NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

Internet Programming and Reserving a Book on The Web Using ASP

Graduation Project
COM- 400

Student: Mohammed Jalal Mohammed
(980887)

Supervisor: Mr. Umit Ilhan

Nicosia - 2002

ACKNOWLEDGMENTS

I would like to thank Mr. Umit for letting me choose the project by my self and also searching for the idea and improve it. It allowed me to improve my self and do the progress of the project in a good manner. Also, I would like to thank the Chairman of Computer Engineering Department Assoc. Prof. Dr Dogan Ibrahim for taking good care of the department's students. Finally, its my pleasure to thank Mr. Tayseer Alshanableh for his support during all my years in the University.

I am pleased to mention in my acknowledgment my family and thank them for encouraging me to do it and perform in it as well as I can.

ABSTRACT

Inernet programming and web designing technology, nowadays, is a very important application. In this project, I will talk about this useful topic in general.

The project is divided into three main chapters. Each chapter includes specific topics that are related to Internet services and designing web pages. Also, including some definitions, examples and applications.

The Internet is a common word that almost every body knows it and uses it in most of his daily life needs. It has all the kinds of information and fun stuff that is available for every person in the world. Although, its easy to use and understandable to any one and some of the web sites on the Internet are available in different languages but most of them are in English, which is almost the language that every body knows.

Today, the Internet is a way to connect people with each other and make them communicate easily. The advantage of this that it doesn't cost lot money and it's cheaper than using the other communication tools such as telephones, faxes...etc.

As it is mentioned above, the project is divided into three different chapters, and each chapter has main topics and sub topics, which are defiantly related to each other in a way, and related to the main topic of the project, which is Web Designing with an Example of (Reserving a Book on The Web Using ASP).

TABLE OF CONTENTS

INTRODUCTION

This project main utility is to create a web page that has a communicating activity to a certain database stored in a specific server. Before designing and programming this important and powerful application, we had to take a tour over the Internet world and its programming tools and languages.

The project is divided into three main chapters including several topics such as Internet services, web pages programming tools, definitions, examples and applications.

The first chapter indicates an over view to World Wide Web (www) and defining the meaning of a web site. It also includes explanation of the web interface definition language (WIDL) with its benefits and the web services description language (WSDL). Finally, the last section of this chapter explains an important feature in transferring data to and from the Internet, which is the File Transfer Protocol (FTP).

The second chapter of the project defines the different Internet programming tools and languages that are used to design and program any web page. This includes HTML, Dynamic HTML (DHTML), Java and Java Scripts, Visual Basic Scripting (VB Script), Common Gateway Interface (CGI) and the most powerful tool, which is the Active Server Pages (ASP).

Finally, the last chapter of the project, which is the third one, indicated the main purpose of this project, which is reserving a book from library on the web using an Active Server Page (ASP). It includes creating the database with its articles that are needed to be stored in the library's server. Then the ASP code to create the active page with its description, and finally the layout of the page after applying the ASP program code.

The project is organized such as it goes in a step by step explaining manner until the desired purpose of this project is reached in the last chapter of the project.

Chapter 1

World Wide Web (www) Overview

1.1 Introduction

WWW " is shorthand for "World Wide Web."

Conceived in 1989 by Tim Berners-Lee, the Web is a way to use the Internet to share files from computers around the world.

No one "owns" the Web, and no one "runs" the Web - at least in the usual sense of that word. It is preserved from being chaotic because there is general agreement on a set of standards for Web pages. These standards have nothing to do with the content of the page, just how it will be transferred from one computer to another and how different computers will read it.

There are two fundamental types of software needed to make the Web a reality. One is the piece of software that is designed to serve Web files to the rest of the world. This software is called a Web "server." There are Web servers designed to run on all the popular operating systems, including Unix, Windows, and Macintosh.

The second piece of software is technically called a "client," but most people know it as a "browser." It is what you are using to read this message. Browsers first dealt with text only. In 1994 a browser called Mosaic generated an explosion in Web growth because it also read images and included a simple, graphical user interface. The most popular Web browser in 1996 is Netscape, but there are several others challenging it, including various versions of Mosaic.

