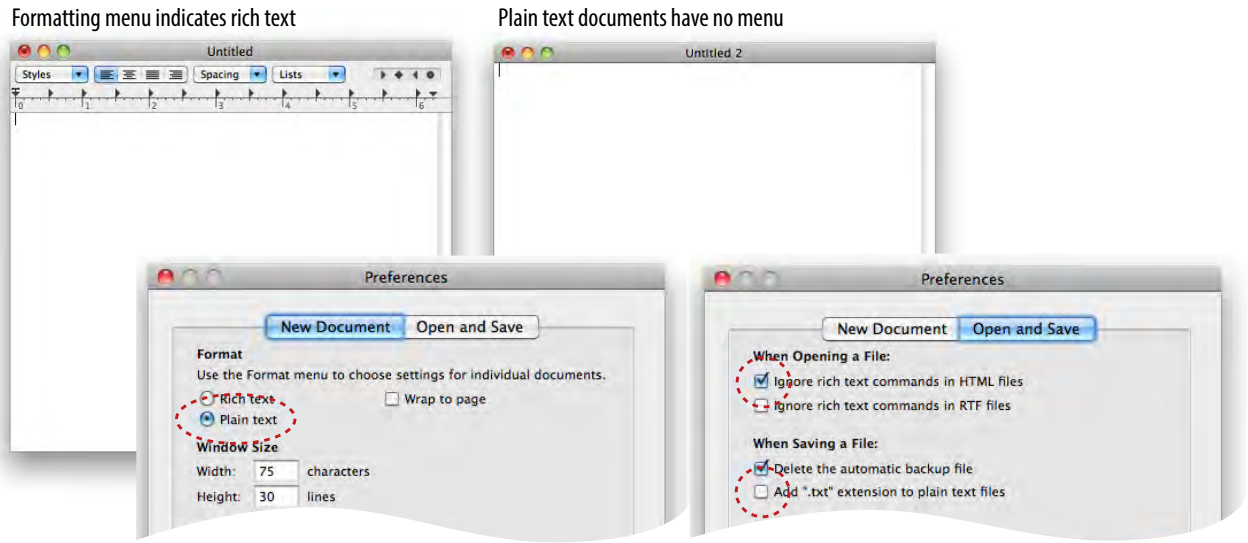


Figure 4-3. Launching TextEdit and choosing Plain Text settings in the Preferences.

Step 1: Start with Content

Now that we have our new document, it's time to get typing. A web page always starts with content, so that's where we begin our demonstration. Exercise 4-1 walks you through entering the raw text content and saving the document in a new folder.

exercise 4-1 | Entering content

1. Type the content below for the home page into the new document in your text editor. Copy it exactly as you see it here, keeping the line breaks the same for the sake of playing along. The raw text for this exercise is available online at *www.learningwebdesign.com/4e/materials/*.

```
Black Goose Bistro

The Restaurant
The Black Goose Bistro offers casual lunch and dinner fare in hip
atmosphere. The menu changes regularly to highlight the freshest
ingredients.

Catering
You have fun... we'll handle the cooking. Black Goose Catering
can handle events from snacks for bridge club to elegant corporate
fundraisers.

Location and Hours
Seekonk, Massachusetts;
Monday through Thursday 11am to 9pm, Friday and Saturday, 11am to
midnight
```

2. Select "Save" or "Save as" from the File menu to get the Save As dialog box (Figure 4-4). The first thing you need to do is create a new folder that will contain all of the files for the site (in other words, it's the local root folder).

   Windows: Click the folder icon at the top to create the new folder.

   Mac: Click the "New Folder" button.

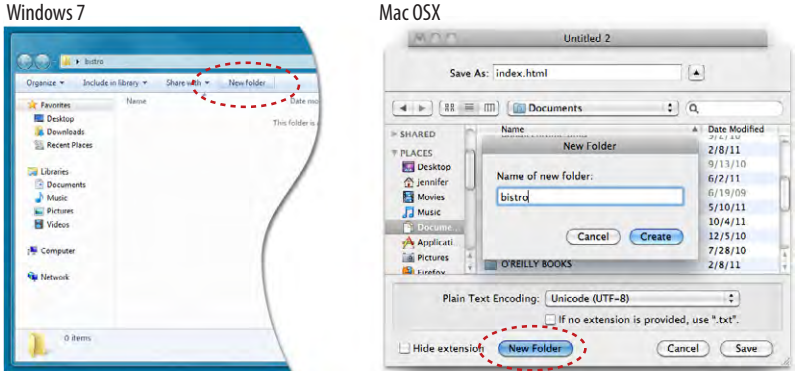


Figure 4-4. Saving index.html in a new folder called "bistro".

Naming Conventions

It is important that you follow these rules and conventions when naming your files:

Use proper suffixes for your files. HTML and XHTML files must end with .html. Web graphics must be labeled according to their file format: *.gif*, *.png*, or *.jpg* (*.jpeg* is also acceptable).

Never use character spaces within filenames. It is common to use an underline character or hyphen to visually separate words within filenames, such as *robbins_bio.html* or *robbins-bio .html*.

Avoid special characters such as **?, %, #, /, :, ;, •,** etc. Limit filenames to letters, numbers, underscores, hyphens, and periods.

Filenames may be case-sensitive, depending on your server configuration. Consistently using all lowercase letters in filenames, although not necessary, is one way to make your filenames easier to manage.

Keep filenames short. Short names keep the character count and file size of your HTML file in check. If you really must give the file a long, multiword name, you can separate words with hyphens, such as *a-long-document-title.html*, to improve readability.

