
Chapter 2

Web Application Basics

Web applications evolved from Web sites or Web systems. The first Web sites, created by Tim Berners-Lee while at CERN (the European Laboratory for Particle Physics), formed a distributed hypermedia system that enabled researchers to have access to documents and information published by fellow researchers, directly from their computers. Documents were accessed and viewed with a piece of software called a browser, a software application that runs on a client computer. With a browser, the user can request documents from other computers on the network and render those documents on the user's display. To view a document, the user must start the browser and enter the name of the document and the name of the host computer where it can be found. The browser sends a request for the document to the host computer. The request is handled by a software application called a Web server, an application usually run as a service, or daemon, that monitors network activity on a special port, usually port 80. The browser sends a specially formatted request for a document (Web page) to the Web server through this network port. The Web server receives the request, locates the document on its local file system, and sends it back to the browser; see Figure 2-1.

This Web system is a hypermedia system because the resources in the system are linked to one another. The term Web comes from looking at the system as a set of nodes with interconnecting links. From one viewpoint, it looks like a spider's web. The links provide a means to navigate the resources of the system. Most of the links connect textual documents, but the system can be used to distribute audio, video, and custom data as well. Links make navigation to other documents easy. The user simply clicks a link in the document, and the browser interprets that as a request to load the referenced document or resource in its place.

A Web application builds on and extends a Web system to add business functionality. In its simplest terms, a Web application is a Web system that allows its users to execute business logic with a Web browser. This is not a very precise definition, but most people's conception of a Web application is not, either. There is a subtle distinction

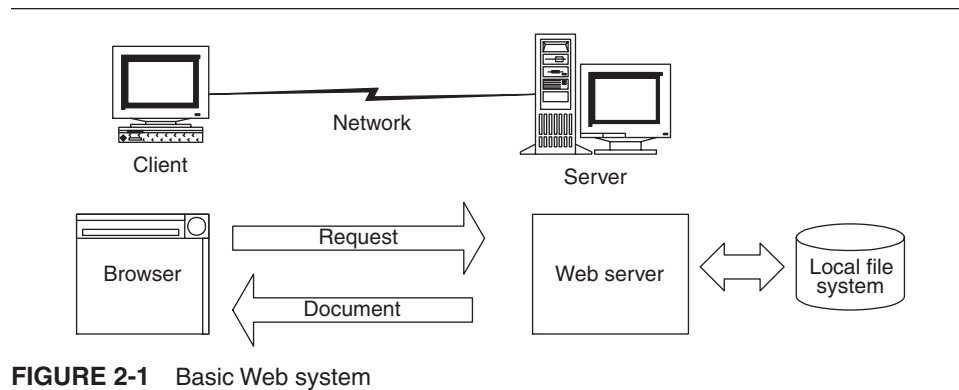


FIGURE 2-1 Basic Web system

between a Web application and a Web site. For the purpose of this book, a Web application is a Web site where user input—navigation through the site and data entry—affects the state of the business: beyond, of course, access logs and hit counters. In essence, a Web application uses a Web site as the front end to a business application.

HTTP

Browsers and Web servers use a special protocol, called the HyperText Transfer Protocol (HTTP), which specifies how a browser should format and send a request to a Web server. The client browser sends a document request consisting of a line of characters terminated by a CR/LF (carriage return/line-feed) pair. A well-behaved server will not require the carriage return character. This request consists of the word GET, a space, and the location of the document relative to the root of the Web server's file system. When a Web server/site is configured, it is usually set up to use a particular directory on the host machine's local file system as the Web site's root directory. Documents are found relative to this directory.

Document Identification

The full identifier for referencing and obtaining the document is called a uniform resource locator (URL). It identifies the protocol (HTTP), host machine name, optional port number, and document name/location. A URL is a single word with no white space. Any further words found on the request line are either ignored or treated according to the full HTTP spec.

A URL is a way to specify an object, or resource, on the network. A URL is like the network equivalent for specifying a file name on a file system. A URL can be used to request many types of objects with different protocols. In addition to HTTP, common Internet protocols include news, FTP, Gopher, and file. Each protocol is specific to the type of information or resource it represents.

When HTTP is specified, the object is a Web page. The following URL requests a Web page from a host identified by `www.wae-uml.org`:

```
http://www.wae-uml.org/specs/wd/cav43.html
```

The document name is `cav43.html` and is located in the directory `/specs/wd/`. This directory is relative to the Web site's root directory on the Web server.

A more explicit reference to this page could include the port number; however, the default port number, 80, is usually assumed for all HTTP requests:

```
http://www.wae-uml.org:80/specs/wd/cav43.html
```

It is possible to configure the Web server to listen to a port other than 80. This is often done to create a “private” Web site. Some Web servers monitor an additional port and use it for Web configuration. This allows Web masters—those responsible for managing a Web server and site—to remotely manage a Web server's configuration with just a browser. This type of configuration tool is an example of a small Web application.

Domain Names

A domain name is simply the textual name used to look up a numeric Internet address. The Internet addressing system in use today is the Internet Protocol (IP).¹ When a client requests a URL, the request must be made directly to the host computer. This means that the host name identified in the URL must be resolved into a numeric IP address. This process is done with a domain name server (DNS). A DNS is a network computer that the client has access to and that has a known IP address. Network infrastructure software on the client knows how to request the IP address from a DNS for a given domain name.

The host name `www.wae-uml.org` is made up of two distinct parts. The rightmost dot in the name is used to separate the host name from its top-level domain (TLD), in this case `com`. The `wae-uml` part is the subdomain. The term domain name often refers to the combination of the top-level domain and the subdomain. In this case, the domain name is `wae-uml.org`, and it is this name that I “own.” As owner of this domain name, I am responsible for ensuring that an IP address is associated with the domain. Reserving a domain name and not using it to host a Web site or application is referred to as “parking” the domain name, with the expectation that a real host will soon respond meaningfully to this domain name.

