Table 2.4 Request methods in HTTP/1.0

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>It gets or retrieves a web page.</td>
</tr>
<tr>
<td>HEAD</td>
<td>It requests the header information of the web page. In other words, the response is the same as that for GET with the body or the content of the web page removed.</td>
</tr>
<tr>
<td>POST</td>
<td>It posts additional data to the web server in the HTTP request message. The additional data is attached after the headers.</td>
</tr>
</tbody>
</table>

Entity_header(s)
Blank_line
Entity(Additional_data)

As described in Table 2.4, Request_method specifies the request method used. Resource_address is essentially the URL that specifies the location of the requested resource in the web server. HTTP/Version_number tells the web server what HTTP protocol the web client is using. There are three types of headers for passing additional information to the web server, namely, General_header, Request_header, and Entity_header. They are described in Tables 2.5, 2.6, and 2.7, respectively. Finally, the web client can post additional data to the server after the Blank_line. This is used in conjunction with the POST request method. For details on the request methods and the request headers, please refer to RFC1945.

Let us look at the following example of an HTTP request message.

GET /vbs.html HTTP/1.0
Accept:image/gif,image/jpeg,*/*

This request message means that the client wants to get a document called "vbs.html" from the server. The document is located at the root directory of the server. Version 1.0 of the HTTP is used. The client can accept any content type as

Table 2.5 General headers in HTTP/1.0

<table>
<thead>
<tr>
<th>Header name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>It specifies when (i.e. date and time) the message was generated.</td>
</tr>
<tr>
<td>Pragma</td>
<td>This header is for specifying implementation-specific directives. For example, if the client does not want to receive a cached copy of the requested resource, it will specify Pragma: No-cache.</td>
</tr>
</tbody>
</table>
Table 2.6 Request headers in HTTP/1.0

<table>
<thead>
<tr>
<th>Header</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization</td>
<td>Used with the later WWW-Authenticate response header, it provides authentication information to the web server. HTTP provides a basic authentication scheme by encoding the username and password in Base64 format.</td>
</tr>
<tr>
<td>From</td>
<td>This header provides the contact e-mail address. (e.g., the e-mail address of the person who generates the request.)</td>
</tr>
<tr>
<td>If-Modified-Since</td>
<td>It asks the web server to provide the requested resource only if it has been modified since the specified time in the header.</td>
</tr>
<tr>
<td>Referer</td>
<td>It indicates where (i.e. URL) did the client obtain the current address. By using this header, a web server can trace back the previous link(s), e.g., for maintenance or administrative purposes.</td>
</tr>
<tr>
<td>User-Agent</td>
<td>It provides information on the user agent (e.g. web browser) used by the web client.</td>
</tr>
</tbody>
</table>

indicated by "*/" but for the image content, GIF is preferred to JPEG (see Chapter 3). Note that no additional data can be enclosed in the HTTP request.

2.6.2 HTTP response

Having processed the web client’s request, the web server returns a response to the client. The general format of the response is as follows (see RFC1945).

```
HTTP/Version_number status_code Result_message (Status line)
General_header(s)
Response_header(s)
```

Table 2.7 Entity header in HTTP/1.0

<table>
<thead>
<tr>
<th>Header</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow</td>
<td>It indicates the request methods (e.g. GET, POST, and HEAD) allowed.</td>
</tr>
<tr>
<td>Content-Encoding</td>
<td>It specifies the encoding method (e.g. compression method) applied to the content.</td>
</tr>
<tr>
<td>Content-Length</td>
<td>It indicates the length of the content in number of octets.</td>
</tr>
<tr>
<td>Content-Type</td>
<td>It indicates the content type or MIME type of the content, e.g., text/html means HTML document in text format.</td>
</tr>
<tr>
<td>Expires</td>
<td>It specifies when (i.e. date and time) the content becomes expired.</td>
</tr>
<tr>
<td>Last-Modified</td>
<td>It specifies when the content (web page) was last modified.</td>
</tr>
</tbody>
</table>
Table 2.8  Commonly used status codes in HTTP/1.0

<table>
<thead>
<tr>
<th>Status code</th>
<th>Result message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>OK</td>
<td>This refers to the normal case in which the request is OK or successful.</td>
</tr>
<tr>
<td>201</td>
<td>Created</td>
<td>The request is processed and the resource is created as requested.</td>
</tr>
<tr>
<td>204</td>
<td>No content</td>
<td>The request is processed but no content is available for the client.</td>
</tr>
<tr>
<td>301</td>
<td>Moved permanently</td>
<td>The resource has been moved permanently to the URL as given in the “Location” header.</td>
</tr>
<tr>
<td>302</td>
<td>Moved temporarily</td>
<td>The resource has been moved temporarily to the URL as given in the “Location” header. As it is only a temporary relocation, future requests should still be sent to the current URL.</td>
</tr>
<tr>
<td>304</td>
<td>Not modified</td>
<td>The requested web page is not returned to the client as it has not been modified since the time as specified in the “If-Modified-Since” header.</td>
</tr>
<tr>
<td>400</td>
<td>Bad request</td>
<td>The request is “bad” because it could not be interpreted by the server most possibly because of syntax errors in the request.</td>
</tr>
<tr>
<td>401</td>
<td>Unauthorized</td>
<td>Used in conjunction with the WWW-Authenticate header field, it indicates that user authentication is required.</td>
</tr>
<tr>
<td>403</td>
<td>Forbidden</td>
<td>Access is forbidden, e.g., the user does not have the access right.</td>
</tr>
<tr>
<td>404</td>
<td>Not found</td>
<td>The requested resource is not found, possibly because it has been deleted from the web server.</td>
</tr>
</tbody>
</table>

Entity_header(s)
Blank_line
Entity_body (e.g., web page)

Again, the HTTP/Version_number indicates the version of HTTP that the server is using. The Status_code indicates the result of the request. The common status codes are given in Table 2.8. The headers General_header(s), Response_header(s), and Entity_header(s) are used to pass additional information to the web client. General_header and Entity_header have been described in Tables 2.5 and 2.7, respectively. Response_header is described in Table 2.9. Following the headers, the response data is enclosed as the Entity_body. Usually this is a hypertext file.

Let us look at an example of a server response. Suppose that the response message was as follows.
Table 2.9  Commonly used response header in HTTP/1.0

<table>
<thead>
<tr>
<th>Response header</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>Used with the status code 301 and 302 etc, it provides the new URL for redirection purpose.</td>
</tr>
<tr>
<td>Server</td>
<td>It provides information about the server software.</td>
</tr>
<tr>
<td>WWW-Authenticate</td>
<td>Used with the unauthorized response message (i.e. status code of 401), it provides the authentication information required for successful authentication.</td>
</tr>
</tbody>
</table>

HTTP/1.0 200 OK
Server: Microsoft-IIS/4.0
Date: Sat, 30 Sep. 2000 09:30:00 GMT
Last-Modified: Sat, 30 Sep. 2000 09:00:00 GMT
Content-Type: text/html
Content-Length: 600

This response message means that the web server is using version 1.0 of HTTP. The request has been processed successfully. The server is Microsoft-IIS/4.0. The current date and time are 30 Sep. 2000 and 09:30:00, respectively. The response document is an HTML file in text format and the file size is 600 bytes. This file has not been modified since 09:00:00 on 30 Sep., 2000.

2.7 GENERATION OF DYNAMIC WEB PAGES

So far, we have talked about how to get a static web page from a web server by using the GET command. In many cases, the returned web page is user-dependent, i.e., it is dynamic rather than static. For example, a user may want to use the search engine of our VBS to search for books about e-commerce. In this case, the returned web page will be dependent on the user’s search criteria. Therefore, we need a method for the web client to pass additional data to the web server. One simple way to do this is to attach the data at the end of the URL by using the following format [Hall, 1998]:

?name_1=value_1&name_2=value_2&...&name_n=value_n

where name_1, name_2, ..., name_n specify the names of the input elements and value_1, value_2, ..., value_n specify the corresponding values.
For example, if a user wants to search for all books on ecommerce published in the year 2000, he can append the search criteria after the URL as shown in the following:


where name1 is “title” and the corresponding value is “ecommerce” and name2 is “year” and its value is “2000.”

By entering these, the web browser will issue the following GET command:

GET /servlet/booksearch?title=ecommerce&year=2000 HTTP/1.0

in the HTTP request message.

In this case, the request is not just for a static web page. Instead, we invoke a program called “booksearch” in order to generate the search result and then return it to the user. The program “booksearch” is a servlet program stored under the logical directory “servlet” of the host www.vbs.com. We will deal with servlets when discussing server-side programming later in the book. At the moment, our concern is how to pass the search criteria to the web server. In some cases, we may need to pass some special characters to the web server as well. The default encoding method is called “application/x-www-form-urlencoded.” In this encoding method, the following rules are used [Hall, 1998]:

- a space becomes a “+”
- a nonalphanumeric character becomes a hexadecimal code preceded by a %

For example, ?Input=%2F%7Ehenry%2Flecture2%2Dnotes.html is equivalent to attaching a name called “Input” with value

/~henry/lecture2-notes.html

to the URL because %2F is “/”, %7E is “~” and %2D is “-”.

An alternative way to pass data to the web server is by using the POST command. In this case, data is appended after the headers in the HTTP request message. For example, if we use the POST command to pass data to the above “booksearch” program, you will find the following in the HTTP request message:

POST /servlet/booksearch HTTP/1.0
Accept */*

title=ecommerce&year=2000
Note that data is appended after a blank line following the header. In this example, there is only one header called “Accept */*”. It specifies that the web client is willing to accept any content type.

2.8 COOKIES

HTTP is a stateless protocol. That means, the web server will not keep user’s state or user’s information. For example, when a web server receives an HTTP request, it does not know whether this request comes from a previous client or a new client. In other words, there is no way to tell whether or not the current request is related to a previous request. In many e-commerce applications, knowing the user’s state is an important requirement. For example, in a shopping cart application, the server needs to know the content of the user’s shopping cart in order to display the items to the user correctly. To address this important issue, Netscape proposed a method called “cookies” for a web server to save state data at the web client. The original specification is stored at http://www.netscape.com/newsref/std/cookie_spec.html, and it has now been standardized in RFC2109. A maximum of 20 cookies are allowed at each domain and each cookie is limited to 4 Kb to prevent overloading the memory of the client’s computer [RFC2109; Hall, 1998].

If a web server wants a web client to save “cookie,” it will send the Set-Cookie header in the HTTP response. The Set-Cookie header is of the form

Set-Cookie: Name=Value

where Name and Value are the name and value of the cookie, respectively. Whenever required, the client will include the cookie in the HTTP request header using the following format:

Cookie: Name=Value

This allows the user’s information to be passed to the server.

Let us look at how cookies can be used to implement a simple shopping cart for our VBS. Suppose that there are already two items in the shopping cart. The first item (Item1) has a product code of 11111 and the second item (Item2) has a product code of 22222. When the client sends a HTTP request to put another item (say an item with product code 33333) into the shopping cart, the server can set a cookie by including the following cookie header:

Set-Cookie: Item3=33333
It means that the third item has a product code of 33333. In the next HTTP request, the user needs to send to the server the following cookie headers:

Cookie: Item1=11111
Cookie: Item2=22222
Cookie: Item3=33333

By reading the cookies, the server knows the content of the shopping cart so that it can be displayed in the returned web page accordingly.

Besides the Set-Cookie header, the following are extra information that can be provided for the cookie(s) (see RFC2109 and Hall [1998]). They can be added on the Set-Cookie header as shown in the later example*.