Self-imposed conventions. It is helpful to develop a consistent naming scheme for huge sites. For instance, always using lowercase with hyphens between words. This takes some of the guesswork out of remembering what you named a file when you go to link to it later.

Name the new folder ***bistro***, and save the text file as ***index.html*** in it. Windows users, you will also need to choose "All Files" after "Save as type" to prevent Notepad from adding a ".txt" extension to your filename. The filename needs to end in ***.html*** to be recognized by the browser as a web document. See the sidebar "Naming Conventions" for more tips on naming files.

3. Just for kicks, let's take a look at ***index.html*** in a browser. Launch your favorite browser (I'm using Google Chrome) and choose "Open" or "Open File" from the File menu. Navigate to ***index.html***, and then select the document to open it in the browser. You should see something like the page shown in Figure 4-5. We'll talk

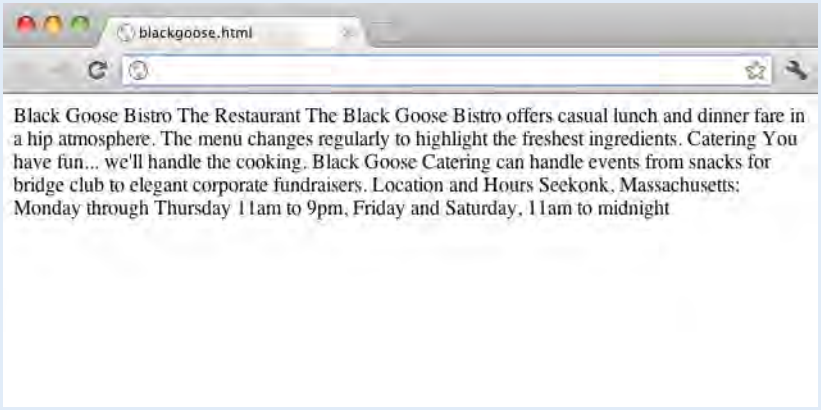


Figure 4-5. *A first look at the content in a browser.*

Learning from step 1

Our content isn't looking so good (Figure 4-5). The text is all run together—that's not how it looked in the original document. There are a couple of things to be learned here. The first thing that is apparent is that the browser ignores line breaks in the source document. The sidebar "What Browsers Ignore" lists other information in the source that is not displayed in the browser window.

Second, we see that simply typing in some content and naming the document *.html* is not enough. While the browser can display the text from the file, we haven't indicated the *structure* of the content. That's where HTML comes in. We'll use markup to add structure: first to the HTML document itself (coming up in Step 2), then to the page's content (Step 3). Once the browser knows the structure of the content, it can display the page in a more meaningful way.

What Browsers Ignore

Some information in the source document will be ignored when it is viewed in a browser, including:

Multiple (white) spaces. When a browser encounters more than one consecutive blank character space, it displays a single space. So if the document contains:

```
long,     long     ago
```

the browser displays:

long, long ago

Line breaks (carriage returns). Browsers convert carriage returns to white spaces, so following the earlier "ignore multiple white spaces rule," line breaks have no effect on formatting the page. Text and elements wrap continuously until a new block element, such as a heading (**h1**) or paragraph (**p**), or the line break (**br**) element is encountered in the flow of the document text.

Tabs. Tabs are also converted to character spaces, so guess what? Useless.

Unrecognized markup. Browsers are instructed to ignore any tag they don't understand or that was specified incorrectly. Depending on the element and the browser, this can have varied results. The browser may display nothing at all, or it may display the contents of the tag as though it were normal text.

Text in comments. Browsers will not display text between the special **<!--** and **-->** tags used to denote a comment. See the Adding Hidden Comments sidebar later in this chapter.

Step 2: Give the Document Structure

We have our content saved in an *.html* document—now we're ready to start marking it up.

Introducing…HTML elements

An element consists of both the content and its markup.

Back in Chapter 2, How the Web Works, you saw examples of HTML elements with an opening tag (** ** for a paragraph, for example) and closing tag (** **). Before we start adding tags to our document, let's look at the anatomy of an HTML element (its syntax) and firm up some important terminology. A generic container element is labeled in Figure 4-6.

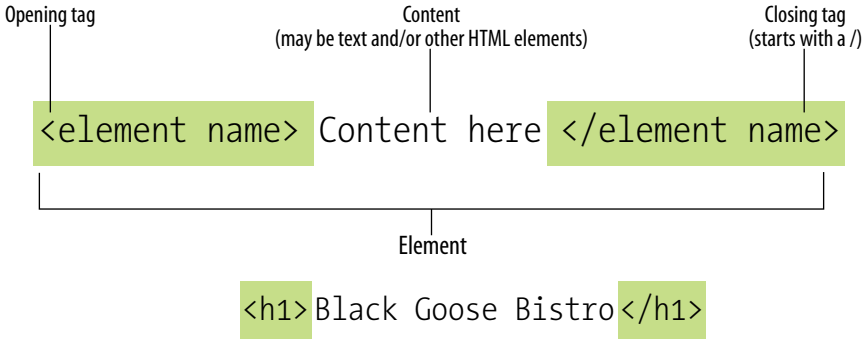


Figure 4-6. *The parts of an HTML container element.*

Elements are identified by tags in the text source. A tag consists of the element name (usually an abbreviation of a longer descriptive name) within angle brackets (**< >**). The browser knows that any text within brackets is hidden and not displayed in the browser window.

The element name appears in the opening tag (also called a start tag) and again in the closing (or end) tag preceded by a slash (**/**). The closing tag works something like an "off" switch for the element. Be careful not to use the similar backslash character in end tags (see the tip Slash vs. Backslash).