When I registered that domain name, I used one of the official registrars delegated the authority to assign domain names. In order to reserve a domain, I had to supply the

1. The version in use today is version 4 (IPv4) and is a 32-bit number. The next version, 6 (IPv6) — version 5 was skipped—is a 128-bit number and is expected to replace IPv4 in the coming years.

IP addresses of two name servers (DNS) that would act as the authoritative source for translating the domain name into a valid IP address. It is on this server that I have the rights to adjust, as necessary, the IP address that I want associated with the domain name and all its multilevel variations. So in my case, I created records in the DNS to equate `wae-uml.org`, `www.wae-uml.org`, and `test.wae-uml.org` to the specific IP address of my host computer.

The `www` and `test` parts are third-level domains, and their usage is up to the discretion of the host owner. The convention is to use `www` for HTML Web sites, `ftp` for FTP servers, `nnntp` for Usenet news servers, and so on, although this shouldn't be confused with the protocol specification part of a URL (see discussion of protocols in this chapter). Third-level domain names serve only as a convenience to host machine administrators and are not part of the domain name ownership process; nor do they impact types of protocols that are used.

Any host can receive the requests for multiple domains. When it makes an HTTP request for a resource to a server, a client application typically includes the full URL that was used to resolve the IP address. The server, with a single IP address, can receive a request and look at the URL to determine which application or separate Web site should handle the request. This is how many Internet service providers (ISPs) can offer basic hosting capabilities to customers with custom domain names on shared machines. I've set up my server to point to a portal application when the request is for `wae-uml.org` and `www.wae-uml.org` and to a simple test HTML Web site for `test.wae-uml.org`.

The two types of top-level domains are generic and country specific. An additional one, `.arpa`, is dedicated to Internet infrastructure. Recently, the list of generic top-level domains was expanded; however, `.com`, `.edu`, `.net`, and `.org` continue to be the ones most of us recognize. Each domain is intended for a particular type of use. For example, `.com` domains are for commercial businesses; `.net`, for Internet service providers; `.org`, for nonprofit organizations; and `.edu`, for educational institutions.

Country-specific domains are managed by organizations in individual countries and can define the usage of the second-level domain in any way they want. For example, the `.uk` top-level domain of the United Kingdom defines a number of second-level domains, such as `.co.uk`, `.me.uk`, `.org.uk`, `plc.uk`, and `ltd.uk`, each with its own expected uses. It isn't until the third-level domain is specified that individual organizations can claim ownership.

TLDs are managed by a single authority: Internet Corporation for Assigned Names and Numbers (ICANN, www.icann.org), with the distribution of Internet numbers managed by Internet Assigned Numbers Authority (IANA, www.iana.org). In the vast majority of situations, you will work through an official registrar—a company responsible for selling domains—or an ISP rather than work directly with these organizations.

Resource Identifiers

Related to the URL is the uniform resource identifier (URI) and the uniform resource name (URN).² Simply put, a URI is a superclass to URLs and URNs, whereas the URI is simply an identifier of a resource on the Internet, nothing more. A URL, on the other hand, is a name that includes an access mechanism: name of host server. The URN is required to “remain globally unique and persistent even when the resource ceases to exist or becomes unavailable.”³ For all practical purposes, we as Web application developers are interested mostly in URLs since it is these identifiers that we use to connect our Web pages to form a system.

Fault Tolerance

One important design goal of Web systems is that they be robust and fault tolerant. In the first Web systems at CERN, Web documents, computers, and network configurations were often subject to change. This meant that it was possible for Web pages to contain links to documents or host computers that no longer existed. It is even possible for the HTML specification itself to change, by adding elements, or tags. The browsers and Web servers of the system have to deal gracefully with these conditions.

This desire for a high degree of fault tolerance led in part to the decision to use a connectionless protocol, such as HTTP, as the principal protocol for managing document requests. HTTP is considered a connectionless protocol because as soon as the request is made and fulfilled, the connection between the client and server is terminated. The connection is broken by the server when the whole document has been transferred. The client can abort the transfer by breaking the connection before the transfer completes, in which case the server doesn’t record any error condition. The server doesn’t need to store information about the request after disconnection. This enables hosts and clients to act more independently and is more resistant to temporary network outages.

HTTP runs over TCP (Transmission Control Protocol), but could run over any connection-oriented service. TCP, a lower-level network protocol used by the Internet and many company networks, enables computers to make connections and to exchange information with one another. TCP, usually combined with IP, is an implementation of layers in the OSI (Open Systems Interconnection) model for network communications.

2. An excellent, detailed discussion of URIs by Tim Berners-Lee can be found at <http://www.w3.org/DesignIssues/Axioms.htm>.

3. The World Wide Web Consortium (W3C) report “URIs, URLs, and URNs: Clarifications and Recommendations,” at <http://www.w3.org/TR/2001/NOTE-uri-clarification-20010921/>, is an excellent resource for exploring this topic further.

HTTPS—HTTP with Secure Sockets Layer (SSL)—is related to HTTP but uses encryption to help “secure” the communication. HTTPS is used on the Internet for handling sensitive data such as personal and financial information. More detailed discussion of security and encryption is in Chapter 5, Security.

HTML

Browsers, in addition to establishing the network connections and protocols for document interchanges, need to render the document on a display. TCP/IP and HTTP don’t address this at all. The rendering of content is managed by the browser. This is where the Hypertext Markup Language (HTML) fits in. HTML, used to express the content and the visual formatting of a Web page, is a tag language based on the Standard Generalized Markup Language (SGML), which is a much broader language used to define markup languages for particular purposes. HTML is simply one specific application of SGML, suited to the presentation of textual documents. HTML contains tags that define how text is to be formatted—font, size, color, and so on—on the display. Tags are used to point to images to include in the display, as well as to define links to other Web pages. Like HTTP, HTML is an evolving standard managed by the World Wide Web Consortium standards body. The current W3C recommendation is HTML 4.01, which is based on the earlier HTML 3.2 and HTML 2.0 specifications.