- **Comment** – provides information on the cookie (e.g. its use)
- **Domain** – specifies in which domain the cookie is effective
- **Expires** – specifies when the cookie will expire
- **Max-age** – specifies the cookie's lifetime in seconds
- **Path** – specifies the URLs to which the web client should return the cookie(s)
- **Secure** – specifies that the cookie is returned only if the connection is secure (e.g., SSL is enabled, see Chapter 8)

Here is a simple example based on similar examples in [Hall, 1998]. Suppose that the VBS web server wants to create a cookie called Credit=111 in order to remember the user's credit. The Set-Cookie header is

```
Set-Cookie: Credit=111; secure; expires=Thursday,
07-Dec-2000 10:00:00 GMT; domain=.vbs.com; path=/
```

It means that the cookie Credit=111 will be returned only to a SSL-enabled server. The expiry date of the cookie is 07-Dec-2000, 10:00:00 GMT. The cookie is effective under the domain name vbs.com. Note that “path=/” means that the cookie applies to any directory under the root directory of the server.

### 2.9 HTTP/1.1

In HTTP/1.1, many enhancements are included to improve the performance of HTTP, to enhance its functionality, and to eliminate the limitations of HTTP/1.0. Generally speaking, HTTP/1.1 works in a similar manner to HTTP/1.0 except that many additional headers are added so HTTP/1.1 is upwardly compatible with HTTP/1.0. Some

---

*We consider the headers in both RFC2109 and the original specification.*
of the major enhancements are summarized as follows according to [Krishnamurthy, Mogul, and Kristol, 1999]:

- **Persistent connections and pipelining:** In HTTP/1.0, a connection is released after a request is served. Obviously this is inefficient because a web client may want to retrieve other web pages from the same web server. In HTTP/1.1, a connection is kept open such that the web client can send multiple requests over the same connection. For example, after accessing the home page of the VBS, the customer may want to read the company information by getting the corresponding web page from the web server. Instead of opening a new connection for this request, it can be sent along the same connection. Furthermore, a web client can send the next request without waiting for the response to the previous request. In other words, HTTP/1.1 allows pipelining of requests and responses. If a web client wants to close a connection, it can specify a “close” option in the Connection request header, i.e., Connection: close.

- **Efficient use of IP addresses:** Currently many small organizations use a web hosting service from ISPs. For example, we may put the VBS in an ISP’s web server such that we do not need to set up and look after a web server ourselves. In HTTP/1.1, a Host header must be included in the HTTP request message to specify the host name in the web server. This enables different organizations to share the same IP address of the web server thus allowing the efficient use of IP addresses.

- **Range request:** HTTP/1.1 allows a web client to retrieve part of the file by using the Range header. For example, if the connection is broken while the web client is receiving a large file, it can request the web server to send the file from the “break point.” Furthermore, the range request function is useful when the web client wants only a portion of a large file.

- **Cache control:** The purpose of caching is to shorten the retrieval time of web pages. It is done by maintaining a cache copy of the previous responses in the web browser or the proxy server so that future requests can be served by the cache copies rather than by the original servers. HTTP/1.0 only supports basic cache control. For example, by using the Expires header, the original server can tell the proxy server when a cache copy should be removed. Furthermore, the web client can tell the proxy server that it does not want a cache copy of the response by using the “Pragma: No-cache” header. In HTTP/1.1, a “Cache-Control” header is included to provide better cache control and cache functions.

- **Support for proxy authentication:** HTTP/1.1 provides the Proxy-Authentication and Proxy-Authorization headers for enabling proxy authentication. In principle, they work in a similar manner to the WWW-Authentication and
Authorization headers in HTTP/1.0, respectively. However, the Proxy-Authentication and Proxy-Authorization headers are used on a hop-by-hop basis.

- **Better support for data compression**: HTTP/1.1 provides better support for data compression. In particular, a web client can specify the encoding method such as the compression scheme(s) that is/are supported and preferred by using the Accept-Encoding header.

- **Better support for language(s)**: In HTTP/1.1, a web client can specify the language(s) that is/are acceptable and preferred.

- **Support for content integrity**: In HTTP/1.1, content integrity can be supported by the Content-MD5 header.

- **Additional request methods**: Four additional request methods are added as described in Table 2.10. However, they are less commonly-used than the GET, POST, and HEAD request methods.

## 2.10 EXAMPLE

Let us summarize these web technologies in the context of building the VBS. To build the VBS, we need to set up a web server. This web server interfaces with the web client as well as the backend system. Customers access the VBS through their web browsers (web clients). Basically, when the URL of the VBS is accessed, the web client makes a TCP connection over the internet to port 80 of the VBS web server. The web client then uses the HTTP GET command to retrieve the homepage of the VBS from the web server. After receiving the request, the VBS web server will return the homepage in the HTTP response message. The homepage will then be displayed to the customer. In the next chapter, we discuss how to build the homepage by using the hypertext markup language. In many cases, the customer may want to search for some books. In this case, he generates another HTTP request to the VBS web server.
This time, additional data about the search criteria will be passed to the web server. This can be done by using the HTTP GET or POST command. In the former case, the additional data is appended at the end of the URL. In the latter case, the additional data is embedded in the HTTP request message. Having received the search request, the web server will process it accordingly with the help of other servers and the backend system. We will discuss this in the chapter on server-side programming. The book search result will be returned to the customer. Unlike the previous homepage, this web page is generated dynamically, based on the customer's input. In order to keep the customer's information, the Cookie method can be used. In particular, this can be used to implement the shopping cart application.

2.11 SUMMARY

In summary, the internet is a collection of networks interconnected by routers. It is based on a layered model consisting of four layers, namely, link layer, network layer, transport layer, and application layer. In a web-based e-commerce system, there are four main components. They are the web client, the web server, the application server, and the backend system. The web client and the web server communicate with each other based on a request/response protocol called HTTP. Basically, a web client passes an HTTP request to the web server with some headers and optional data. The web server then returns a response to the web client with some headers and the requested resource such as the requested web page. The web client can also pass additional data to the web server by appending it after the URL or embedding it inside the HTTP request message. This can be used to generate dynamic web pages. As the HTTP is stateless, a "Cookie" method can be used to keep track of a user's state. This is important for many e-commerce applications such as building a shopping cart.

REFERENCES

RECOMMENDED READING

There are many good books on computer networks in general and the internet in particular.


This book gives a good overview on internet standards.


These are some popular textbooks on computer networks.


These are classic references to explore further about TCP/IP protocols.


This provides a good comparison between HTTP/1.0 and HTTP/1.1.


This book gives a good overview of HTTP and cookies. It also provides some Java programs to build a simple web server.


This is a very good paper on IP addressing.

The RFCs for internet standards are maintained in the IETF homepage at www.ietf.org.

In particular, the standards for HTTP/1.0 and HTTP/1.1 are given in RFC1945 and RFC2616, respectively. There is also a good TCP/IP tutorial in RFC1180. Web-related technologies and standards can be found at the W3 homepage at www.w3.org.
This Page Intentionally Left Blank
In Chapter 2, it was explained that the web browser downloads the required information and provides the basic display on the client site. To do this, it downloads web pages including Hypertext Markup Language (HTML) codes and other web page elements (e.g. Java Applets) from the respective server(s). For example, a customer may access the URL of our VBS through his web browser. After receiving the request, the web server of our VBS will return the web page to the customer. The web page
contains information on the VBS as well as other forms of input elements. Essentially, it provides the user interface. Hence, we need to create the web site so that when the browser accesses it, the required user interface can be displayed to the client. This process of setting up the web pages or programs at the web sites for clients to see the required display is referred to in this book as client-side programming. In general, this is different from setting up the user interface in conventional client–server architecture. In that case, the user interface programming code actually resides at the client side (i.e., it is not downloaded and displayed within a browser). This is the reason that client-side programming in e-commerce is sometimes referred to as web Programming. This is a very important distinction in client-side programming, as carried out in e-commerce systems.

Besides static web pages, one could also create interactive or dynamic web pages by downloading program codes written in an appropriate language such as JavaScript or Java. In addition, multimedia elements such as image, video, animation, and sound could be included on the web pages.

In terms of e-commerce applications, client-side programming is mainly used for processing a sale transaction, providing information on your business, and updating information in the backend system (e.g. a database). This involves disseminating information or carrying out interactions with the server side over the internet.

3.1 Important Factors in Client-Side or Web Programming

To carry out client-side programming in e-commerce applications, there are several different ways, which include using HTML, JavaScript, Java Applets, and ActiveX controls. Furthermore, one could also use plugins, which are applications of different sorts that are embedded in a web page for performing special functions (e.g. showing animations). In this chapter, we will focus on HTML because it forms the basis of nearly all the client-side programming techniques. Furthermore, we will give an overview of JavaScript towards the end of this chapter. It enhances the functions of HTML and makes a web page more interactive and dynamic. While there are many client-side programming techniques available, HTML and JavaScript are currently the most commonly used programming techniques for building the user interface at the client side.

A very important factor in client-side programming for e-commerce applications is downloading time. This is the time required to download a web page and its associated elements from the server side to the client side over the internet. This depends on many factors, including the quality of the network and the type of connection to the network. For B2C applications, many clients are likely to access the internet using dial-up modems, working at 56.6 Kbps or below. In B2B applications,
most of the clients could access the internet through Ethernet connections or leased lines with a much higher data rate.

In general, the downloading time should be kept within 15 s, otherwise it may become unacceptable from the client's point of view. This limits or greatly influences the choice of web programming techniques. Generally speaking, the downloading time for Java Applets is much higher than that of HTML or JavaScript. For this reason, while early e-commerce applications may have sometimes utilized Java Applets, they are now only very sparingly utilized, if at all, to address special requirements. In fact, JavaScript can perform many interactions at the client side already.

Data validation is another important factor to be considered when developing a user interface. It can involve several aspects and includes

1. type checking (e.g., integer)
2. range checking (e.g., between two numbers, say $n_1$ and $n_2$)
3. sequence checking (e.g., one cannot initiate an event in the past retrospectively)
4. business requirements checking

Generally speaking, points 1–3 can be validated at the client side by embedding programming code such as JavaScript within the HTML document, as discussed later in this chapter. For point 4, it is often performed at the server side because it frequently requires additional information from the backend system (e.g. database).

It is important to note that in an e-commerce system, a client is likely to be at a physically different location from the e-commerce application servers and hence technical support. In fact, he could be on the other side of the world. Thus, the "usability" of a client interface has to be kept at a very high standard.

The *usability* of a computer software “is measured by how easily and how effectively it can be used by a specific set of users, given particular kinds of support, to carry out a defined set of tasks, in a defined set of environments” [Shackel, 1981].

There is a large body of literature on the factors that make up a usable user interface. Molich and Nielson [1990] and Bevan and Macleod [1994] have all identified factors based on different empirical studies. A study of the literature indicates that the following list of factors would give a comprehensive coverage of the notion of usability. These factors include

1. system feedback
2. consistency
3. error prevention
4. performance/efficiency
5. user like/dislike
6. error recovery
**System feedback**: The purpose of system feedback is to inform the users what is going on in the system at any time. A well-designed system should always provide users with appropriate feedback, including immediate system feedback, acknowledgements, follow-ups, and indications that an action request has been carried out.

System feedback is characterized by several aspects in particular to address the following issues:

- Where does the error occur (i.e. error localization)?
- If an action is not allowed, does the system give the reason?
- Does the system give prompts on how to proceed?
- Does the system let one know where one is?
- Does the system explain why an action cannot be performed?
- Does the system acknowledge that an action requested has been carried out?