The tags added around content are referred to as the markup. It is important to note that an element consists of both the content *and* its markup (the start and end tags). Not all elements have content, however. Some are empty by definition, such as the **img** element used to add an image to the page. We'll talk about empty elements a little later in this chapter.

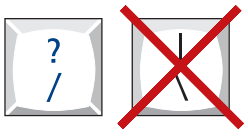
One last thing...capitalization. In HTML, the capitalization of element names is not important. So **<img>**, **<Img>**, and **<IMG>** are all the same as far as the browser is concerned. However, in XHTML (the stricter version of HTML) all element names must be all lowercase in order to be valid. Many web developers have come to like the orderliness of the stricter XHTML markup rules and stick with all lowercase, as I will do in this book.

TIP

Slash vs. Backslash

HTML tags and URLs use the slash character (/). The slash character is found under the question mark (?) on the standard QWERTY keyboard.

It is easy to confuse the slash with the backslash character (\), which is found under the bar character (|). The backslash key will not work in tags or URLs, so be careful not to use it.



Basic document structure

Figure 4-7 shows the recommended minimal skeleton of an HTML5 document. I say "recommended" because the only element that is *required* in HTML is the **title**. But I feel it is better, particularly for beginners, to explicitly organize documents with the proper structural markup. And if you are writing in the stricter XHTML, all of the following elements except **meta** must be included in order to be valid. Let's take a look at what's going on in Figure 4-7.

❶ I don't want to confuse things, but the first line in the example isn't an element at all; it is a document type declaration (also called DOCTYPE declaration) that identifies this document as an HTML5 document. I have a lot more to say about DOCTYPE declarations in Chapter 10, What's Up, HTML5?, but for this discussion, suffice it to say that including it lets modern browsers know they should interpret the document as written according to the HTML5 specification.

❷ The entire document is contained within an **html** element. The **html** element is called the root element because it contains all the elements in the document, and it may not be contained within any other element. It is used for both HTML and XHTML documents.

❸ Within the **html** element, the document is divided into a head and a body. The **head** element contains descriptive information about the document itself, such as its title, the style sheet(s) it uses, scripts, and other types of "meta" information.

❹ The **meta** elements within the **head** element provide information *about* the document itself. A **meta** element can be used to provide all sorts of information, but in this case, it specifies the character encoding (the standardized collection of letters, numbers, and symbols) used in the document. I don't want to go into too much detail on this right now, but know that there are many good reasons for specifying the **charset** in every document, so I have included it as part of the minimal document structure.

❺ Also in the **head** is the mandatory **title** element. According to the HTML specification, every document must contain a descriptive title.

❻ Finally, the **body** element contains everything that we want to show up in the browser window.

Are you ready to add some structure to the Black Goose Bistro home page? Open the *index.html* document and move on to Exercise 4-2.

NOTE

Prior to HTML5, the syntax for specifying the character set with the **meta** *element was a bit more elaborate. If you are writing your documents in HTML 4.01 or XHTML 1.0, your* **meta** *element should look like this:*

```
<meta http-equiv="content-
type" content="text/html;
charset=UTF-8">
```

Figure 4-7. The minimal structure of an HTML document.

```
❶ <!DOCTYPE html>

  <html>

❸   <head>
❹   <meta charset="utf-8">
❺   <title>Title here</title>
    </head>

❷

❻   <body>
    Page content goes here.
    </body>

  </html>
```

exercise 4-2 | Adding basic structure

1. Open the newly created document, *index.html*, if it isn't open already.
2. Start by adding the HTML5 DOCTYPE declaration:

```
<!DOCTYPE html>
```

3. Put the entire document in an HTML root element by adding an **<html>** start tag at the very beginning and an end **<html>** tag at the end of the text.
4. Next, created the document head that contains the title for the page. Insert **<head>** and **</head>** tags before the content. Within the head element, add informatino about the character encoding **<meta charset="utf-8">**, and the title, "Black Goose Bistro", surrounded by opening and closing **<title>** tags.

   The correct terminology is to say that the **title** *element is nested within the* **head** *element. We'll talk about nesting more in later chapters.*

5. Finally, define the body of the document by wrapping the content in **<body>** and **</body>** tags. When you are done, the source document should look like this (the markup is shown in color to make it stand out):

```
<!DOCTYPE html>
<html>

<head>
<meta charset ="utf-8">
<title>Black Goose Bistro</title>
</head>

<body>
Black Goose Bistro

The Restaurant
The Black Goose Bistro offers casual lunch and dinner fare in
a hip atmosphere. The menu changes regularly to highlight the
freshest ingredients.

Catering Services
You have fun... we'll do the cooking. Black Goose catering can
handle events from snacks for bridge club to elegant corporate
fundraisers.
Location and Hours
Seekonk, Massachusetts;
Monday through Thursday 11am to 9pm, Friday and Saturday, 11am to
midnight
</body>

</html>
```

6. Save the document in the bistro directory, so that it overwrites the old version. Open the file in the browser or hit "refresh" or "reload" if it is open already. Figure 4-8 shows how it should look now.

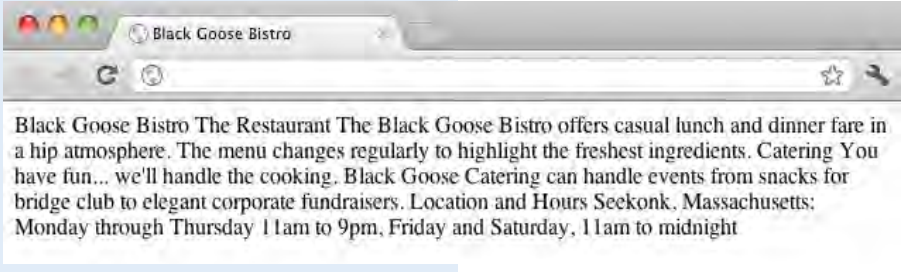


Figure 4-8. The page in a browser after the document structure elements have been defined.