One important thing to note is that HTML is a language that specifies how documents should be displayed on a computer screen. This raises several problems when the Web system needs to enable users to print formatted documents. Many documents, especially forms, have strict printing requirements. If a Web application needs to allow users to print forms or documents in which the typesetting is important—page breaks, font sizes, margins, and so on—additional elements will have to be added to the system. HTML does not address printing in detail. Several attempts to make HTML more suitable for printing have been made, but so far with little more than page break support, so Web applications that need strict printing capabilities must include additional components in their architecture.

HTML defines a set of tags that can be used either to tell the browser how to render something or to define a link to another Web page. All tags are enclosed by angle brackets (< and >). Tags are usually used in pairs, with a beginning and an ending tag. For example, to make a word italicized, the emphasis tag——is used. A sample sentence and the HTML to render it follow:

```
This is really neat.  
This is really <em>neat</em>.
```

Some tags accept parameters, which are placed inside the brackets and are usually a parameter name, followed by an equal sign and then the value enclosed by double

quotation marks. The following HTML for a hyperlink to another Web document uses the anchor tag, `<a>`:

The HTML 4 spec can be found at the `W3C Web site`.

Most browsers render the hyperlink with an underscore:⁴

The HTML 4 spec can be found at the W3C Web site.

The anchor tag uses the parameter `href` to define the location and the type of the link.

HTML pages are usually text files on the Web server's file system. The language was originally intended to be easy to learn, so that people interested in publishing content could easily specify how the content should be rendered. The key points here are "easy to learn" and "any display terminal." Because a Web system is, potentially, made up of many types of computers, a device-independent way was needed to specify basic formatting commands. For example, specifying a font by name would be a problem if the browser's computer didn't recognize the font name or didn't have the ability to render that font.

This was not a problem for early Web page writers, who were more interested in content than presentation. The language was simple enough to express only the basic formatting capabilities expected for the scientific community. The first generation of Web pages were all written manually, without the aid of WYSIWYG editors.

When the Internet and the Web became commercial, this simple language and its limitations did become a problem. Exact formatting of pages is very important to companies, especially e-commerce companies on the Internet. As in print advertisement, the look of a Web page is very important to potential customers and clients. HTML evolved to meet these needs, enabling more precise formatting of content by introducing new tags and parameters to the language.

What didn't change, though, was that the document content and the presentation information are coupled. The content of an HTML-formatted Web page is a mix of document content—the text or pictures that are displayed by the browser—and rendering instructions, such as bold, indent, font size, and so on. Figure 2-2 shows how HTML renders in a browser.

All the formatting commands are embedded with the content of the document. It is possible to separate some of the formatting specifics from the content with style sheets; even with their use, however, complete separation is not possible. A related term and technology—content management, discussed in Chapter 7, Defining the Architecture—provides more concrete examples of how the separation of content from formatting is an ongoing and important topic to Web application developers.

4. This is the default behavior. For any given Web page and browser instance, this behavior is determined by a combination of browser default settings and style sheets referenced or included in the HTML document.

(a) `<p>The new HTML
4.0 specification includes
additional support for</p>

 Style sheets
 Internationalization
 Accessibility
 Tables and forms
 Scripting and multimedia
`

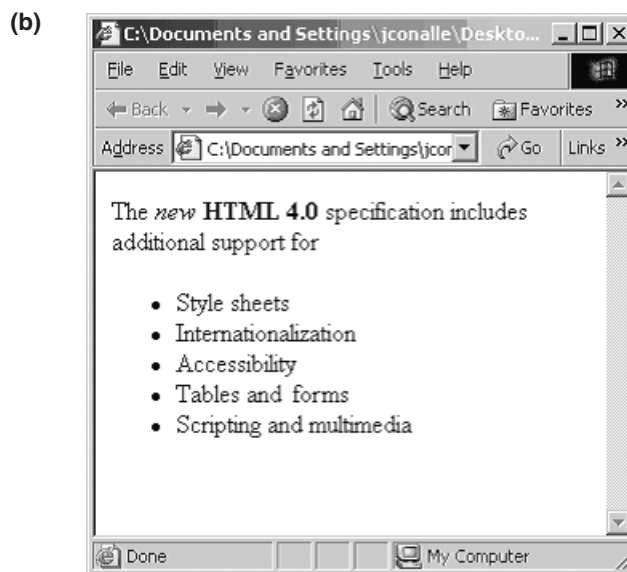


FIGURE 2-2 (a) HTML content (b) rendered in a browser

Style sheet specifications allow Web page authors to define a separate layout, such as color, font, and margins, template that could be used by many other content documents. This helps maintain a consistent look-and-feel across a Web site.

It should be noted that some browsers, especially older and nontraditional browsers, such as PDAs (personal digital assistants) and mobile phones, do not support the use of style sheets. Even so, the W3C recommends the use of style sheets, and is continuing to refine HTML so that in the future, there will be an even further separation of content and presentation. The emergence of XML (Extensible Markup Language), discussed in Chapter 4, Beyond HTTP and HTML, plays a key role in this separation.