In 1993-94 there was much talk about how we should build an "Information Superhighway." You don't hear that term too much any more, for as people were talking about it as something to be done in a more distant future, others knew it was already here. It was the Internet and its most exciting use was the Web.

The World Wide Web (WWW or Web) is defined by two characteristics:

1. It is a "hyperlinked" communications service that piggy-backs on top of the Internet's TCP/IP communications technology.

2. It is composed of millions of hyperlinked, graphical Web pages that may host a wide range of text, image, audio, and video media.

"Hyperlinks" are a way of actively linking text or graphical documents (or other kinds of files) that contained active communications links ("hyperlinks") to other documents or files (usually Web pages) on other computers (usually called Web "hosts" or "servers") across the Internet.

"Hypertext" documents (usually Web pages) on the WWW are files that contain active hyperlinks to other documents or files, which, in turn, may contain links to other documents, etc.

Clicking on a link, which may be text (usually blue and underlined) or an image, takes the user to another document.

- Specifically, it cause a request to be send to the computer hosting the other documents or Web pages. The request asks for the Web page and related files (such as images on the Web page) to be sent.
- The browser on the local computer then displays the Web page.

The "Hypertext Transfer Protocol" (http) is the communications protocol that makes this possible.

Http runs on top of the Internet's TCP/IP protocol and defines how different types of hyperlinked data (text and multimedia) are transmitted and accessed.

Internet communications equipment and wiring is equivalent to the letter carriers and trucks and airplanes that deliver the mail.

TCP/IP, the protocol defining the format for carrying information on the Internet, is the envelope.

Http, the protocol defining the format for information on the Web, is the letter.

Hyperlinked, graphical Web pages:

The millions of hyperlinked, graphical Web pages containing text, image. audio, and video media are displayed by browsers that have sufficient sophistication to handle the various media.

Graphical "hyperlinked" Web pages are created and displayed mostly through the use of the Hypertext Markup Language (HTML).

1.2 Website

A Web site (we prefer the two words rather than *Website*) is a collection of Web files on a particular subject that includes a beginning file called a home page. For example, most companies, organizations, or individuals that have Web sites have a single address that they give you. This is their home page address. From the home page, you can get to all the other pages on their site. For example, the Web site for IBM has the home page address of http://www.ibm.com. (In this case, the actual file name of the home page file doesn't have to be included because IBM has named this file *index.html* and told the server that this address really means http://www.ibm.com/index.html.)

Since it sounds like geography is involved, a Web site can be confused with a Web server. A server in this context is a computer that holds the files for one or more sites. A very large Web site may reside on a number of servers located in many different geographic places. IBM is a good example; its Web site consists of thousands of files spread out over many servers in worldwide locations. But a more typical example is probably the site you are looking at, whatis.com. We reside on a commercial space provider's server with a number of other sites that have nothing to do with Internet glossaries.

A synonym and less frequently used term for Web site is "Web presence." That term seems to better express the idea that a site is not tied to specific geographic location, but is "somewhere in cyberspace." However, "Web site" seems to be used much more frequently.

1.3 Web Interface Definition Language (WIDL)

The purpose of the Web Interface Definition Language (WIDL) is to enable automation of all interactions with HTML/XML documents and forms, providing a general method of representing request/response interactions over standard Web protocols, and allowing the Web to be utilized as a universal integration platform.

A central feature of WIDL is that programmatic interfaces can be defined and managed for data (HTML, XML or text files) and services (CGI-bin, database, or other back end systems) that are not under the direct control of programs that require such access. WIDL definitions can be co-located with client programs, centrally managed in client/server architecture, or referenced directly from HTML/XML documents.

WIDL definitions provide a mapping between Web resources and applications written in conventional programming languages such as C/C++, COBOL, Visual Basic, Java, JavaScript, etc., enabling automatic and structured Web access by compatible client programs, including mainstream business applications, desktop applications, applets, Web agents, and server-side Web programs (CGI, etc.).

Automatic means that complex interactions with Web servers do not require human intervention; programs can request Web data and services by making local calls to functions which encapsulate standard Web access protocols and utilize WIDL definitions to provide naming services, change management, error handling, condition processing and intelligent data binding.