Not much has changed after structuring the document, except that the browser now displays the title of the document in the top bar or tab. If someone were to bookmark this page, that title would be added to his Bookmarks or Favorites list as well (see the sidebar Don't Forget a Good Title). But the content still runs together because we haven't given the browser any indication of how it should be structured. We'll take care of that next.

Step 3: Identify Text Elements

With a little markup experience under your belt, it should be a no-brainer to add the markup that identifies headings and subheads (`h1` and `h2`), paragraphs (`p`), and emphasized text (`em`) to our content, as we'll do in Exercise 4-3. However, before we begin, I want to take a moment to talk about what we're doing and not doing when marking up content with HTML.

Introducing…semantic markup

The purpose of HTML is to add meaning and structure to the content. It is *not* intended to provide instructions for how the content should look (its presentation).

Your job when marking up content is to choose the HTML element that provides the most meaningful description of the content at hand. In the biz, we call this semantic markup. For example, the most important heading at the beginning of the document should be marked up as an `h1` because it is the most important heading on the page. Don't worry about what that looks like in the browser...you can easily change that with a style sheet. The important thing is that you choose elements based on what makes the most sense for the content.

In addition to adding meaning to content, the markup gives the document structure. The way elements follow each other or nest within one another creates relationships between the elements. You can think of it as an outline (its technical name is the DOM, for Document Object Model). The underlying document hierarchy is important because it gives browsers cues on how to handle the content. It is also the foundation upon which we add presentation instructions with style sheets and behaviors with JavaScript. We'll talk about document structure more in Part III, when we discuss Cascading Style Sheets, and in Part IV in the JavaScript overview.

Although HTML was intended to be used strictly for meaning and structure since its creation, that mission was somewhat thwarted in the early years of the web. With no style sheet system in place, HTML was extended to give authors ways to change the appearance of fonts, colors, and alignment using markup alone. Those presentational extras are still out there, so you may run across them if you view the source of older sites or a site made with old tools.

Don't Forget a Good Title

Not only is a `title` element required for every document, it is quite useful as well. The title is what is displayed in a user's Bookmarks or Favorites list and on tabs in desktop browsers. Descriptive titles are also a key tool for improving accessibility, as they are the first thing a person hears when using a screen reader. Search engines rely heavily on document titles as well. For these reasons, it's important to provide thoughtful and descriptive titles for all your documents and avoid vague titles, such as "Welcome" or "My Page." You may also want to keep the length of your titles in check so they are able to display in the browser's title area. Another best practice is to put the part of the title with more specific information first (for example, the page description ahead of the company name) so that the page title is visible when multiple tabs are lined up in the browser window.

In this book, however, we'll focus on using HTML the right way, in keeping with the contemporary standards-based, semantic approach to web design.

OK, enough lecturing. It's time to get to work on that content in Exercise 4-3.

exercise 4-3 | Defining text elements

1. Open the document ***index.html*** in your text editor, if it isn't open already.
2. The first line of text, "Black Goose Bistro," is the main heading for the page, so we'll mark it up as a Heading Level 1 (**h1**) element. Put the opening tag, **<h1>**, at the beginning of the line and the closing tag, **</h1>**, after it, like this:

   ```
   <h1>Black Goose Bistro</h1>
   ```

3. Our page also has three subheads. Mark them up as Heading Level 2 (**h2**) elements in a similar manner. I'll do the first one here; you do the same for "Catering" and "Location and Hours".

   ```
   <h2>The Restaurant</h2>
   ```

4. Each **h2** element is followed by a brief paragraph of text, so let's mark those up as paragraph (**p**) elements in a similar manner. Here's the first one; you do the rest.

   ```
   <p>The Black Goose Bistro offers casual lunch and
   dinner fare in a hip atmosphere. The menu changes
   regularly to highlight the freshest ingredients.
   </p>
   ```

5. Finally, in the Catering section, I want to emphasize that visitors should just leave the cooking to us. To make text emphasized, mark it up in an emphasis element (**em**) element, as shown here.

   ```
   <p>You have fun... <em>we'll handle the cooking
   </em>. Black Goose Catering can handle events
   from snacks for bridge club to elegant corporate
   fundraisers.</p>
   ```

6. Now that we've marked up the document, let's save it as we did before, and open (or refresh) the page in the browser. You should see a page that looks much like the one in Figure 4-9. If it doesn't, check your markup to be sure that you aren't missing any angle brackets or a slash in a closing tag.

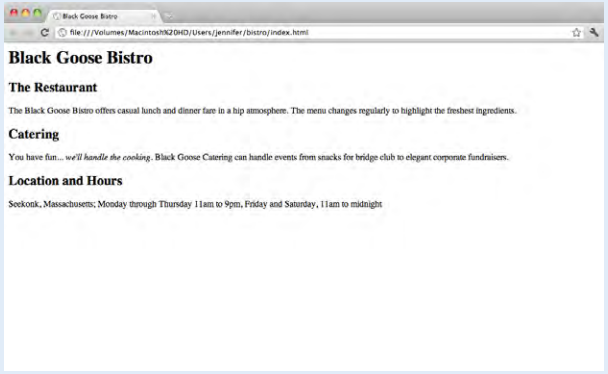


Figure 4-9. *The home page after the content has been marked up with HTML elements.*

Now we're getting somewhere. With the elements properly identified, the browser can now display the text in a more meaningful manner. There are a few significant things to note about what's happening in Figure 4-9.