One mechanism for separating the content of an HTML page from its presentation is server-side includes (SSI). The NCSA (National Center for Supercomputing Applications) has defined a simple tag that can be used to include HTML fragments that need to be shared across a number of response pages. Typically, these are standard

prologs, epilogs, or legal disclaimers and copyright notices. By spreading the content on several shared pages, it is possible to manage a single point of update.⁵ The tag simply specifies the name of a file in the Web site's file system:

```
<!--#include file="banner.html"-->
```

HTML has special tags that allow the Web page author to use multimedia information such as images, sound, and video, in addition to textual information. Instead of embedding the multimedia data in the page with the text, the tags specify separate URLs for each item. This means that the browser will make an additional request and connection to the server for each image or multimedia item mentioned in the original Web page. To the Web application architect and designer, this means that certain exceptions need to be taken into account when designing the application. These situations usually are invalid URLs or missing files. It also implies that Web applications with lots of images and multimedia types may cause significant network performance penalties when establishing so many discrete connections. Remember, the choice of a connectionless protocol benefits the robustness of the system, not its performance.

A full discussion of HTML and its tags will not be presented here; the topic is beyond the scope of this book and has been done very well in countless other books. What does need to be discussed here is the architecturally significant elements of the language, especially as they relate to Web applications. Like separating content from presentation, the architecturally significant elements of a Web page need to be brought out and modeled as such. For example, a design model of a Web application is not very interested in the font size or color of the text used in a display, but it is very interested in the sets of Web pages that can be navigated to. The following sections discuss these important elements of HTML.

Anchors

A hyperlink to a Web page is created with the HTML anchor tag: `<a>`. The tag uses several parameters, the most important of which is `href`. This tag can be used without an `href` value, but in this case, the tag is being used as an internal bookmark of the page and as such is not architecturally significant to the design model but could be significant in the user experience model (see Chapter 9). The `href` parameter, which specifies the linked document's URL, may contain a relative URL, which doesn't specify the full URL, including the host's name, but rather expects the browser to use as the host the host that supplied the page containing the hyperlink. For example, the following anchor tag is perfectly acceptable:

We have a full line of `products` to choose from.

5. See the discussion on content management in Chapter 9, The User Experience, for additional ways to manage content as separate from the rest of the system.

In this example, the link is to a page on the same machine and in the same directory as the current page.

In addition to the location and the name of the Web page, an anchor can pass along parameters with the page request. When parameters are specified in a page request, it usually means that the requested page is executable. The requested Web page is capable of accessing the parameter information and using it to build the returned page. Parameters are passed with the request as name/value pairs separated by the ampersand symbol (&). The parameters are separated from the Web page by a question mark (?). The following page request passes along two parameters: ProductID and RateCode. The ProductID is assigned the value 452, and the RateCode is given a value of B.

```
http://www.mystore.com/catalog/products.jsp?ProductID=452&RateCode=B
```

The requested page is `products.jsp`. The extension of the Web page gives a clue that the enabling technology used by the executable page is JavaServer Pages.

In addition to `href`, the other significant parameter is `target`. When a hyperlink is selected, it typically loads the new document in the same browser window as the original document. This is not always the case. When frames are used to divide up a browser's display area, each frame displays a separate Web page. Frames are discussed in detail later.

When frames or multiple browsers are used, it is possible for a hyperlink to specify a frame or browser instance to load in. Frames and browser instances can be assigned any name, although a few target names are reserved.

- | | |
|----------------------|--|
| <code>_blank</code> | Makes the link load into a new blank browser window. The new window is not assigned a name. |
| <code>_parent</code> | Makes the link load in the immediate frameset parent of the document. It defaults to <code>_self</code> if the document has no parent. |
| <code>_self</code> | Makes the link load in the same window the anchor was clicked in. This is the default behavior of an anchor. |
| <code>_top</code> | Makes the link load in the full body of the window. It is a way to "break out" of a frameset. |

In the following example, the anchor tag specifies a named `target`: `maindoc`.

```
<a href="chap1.html" target="maindoc" >  
    Chapter 1, Web Application Basics.  
</a>
```

Forms

HTML form elements distinguish a Web site from a Web application. The HTML form part of a Web page can accept user input. An HTML form is a collection of fields that allow users to enter text or to select from a list. In addition to text boxes, form

fields can be rendered by buttons, check boxes, and radio buttons. If a Web page has a form in it, the browser will render that form with the appropriate user interface controls and allow the user to enter and to change its values. Most forms have a special button in them that, when clicked by the user, submits the form and its contents to the Web server as part of another Web page request. The Web server receives a request for a special Web page, or executable module, that is capable of reading the form's field values and processing them on the server. The ultimate result is a new HTML page that is sent back to the requesting browser; see Figure 2-3.

More detailed discussions of form-processing enabling technologies are given later in this book. The general concept, however, is that the executable page is used by the Web server to process the form's values and to produce a new HTML page to send back to the browser. Most often, the processing involves communicating with objects on the server or with databases. Forms are a key mechanism in the interaction of Web application users but by no means the only one. More sophisticated mechanisms for getting user input are discussed in Chapter 3, Dynamic Clients.

A form is defined by the `<form>` tag. The two principal parameters are `action` and `method`. The `action` parameter is the URL of the executable Web page that processes the form. The `method` parameter specifies how the data will be sent to the server. The two valid values are `GET` and `POST`. When `GET` is used, the values of all the fields in the form are appended to the URL as parameters. The Web server sees the form submission as a typical `GET` request, as if it were from a standard anchor tag. The W3C does not recommend using `GET`, as it has some internationalization problems and will not work for large forms. Instead, the value `POST` is preferred. The `POST` method tells the browser to package up the field values in a special section of the request called the data body.