Structured means that Web data and services are described as interfaces with well defined input and output variables.

Standard Web access protocols means HTTP and HTTPS.

Compatible means any program that both utilizes WIDL definitions to define the location of Web services and the structure of data that is returned by standard HTTP and HTTPS requests, and allows WIDL definitions to be managed locally, centrally, or by individual service providers.

WIDL describes business objects on the Web, providing the basis for a common API across Web servers, legacy systems, databases, and middleware infrastructures, and effectively transforming the Web from an access medium into an integration platform.

1.3.1 Benefits of WIDL

A major part of the value of an Interface Definition Language (IDL) is that it can define services offered by applications in an abstract but highly usable fashion. WIDL brings to the Web many of the features of IDL concepts that have been implemented in distributed computing and transaction processing platforms including DCE, and CORBA.

1- Business-to-Business Integration

WIDL makes it easy for organizations to automate business transactions with customers and suppliers. WIDL describes and automates interactions with services hosted by Web servers on intranets, extranets and the Internet; it transforms the Web into a standard integration platform and provides a universal API for all Web-enabled systems.

Using HTML, XML, HTTP and HTTPS as corporate standards glue, WIDL requires only that target systems be Web-enabled. There are hundreds of products in the market today which Web-enable existing systems, from mainframes to client/server applications. The use of standard Web technologies empowers various IT departments to make independent technology selections. This has the effect of lowering both the technical and 'political' barriers that have typically derailed cross-organizational integration projects.

A number of analysts have already warned that proprietary e-commerce platforms could lock suppliers into relationships by forcing them to integrate their systems with one infrastructure for business-to-business integration, making it costly for them to switch to or integrate with other partners who have selected alternate e-commerce platforms. Buyer-supplier integration issues involve many-to-many relationships, and demand a standard platform for functional integration and data exchange.

A service defined by WIDL is equivalent to a function call in standard programming languages. At the highest level, WIDL files describe the locations (URLs) of services, input parameters to be submitted (via Get or Post methods) to each service, conditions for successful processing, and output parameters to be returned by each service.

WIDL provides the following features:

- A browser is not required to drive Web applications.
- WIDL definitions are dynamically interpreted and can be centrally managed.
- Client applications are insulated from changes in service locations and data extraction methods.
- Developers are insulated from network programming concerns.
- Application resources can be integrated through firewalls and proxies.

WIDL can be used to describe interfaces and services for:

- Static documents (HTML, XML, and plain text files)
- Dynamically generated documents (HTML, XML, and plain text files)
- HTML forms
- URL directory structures

WIDL can be used:

- To automate interactions with Web servers
- For both on-demand and scheduled extraction of targeted Web data

- To aggregate data from a number of Web sources
- To chain services across multiple Web sites
- To rapidly integrate Web resources with traditional application development languages and environments

WIDL has the ability to specify conditions for successful processing, and error messages to be returned to calling programs. Conditions further enable services to be defined that span multiple documents.

2- Change Management

One of WIDL's most significant benefits is its ability to insulate client programs from changes in the format and location of Web documents. Unlike the way CORBA and DCE IDL are normally used, WIDL is interpreted at runtime; as a result, service URLs, object references in variables, definitions of document regions, success/failure conditions, and directives for service chaining can all be administered without requiring modification of client code. This usage model supports application-to-application linkages that are far more robust and maintainable than if they were coded by hand.

There are three models for WIDL management:

- Client side - where WIDL files are co-located with a client program
- Naming service - where WIDL definitions are centrally managed and referenced via directory services, i.e. LDAP
- Server side - where WIDL files are referenced by, co-located with, or embedded within Web documents.

WIDL does not require that existing Web resources be modified in any way. Flexible management models allow organizations to describe and integrate Web sites that are uncontrolled, as well as to provide their business partners with interfaces to services that are controlled. The ability to seamlessly migrate from independent to shared management eases the transition from informal to formal business-to-business integration.