Inadequate system feedback has several components and these include

- number of times dialogue/feedback is missing
- number of times dialogue/feedback is unnecessary
- number of times system feedback confuses the user
- number of messages that are irrelevant
- number of actions taken which lead to repeated feedback message
- number of times the user makes the wrong choice of action due to improper system feedback

**Consistency**: The interface should be consistent in terms of the look, feel, and behavior throughout the application and with other applications in the same domain. Most guidelines seek to fulfill this important goal. This consistency should be maintained across a variety of issues such as message display methods, color use, key definition, data entry methods, etc. If the user interface is consistent, it reduces the amount of uncertainty that the user faces when using the interface. It is also likely to reduce the number of erroneous interpretations or actions that the user makes. Consistency of the interface has a number of components and these include consistency with respect to the following:

1. Message display methods (prompts, warnings, helps)
2. Color use (entry form, menu and submenu, foreground/background)
3. Keys definition
4. Data entry method
5. Menu, dialogue, and window display methods
6. Menu hierarchy that is consistent with the real world
7. Terminology used is the same as in real life in that domain
8. Menu options have to be consistent with Menu Title

The issue of consistency with other applications in the same domain is very important for e-commerce applications, particularly B2C applications as the user is unlikely to remember what different things meant between one visit and another.

Error prevention: Error prevention is an important goal of the design of the client user interface. If the user interface specifically helps the user to avoid making errors, it increases his efficiency. It will also reduce the level of frustration the user is likely to experience with the user interface and therefore bring about greater acceptance of the user interface by the user. There are several aspects that need to be taken into account in error prevention and these include

- number of errors encountered during task
- number of wrong key strokes/press causing error messages
- number of times the same key is pressed without the desired response
- number of extra key presses that are unnecessary
- number of times the same error encountered
- number of steps missing compared with real-world execution

Performance/efficiency: Performance or efficiency is a quality of the user interface that determines how effectively or efficiently the user can complete his tasks. Performance and efficiency have a number of components and these are as follows:

1. number of goals/tasks not achieved
2. time taken for task completion
3. unproductive period
4. percentage of tasks not completed

Like/dislike: Unlike the aforementioned factors, which characterize the manner in which the user interface facilitates user effectiveness or efficiency, the like/dislike factor measures user preference. This essentially indicates the level of satisfaction that the user feels with the system and the user interface.

Error recovery: Error recovery is that quality of the system of the user interface which allows the user to exit from a situation that the user did not intend to be in.
Users frequently choose the wrong option or enter the wrong data and they are likely to find themselves in an error state from which they need to recover. The manner in which the system facilitates this recovery from error could reduce the time the user spends recovering from this error state. Recovery from error consists of a number of components and these include

- number of times the user has to redo the task
- number of times the user did not continue
- number of actions taken that do not solve the problem
- number of minutes (hours) spent on one error recovery
- percentage of all time spent on error recovery
- number of times the user has to re-robot/start again

In addition to these factors, one also needs to add the following four in the case of client-side programming on the internet, namely

- browser compatibility
- attractiveness
- suitable navigational structure
- a site search engine

Browser compatibility relates to the fact that the appearance of a web page may change depending on the browser the client is using. The reason for this is that the HTML code and tags merely provide information on the type of information to the browser, e.g., a heading, paragraph, etc., in order to allow the browser to lay out the information. However, the manner in which the browser actually lays out the information is dependent on the browser itself.

The attractiveness of the layout of web pages is an important issue. This is true for the whole web site and even more so for the homepage. This arises from the fact that the web display seen by a customer in an e-commerce site is not only a software user interface in the traditional sense but also a marketing tool for the company to the client. Thus, the homepage and entire web site should be designed to project a specific image that is compatible with one that the company is trying to project. Several of the issues related to this will be discussed in Chapter 14 on Web Publishing and Advertising.

The creation of a navigational structure through the web site, which suits the particular audiences who visit that site, is of considerable importance and is discussed in detail in a later section.

It is important that there be, if possible, a site search engine that allows one to find information required, e.g., particular products using keyword search.
Last but not least, it is important that the design and architecture of the web site be such that it allows for evolution and change. As further changes and developments always take place, the initial design should allow for easy integration.

3.2 WEB PAGE DESIGN AND PRODUCTION

Before one begins programming a web site or the client-side of an e-commerce application, it is important to develop a careful design. These are briefly discussed here and in much greater detail in Chapter 14. In general, the steps in web site design and production can be described as follows.

3.2.1 Define the audiences and the information requirements

An effective web site should adopt an audience-centered design. Each web site may have many different types of audience, each having a different requirement. For example, a repeat customer may want to reorder something from the web site, but a new customer may want to learn more about the company. Having decided different types of audience, we should then formulate the information requirements for each type of audience accordingly.

3.2.2 Develop the logical design of the web site

Technically, the logical design of the client-side user interface involves the identification of abstract user interface objects and their interaction with other abstract user interface objects as well as other objects in the system such as domain objects or data management objects. The abstract user interface objects specify what the user interface does, not how the user interacts with the system to perform his/her tasks.

The logical design involves, among other things, characterization of the flow of interaction. The flow of interaction requires an understanding of the possible sequences involved in the interaction between the application and the software. This is frequently embedded in the user's perception of how the tasks should be carried out. It could be considered as a delegation/monitoring/control paradigm of the task to the machine. The dynamics of the interaction are determined by this aspect of the user's model of the task.

Among other things, the logical design should give one

- the site architecture
- the navigational structure
- the data input, data displayed, and actions that can be taken from each page
An example of the site architecture for the VBS is given in Figure 3.1. To do the design, one needs a mechanism for representing the abstract user interface objects and the flow of interaction using FINs. These are discussed in greater detail in Chapter 14.

3.2.3 Develop the perceptual design

The perceptual level of the design involves the choice of user interface widgets and the appearance of each particular page and its constituent parts. The user interface widgets employed in a particular user interface are sometimes referred to as concrete user interface objects. They define the actual items the user sees and interacts with on the screen such as buttons, pop-up menus, dialogue boxes, check boxes, entry forms, list boxes, etc. Therefore, they define how the user interacts with the system. In addition to actually choosing these concrete user interface objects, the actual arrangement of these on a two-dimensional surface must be specified during perceptual design. This is known as the layout of the page.

The outputs of the perceptual design should be the layout, the choice of graphics, images, etc. for the display of information, and the choice of colors, sizes, posters, etc.
3.2.4 Content creation

Having developed the logical design and the perceptual design of the web site, we need to create the content in accordance with the information requirements as defined in the first step.

3.2.5 Programming

This involves putting the content and other elements of the web pages together. As mentioned, we will discuss HTML and JavaScript later in this chapter. During the programming stage, it is important to evaluate the usability of the web pages and make enhancements accordingly. Usability evaluation consists of getting a set of prospective users to carry out a designated set of tasks on the client-side interface and the monitoring and evaluation of them to determine

1. the user acceptance of the interface
2. user efficiency in carrying out the task

It also helps identify specific deficiencies in the design of the pages, which can then be rectified.

It is useful to prototype an initial set of client-side screens using a rapid development tool such as Cold Fusion, Dreamweaver, or Front page and get user criticism and evaluation. This permits the screens to be changed without a great deal of effort. Once the final layout of the screens and the underlying web pages have been determined, one can proceed to code them using HTML or JavaScript or some combination of these or build them using the software tools such as those referred to earlier.

The overall structure of the client-side design is shown in Figure 3.2.

3.2.6 Posting and hosting the site

Finally, the web sites will be set up accordingly. Basically, we can either host the web site in a private web server or use a third-party’s web hosting service.

3.3 Overview of HTML

The rest of this chapter is about building the user interface using HTML. We only give an overview of HTML. This is similar to several other books including Ray and Ray [1997] Lemay [1999] Zakour et al. [1997] and Deitel et al. [2000]. For readers
who are interested in a more detailed discussion of HTML, the books by Ray and Ray [1997] and Lemay [1999] are highly recommended. Furthermore, we will also give an overview of JavaScript. HTML is the commonly used markup language for web publishing, and JavaScript is often used to enhance the functionality of HTML. At the time of writing, HTML 4.0, is the most updated version. We will give an overview of HTML 4.0, in particular the most commonly used elements for building an e-commerce system.

HTML is a markup language for telling a Web browser how to format and display a Web page. It can be viewed as a subset of the Standard Generalized Markup Language (SGML), which is for defining general document format [Tanenbaum, 1996]. Predefined tags are employed to describe the format of a document. For example, by putting the word “Italics” inside the `<I></I>` tag pair (i.e., `<I>Italics</I>`), the word “Italics” will be displayed by the Web browser in Italics form. Most tags have a “start tag” and an “end tag” (also called container tags) and the content is embedded between the two tags. Some tags are standalone only without any content [Naik, 1998]. An example is the `<HR>` tag, which adds a horizontal rule. For most tags, one can also specify its attributes so as to define additional properties about the tag [Ray and Ray, 1997]. For example, one can change the font face by applying the FACE attribute of the `<FONT>` tag as shown below:

```html
<FONT FACE="Arial"> The font face is Arial.</FONT>
```

### 3.4 Basic Structure of an HTML Document

Basically, the structure of an HTML document is defined as follows [Naik, 1998]
The `<DOCTYPE>` tag specifies the version of the HTML document and other related information. `<HTML>` and `</HTML>` tags define the start and the end of an HTML document, respectively. Within the `<HTML>` tag pair, there are two main sections namely the HEAD section as included inside the `<HEAD></HEAD>` tag pair and the BODY section as included inside the `<BODY></BODY>` tag pair. The HEAD section provides information (e.g. the document title) for the web browser to process the document but the information is not displayed. The web browser only displays the information within the `<BODY></BODY>` tag pair. In an HTML document, users can also insert comments inside the comment tag pair: `<!--` and `-->`. The comments are not displayed by the web browser. In the above example, comments are used to tell you what should be put inside the respective tags.

The common attributes of the `<BODY>` tag are given as follows:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BACKGROUND</td>
<td>Specifies the URL of the background image</td>
</tr>
<tr>
<td>BGCOLOR</td>
<td>Specifies the background color to be used</td>
</tr>
<tr>
<td>TEXT</td>
<td>Specifies the default color of the text in the Web page</td>
</tr>
<tr>
<td>LINK</td>
<td>Specifies the color of the hyperlinks that have not been visited</td>
</tr>
<tr>
<td>ALINK</td>
<td>Specifies the color of the active hyperlinks</td>
</tr>
<tr>
<td>VLINK</td>
<td>Specifies the color of the visited hyperlinks</td>
</tr>
</tbody>
</table>

In conventional HTML, colors are represented by a hexadecimal code. For example, black and white colors are represented by the codes "#000000" and "#FFFFFF," respectively.
3.5 BASIC TEXT FORMATTING

Let us first look at a simple example that shows most of the basic formatting features. The corresponding web page is shown in Figure 3.3.

```html
<HTML>
<HEAD>
<TITLE>A simple web page</TITLE>
</HEAD>

<BODY>
<H1 ALIGN="center">First level heading</H1>
<H2>Second level heading</H2>
<P><FONT FACE="Courier" SIZE="4">The font face is Courier and
the font size is 15pt.</FONT></P>
<P><FONT COLOR="#FF0000">The font color is red.</FONT></P>
<P><EM>The text is in italic form.</EM></P>
<P><U>The text is underlined.</U></P>
<P><STRONG>The text is expressed in bold face.</STRONG></P>
<P>This is<SUP>superscript</SUP> and this
is<SUB>subscript</SUB>.</P>
<P><BLINK>This text is blinking.</BLINK></P>

<P>This is a simple bullet list:

<UL>
  <LI>First item</LI>
  <LI>Second item</LI>
</UL>

<P>This is a simple numbered list:

<OL>
  <LI>First item</LI>
  <LI>Second item</LI>
</OL>
</BODY>
</HTML>
```
3.5.1 Heading

The heading tags `<H1>` and `<H2>` are used to create a first and second level heading, respectively. Altogether there are six levels of headings: `<H1>`, `<H2>`, ..., `<H6>`. The `ALIGN` attribute specifies the alignment method. Possible options are `LEFT`, `CENTER`, and `RIGHT`. For instance, in the earlier example, the first level heading is aligned to the center whereas the second level heading is aligned to the left because no attribute is specified, so the default alignment method is used.