Plain-vanilla HTML has only a few core form elements: `<input>`, `<select>`, and `<textarea>`. The `<select>` tag specifies either a list box or a drop-down list from which the user can select something. The `<textarea>` tag is a multiline text input

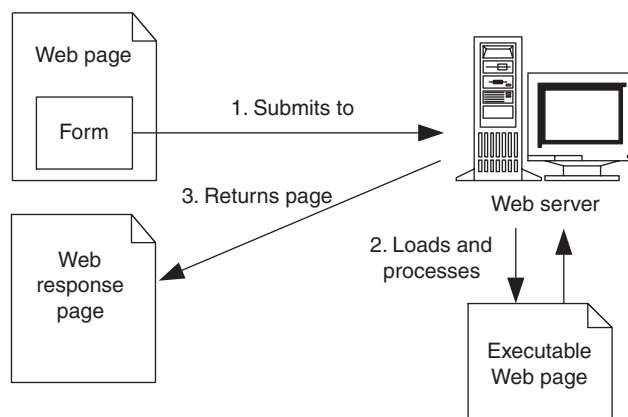


FIGURE 2-3 An executable Web page processing a form's data

control and allows users to enter in large blocks of text. The `<input>` tag is really an overloaded tag that could be configured to act like a push button, a check box, a radio button, or a single-line text entry field.

The `<input>` tag's name parameter identifies the field's name. This name is used by the executable Web page when processing the form's data. The `<input>` tag's type parameter determines what type of user interface control should be used and what type of data to accept. The most common values are:


Checkbox	Displays a check box. If the user checks the box, the field's value will be assigned the value specified by the <code><input></code> tag's value parameter.
Hidden	Does not display a user interface control. Values are usually set here by either the executable page that created it or client-side scripting and dynamic HTML.
Password	Displays a password-style entry field. Characters entered here are not displayed to the user.
Radio	Displays a radio button. The tag's name value defines the radio button group. When selected, the field associated with the radio button group is assigned the value specified by the value parameter.
Submit	Displays a push button. When the user clicks this button, the form and all its values are submitted to the server.
Text	Displays a single-line text entry box.

Other types exist, but these define the core set of input types that are used and available in most browser implementations. The key point here is that simple forms collect textual information from the user—either directly or by translating a check box, a button, or a list box selection into one—and define the mechanism by which it gets submitted to the server.

The following HTML fragment defines a simple form for collecting log-on information:

```
<form method="POST" action="cgi-bin/logon.pl">
<p>Username:
<input type="text" name="username" size="10"></p>
<p>Password:
<input type="password" name="password" size="10"></p>
<p>Logon as:</p>
<p>
<input type="radio" name="Role" value="Supervisor">Supervisor<br>
<input type="radio" name="Role" value="Clerk">Clerk<br>
<input type="radio" name="Role" value="Guest">Guest
</p>
<p><input type="submit" value="Logon" name="LogonBtn"></p>
</form>
```

The browser's rendering of the form is shown in Figure 2-4.



Username:

Password:

Log on as:

☒ Supervisor
☐ Clerk
☐ Guest

FIGURE 2-4 Rendered HTML form

Frames

A controversial element in the HTML arsenal is a frameset. A frameset divides up the browser's display area into rectangular regions, each rendering its own HTML document. The `<frameset>` tag defines the number of frames the display should be broken up into and their sizes or proportions. Separate `<frame>` tags identify each of the frames with a target name. The `<frame>` tags also tell the browser which Web pages to request for each frame when initializing the page. Once a frameset page is loaded, with all its individual frames loaded as well, the user can work with the page. The user can select a hyperlink in any of the displayed pages. The link might specify a new document for the frame that it is in or a page to be loaded in another named frame.

The most common use of frames is to define a table of contents and main document frame. The Web page in the table of contents frame is typically a long list of table of content entries, each a hyperlink to an area of the Web site. Each of the links specifies that the "main document" frame, or target, is where the linked Web page should be rendered. The output produced from the JavaDoc application is another excellent example of the use of frames.

The parameters `cols` and `rows` of the `<frameset>` tag define the initial size and the number of frames in the set. For example, the value `"20%,50%,*"` specifies three frames to be defined. The first occupies 20 percent of the screen; the second, 50 percent; the third, the remaining space, 30 percent. Instead of percentages, explicit widths can also be defined, and that's where the `*` value becomes useful. Of course, it is entirely possible for a frame to contain another frameset. This allows designers a little more freedom from a simple matrix frame design and can be used to produce any combination of rectangular regions in the browser's display.

The following HTML fragment defines a simple table of contents-like page. The table of contents appears in the leftmost frame and occupies 20 percent of the display.

```
<frameset cols="20%,80%">
  <frame name="toc" src="toc.html">
  <frame name="maindoc" src="intro.html">
</frameset>
```

Names are specified for each frame. A link in the table of contents frame would specify "maindoc" as the target for the link. For example, clicking on the following HTML link would display the Chapter 1 page in the main document frame. The table of contents frame would remain the same.

```
<a target="maindoc" href="chapter1.html">Chapter 1. Web Application Basics</a>
```

The frameset Web page itself typically doesn't contain content like other Web pages do. Most do contain enough content to tell the user that a frames-capable browser is needed to view the page and to provide a link to a page that doesn't require one. This is sometimes necessary on the Internet, as not all browsers support frames.

The controversy over frames centers on user interface preferences and complexity. Some people don't like frames. Frames do, however, raise the level of complexity a bit, as now multiple Web pages are interacting with the user at the same time. This is what makes frames an architecturally significant element.

Web Applications

Web applications use enabling technologies to make their content dynamic and to allow users of the system to affect business logic on the server. The distinction between Web sites and Web applications is subtle and relies on the ability of a user to affect the state of the business logic on the server. Certainly, if no business logic exists on a server, the system should not be termed a Web application. For those systems on which the Web server—or an application server that uses a Web server for user input—allows business logic to be affected via Web browsers, the system is considered a Web application. For all but the simplest Web applications, the user needs to impart more than just navigational request information; typically, Web application users enter a varied range of input data: simple text, check box selections, or even binary and file information.