Block and inline elements

Although it may seem like stating the obvious, it is worth pointing out that the heading and paragraph elements start on new lines and do not run together as they did before. That is because by default, headings and paragraphs display as block elements. Browsers treat block elements as though they are in little rectangular boxes, stacked up in the page. Each block element begins on a new line, and some space is also usually added above and below the entire element by default. In Figure 4-10, the edges of the block elements are outlined in red.

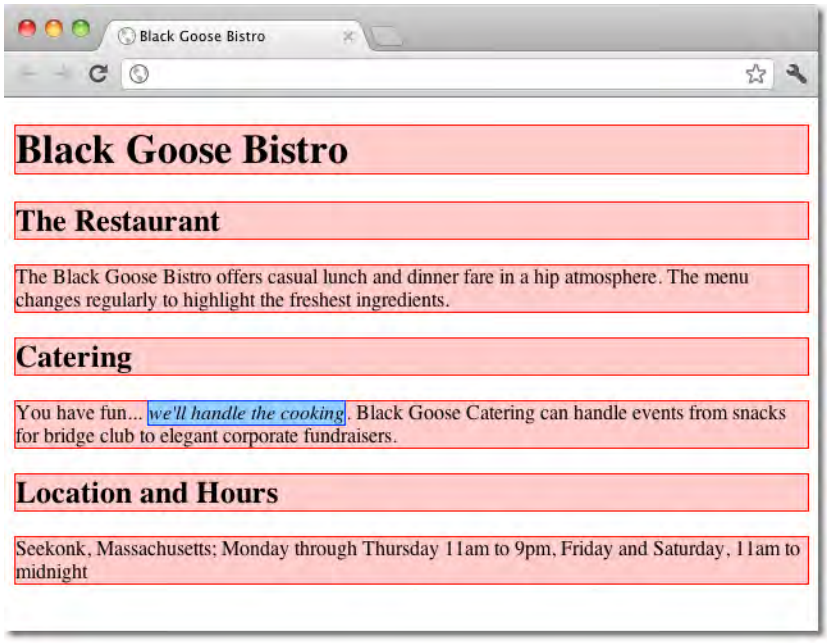


Figure 4-10. The outlines show the structure of the elements in the home page.

By contrast, look at the text we marked up as emphasized (`em`). It does not start a new line, but rather stays in the flow of the paragraph. That is because the `em` element is an inline element. Inline elements do not start new lines; they just go with the flow. In Figure 4-10, the inline `em` element is outlined in light blue.

Default styles

The other thing that you will notice about the marked-up page in Figures 4-9 and 4-10 is that the browser makes an attempt to give the page some visual hierarchy by making the first-level heading the biggest and boldest thing on the page, with the second-level headings slightly smaller, and so on.

How does the browser determine what an `h1` should look like? It uses a style sheet! All browsers have their own built-in style sheets (called user agent style sheets in the spec) that describe the default rendering of elements. The default rendering is similar from browser to browser (for example, `h1`s are always big and bold), but there are some variations (long quotes may or may not be indented).

If you think the `h1` is too big and clunky as the browser renders it, just change it with a style sheet rule. Resist the urge to mark up the heading with another element just to get it to look better, for example, using an `h3` instead of an `h1` so it isn't as large. In the days before ubiquitous style sheet support, elements were abused in just that way. Now that there are style sheets for controlling the design, you should always choose elements based on how

Adding Hidden Comments

You can leave notes in the source document for yourself and others by marking them up as comments. Anything you put between comment tags (`<!-- -->`) will not display in the browser and will not have any effect on the rest of the source.

```
<!-- This is a comment -->
<!-- This is a
  multiple-line comment
  that ends here. -->
```

Comments are useful for labeling and organizing long documents, particularly when they are shared by a team of developers. In this example, comments are used to point out the section of the source that contains the navigation.

```
<!-- start global nav -->
<ul>
 ...
</ul>
<!-- end global nav -->
```

Bear in mind that although the browser will not display comments in the web page, readers can see them if they "view source," so be sure that the comments you leave are appropriate for everyone. It's probably a good idea just to strip out notes to your fellow developers before the site is published. It cuts some bytes off the file size as well.

accurately they describe the content, and don't worry about the browser's default rendering.

We'll fix the presentation of the page with style sheets in a moment, but first, let's add an image to the page.

Step 4: Add an Image

What fun is a web page with no image? In Exercise 4-4, we'll add an image to the page using the **img** element. Images will be discussed in more detail in Chapter 7, Adding Images, but for now, it gives us an opportunity to introduce two more basic markup concepts: empty elements and attributes.

Empty elements

So far, nearly all of the elements we've used in the Black Goose Bistro home page have followed the syntax shown in Figure 4-1: a bit of text content surrounded by start and end tags.

A handful of elements, however, do not have text content because they are used to provide a simple directive. These elements are said to be empty. The image element (**img**) is an example of such an element; it tells the browser to get an image file from the server and insert it at that spot in the flow of the text. Other empty elements include the line break (**br**), thematic breaks (**hr**), and elements that provide information about a document but don't affect its displayed content, such as the **meta** element that we used earlier.

Figure 4-11 shows the very simple syntax of an empty element (compare to Figure 4-4). If you are writing an XHTML document, the syntax is slightly different (see the sidebar Empty Elements in XHTML).

Example: The `br` element inserts a line break.

```
<p>1005 Gravenstein Highway North<br>Sebastopol, CA 95472</p>
```

Figure 4-11. Empty element structure.