3.5.2 Paragraph

You can define the start and the end of a paragraph by using the `<P>` and `</P>` tags, respectively. Similar to the heading tag, the `ALIGN` attribute can be used to specify the alignment method.
3.5.3 Font

As shown in the example, the `<FONT>` tag is used to define the font properties. Its common attributes are as follows:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR</td>
<td>Defines the font color (i.e. color of the text)</td>
</tr>
<tr>
<td>FACE</td>
<td>Defines the font style listed in order of preference. For example, “Arial, Courier” means that Arial is preferred to Courier (i.e., Courier is used if Arial is not supported.)</td>
</tr>
<tr>
<td>SIZE</td>
<td>Defines the default font size (an absolute scale of 1–7 can be used)</td>
</tr>
</tbody>
</table>

3.5.4 Other special tags for formatting text

We can also set the text in italics by using either the `<EM>` or `<I>` tag. For underlining the text, the `<U>` tag can be used. The `<STRONG>` tag can be used to display the text using bold face. Alternatively, the `<B>` tag can be used for the same purpose. The `<SUB>` and `<SUP>` tags are for displaying text as subscript and superscript, respectively. To create blinking text, we can put the text between the `<BLINK>` and `</BLINK>` tags.

3.5.5 Horizontal rule

To insert a horizontal rule, the `<HR>` tag is used. Its common attributes are as follows:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALIGN</td>
<td>Defines the alignment method: LEFT, CENTER, or RIGHT</td>
</tr>
<tr>
<td>NOSHADE</td>
<td>Display the rule using solid color (black)</td>
</tr>
<tr>
<td>SIZE</td>
<td>Sets the thickness in pixels</td>
</tr>
<tr>
<td>WIDTH</td>
<td>Sets the width</td>
</tr>
</tbody>
</table>

Note that the width of an horizontal rule can be set in terms of pixels or a percentage of the web page width.

3.5.6 Lists

To format text using a list for ease of reading, the `<OL>` or `<UL>` tag can be used. `<OL>` and `<UL>` are for creating an ordered list and unordered list, respectively. After creating the list tag, the list items are specified by the `<LI>` tag.
The **TYPE** attribute of the `<OL>` tag specifies the numbering style of the list. The available ones are [Ray and Ray, 1997; Lemay, 1999]

- **TYPE=1** – Arabic numerals starting from 1
- **TYPE=a** – alphabetical order starting from a (i.e. small letters are used)
- **TYPE=A** – alphabetical order starting from A (i.e. capital letters are used)
- **TYPE=i** – lowercase Roman numerals starting from i
- **TYPE=I** – uppercase Roman numerals starting from I

To specify the starting value of the ordered list, we can use the **START** attribute of the `<OL>` tag. For example, `<OL TYPE=1 START=6>` means that the first item of the list starts at 6.

For the `<UL>` tag, the **TYPE** attribute specifies the bullet shape. The available ones are

- **TYPE=CIRCLE** – the bullet type is a circle
- **TYPE=DISC** – the bullet type is a disc
- **TYPE=SQUARE** – the bullet type is a square

The `<LI>` tag has the same attributes as the corresponding `<OL>` or `<UL>` tag. Therefore, we can override the original attribute by applying the new attribute in the respective `<LI>` tag. Furthermore, for the `<LI>` tag of the ordered list, we can assign a new value with the **VALUE** attribute. For example, `<LI TYPE=1 VALUE=11>` specifies that the respective item is assigned as the 11th item irrespective of the previous order [Ray and Ray, 1997].

### 3.6 **LINKS**

Links (or hyperlinks) are the most powerful feature of an HTML document. They are used to link web pages. For example our VBS web page can be linked to a publisher’s web page. The two web pages can be situated in two different web servers that may be distant from each other. The computer screen is two-dimensional but the internet space is infinite because the hyperlinks give HTML files an infinite depth. The following gives an overview of hyperlinks. More details can be found in the book by Ray and Ray [1997].

Links are defined by the anchor tag pair: `<A>` and `</A>`. For example, a basic hyperlink looks like the following:

```
<A HREF=http://www.vbs.com/Books/Art.html>Art</A>
```
In this example, the word "Art" is underlined and colored. When it is clicked, the browser links to the URL http://www.vbs.com/Books/Art.html as specified by the HREF attribute. Recall that in the URL, "http" stands for the hypertext transfer protocol; "www.vbs.com" is the server name; "Books" is the directory; and, "Art.html" is the HTML file.

We can also set up an email link with the <A> tag. It is widely supported by most browsers such as Netscape and Internet Explorer. For example, we can set up the following email link for customers to send emails to the Webmaster of our VBS:

```html
<A HREF=mailto:webmaster@vbs.com>Please send any question to the Webmaster</A>
```

A URL can be relative or absolute. For a relative URL, the location of the requested web page is relative to the current web page. For example, if the current web page "Art.html" contains a hyperlink say

```html
<A HREF=Science.html>Science</A>,
```

the file "Science.html" will be accessed under http://www.vbs.com/Books, i.e., at the same directory as the current document "Art.html." For an absolute URL, the full address is specified just like the full address specified in the above example.

If a file such as "Welcome.html" is at the server root, we can also specify the link as <A HREF=/Welcome.html></A> where "/" denotes the server root. To access documents at another directory, we can use "/" to go back one directory, "../.." to go back two directories, and so on. For example, the following hyperlink <A HREF=../Welcome.html></A> means that to get the web page “Welcome.html,” we need to go back one directory relative to the current web page. It is worth mentioning that it is possible to link to a file in a local disk as well. For example, if we want to link to a file called "thefile" stored under "c:\directory\subdirectory" in a local disk, the URL is

```html
file:///c\directory/sub-directory/thefile
```

Note that an additional backslash ("\") is used to replace the normal host name.

With HTML 4.0, it is possible to link to different locations within the same web page by using the anchor tag. It can best be explained with an example. Suppose that in "Science.html," we want to identify a section called "Computing" such that we can link to this section from other parts of the web page. We can define the name anchor by specifying the name attribute in the <A> tag as follows:
Note that unlike the hyperlinks, the words "Books on computing" will still appear as normal text (i.e., it will not be underlined and colored). Within the same web page (i.e. Science.html), we can link to the "Computing Section" by using the following tag:

```
<A HREF="#Computing">
```

In general, we can link to the "Computing Section" by specifying the following URL within the `<A>` tag as follows:

```
<A HREF="http://www.vbs.com/books/science.html#Computing">
```

Computing Section</A>

Finally, let us discuss a major advantage of using relative URLs concerning movement and change. Sometimes we need to move documents between servers. Suppose that we need to migrate all the files from www.vbs.com to another server say www.xyz.com. If all the links are specified using relative URLs and the directory structure remains unchanged, basically we do not need to update the hyperlinks inside the files. However, if the links are specified using their absolute addresses, they have to be updated with the new server name. Of course, the absolute URL approach must be used for accessing documents in other external servers.

### 3.7 Images

Currently, there are two major image formats for the web, namely, Graphics Interchange Format (GIF) and Joint Photographic Experts Group (JPEG) [Ray and Ray, 1997; Lemay, 1999]. They are used to store images in compressed form so that the file size can be reduced. In general, GIF is more commonly used for a number of reasons. First, the compression method (called LZW) employed by GIF is lossless. This means that images can be compressed while preserving the data (i.e. no loss). Second, by using special image-editing tools (e.g. Paint Shop Pro), animated GIF files can be created for presenting simple animation effects on a web page. Third, a GIF image can be made to look transparent in a web page by specifying its background color to match with that of the web page. However, GIF works well only for simple images such as logos but not for complex images such as photographs because it only provides 256 colors. In contrast, JPEG employs a lossy compression method. While the file size can be reduced further, some data are lost during compression. In general, JPEG is well-suited for photographs because it provides significantly more colors than GIF. GIF is used for general purpose images.
The `<IMG>` tag is used for including images in a web page. Its common attributes are as follows [Zakour et al., 1997]:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALIGN</td>
<td>Defines the alignment method (i.e., LEFT, RIGHT, TOP, MIDDLE, or BOTTOM)</td>
</tr>
<tr>
<td>ALT</td>
<td>Gives a description of the image, which is displayed when the image is not available (e.g., during loading)</td>
</tr>
<tr>
<td>BORDER</td>
<td>Sets the border size in pixels</td>
</tr>
<tr>
<td>HEIGHT</td>
<td>Fixes the height of the image in pixels</td>
</tr>
<tr>
<td>HSPACE</td>
<td>Specifies the additional horizontal space to be added</td>
</tr>
<tr>
<td>SRC</td>
<td>Specifies where (i.e., the URL) to load the source image</td>
</tr>
<tr>
<td>VSPACE</td>
<td>Specifies the additional vertical space to be added</td>
</tr>
<tr>
<td>WIDTH</td>
<td>Fixes the width of the image in pixels</td>
</tr>
</tbody>
</table>

Here is an example of inserting the image of our VBS Logo (`vbs.gif`):

```html
<IMG SRC="vbs.gif" HEIGHT="100" WIDTH="100" ALIGN="LEFT" BORDER="1" ALT="VBS">
```

It means that the height and width of the image is 100 pixels each, the alignment is to the left, and the border width is 1 pixel. When the image is not available (e.g., when the browser is getting the image), the text "VBS" is displayed.

One can also create an image link by embedding the `<IMG>` tag inside the `<A>` tag pair. For example, for creating an image link using the VBS logo, one can write it as

```html
<A HREF="main.html"><IMG SRC="vbs.gif"></A>
```

### 3.8 IMAGEMAP

Apart from using hyperlinks, we can also create an ImageMap for linking to different URLs. There are two types of image maps, namely, server-side and client-side image maps. As the latter is more commonly used, we will focus on client-side image maps in this book.

A client-side image map (referred to as image map hereafter) is defined by using the `<MAP>` tag pair as follows [Lemay, 1999]:

```html
<MAP NAME="map_ID">
  <AREA SHAPE="CIRCLE/RECT/POLY" COORDS="..." HREF="URL">
```
Between the <MAP> and </MAP> tags, the <AREA> tag is used to define the areas for the image map. As specified by the SHAPE attribute, each area can be a circle (CIRCLE), a rectangle (RECT), or a polygon (POLY). Depending on the shape of the area, the boundary is specified by the respective coordinates as follows:

- **Circle**: SHAPE="CIRCLE" COORDS="x, y, r" where (x, y) defines the center and r is the radius of the circle.
- **Rectangle**: SHAPE="RECT" COORDS="XL, yL, XR, yR" where (XL, yL) and (XR, yR) define the top-left corner and bottom-right corner of the rectangle, respectively.
- **Polygon**: SHAPE="POLY" COORDS="x1, y1, x2, y2, ..., xi, yi" where (x1, y1), (x2, y2), ..., (xi, yi) specifies the i-th corners of the polygon.