The distinction becomes even more subtle in the case of search engines, on which users do enter in relatively sophisticated search criteria. Search engines that are Web sites simply accept this information, use it in some form of database SELECT statement, and return the results. When the user finishes using the system, there is no noticeable change in the state of the search engine—except, of course, in the usage logs and hit counters. This is contrasted with Web applications that, for example, accept online registration information. A Web site that accepts course registration information from a user has a different state when the user finishes using the application.

The architecture for a Web site is straightforward. It contains the same principal components of a Web site: a Web server, a network connection, and client browsers. Web applications also include an application server. The addition of the application

server enables the system to manage business logic and state. A more detailed discussion of Web application architectures is given in Chapter 7, *Defining the Architecture*.

Client State Management

One common challenge of Web applications is managing client state on the server. Owing to the connectionless nature of client and server communications, a server doesn't have an easy way to keep track of each client request and to associate it with the previous request, since each and every Web page request establishes and breaks a completely new set of connections.

Managing state is important for many applications; a single use case scenario often involves navigating through a number of Web pages. Without a state management mechanism, you would have to continually supply all previous information entered for each new Web page. Even for the most simple applications, this can get tedious. Imagine having to reenter the contents of your shopping cart every time you visit it or to enter in your user name and password for each and every screen you visit while checking your Web-based e-mail.

To address this common problem, the W3C has proposed an HTTP state management mechanism.⁶ This mechanism, more commonly known as “cookies,” has received quite a bit of attention from privacy advocates in the past few years and will most likely continue to as more and interesting uses of this mechanism are found. This book isn't about privacy concerns but rather is focused on the technology around Web applications, so I'll focus on describing the technology and leave the philosophy to you.

Cookies

A cookie is a piece of data that a Web server can ask a Web browser to hold on to, and to return every time the browser makes a subsequent request for an HTTP resource to that server. Typically, the size of the data is small, between 100 and 1K bytes; however, the official limit is around 4K. Initially, a cookie is sent from a server to a browser by adding a line to the HTTP headers:

```
Content-type: text/html
```

```
Set-Cookie: sessionId=12345; path=/; expires Mon, 09-Dec-2002 11:21:00 GMT; secure
```

If the browser is configured to accept cookies, the line is accepted and stored somewhere on the client's machine, depending on the browser vendor. After that, each and every HTTP request to the server is sent back the values of these cookies.

When it is sent to a client, a cookie can have up to six parameters passed with it:

- Name (required)
- Value (required)

6. All the gory details of this mechanism can be found in the RFC 2109 document at <http://www.w3.org/Protocols/rfc2109/rfc2109>.

- Expiration date
- Path
- Domain
- Requires a secure connection

The Set-Cookie header is a string that contains characters, including white space, commas, and the semicolon. The name and value parameters are required and must not contain white space, commas, or semicolons. The expiration date tells the browser how long to keep this information. The path and the domain are a way of determining which servers, or domains, to send the cookies back to. If the domain is not set explicitly, it defaults to the full domain of the document creating the cookie. The path helps organize cookies within a domain. Only when a resource is requested in the domain under the path will the cookie be sent back to the server.

The server sending the cookie must be a member of the domain that is specified. Thus, a server in the domain `www.myserver.com` cannot set a cookie for the domain `www.otherserver.com`. If it could, one company would be able to set cookies in another company's domain.

The server can send multiple Set-Cookie headers with an HTTP response. When the browser responds with the Set-Cookie header, all cookies for the domain and the path are returned. For example, the server could have included the following Set-Cookie headers:

```
Set-Cookie: sessionId=12345; path=/; expires Mon, 09-Dec-2002 11:21:00 GMT
Set-Cookie: colorPref=Blue; path=/; expires Mon, 09-Dec-2002 11:21:00 GMT
```

When it requests a URL in the path `/` on this same server, the client sends with its HTTP request the following:

```
Cookie: sessionId=12345; colorPref=Blue
```

When the response is returned, the server might set another cookie with:

```
Set-Cookie: rateCode=B; path=/order
```

When it requests a URL in path `/order`, a client sends:

```
Cookie: sessionId=12345; colorPref=Blue; rateCode=B
```

Note that all three cookies are sent with the request because the first two are in a higher path and are “inherited” in the mapping.

In addition to the server's being able to set a cookie value, so too can JavaScript. Chapter 3, *Dynamic Clients*, describes the capabilities of client-side scripting in more detail; here, however, it is sufficient to say that cookies can be set and obtained in multiple ways. The specific mechanisms for setting and accessing cookies are typically

provided by the development environment and architecture, by a single function call to an accessible object.

This mechanism is not without faults. Privacy advocates point to cookies as the primary mechanism supporting the tracking of unknowing users across multiple Web sites. In fact, while writing this chapter, I wanted to look at some sample cookies on my machine. When I scanned the list, I was surprised to find a few cookies from domains that I know I had never visited. Of course, this piqued my curiosity, and as I looked at the data in the cookies, I noticed name/value pairs that included URLs from sites I do remember visiting. Investigating a little further, I found out that these cookies were placed on my machine through the use of banner ads that appeared in the sites that I did visit.

The reality here is that the images in most banner ads are not hosted by the sites that referenced them. Rather, companies specializing in banner ads sell a service to Web sites. When someone visits those sites, the companies provide most of the content of the Web page, as well as a reference to an image stored on the advertisement company's server. Because the image is obtained with a standard HTTP request, the exchange of cookies also happens with this "other" server. So when you visit a Web page that has banner ads in it, they most likely are coming from another company's server and are being collected and managed by that company. After a while, you will visit enough Web sites using the same advertiser's server that the banner ad company can start to build a profile of the sites you visit most and begin to target more appropriate advertisement for you.

Using cookies in this way is very controversial and has led to the heated debate on the use of cookies, privacy, and the Internet. But we won't focus on that type of usage here. Instead, we'll look at how cookies were intended to be used: to manage client state in the context of a single use case or set of use cases.