Empty Elements in XHTML

In XHTML, all elements, including empty elements, must be closed (or terminated, to use the proper term). Empty elements are terminated by adding a trailing slash preceded by a space before the closing bracket, like so: **<img />**, **
, **<meta />, and **<hr />**. Here is the line break example using XHTML syntax.

```
<p>1005 Gravenstein Highway
North <br />Sebastopol, CA
95472</p>
```

Attributes

Let's get back to adding an image with the empty **img** element. Obviously, an **<img>** tag is not very useful by itself—there's no way to know which image to use. That's where attributes come in. Attributes are instructions that clarify or modify an element. For the **img** element, the **src** (short for "source") attribute is required, and specifies the location (URL) of the image file.

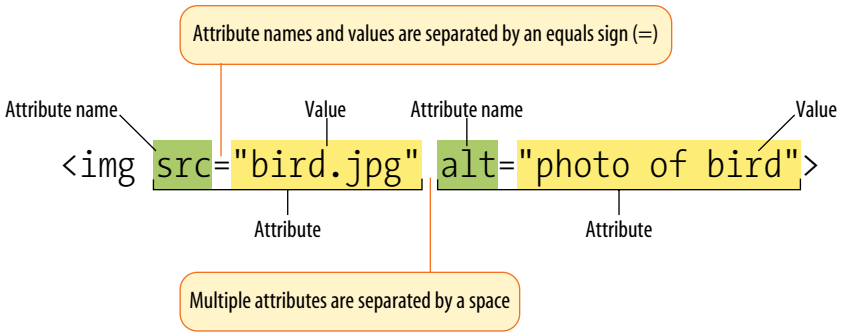


Figure 4-12. An `img` element with attributes.

The syntax for an attribute is as follows:

```
attributename="value"
```

Attributes go after the element name, separated by a space. In non-empty elements, attributes go in the opening tag only:

```
<element attributename="value">
<element attributename="value">Content</element>
```

You can also put more than one attribute in an element in any order. Just keep them separated with spaces.

```
<element attribute1="value" attribute2="value">
```

For another way to look at it, Figure 4-12 shows an **`img`** element with its required attributes labeled.

Here's what you need to know about attributes:

- Attributes go after the element name in the opening tag only, never in the end tag.
- There may be several attributes applied to an element, separated by spaces in the opening tag. Their order is not important.
- Most attributes take values, which follow an equals sign (=). In HTML, some attribute values can be reduced to single descriptive words, for example, the **`checked`** attribute, which makes a checkbox checked when a form loads. In XHTML, however, all attributes must have explicit values (**`checked="checked"`**). You may hear this type of attribute called a Boolean attribute because it describes a feature that is either on or off.
- A value might be a number, a word, a string of text, a URL, or a measurement, depending on the purpose of the attribute. You'll see examples of all of these throughout this book.
- Some values don't have to be in quotation marks in HTML, but XHTML requires them. Many developers like the consistency and tidiness of quotation marks even when authoring HTML. Either single or double quotation marks are acceptable as long as they are used consistently; however,

double quotation marks are the convention. Note that quotation marks in HTML files need to be straight (") not curly (”).

- Some attributes are required, such as the **src** and **alt** attributes in the **img** element.
- The attribute names available for each element are defined in the HTML specifications; in other words, you can't make up an attribute for an element.

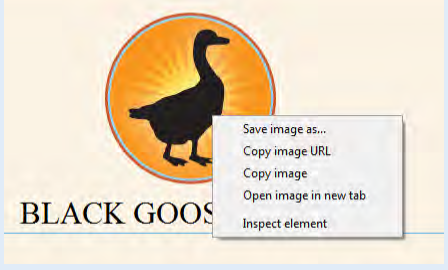
Now you should be more than ready to try your hand at adding the **img** element with its attributes to the Black Goose Bistro page in the next exercise. We'll throw a few line breaks in there as well.

exercise 4-4 | Adding an image

1. If you're working along, the first thing you'll need to do is get a copy of the image file on your hard drive so you can see it in place when you open the file locally. The image file is provided in the materials for this chapter. You can also get the image file by saving it right from the sample web page online at *www.learningwebdesign.com/4e/chapter04/bistro*. Right-click (or Ctrl-click on a Mac) on the goose image and select "Save to disk" (or similar) from the pop-up menu as shown in Figure 4-13. Name the file ***blackgoose.png***. Be sure to save it in the ***bistro*** folder with ***index.html***.
2. Once you have the image, insert it at the beginning of the first-level heading by typing in the **img** element and its attributes as shown here:

```
<h1><img src="blackgoose.png" alt="Black Goose logo">Black Goose
Bistro</h1>
```

   The **src** attribute provides the name of the image file that should be inserted, and the **alt** attribute provides text that should be displayed if the image is not available. Both of these attributes are required in every **img** element.



Windows:
Right-click on the image to access the pop-up menu

Mac:
Control-click on the image to access the popup menu. The options my vary by browser.

Figure 4-13. Saving an image file from a page on the Web.

3. I'd like the image to appear above the title, so lets add a line break (**br**) after the **img** element to start the headline text on a new line.

```
<h1><img src="blackgoose.png" alt="Black Goose logo"><br>Black
Goose Bistro</h1>
```

4. Let's break up the last paragraph into three lines for better clarity. Drop a **
** tag at the spots you'd like the line breaks to occur. Try to match the screenshot in Figure 4-14.
5. Now save ***index.html*** and open or refresh it in the browser window. The page should look like the one shown in Figure 4-14. If it doesn't, check to make sure that the image file, ***blackgoose.png***, is in the same directory as ***index.html***. If it is, then check to make sure that you aren't missing any characters, such as a closing quote or bracket, in the **img** element markup.