Note that (0, 0) is the co-ordinate of the top-left corner of the web page. The destined URL is defined by the HREF attribute.

Having defined the image map, the respective image can be put into the web page using the <IMG> tag as follows:

```html
<IMG SRC="image_file" USEMAP="#map_ID">
```

where "image_file" specifies the source of the image file and "#map_ID" gives the corresponding image map (i.e., the one defined earlier). Note that the "#" symbol indicates that the image map is situated in the current web document rather than the web server.

### 3.9 TABLES

Tables are typically used to organize information in a structural manner for ease of reading. In addition, they are commonly used for facilitating the layout of web page components. For instance, we can format a web page in a two-column format using a borderless table and then place the navigation buttons and the content into the left and right column, respectively. Note, however, that unlike using frames, the navigation buttons and the content are contained in the same web page. Basically,
a table is created by using the following tags:

```html
<TABLE>
  <TR>
    <TH>Heading cell (first row, first column)</TH>
    <TH>Heading cell (first row, second column)</TH>
  </TR>
  <TR>
    <TD>Data cell (second row, first column)</TD>
    <TD>Data cell (second row, second column)</TD>
  </TR>
</TABLE>
```

The `<TABLE>` tag pair specifies the beginning and the end of a table, respectively. The `<TR></TR>` tag pair defines a table row and the `<TD></TD>` tag pair defines a data cell. Therefore, this table has two columns and two rows because there are two `<TR></TR>` tag pairs and each row contains two data cells. In the first row, the `<TH></TH>` tag pair specifies the heading cell.

The common attributes of the `<TABLE>` tag are given as follows. They are used for formatting a table [Zakour et al., 1997].

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALIGN</td>
<td>Specifies the alignment method (LEFT, CENTER, or RIGHT), e.g., ALIGN = CENTER means that the table is aligned to the center of the document</td>
</tr>
<tr>
<td>BGCOLOR</td>
<td>Specifies the background color for the table</td>
</tr>
<tr>
<td>BORDER</td>
<td>Sets the border thickness in pixels (Note: BORDER = 0 means that there is no border)</td>
</tr>
<tr>
<td>CELLPADDING</td>
<td>Sets the padding (in pixels) between the cell border and content, e.g., CELLPADDING = 2 means that a cell padding of 2 pixels is used</td>
</tr>
<tr>
<td>CELLSMAPPING</td>
<td>Sets the spacing (in pixels) between data cells, e.g., CELLSMAPPING = 3 means that a cell spacing of 3 pixels is used</td>
</tr>
<tr>
<td>COLS</td>
<td>Specifies how many columns the table has</td>
</tr>
<tr>
<td>WIDTH</td>
<td>Defines the table width (in terms of a percentage of the document width or pixels), e.g., WIDTH = 50 means that the table width is 50 pixels</td>
</tr>
</tbody>
</table>

After setting up the basic structure of a table, the `<TR>`, `<TH>`, and `<TD>` tags are used to construct the rows, the heading cells, and the data cells, respectively. If we specify an attribute for the `<TR>` tag, it will be applied to all the data cells of that row.
For changing the attribute of a data cell or a heading cell, we can override the original attribute with a new attribute in the respective `<TH>` or `<TD>` tag. If we do not specify any attribute, the web browser will use the default attributes. The common attributes for these tags are given as follows.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALIGN</td>
<td>Defines the cell alignment method (LEFT, CENTER, RIGHT, JUSTIFY, or CHAR), e.g., ALIGN = CENTER means that the text is aligned to the center of the cell.</td>
</tr>
<tr>
<td>BGCOLOR</td>
<td>Specifies the background color.</td>
</tr>
<tr>
<td>COLSPAN*</td>
<td>Specifies the number of columns spanned by the data cell, e.g., COLSPAN = 2 means that the data cell covers two columns.</td>
</tr>
<tr>
<td>ROWSPAN*</td>
<td>Specifies the number of rows spanned by the data cell, e.g., ROWSPAN = 2 means that the data cell covers two rows.</td>
</tr>
<tr>
<td>VALIGN</td>
<td>Defines the vertical alignment method (TOP, MIDDLE, BOTTOM), e.g., VALIGN = MIDDLE means that the vertical alignment method is middle.</td>
</tr>
<tr>
<td>WIDTH*</td>
<td>Sets the cell width in terms of the table width percentage or pixels, e.g., WIDTH = 20 means that the cell width is 20 pixels.</td>
</tr>
</tbody>
</table>

Figure 3.4 shows an example of a simple table incorporating the table tags and the common attributes.

### 3.10 FRAMES

Very often, a company wants to display multiple Web pages on a browser. In HTML, frames are available for satisfying this requirement. For example, a company may set up a LEFT frame and a RIGHT frame for displaying the navigation buttons and the content, respectively. The navigation buttons and the content can be written in two different HTML files. To set up frames, we need to use the `<FRAMESET>` tag to define the frame format and then use the `<FRAME>` tag to define the frame content.

The common attributes of the `<FRAMESET>` tag are:

- **COLS**="c₁, c₂ .... cₚ" (i.e. there are p columns)
- **ROWS**="r₁, r₂ .... rₘ" (i.e. there are m rows)

where cₓ and rₓ and are the sizes of the x-th column and row, respectively. The units are expressed in terms of a percentage of the document width or pixels. Furthermore,

* These attributes are for `<TD>` and `<TH>` only.
we can use * to tell the browser how to allocate the available space. Let us look at a few examples. `<FRAMESET COLS="*, *">` can be used to set up two column frames (left and right) with equal width. If we use 3* instead of * for the right frame (i.e., `<FRAMESET COLS="*, 3*">`), this will set the right frame three times the size of the left frame. What does `<FRAMESET ROWS="20%, *, 3*">` mean? It means that there are three row frames. Starting from the top of the browser, the first frame is 20% of the browser window in height. For the last two frames, the third one is three times the size of the second one.

After defining the frame layout using the `<FRAMESET>` tag, we need to specify the frame content by using the `<FRAME>` tag. The common attributes of the `<FRAME>` tag are as follows:
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRAMEBORDER</td>
<td>Indicates whether a frame border is used (1) or not used (0)</td>
</tr>
<tr>
<td>HEIGHT</td>
<td>Sets the frame height</td>
</tr>
<tr>
<td>MARGINHEIGHT</td>
<td>Specifies the top and bottom margins</td>
</tr>
<tr>
<td>MARGINWIDTH</td>
<td>Specifies the left and right margins</td>
</tr>
<tr>
<td>NAME</td>
<td>Specifies a name for identification purpose</td>
</tr>
<tr>
<td>NORESIZE</td>
<td>Does not allow users to change the size of the frame</td>
</tr>
<tr>
<td>SCROLLING</td>
<td>Indicates whether a scrollbar is used (YES, NO, or AUTO)</td>
</tr>
<tr>
<td>SRC</td>
<td>Specifies the URL of the initial HTML file to be loaded</td>
</tr>
<tr>
<td>WIDTH</td>
<td>Sets the frame width</td>
</tr>
</tbody>
</table>

Again let us look at some examples. If we want to specify three column frames (called, say, LEFT, MIDDLE, and RIGHT) of equal width, the corresponding HTML code is as follows:

```html
<FRAMESET COLS="*,*,*"
    <FRAME NAME=LEFT SRC=Left.html>
    <FRAME NAME=MIDDLE SRC=Middle.html>
    <FRAME NAME=RIGHT SRC=Right.html>
</FRAMESET>
```

where SRC specifies the initial HTML file for the frame.

Many companies like to divide their web pages into three frames: a banner frame, for displaying the company banner; an index frame, for showing the navigation buttons; and a content frame, for presenting the content. This can be done using nested frames as follows and Figure 3.5 shows the corresponding web page.

```html
<HTML>
<FRAMESET ROWS="20%,80%">
    <FRAME NAME="BANNER_FRAME" SRC=BANNER.html>
    <FRAMESET COLS="30%,70%">
        <FRAME NAME="INDEX_FRAME" SRC=INDEX.html>
        <FRAME NAME="CONTENT_FRAME" SRC=CONTENT.html>
    </FRAMESET>
</FRAMESET>
</HTML>
```

When a hyperlink is clicked, the TARGET attribute of the `<A>` tag determines in which frame the corresponding HTML page is opened. For example, suppose that
there are two frames, namely,

```html
<FRAME NAME="left" SRC="left.html">
<FRAME NAME="right" SRC="right.html">
```

If a hyperlink is defined as follows:

```html
<A HREF="page.html" TARGET="left">Choice</A>
```

the web page "page.html" will be opened in the left frame.

Other target attributes include the following:

- `_blank`: display the HTML page in another browser window
- `_self`: display the HTML page in the originating frame
- `_top`: display the HTML page in the full browser window without any frame (i.e., eliminate the previous frames)
You can also specify the default target attribute in the <BASE> tag within the <HEAD> section of the HTML document. For example,  
<br>  
<BASE TARGET="CONTENT"> means that unless otherwise specified, all documents should be displayed in the frame named "CONTENT."
  
While frames provide many advantages, they should be used with caution. Some older browsers do not support frames, so it is a good practice to include the <NOFRAMES></NOFRAMES> tag pair to cater for this situation. For example, you can include a statement like this:
  
<br>  
<NOPRAMES>
Sorry. You cannot view this web page because your browser does not support frames.
Please choose the nonframe version.
</NOFRAMES>
  
Furthermore, a framed web page may be displayed differently if a different browser is used or the screen resolution is changed; so framed web pages should be tested more carefully. It is also a common practice to provide a "non-frame" version in case the user encounters difficulties in displaying the framed web pages.

### 3.11 FORM

Forms are generally used to obtain data from the client for submission to the server. Typically, a program in the server is invoked to process the data, possibly with the assistance of the backend system. The result (in most cases, an HTML file) will then be passed to the web client by using the HTTP.

In general, an HTML form has the following format [Naik, 1998]:

```
<FORM ACTION="Program URL" METHOD="GET or POST">
<!-- Put form input elements here -->
</FORM>
```

The ACTION attribute provides the URL of the program for processing the form data, and the METHOD attribute specifies the method for passing data to the server (i.e., by using GET or POST). Recall that if GET is used, data is attached to the destined URL using a query string ([Hall, 1998]). If POST is used, data will be embedded inside the HTTP request message.

Between the <FORM> and </FORM> tags, different form input elements can be included. Most of them are defined by the <INPUT> tag. The common attributes of
the `<input>` tag are as follows:

- `type`: the type of the input such as "text" and "checkbox" as described later
- `name`: the name of the input
- `value`: the value of the input

3.11.1 Textbox

A textbox is used to collect simple text input. The length of the textbox is specified by the `size` attribute. We can also use the `maxlength` attribute to specify the maximum number of characters that can be entered into a textbox. Here is an example of a simple textbox:

```html
<input type="text" name="InputName" size="15">
```

It means that the textbox is 15 characters long, the NAME of the input is called "InputName" and its value is the data entered in the textbox.