Sessions

A session represents a single cohesive use of the system. A session usually involves many executable Web pages and a lot of interaction with the business logic on the application server. Because achieving a use case goal often requires the successful execution of a number of executable Web pages, it is often useful to keep track of a client's progress throughout the use case session.

The most common example of keeping client state on the server can be found on the Internet at any e-commerce site. The use of virtual shopping carts is a nice feature of an online store. A shopping cart contains all the items an online customer has selected from the store's catalog. In most sites, the shopper can check the contents of the cart at any time during the session. This feature requires that the server be capable of maintaining some state about the client across a series of Web page requests.

Session state in a Web application can be maintained in four common ways, two of which require the use of cookies:

1. Place all state values in cookies.
2. Place a unique key in the cookie and use with a server-managed dictionary or map.

3. Include all state values as parameters in every URL of the system.
4. Include a unique key as a parameter in every URL of the system and use with a server-managed dictionary or map.

When you place all state values in cookies, you are first limited by size (4K) and at most 20 cookies per domain. All state data must be encoded into simple text: no white space, semicolons, and so on. You can't directly use higher-level objects in the session state. The real limitation, however, is that many clients' security settings don't allow the automatic storing of cookies. If the application is an Internet application targeting the consumer market, you don't want to automatically turn away a significant number of potential customers without a good reason.

When a unique key is used in a cookie and then used on the server as a key into a dictionary or a map, any type of server-side object can be part of the session state. This is the default mechanism used by most Web application-enabling environments, such as ASP and JSP. It is very effective and flexible; however, like any cookie-based method, it depends on the willingness of clients to accept cookies.

URL redirection is the other class of session management. In this mechanism, all URLs in the system are dynamically constructed to include parameters that contain either the entire session state or only one key into a server-side dictionary.

Each mechanism has tradeoffs. Keeping a dictionary in memory for every user of the system could be very expensive if it never expired. For practical reasons, most session dictionaries are removed when the Web application user either finishes the process or stops using the system for a set period of time. A session timeout value of 15 minutes is typical. No matter what technique is used, the management of client state on the server is almost always an issue in Web applications.

Enabling Technologies

The enabling technologies for Web applications are varied and differentiated principally by the vendor. Enabling technologies are, in part, the mechanism by which Web pages become dynamic and respond to user input. Of the several approaches to enabling a Web application, the earliest involved the execution of a separate module by a Web server. Instead of requesting an HTML-formatted page from the file system, the browsers would request the module, which the Web server interpreted as a request to load and to run the module. The module's output is usually a properly formatted HTML page but could be image, audio, video, or other data.

The original mechanism for processing user input in a Web system is the Common Gateway Interface (CGI), a standard way to allow Web users to execute applications on the server. Because letting users run applications on your Web server might not be the safest thing in the world, most CGI-enabled Web servers require CGI modules to reside in a special directory, typically named `cgi-bin`. CGI modules can be written in any language and can even be scripted. In fact, the most common language for small-scale CGI modules is Perl (practical extraction and reporting language), which is interpreted each time it is executed.

Even though HTML documents are the most common output of CGI modules, they can return any number of document types. They can send to the client an image, plaintext—an ASCII document with no special formatting—audio, or even a video clip. They can also return references to other documents. In order for it to interpret the information properly, the browser must know what kind of document it is receiving. In order for the browser to know this, the CGI module must tell the server what type of document it is returning.

In order to tell the server what kind of document is being sent back—a full document or a reference to one—CGI requires a short header on the output. This header is ASCII text, consisting of separate lines followed by a single blank line. For HTML documents, the line would be

```
Content-type: text/html
```

If it does not build the returning HTML Web page, the CGI module can redirect the Web server to another Web page on the server or even another CGI module. To accomplish this, the CGI module simply outputs a header similar to

```
Location: /responses/default.html
```

In this example, the Web server is told to return the page `default.html` from the `responses` directory.

The two biggest problems with CGI are that it doesn't automatically provide session management services and that every execution of the CGI module requires a new and separate process on the application/Web server. Creating a lot of processes can be expensive on the server.

All the available solutions overcome the multiprocess problems of CGI by adding plug-ins to the Web server. The plug-ins allow the Web server to concentrate on servicing standard HTTP requests and deferring executable pages to another, already running process. Some solutions, such as Microsoft's Active Server Pages, can even be configured to run in the same process and to address space as the Web server itself, although this is not recommended.

Two major approaches to Web application-enabling technologies are used today: compiled modules and interpreted scripts. Compiled-module solutions are CGI-like modules that are compiled loadable binaries executed by the Web server. These modules have access to APIs that provide the information submitted by the request, including the values and names of all the fields in the form and the parameters on the URL. These modules produce HTML output that is sent to the requesting browser. Some popular implementations of this approach are Microsoft's Internet Server API (ISAPI), Netscape Server API (NSAPI), and Java servlets.

ISAPI and NSAPI server extensions can also be used to manage user authentication, authorization, and error logging. These extensions to the Web server are essentially a filter placed in front of the normal Web server's processing.

Compiled modules are an efficient, suitable solution for high-volume applications. The biggest drawbacks are related to development and maintenance. These modules usually combine business logic with HTML page construction. The modules often contain many print lines of HTML tags and values, which can be confusing and difficult for a programmer to read.

The other problem is that each time the module needs to be updated, or fixed, the Web application has to be shut down and the module unloaded. For most mission-critical applications, this is not much of a problem; the rate of change in the application should be small. Also, it's likely that a significant effort would have been made by the QA/test team to ensure that the delivered application was free of bugs. For smaller, internal intranet applications, however, the rate of change might be significant. For example, the application might provide sets of financial or administrative reports. The logic in these reports might change over time, or additional reports might be requested.