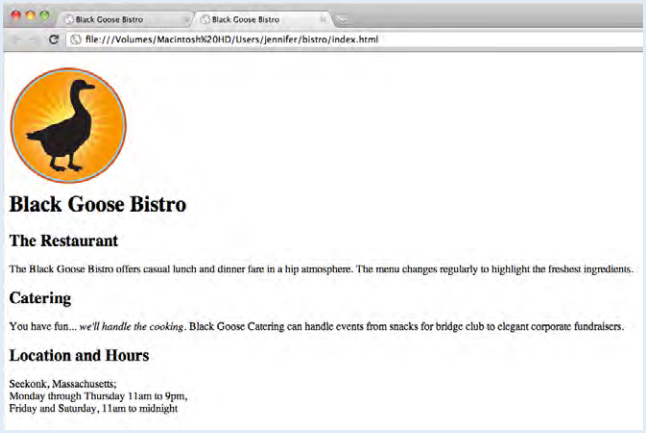


Figure 4-14. The Black Goose Bistro page with the logo image.

Step 5: Change the Look with a Style Sheet

Depending on the content and purpose of your website, you may decide that the browser's default rendering of your document is perfectly adequate. However, I think I'd like to pretty up the Black Goose Bistro home page a bit to make a good first impression on potential patrons. "Prettying up" is just my way of saying that I'd like to change its presentation, which is the job of Cascading Style Sheets (CSS).

In Exercise 4-5, we'll change the appearance of the text elements and the page background using some simple style sheet rules. Don't worry about understanding them all right now; we'll get into CSS in more detail in Part III. But I want to at least give you a taste of what it means to add a "layer" of presentation onto the structure we've created with our markup.

exercise 4-5 | Adding a style sheet

1. Open *index.html* if it isn't open already.
2. We're going to use the **style** element to apply a very simple embedded style sheet to the page. (This is just one of the ways to add a style sheet; the others are covered in Chapter 11, Style Sheet Orientation.)

   The **style** element is placed inside the **head** of the document. Start by adding the **style** element to the document as shown here:

```
<head>
  <meta charset="utf-8">
  <title>Black Goose Bistro</title>
  <style>

  </style>
</head>
```

3. Now, type the following style rules within the **style** element just as you see them here. Don't worry if you don't know exactly what is going on (although it is fairly intuitive). You'll learn all about style rules in Part III.

```
<style>

body {
  background-color: #faf2e4;
  margin: 0 15%;
  font-family: sans-serif;
  }

h1 {
  text-align: center;
  font-family: serif;
  font-weight: normal;
  text-transform: uppercase;
  border-bottom: 1px solid #57b1dc;
  margin-top: 30px;
}

h2 {
  color: #d1633c;
  font-size: 1em;
}

</style>
```

4. Now it's time to save the file and take a look at it in the browser. It should look like the page in Figure 4-15. If it doesn't, go over the style sheet code to make sure you didn't miss a semicolon or a curly bracket.

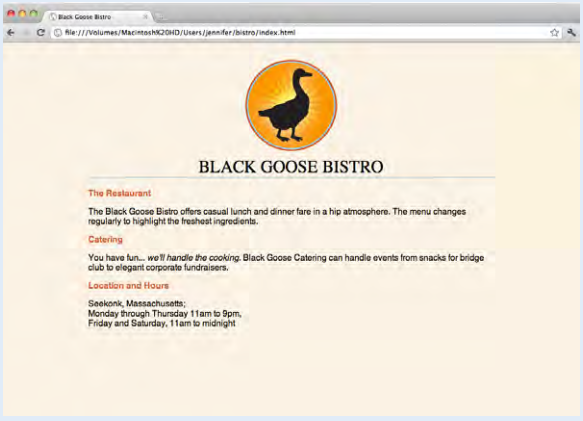


Figure 4-15. The Black Goose Bistro page after CSS style rules have been applied.

We're finished with the Black Goose Bistro page. Not only have you written your first web page, complete with a style sheet, but you've learned about elements, attributes, empty elements, block and inline elements, the basic structure of an HTML document, and the correct use of markup along the way. Not bad for one chapter!

When Good Pages Go Bad

The previous demonstration went smoothly, but it's easy for small things to go wrong when typing out HTML markup by hand. Unfortunately, one missed character can break a whole page. I'm going to break my page on purpose so we can see what happens.

NOTE

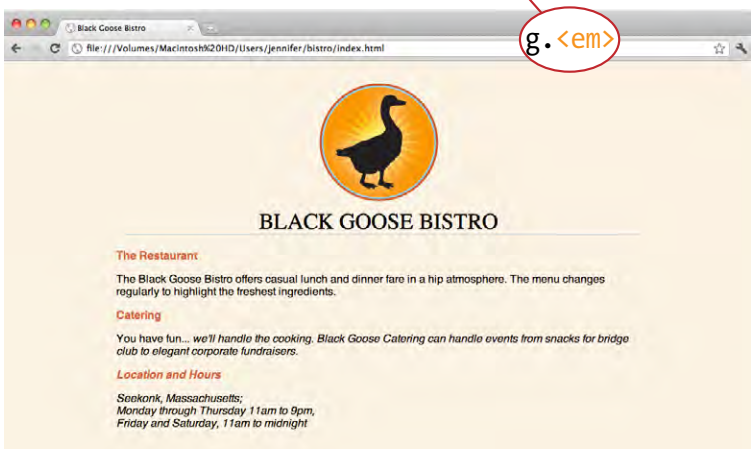
Omitting the slash in the closing tag (or even omitting the closing tag itself) for block elements, such as headings or paragraphs, may not be so dramatic. Browsers interpret the start of a new block element to mean that the previous block element is finished.