3.11.2 Password textbox

Password textbox is a special textbox for entering a password. It is almost the same as the textbox except that the entered characters appear as * so that they cannot be seen by others. Here is an example:

```html
<input type="password" name="InputName" size="20">
```

3.11.3 Checkbox

Checkbox is for making a choice out of a number of items. It is defined by `type="checkbox"`. If a checkbox is checked, the specified `name` and the corresponding `value` will be passed to the server for processing. If you want to set a checkbox by default, you can set the `checked` attribute within the `<input>` tag. By setting this attribute, the checkbox will appear as checked, otherwise the checkbox will remain blank. In the following example, the checkbox is checked by default. If the user chooses this default choice, the parameter "InputName=InputValue" will be passed to the server for processing.

```html
<input type="checkbox" name="InputName" value="InputValue" checked>
```
3.11.4 Radio button

Radio button functions like the checkbox. An example is shown below.

\[
<\text{INPUT TYPE}="\text{RADIO}" \text{NAME}="\text{InputName}" \text{VALUE}="\text{InputValue}" \text{CHECKED}>
\]

3.11.5 Submit button

A submit button is for submitting a form to the web server. That means, when a submit button is clicked, the input data (i.e., all the NAME/VALUE pairs in the form) will be sent to the server using the METHOD specified in the \text{<FORM>} tag. The following is a simple example of the submit button. Users will see the word "Submit" on the button. When the button is clicked, the input data on the form as well as the parameter "\text{InputName}=\text{Submit}" will be passed to the server for processing.

\[
<\text{INPUT TYPE}="\text{SUBMIT}" \text{NAME}="\text{InputName}" \text{VALUE}="\text{Submit}">
\]

3.11.6 File input field

The file input field is for uploading a file (e.g. an image file) to the server. It is defined by \text{TYPE}="\text{FILE}". The name of the file is specified by the \text{NAME} attribute. The length and maximum size of the field is defined by the \text{SIZE} and \text{MAXLENGTH} attributes, respectively.

Furthermore, the allowable file type(s) can be set with the \text{ACCEPT} attribute. Here is an example:

\[
<\text{INPUT TYPE}="\text{FILE}" \text{NAME}="\text{BOOK1}" \text{ACCEPT}="\text{image/gif}\ " \text{MAXLENGTH}="40" \text{SIZE}="15">\]

It means that the name of the file is \text{BOOK1}, the length of the input field is 15 characters, and only a maximum of 40 characters can be entered. The uploaded file must be an image file in GIF format.

3.11.7 Hidden form field

Hidden form fields are hidden (i.e. not displayed). They are used mainly to pass additional data to the server. For example, they can be used to pass user's state to the
server like cookies do. An hidden field is defined by TYPE="HIDDEN." An example is given below.

<INPUT TYPE="HIDDEN" NAME="User_ID" VALUE="123456">

In this example, the hidden form field is used to tell the server that this request is from User_ID=123456.

### 3.11.8 Textarea (Comment box)

Textarea is not defined by the <INPUT> tag but by the <TEXTAREA> tag. It is typically used for collecting customer’s comments. The ROWS and COLS attributes specify the number of rows and columns for the textarea, respectively. For example, the following TEXTAREA has six rows and ten columns.

<TEXTAREA NAME="InputName" ROWS="6" COLS="10"></textarea>

### 3.11.9 Select menu

A select menu allows users to choose one or more choice(s). It is defined by the <SELECT> tag as shown below. The SIZE attribute inside the <SELECT> tag defines how many field(s) in the select menu is/are displayed. Within the <SELECT></SELECT> tag pair, the <OPTION> tag is used to define the options available for selection. Here is an example:

<SELECT NAME="InputName" SIZE="1">
  <OPTION VALUE="Value1">Value 1</OPTION>
  <OPTION VALUE="Value2">Value 2</OPTION>
</SELECT>

In this example, the select menu has only two fields. As the SIZE attribute is "1", only the first one is displayed. We need to scroll down the select menu in order to select the second item.

### 3.12 CASCADING STYLE SHEETS

In general, a web page has three main components, namely “presentation structure” (referred to as structure for simplicity), “style,” and “content.” In the early HTML
versions, the structure and style are integrated. Let us look at the following example:

```html
<P ALIGN=CENTER>This is a paragraph</P>
```

In this example, the structure is defined by the `<P>` tag (it defines that this is a paragraph), the style is given by the attribute inside the `<P>` tag and the content is the sentence "This is a paragraph."

In HTML 4.0, Cascading Style Sheet (CSS) is available to separate the style from the structure. Hence, it enables web designers to control the style of a web page in a more flexible manner. For example, by using CSS, a single style sheet can be applied to different web pages that require the same style (i.e., to fulfill the consistency requirement as discussed earlier). There are three types of CSS, namely

- external style sheets
- embedded style sheets
- inline style sheets

Lemay [1999] gives a very good and detailed explanation of stylesheet. Here we give an overview based on Lemay's approach.

### 3.12.1 External style sheets

For external style sheets, the style definitions are stored in a separate file with a file extension of "*.css." Using external style sheets, our VBS can maintain a consistent style by applying the same style sheet to all the HTML files.

To create an external style sheet for an HTML file, there are two basic steps:

- **Step 1:** Create the following `<LINK>` tag in the `<HEAD>` section of the HTML file:

  ```html
  <LINK REL="stylesheet" HREF="external_stylesheets.css">
  ```

  where `REL` specifies that a style sheet is to be used and `HREF` specifies the URL of the external style sheet.

- **Step 2:** Create the external style sheet and save it as the specified file.

  Let us create an external style sheet called "style1.css" as follows:

  ```css
  BODY { font-color: blue; font-family: Times New Roman;
          font-size: 20 pt}
  A:link { color: red }
  H1 { font-weight: bolder }
  H2 { font-weight: bold }
  ```
In this example, the external style sheet specifies the font color of the BODY section to be "blue", the default font to be "Times New Roman," and the default font size to be 20 pt. For the reference links, the default color is "red". The font-weight of the first and second level headings is set to be bolder and bold, respectively.

Besides the background color, font style, and font properties, a web designer can also apply the following style properties to a web page:

- Page layout properties on the page margin:
  margin-x
  where "x" is the attribute including left, right, top, or bottom.

- Other background properties:
  Background-x
  where "x" is the attribute including color or image.

- Font properties:
  font-x
  where "x" is the attribute including style, family, size, or weight.

- Border properties:
  border-x
  where "x" is the attribute including style, width, or color.

Details of the CSS formatting styles and rules can be found in W3C CSS homepage:
http://www.w3.org/Style/.

### 3.12.2 Embedded style sheets

Instead of using an external style sheet, we can also embed the style definitions inside the HEAD section of the HTML file. This is called embedded style sheets.

In this case, the style definitions are put between the <STYLE> and </STYLE> tags within the HEAD section of the HTML file as shown below:

```
<HEAD>
...
<STYLE TYPE="text/css">
<!--
  Put the style definitions here
-->
</STYLE>
</HEAD>
```
The TYPE attribute specifies the content (MIME) type of the style. In most cases, it is "text/css." As you can see, all the style definitions are embedded within the comment tags "<!--" and "-->. This allows browsers that cannot support CSS to ignore the style definitions by processing them as comments.

The previous style sheet can be embedded within the HEAD section as follows to create the same style.

```html
<HEAD>
<TITLE>VIRTUAL BOOKSTORE (VBS): WELCOME</TITLE>
<style type="text/css">
<!--
body {
font-color: blue; font-family: Times New Roman;
font-size: 20 pt}
A:link {
color: red }
H1 {
font-weight: bolder }
H2 {
font-weight: bold }
--> 
</style>
</HEAD>
```

In general, if we want to create a set of web pages with the same style, the external style sheet should be used. On the other hand, if we want to design a web page with a unique style, the embedded style sheet provides a better solution.

### 3.12.3 Inline style

In some situations, we may want to apply a style rule to part of a web page (e.g., a paragraph). In this case, the inline style can be used. Suppose we want to set the style of a heading, this can be done by the `<STYLE>` attribute as follows:

```html
<h1 style="font-family: Helvetica, sans-serif; font-size: 18pt">
The specific style for this heading.</h1>
```

By doing so, the content between the `<h1>` and `</h1>` tags is reformatted using the style rule as defined by the `STYLE` attribute.

Apart from using the `STYLE` attribute, users can also create the same effect by using the `CLASS` attribute. This can be used in an external style sheet as well as in an embedded style sheet. By using the `CLASS` attribute, the defined style rule can be reused elsewhere in a web page. It is best to explain with an example. In this example,
a user can specify two different styles for the <H1> tag using embedded style sheet as shown below:

```html
<STYLE TYPE="text/css">
<!--
H1 {font-family: Helvetica; font-style: normal}
H1.italic {font-family: Helvetica; font-style: italic}
-->
</STYLE>
```

The first one and the second one give the default style and the style for the first level heading requiring italic display, respectively. If we want to display a first level heading in italic, the CLASS attribute can be specified as follows:

```html
<H1 CLASS="italic">
The words will be displayed in italic.
</H1>
```

The STYLE and CLASS attributes work fine for text within container tags (i.e., those with open and close tags). For noncontainer tags, we can use the <DIV></DIV> tag pair to apply the style as follows:

```html
<DIV STYLE="...">...</DIV>
<DIV CLASS="NameOfClass">...</DIV>
```

In the first case, a user can specify the required inline style for the text between the <DIV> and </DIV> tags. In the second case, the style as defined by the "NameOfClass" will be applied to the content between the <DIV> and </DIV> tags. Alternatively, we can use the <SPAN></SPAN> tag pair for the same purpose.

### 3.13 **JAVASCRIPT**

#### 3.13.1 What is JavaScript?

JavaScript is a scripting language proposed by Netscape to enhance the functions of HTML (e.g. form validation). It is often called an object-oriented (OO) scripting language with syntax looking like Java. In particular, it can be used to make a
web page more interactive and dynamic. It is supported by most commonly used browsers including Microsoft’s Internet Explorer and Netscape’s Navigator. Unlike the server-side programs, a JavaScript code is included in an HTML document and executed on the client side. In this section, we will give an overview of JavaScript by using some examples. For the detailed documentation and the latest developments of JavaScript, please refer to Netscape’s site on JavaScript (http://developer.netscape.com/docs/manuals/index.html).

### 3.13.2 Basic structure of JavaScript

A JavaScript code is embedded between the `<SCRIPT>` and `</SCRIPT>` tags as follows [Lemay, 1999]:

```html
<HTML>
<HEAD>
<TITLE>HTML file with JavaScript code</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
    Put the JavaScript code here.
//-->
</SCRIPT>
</HEAD>
</HTML>
```

In the example, the LANGUAGE attribute specifies that JavaScript is used. Other scripting languages such as VBScript can also be used. The JavaScript code is put between the comment tag pair, i.e., "<!--" and "//-->" so that if the browser does not support JavaScript, the code will just be processed as a comment rather than an error. For complex JavaScript codes, they can be stored in a separate file with a file extension of ".js." In this case, the JavaScript code(s) can be linked to the HTML file by using the `SRC` attribute of the `<SCRIPT>` tag as follows:

```html
<SCRIPT LANGUAGE="JavaScript" SRC="JavaScript_URL">
```

where `JavaScript_URL` specifies the URL of the JavaScript code.
3.13.3 A simple JavaScript example

In JavaScript, there are three main objects, document, form, and location, as described briefly here [Lemay, 1999]:

- **Document object** – for providing information on the document, such as page characteristics, links, etc.
- **Form object** – for providing information on the form(s) used in the current web page, such as information on a particular form element.
- **Location object** – for providing location related information for the current web page, such as URL, host name, directory path, etc.