3- Language Bindings

The primary purpose of WIDL is integration of Web resources with corporate business applications. In much the same way that DCE or CORBA IDL is used to generate code fragments, or 'stubs', to be included in application development projects, WIDL provides the structure necessary for generating client code in languages such as C/C++, Java, COBOL, and Visual Basic. Developers can thus be insulated from the need to understand both HTML/XML parsing and Web protocols. This capability enables the existing skills of innumerable programmers to be rapidly leveraged in the utilization of Web based resources.

1.4 Web Services Description Language (WSDL)

WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate, however, the only bindings described in this document describe how to use WSDL in conjunction with SOAP 1.1, HTTP GET/POST, and MIME.

As communications protocols and message formats are standardized in the web community, it becomes increasingly possible and important to be able to describe the communications in some structured way. WSDL addresses this need by defining an XML grammar for describing network services as collections of communication endpoints capable of exchanging messages. WSDL service definitions provide documentation for distributed systems and serve as a recipe for automating the details involved in applications communication.

A WSDL document defines **services** as collections of network endpoints, or **ports**. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: **messages**, which are abstract descriptions of the data being exchanged, and **port types**, which are abstract collections of **operations**. The concrete protocol and data format specifications for a particular port type constitutes a reusable **binding**. A port is defined by associating a network address with a reusable binding, and a collection of ports defines a service. Hence, a WSDL document uses the following elements in the definition of network services:

- Types– a container for data type definitions using some type system (such as XSD).
- Message– an abstract, typed definition of the data being communicated.
- Operation– an abstract description of an action supported by the service.
- Port Type–an abstract set of operations supported by one or more endpoints.
- Binding– a concrete protocol and data format specification for a particular port type.
- Port– a single endpoint defined as a combination of a binding and a network address.
- Service– a collection of related endpoints.

It is important to observe that WSDL does not introduce a new type definition language. WSDL recognizes the need for rich type systems for describing message formats, and supports the XML Schemas specification (XSD) as its canonical type system.

However, since it is unreasonable to expect a single type system grammar to be used to describe all message formats present and future, WSDL allows using other type definition languages via extensibility.

In addition, WSDL defines a common **binding** mechanism. This is used to attach a specific protocol or data format or structure to an abstract message, operation, or endpoint. It allows the reuse of abstract definitions.

In addition to the core service definition framework, this specification introduces specific binding extensions for the following protocols and message formats:

- SOAP 1.1
- HTTP GET / POST
- MIME

1.4.1 SOAP Binding

WSDL includes a binding for SOAP 1.1 endpoints, which supports the specification of the following protocol specific information:

- An indication that a binding is bound to the SOAP 1.1 protocol.
- A way of specifying an address for a SOAP endpoint.
- The URI for the SOAPAction HTTP header for the HTTP binding of SOAP.
- A list of definitions for Headers that are transmitted as part of the SOAP Envelope.

This binding grammar it is not an exhaustive specification since the set of SOAP bindings is evolving. Nothing precludes additional SOAP bindings to be derived from portions of this grammar. For example:

- SOAP bindings that do not employ a URI addressing scheme may substitute another addressing scheme by replacing the soap.
- SOAP bindings that do not require a SOAPAction.

1.4.2 HTTP GET & POST Binding

WSDL includes a binding for HTTP 1.1's GET and POST verbs in order to describe the interaction between a Web Browser and a web site. This allows applications other than Web Browsers to interact with the site. The following protocol specific information may be specified:

- An indication that a binding uses HTTP GET or POST.
- An address for the port.
- A relative address for each operation (relative to the base address defined by the port).

1.4.3 MIME Binding

WSDL includes a way to bind abstract types to concrete messages in some MIME format. Bindings for the following MIME types are defined:

- Multipart/related.
- Text/xml.
- Application/x-www-form-unrecorded (the format used to submit a form in HTML).
- Others (by specifying the MIME type string).

The set of defined MIME types is both large and evolving, so it is not a goal for WSDL to exhaustively define XML grammar for each MIME type. Nothing precludes additional grammar to be added to define additional MIME types as necessary. If a MIME type string is sufficient to describe the content, the mime element defined below can be used.