What if I had forgotten to type the slash (/) in the closing emphasis tag ()? With just one character out of place (Figure 4-16), the remainder of the document displays in emphasized (italic) text. That's because without that slash, there's nothing telling the browser to turn "off" the emphasized formatting, so it just keeps going.

I've fixed the slash, but this time, let's see what would have happened if I had accidentally omitted a bracket from the end of the first <h2> tag (Figure 4-17).

Figure 4-16. When a slash is omitted, the browser doesn't know when the element ends, as is the case in this example.

```
<h2>Catering</h2>
<p>You have fun... <em>we'll handle the cooking.<em> Black Goose
Catering can handle events from snacks for bridge club to elegant
corporate fundraisers.</p>
```



Browsers don't display any text within a tag, so my heading disappeared. The browser just ignored the foreign-looking element name and moved on to the next element.

Making mistakes in your first HTML documents and fixing them is a great way to learn. If you write your first pages perfectly, I'd recommend fiddling with the code as I have here to see how the browser reacts to various changes. This can be extremely useful in troubleshooting pages later. I've listed some common problems in the sidebar Having Problems? Note that these problems are not specific to beginners. Little stuff like this goes wrong all the time, even for the pros.

Figure 4-17. A missing end bracket makes all the following content part of the tag, and therefore it doesn't display.

```
<h2The Restaurant</h2>
<p>The Black Goose Bistro offers casual lunch and dinner fare
in a hip atmosphere. The menu changes regularly to highlight
the freshest ingredients.</p>
```

Without the bracket, all the following characters are interpreted as part of a long, unrecognizable element name, and "The Restaurant" disappears from the page.

Validating Your Documents

One way that professional web developers catch errors in their markup is to validate their documents. What does that mean? To validate a document is to check your markup to make sure that you have abided by all the rules of whatever version of HTML you are using (there are more than one, as we'll discuss in Chapter 10, What's Up, HTML5?). Documents that are error-free are said to be valid. It is strongly recommended that you validate your documents, especially for professional sites. Valid documents are more consistent on a variety of browsers, they display more quickly, and they are more accessible.

Right now, browsers don't require documents to be valid (in other words, they'll do their best to display them, errors and all), but any time you stray from the standard you introduce unpredictability in the way the page is displayed or handled by alternative devices.

So how do you make sure your document is valid? You could check it yourself or ask a friend, but humans make mistakes, and you aren't really expected to memorize every minute rule in the specifications. Instead, you use a validator, software that checks your source against the HTML version you specify. These are some of the things validators check for:

- The inclusion of a DOCTYPE declaration. Without it the validator doesn't know which version of HTML or XHTML to validate against.
- An indication of the character encoding for the document.
- The inclusion of required rules and attributes.
- Non-standard elements.
- Mismatched tags.
- Nesting errors.
- Typos and other minor errors.

Developers use a number of helpful tools for checking and correcting errors in HTML documents. The W3C offers a free online validator at *validator.w3.org*. For HTML5 documents, use the online validator located at *html5.validator.nu*. Browser developer tools like the Firebug plug-in for Firefox or the built-in developer tools in Safari and Chrome also have validators so you can check your work on the fly. If you use Dreamweaver to create your sites, there is a validator built into that as well.

Test Yourself

Now is a good time to make sure you understand the basics of markup. Use what you've learned in this chapter to answer the following questions. Answers are in Appendix A.

1. What is the difference between a tag and an element?

2. Write out the recommended minimal structure of an HTML5 document.

Having Problems?

The following are some typical problems that crop up when creating web pages and viewing them in a browser:

I've changed my document, but when I reload the page in my browser, it looks exactly the same.

It could be you didn't save your document before reloading, or you may have saved it in a different directory.

Half my page disappeared.

This could happen if you are missing a closing bracket (**>**) or a quotation mark within a tag. This is a common error when writing HTML by hand.

I put in a graphic using the `img` *element, but all that shows up is a broken image icon.*

The broken graphic could mean a couple of things. First, it might mean that the browser is not finding the graphic. Make sure that the URL to the image file is correct. (We'll discuss URLs further in Chapter 6, Adding Links.) Make sure that the image file is actually in the directory you've specified. If the file is there, make sure it is in one of the formats that web browsers can display (GIF, JPEG, or PNG) and that it is named with the proper suffix (***.gif***, ***.jpeg*** or ***.jpg***, or ***.png***, respectively).

3. Indicate whether each of these filenames is an acceptable name for a web document by circling "Yes" or "No." If it is not acceptable, provide the reason why.

   a. *Sunflower.html* Yes No

   b. *index.doc* Yes No

   c. *cooking home page.html* Yes No

   d. *Song_Lyrics.html* Yes No

   e. *games/rubix.html* Yes No

   f. *%whatever.html* Yes No

4. All of the following markup examples are incorrect. Describe what is wrong with each one, and then write it correctly.

   a. `<img "birthday.jpg">`

   b. `<i>Congratulations!<i>`

   c. `<a href="file.html">linked text</a href="file.html">`

   d. `<p>This is a new paragraph<\p>`

5. How would you mark up this comment in an HTML document so that it doesn't display in the browser window?

   ```
   product list begins here
   ```

Element Review: Document Structure

This chapter introduced the elements that establish the structure of the document. The remaining elements introduced in the exercises will be treated in more depth in the following chapters.

Element	Description
`body`	Identifies the body of the document that holds the content
`head`	Identifies the head of the document that contains information about the document
`html`	The root element that contains all the other elements
`meta`	Provides information about the document
`title`	Gives the page a title