In many cases, a JavaScript code is invoked when a certain event occurs (e.g., when a form is submitted or the mouse is clicked). JavaScript provides a number of “event handlers” for handling this requirement. Some common event handlers are as follows:

- **onClick** – indicates that the mouse is clicked
- **onMouseOver** – indicates that the mouse is moved over a specific element
- **onSubmit** – indicates that the form is submitted
- **onKeyPress** – indicates that a key is pressed

Let us first look at a simple JavaScript example for displaying a welcome message, the URL of the current web page, and the current date. The JavaScript code is as shown below:

```html
<HTML>
<HEAD>
<TITLE>JavaScript Hello World</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
document.write("<HR ALIGN='left' WIDTH=80%><BR>");
document.write("<H1>JavaScript : Hello World!!</H1><BR>");
document.write("<HR ALIGN='left' WIDTH=80%><BR>");
document.write("Current URL is:");
document.write(location.toString( ));
document.write("<BR>Current time is:");
document.write(Date( ));
//-->
</SCRIPT>
```
Figure 3.6 gives a snapshot of the display when the web page is loaded by a browser.

In this example, the \texttt{write} method of the \texttt{document} object is used to write the HTML file using the standard HTML tags. By using the \texttt{toString()} method of the \texttt{location} object, the current URL can be displayed. Furthermore, we can print out the current time by using the \texttt{built-in Date()} method or function.

### 3.13.4 Form validation using JavaScript

An important application of JavaScript is for form validation. In this section, we present a simple form validation example. Suppose that a form is created as follows:
As you can see, this form is used to collect the name, registration status, date of birth, and e-mail address of a customer. To validate the input data, we need to check that the "Name" field is not empty, one of the radio buttons is selected, the "e-mail" field contains the character "@" and the "Date" field contains a valid date. These are the commonly used form validation procedures. The corresponding JavaScript code is shown in Figure 3.7. Let us explain the key points as follows.

The form is validated by a function called "validation" and the input argument is called "qform." The value of the "Name" field is identified by "qform.name.value." Hence, if it is empty, an alert message will be displayed by using the alert function. Note that the alert function is provided by JavaScript and the input argument is the alert message. Following Lemay's [1999] approach, for the radio buttons, "qform.register[i].status" (i=0,1) can be used to determine whether the (i+1)-th radio button is checked. If it is not checked, the status will be "false." By checking the status of the radio buttons, we can determine whether the customer has been registered. If not, an alert message will be displayed accordingly.

To check whether a valid date is entered for the "Date of birth" field, we can combine the field elements (i.e., qform.year, qform.month, and qform.day) into a date variable using the Date() method as follows:

```javascript
var datefield = new Date(qform.year.value, qform.month.value, qform.day.value);
```

Then we can check whether the "datefield" is valid or not using the "isNaN()" method as follows:
<SCRIPT LANGUAGE="JavaScript">
<!--
function validation( qform ) {
// Check name
if(qform.name.value==""){
    alert("Please input your name!");
    return false;
}

// Check registration
if(qform.register[0].status==false &amp; qform.register[1].status==false){
    alert("Please indicate your registration status!");
    return false;
}

// Check date, month and year
var datefield = new Date(qform.year.value,qform.month.value,qform.day.value);
if(isNaN(datefield.valueOf())){
    alert("Please enter a valid date!");
    return false;
}

// Check Email address
if(qform.email.value.indexOf('@')==-1){
    alert("Please enter a valid email address!");
    return false;
}
return true
}
//-->
</SCRIPT>

Figure 3.7 JavaScript for form validation

if(isNaN(datefield.valueOf())){
    alert("Please enter a valid date!");
    return false;
}

If it is invalid, the results will be "NaN." In this case, an alert message will be displayed accordingly.
To check whether the "e-mail" field contains the character "@," we can use the "indexOf" method as follows:

```javascript
qform.email.value.indexOf("@") == -1
```

If this statement is true, it means that the e-mail field does not contain the character "@," so the corresponding alert message should be displayed.

In the `<FORM>` tag, we need to activate the validation function as follows:

```html
<FORM ACTION="/servlet/vbs.processform" METHOD="POST"
onSubmit="return validation(this)">
```

It means that when the form is submitted (as detected by the onSubmit event handler), the "validation" function is triggered to validate the form. Note that the "this" parameter refers to the current form object. If the validation result is "false" (i.e., the validation fails), the submission will not be proceeded, otherwise the form will be submitted to the server for processing by the program "vbs.processform" stored under the "servlet" directory.

The above is a simple but a commonly used example to show how JavaScript can be used for form validation. In fact, a complete form validation library using JavaScript can be found in the Netscape JavaScript developer web site (http://developer.netscape.com/docs/examples/JavaScript/formval/overview.html).

### 3.14 SUMMARY

In this chapter, we have given an overview of the web page design process as well as HTML 4.0 and JavaScript for client-side programming or web publishing. HTML makes use of tags to format a document. For most tags, additional attributes can be defined. Basically, HTML can be used to format text, to link documents, to add images, and to construct tables. In addition, frames are often used to display multiple web pages in the same browser window. With HTML 4.0, web page designers can control the styles of web pages more effectively by using CSSs. Besides formatting web pages, HTML provides various form input elements for collecting user's input. By submitting a form to the server, a server-side program can be invoked to process the user's request with the input data. We will describe server-side programming techniques in the next chapter. JavaScript can be used to enhance the functionality of HTML in particular to make a web page more dynamic and interactive. A typical application of JavaScript is form validation.
REFERENCES


RECOMMENDED READING


This book provides a good introduction to HTML.

These books, particularly the first two, give an in-depth overview of the standards concerning HTML 4.0 and JavaScript. They are highly recommended. The last book includes a good HTML quick reference table in the appendix.


The HTML specification and its latest development can be found at the www.w3.org. This web site also includes a short tutorial on HTML 4.0.
In the last chapter, we explored how to build the web system using various client-side programming and web publishing techniques. However, in e-commerce applications, client-server programming is of the utmost importance. The facilities provided range from simple electronic form submission systems to more sophisticated shopping cart systems in an interactive cyber store. When designing server-side applications, we
need to consider many different factors such as efficiency, security, cost-effectiveness, and compatibility. Traditional Common Gateway Interface (CGI) programming techniques may become deficient under these considerations.

In this chapter, we initially discuss common server-side programming techniques. In particular, we will introduce Java Servlets – an effective programming technology that extends the functionality of Java to the server-side. We will also explain the advantages of servlets over other techniques for developing web-based e-commerce applications. We also discuss the basic framework of the servlet model and how it can be integrated with other technologies to implement e-commerce applications. Finally, we will present two simple Java Servlet programs.

4.1 REVISITING THE THREE-TIER MODEL

In previous chapters, we introduced the “three-tier model” for building e-commerce applications. As an introduction to server-side programming, let us revisit the three-tier model before discussing various server-side programming techniques.

To achieve the purposes of modular design and platform independence, web-based e-commerce applications are usually developed based on the three-tier model as shown in Figure 4.1. By means of the three-tier model, we can separate the business logic of the web applications from the “frontend” (i.e. web client) and the “backend” (i.e. database systems). This gives us a more flexible and scaleable system.

In summary, the three-tier model has the following components:

1. The first tier – Web client: The first tier provides a web-based Graphical User Interface (GUI) displayed through a web browser in the client computer. Implementation of the web client in the web application is usually referred to as “Web publishing” and “Client-side programming,” which has been extensively discussed in Chapter 3.

2. The second tier – Server-side application (SSA): The second tier consists of server-side applications that run on a web server or a dedicated application server. In general, these applications implement the business logic of the web system. In this chapter, we will give an overview of the following server-side programming techniques:
   - Common Gateway Interface (CGI)
   - Active Server Page (ASP)
   - Java Servlets

   We will compare their advantages and shortcomings. As an example, we will use Java Servlets as the server-side programming tool throughout the book.
3. *The third tier* – *Database management systems (DBMS):* The third tier provides data storage/retrieval services for the second tier so that dynamic web pages can be created. Depending on the system requirements, the third tier may consist of one database or a group of databases (i.e. database cluster). To “bridge” the second tier server-side applications and the “backend” DBMS, there are numerous ways to provide the database connectivity. A popular method is by means of JDBC such as a JDBC-ODBC (Java Database Connectivity–Open Database Connectivity) bridge. Alternatively, other techniques such as Proprietary Network Protocol drivers and Native API drivers can also be used. To facilitate communication with a database, the Structural Query Language (SQL) is often used. This will be discussed in Chapter 5.
In general, the three-tier model has the following advantages over the traditional single-tier or two-tier model, especially for web applications:

- Its modular design or layered architecture facilitates the change or replacement of one tier without affecting the other tiers.
- Using browsers as the web clients allows different applications to share the same look and feel.
- As web browsers can be found in almost all computers, web applications can be accessed from almost anywhere.

Another important requirement in e-commerce applications is “state tracking,” or “session tracking.” As HTTP is a stateless protocol (i.e., it does not keep track of the user’s state), session tracking and management techniques are required for supporting many e-commerce application functions such as user login and shopping carts. In Chapter 6, we will discuss different types of session tracking techniques for building e-commerce applications. These include

- Hidden form fields
- Cookies
- URL rewriting
- HTTP user authorization

We will compare their advantages and disadvantages in terms of usability and ease of implementation. We also explore the Java Servlet Session Tracking API—a platform-independent Java Servlet library for supporting session tracking. We discuss how it works and why it is more effective than other session tracking techniques. In particular, we will use a simple shopping cart example to illustrate how the Java Servlet Session Tracking API can be used in the three-tier model for building interactive e-commerce applications.

### 4.2 COMMON GATEWAY INTERFACE (CGI)

#### 4.2.1 CGI fundamentals

Early web pages were “static.” In other words, a client could request only a static HTML document from the web server as shown in Figure 4.2. Later, CGI programming techniques were introduced to eliminate this constraint. CGI programming allows a web client to pass data to a server-side application so that a dynamic web page can be returned to the client according to the input data.
Figure 4.2 Static web page retrieval

Figure 4.2 explains the retrieval of a static web page. When the web browser receives a client request, it goes to the designated URL on a specific web server to retrieve the required static HTML document. Note that the content is independent of the request, in the sense that everyone who makes a request of that particular URL gets the same document.

If we are to allow interactivity between the web client and the web server, one needs a server-side programming technique to generate dynamic web pages. Such interactivity is of particular importance in e-commerce systems for purposes such as order submission or data input. CGI programming provided one of the first techniques that was utilized for this interactivity (Figure 4.3).

In a typical CGI-based web application, to provide this interactivity, a client invokes a CGI script to perform a specific action on the server side. For example,
a "visit counter" can be included for displaying the number of visits to a particular web page, which can be done with the following image tag:

```html
<IMG SRC="/cgi-bin/visit-counter">
```

This causes the web browser to start a CGI script on the server side on encountering the `<IMG>` tag. This CGI script updates the counter value and returns the current counter value to the client in the form of a GIF image.

Another common approach to invoke a CGI program is by using an HTML form. As discussed in Chapter 3, we can invoke a server-side program by using the `ACTION` attribute in an HTML form. To pass data from the web client to the web server for data processing using HTML forms, one can include the CGI program called "order.pl" in the `<FORM>` tag as follows:

```html
<Form METHOD="POST" ACTION="/cgi-bin/order.pl">
```

Note that the `ACTION` attribute specifies the server-side application or script to be invoked. In this case, it is a Perl script called "order.pl" stored under the "cgi-bin" directory of the web server.

The `METHOD` attribute of the `<FORM>` tag (i.e. `POST`) tells the browser how to send the information to the server. In this case, the data is embedded in the HTTP request message. If one uses the `METHOD` attribute `GET` instead, the data will be appended to the end of the URL.