The other category of solutions is scripted pages. Whereas the compiled-module solution looks like a business logic program that happens to output HTML, the scripted-page solution looks like an HTML page that happens to process business logic. A scripted page, a file in the Web server's file system, contains scripts to be interpreted by the server; the scripts interact with objects on the server and ultimately produce HTML output. The page is centered on a standard HTML Web page but includes special tags, or tokens, that are interpreted by an application server. Typically, the file name's extension tells the Web server which application server or filter should be used to preprocess the page. Some popular vendor offerings in this category are JavaServer Pages, Microsoft's Active Server Pages, and PHP.

Figure 2-5 shows the relationship between components of the enabling technology and the Web server. The database in the figure, of course, could be any server-side resource, including external systems and other applications. This figure shows how the compiled-module solution almost intercepts the Web page requests from the Web server and in a sense acts as its own Web server. In reality, the compiled module must be registered with the Web server before it can function. Nonetheless, the Web server plays only a small role in the fulfillment of these requests.

The scripted-page solution, however, is invoked by the Web server only after it has determined that the page does indeed have scripts to interpret. Typically, this is indicated by the file name extension: `.aspx`, `.jsp`, `.php`. When it receives a request for one of these pages, the Web server first locates the page in the specified directory and then hands that page over to the appropriate application server engine, or filter. The application server preprocesses the page, interpreting any server-side scripts in the page and interacting with server-side resources, if necessary. The results are a properly formatted HTML page that is sent to the requesting client browser.

Even though JavaServer Pages are scripted, they get compiled and loaded as a servlet the first time they are invoked. As long as the server page doesn't change, the Web server will continue to use the already compiled server page/servlet. This gives JavaServer Pages some performance benefits over the other scripted-page offerings.

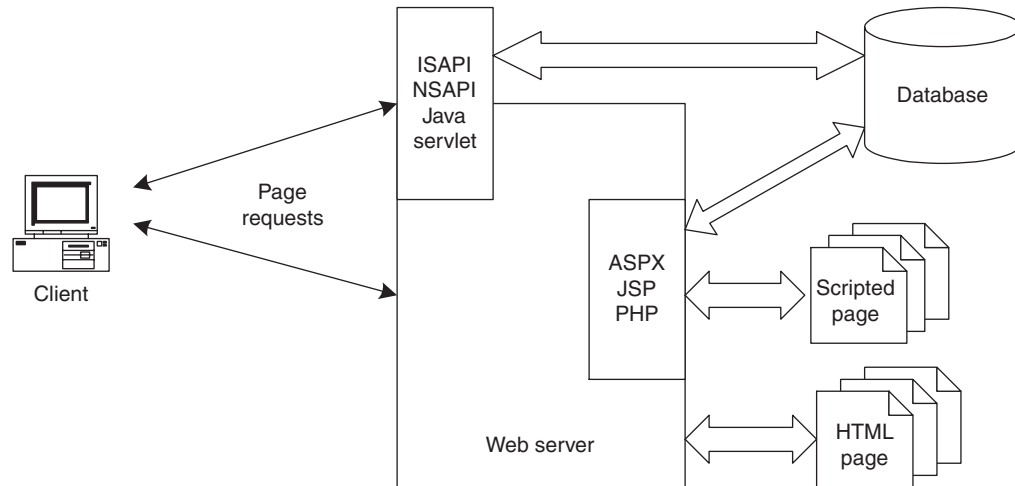


FIGURE 2-5 Web server-enabling technologies

The real appeal of scripted pages, however, is not their speed of execution but their ease of development and deployment. Typically, scripted pages don't contain most of the application's business logic, which instead is often found in compiled business objects that are accessed by the pages. Scripted pages are used mostly as the glue that connects the HTML user interface aspects of the system with the business logic components.

In any Web application, the choice of technologies depends on the nature of the application, the organization, and even the development team itself. On the server, a wealth of technologies and approaches may be used, many of them together. Regardless of the choices, they need to be expressed in the larger model of the system. The central theme in this book is that all the architecturally significant components of a Web application need to be present in the system's models. Servers, browsers, Web pages, and enabling technologies are architecturally significant elements and must be part of the model.



Summary

- ❑ Web systems are hypertext document systems that are relatively new to the computing world.
- ❑ The fundamental elements of a Web system are the client browser, the network, and a Web server.

- ❑ A Web application is a Web system that delivers business logic.
- ❑ HTTP is the primary protocol for communication between clients and servers.
- ❑ HTTP is a connectionless protocol. Each time a client requests a document from a Web server, a new connection must be established with the server.
- ❑ As identifiers of network resources, URLs or URIs can reference Web pages, images, applets, and other resources on the network.
- ❑ Web pages are documents that are requested by the client and delivered by the Web server.
- ❑ The principal formatting language for Web pages is HTML. It tells the browser how to render the content on the screen.
- ❑ Forms allow users to enter input through a Web page.
- ❑ Frames allow multiple Web pages to be active for a user at the same time.
- ❑ Session management is the tracking of a particular user's progress through the application, even across Web pages. Cookies can be used to help manage client state on the server.
- ❑ Enabling technologies are development environments that provide the infrastructure for building Web applications. Some popular technologies are CGI, Active Server Pages, JavaServer Pages, servlets, and Web server APIs.

Discussion

1. What are the advantages of a connectionless protocol, such as HTTP? What are the disadvantages?
2. Discuss the differences in the main client tier—what happens exclusively in the browser—and the presentation tier—the handling of URL resource requests—paradigms of operation?
3. What types of side effects might you expect to see in a Web application when the client browser's Back and Forward buttons are used? How might book-marked pages affect the application?

Activities

1. Examine the cookie file(s) on your machine. Look for and examine cookies left by advertising companies. Compare these cookies with other cookies from sites and applications that you know well.
2. Develop a long-term persistent session management strategy that will allow users to engage in a use case scenario over a period of days, with long breaks in continuity.