### 4.2.2 CGI languages

CGI languages can be interpreted scripted languages (e.g., Perl, Apple Script, Unix Shell Scripting, and TCL) or compiled languages (e.g., C, C++, and Visual Basic). Perl (Practical Extraction and Report Language) has evolved to become a web programming language and is one of the most widely used CGI interpreted scripting languages. Running a script requires an interpreter to interpret the script before performing the required tasks. This makes for slow execution. However, scripts are easier to learn. Compiled languages produce a compact binary executable code from the source CGI code, and execution of this binary executable code leads to faster execution.

### 4.3 ACTIVE SERVER PAGE (ASP)

To develop interactive web applications, Microsoft introduced a server-side programming tool called Active Server Page (ASP). ASP is a "scripting" technique that runs on web servers rather than web clients. This contrasts with VBScript and JavaScript,
which run on web clients. It basically generates dynamic HTML documents for the web client. Execution of the ASP code by the server returns the corresponding HTML document to the client. The server-side code written in ASP can be embedded in the HTML document, which allows one to insert it into web pages even though it is executed on the server. As ASP is a Microsoft product, it can easily be integrated with other Microsoft web development tools and ActiveX controls. Figure 4.4 shows the schematic diagram of the ASP model.

However, ASP has two disadvantages. Firstly, it is not a formal programming language, so debugging can be more difficult. Secondly, it is not object-oriented. In the next section, we will introduce an effective server-side programming technique called Java Servlet. As it is an object-oriented programming technique based on Java, it has a number of advantages over the other server-side programming techniques.

4.4 OVERVIEW OF JAVA SERVLET

Java was originally introduced by Sun Microsystems Inc. with the aim of enhancing interactivity in the web, particularly on the client side. To accomplish this, Sun developed a small client-side application called Applet ("App" means applications and "let" means small). Although applets can enhance client-side interactivity, this is done at the expense of long downloading time; hence, they are not attractive for most e-commerce applications. Furthermore, in many e-commerce applications, a client's request is often required to be processed in conjunction with the backend databases. For example, a customer may want to look up a particular product from the backend database. Obviously, it is not effective to download the whole database to the client side for processing. Hence, there is a strong need for server-side Java to cater for these requirements.
A servlet is a small piece of server-side application, which can be viewed as the server-side analog of an applet. In a typical servlet application, a servlet-enabled web server receives an HTTP request from the client (see Figure 4.5). It then forwards the request to the servlet engine for performing the necessary operations as specified by the program. Finally, it returns a response (e.g. HTML document) to the client via the web server.

Let us look at a simple book ordering system for our VBS based on Java Servlet. The servlet is invoked by using an HTML form. The main steps are described as follows:

1. Using his browser, the customer accesses our VBS web server through HTTP.
2. The book ordering form is forwarded to the customer's browser.
3. The customer fills in the book ordering form (electronically) and sends the form to the server by pressing the "Submit" button.
4. This HTTP request is forwarded to the servlet engine by the VBS web server.
5. The servlet processes the request by performing the necessary operations, e.g. updating the order transaction database and invoking the payment gateway for the internet payment, etc.
6. After processing, the corresponding response is returned to the client via the web server.

Compared to other server-side programming techniques, particularly traditional CGI programming, Java Servlet has the following advantages:

- As it is a formal programming language, debugging is easier. Furthermore, its object-oriented features can greatly facilitate program design.
Each servlet can handle multiple requests. In other words, once a servlet is invoked, it will remain in the system and can be used by different requests requiring the same servlet.

Traditional CGI programming techniques such as Perl are usually platform-dependent. Java Servlets, on the other hand, are based on the philosophy "written once, run everywhere."

As part of the Java family, servlets can use the Java security APIs if necessary and can be easily integrated with other Java-based programming techniques such as CORBA, RMI, JDBC, and JCA (Java Cryptography Architecture) to build a comprehensive e-commerce system.

However, compared with other server-side scripting techniques such as Perl and ASP, the writing of servlets generally requires more programming effort and hence longer development time.

### 4.5 JAVA SERVLET ARCHITECTURE

A servlet is a server-side Java program running inside a Java Virtual Machine (JVM). Through the servlet engine, it can interact with the server and also the HTTP. Like other CGI applications, a servlet is invoked by a client from the client browser (e.g. via an HTML form). It may also be invoked by other servlets or Java programs. As mentioned earlier, unlike traditional CGI applications, which need to set up multiple processes for handling multiple requests, a servlet can handle multiple requests under different threads. Therefore, servlets provide a solution that is more scaleable. Furthermore, servlets can interact closely with the server to do things that may be performed easily with the traditional CGI programming techniques.

To run servlets, there are basically two alternatives. The first one is to use a servlet-enabled web server, i.e., a web server that can support the servlet APIs directly. The second solution is to use a "plug-in" servlet engine in a nonservlet-enabled web server. Tables 4.1 and 4.2 give some examples of the commonly used servlet-enabled web servers and the "plug-in" servlet engines [Moss, 1998; Hunter and Crawford, 1998].

### 4.6 OVERVIEW OF THE SERVLET API

The life cycle of a servlet looks like this. Upon receiving a request to invoke a servlet, the server will create the servlet, call the `init()` method, and then the `service()` method of the servlet. The `init()` method is for performing initialization actions. If a
Table 4.1 Example of servlet-enabled web servers

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Product</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM</td>
<td>Visual Age WebRunner</td>
<td><a href="http://www.ibm.com/software/ad/webrunner/">http://www.ibm.com/software/ad/webrunner/</a></td>
</tr>
<tr>
<td>O'Reilly</td>
<td>Website Professional</td>
<td><a href="http://www.oreilly.com/catalog/webpro2/">http://www.oreilly.com/catalog/webpro2/</a></td>
</tr>
<tr>
<td>W3C</td>
<td>Jigsaw HTTP Server</td>
<td><a href="http://www.w3.org/Jigsaw/">www.w3.org/Jigsaw/</a></td>
</tr>
<tr>
<td>Web Easy</td>
<td>WEASAL</td>
<td><a href="http://www.webeasy.com/products/weasel.htm">http://www.webeasy.com/products/weasel.htm</a></td>
</tr>
</tbody>
</table>

After a servlet has been invoked before, it can be reused. In this case, the service() method will be called directly for processing the request. Finally, if a servlet is to be removed, the destroy() method is called before removing it. In this book, we will only cover the servlet basics, in particular how servlets can be used to build the VBS. For details on servlet programming, please refer to the references at the end of this chapter.

There are two main packages in the Servlet API, namely javax.servlet and javax.servlet.http. They can be downloaded from http://java.sun.com/products/servlet as part of the Java Servlet Development Kit (JSDK). The two packages are for developing generic servlets and servlets for the HTTP (i.e., under

Table 4.2 Example of third-party middleware servlet engines

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Product</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM</td>
<td>WebSphere Application Server</td>
<td><a href="http://www-4.ibm.com/software/webservers/appserv/">http://www-4.ibm.com/software/webservers/appserv/</a></td>
</tr>
<tr>
<td>Live Software</td>
<td>JRun</td>
<td><a href="http://www.jrun.com/Products/Jrun/">http://www.jrun.com/Products/Jrun/</a></td>
</tr>
</tbody>
</table>
Figure 4.6 A generic servlet model (based on the work of Hunter and Crawford [1998])

As shown here, there are two object parameters, namely ServletRequest and ServletResponse. The former object is related to the request to the servlet program (e.g., it contains the client's information). On the other hand, the ServletResponse object is used to handle the response returned to the client via the server. This is shown in Figure 4.6.

For HTTP servlets, the service() method passes the requests to the corresponding doREQ() method where REQ is the HTTP request command. In particular, the doGet() and doPost() methods are invoked by the HTTP GET and POST commands, respectively. For HTTP servlets, it is preferable to override the doREQ() methods rather than the service() method. This allows actions to be taken based on the type of HTTP request received. A schematic overview of the HTTP servlet model is shown in Figure 4.7.

Besides the doGet() and doPost() methods, a variety of methods corresponding to different HTTP commands are available as shown in Table 4.3. Essentially, these methods allow a servlet to perform actions according to the type of HTTP request received. The most commonly used methods are doGet(), doPost(), and doHead().

4.7 BUILDING THE VIRTUAL BOOKSTORE – STEP BY STEP

Throughout this book, we will demonstrate how to build a simple Virtual Bookstore (VBS) by using Java Servlets as the core server-side programming tool. We
will illustrate how to build this dynamic web-based application using the following programming techniques:

- Client-side programming using HTML and JavaScript (Chapter 3).
- Server-side programming using Java Servlets (this chapter and chapters 5 and 6).
- JDBC programming with servlets to implement database connectivity (Chapter 5).
- Session tracking technique using the servlet session tracking API to implement user login and shopping cart applications (Chapter 6).

Table 4.3 A summary of methods used to create a functional HTTP servlet

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service()</td>
<td>This method is invoked for all generic servlets when the server receives a request. In case of HTTP servlets, this method will usually be overridden by one of the following methods associated with various HTTP requests (e.g. POST or GET).</td>
</tr>
<tr>
<td>doGet()</td>
<td>This method is invoked when the server receives an HTTP GET command.</td>
</tr>
<tr>
<td>doPost()</td>
<td>This method is invoked when the server receives an HTTP POST command.</td>
</tr>
<tr>
<td>doPut()</td>
<td>This method is invoked when the server receives an HTTP PUT command.</td>
</tr>
<tr>
<td>doDelete()</td>
<td>This method is invoked when the server receives an HTTP DELETE command.</td>
</tr>
<tr>
<td>doTrace()</td>
<td>This method is invoked when the server receives an HTTP TRACE command.</td>
</tr>
<tr>
<td>doOptions()</td>
<td>This method is invoked when the server receives an HTTP OPTIONS command.</td>
</tr>
</tbody>
</table>
YOUR FIRST SERVLET – WELCOME TO VBS

Like learning other programming languages, let us first show you how to print a simple "Welcome to VBS" message. This programming example shows you how to display a simple message "Welcome to VBS" on the client browser using a servlet program called "WelcomeVBS.java." The program listing is given in Figure 4.8.

Figure 4.9 shows a simple HTML file "WelcomeVBS.html" to invoke the servlet called WelcomeVBS. As indicated in the package statement inside the program, this servlet is stored under the directory "/vbs/servlet1/.

Before studying the servlet program in detail, let us briefly look at the caller file, WelcomeVBS.html. It is a very simple HTML form. In the FORM tag, the URL of the servlet file "WelcomeVBS" is specified using the Unix naming convention. In this case, "/servlet/" is used to inform the web server that the file WelcomeVBS is a servlet program. It is a logical directory in the server. In the HTML form, a "Submit" button is provided for the user to invoke the servlet program by using the GET method. In later examples, we will use multiple buttons to handle multiple actions.

The servlet program "WelcomeVBS.java" contains only the "doGet()" method. When a user clicks the "Submit" button on the form, an HTTP GET request will be sent to the web server. By reading the ACTION attribute of the HTML form, the web server knows that a servlet program is to be invoked so the request is passed to the servlet engine accordingly. The doGet() method of the corresponding servlet will be invoked as this is an HTTP GET request. Moreover, two object parameters, namely HttpServletRequest and HttpServletResponse will be created.

In general, data sent by the client, including the corresponding HTTP request information is provided by the HttpServletRequest object. As shown in later examples, this object can be used to generate dynamic web pages based on the client’s input data, which is important for e-commerce applications.