1.5 FTP (File Transfer Protocol)

FTP (File Transfer Protocol) is a method of copying files between two different locations. In order to use FTP you must have an FTP client on the computer you are using, and the computer you wish to access must have an FTP server.

You can obtain FTP servers to run on your own computer so that you can access files on your computer from other computers. You should not use these unless you are very comfortable with networking and computer setup since it may give other people access to your computer's files.

To use FTP you run the client and tell it the location of the FTP server you want to connect to. All Princeton FTP servers will prompt you for a user name and password. (This is the same as the user name and password you use for email.) The client connects to the server and displays a list of all the files and folders on the computer where the server is running. Once this connection is established you can copy files from the server to the client or vice versa. Most clients also give you the ability to do common file tasks like delete, rename, create new folders, etc.

FTP is useful because you can connect to an FTP server from any computer that has an FTP client. Thus, you could store all your files on one of the UNIX systems and use FTP to access those files from any computer on campus whenever you need them. If you create a document in a cluster you can FTP it to Arizona, then later FTP it from Arizona to your computer in your room. Since your files always stay on a hard drive, you avoid the danger of lost or damaged floppies. In addition, FTP has no inherent limit on the size of the files it can transfer; files too large for a floppy can easily be moved using FTP.

Chapter 2

TOOLS and LANGUAGES of INTERNET PROGRAMMING

2.1 HTML

HyperText Markup Language. This is the code that World Wide Web documents are written in. A Web browser interprets this code to decide where and how to display images and links in a Web document and how to layout the text. Web browsers are not standard, however, and some support certain HTML tags in different ways, or don't support them at all.

If you view the source of a Web document (this should be an option for your browser) you can see what an HTML document looks like. All HTML code is enclosed in the "<" and the ">" characters.

Short for ***HyperText Markup Language***, the authoring language used to create documents on the World Wide Web. HTML is similar to SGML, although it is not a strict subset.

Language is a system for communicating. Written languages use symbols (that is, characters) to build words. The entire set of words is the language's *vocabulary*. The ways in which the words can be meaningfully combined is defined by the language's syntax and *grammar*. The actual meaning of words and combinations of words is defined by the language's semantics. In computer science, human languages are known as natural languages. Unfortunately, computers are not sophisticated enough to understand natural languages. As a result, we must communicate with computers using special computer languages. There are many different classes of computer languages, including machine languages, programming languages, and fourth-generation languages.

Document in the PC world, a file created with a word processor. In addition to text, documents can contain graphics, charts, and other objects. Increasingly, the line separating word processing files from files produced by other applications is becoming blurred. A word processing application can

produce graphics and a graphics application can produce words. This trend is accelerating with new technologies such as OLE and OpenDoc that allow an application to combine many components. Consequently, the term document is used more and more to describe any file produced by an application. Interestingly, this is the way the term has always been used in Macintosh environments.

World Wide Web (www) is a system of Internet servers that support specially formatted documents. The documents are formatted in a script called HTML (*HyperText Markup Language*) that supports links to other documents, as well as graphics, audio, and video files. This means you can jump from one document to another simply by clicking on hot spots. Not all Internet servers are part of the World Wide Web. There are several applications called Web browsers that make it easy to access the World Wide Web; Two of the most popular being Netscape Navigator and Microsoft's Internet Explorer.

SGML is the Abbreviation of *Standard Generalized Markup Language,* a system for organizing and tagging elements of a document. SGML was developed and standardized by the International Organization for Standards (ISO) in 1986. SGML itself does not specify any particular formatting; rather, it specifies the rules for tagging elements. These tags can then be interpreted to format elements in different ways.

SGML is used widely to manage large documents that are subject to frequent revisions and need to be printed in different formats. Because it is a large and complex system, it is not yet widely used on personal computers. However, the growth of Internet, and especially the World Wide Web, is creating renewed interest in SGML because the World Wide Web uses HTML, which is one way of defining and interpreting tags according to SGML rules.

HTML defines the structure and layout of a Web document by using a variety of tags and attributes. The correct structure for an HTML document starts with <HTML><HEAD>(enter here what document is about)</HEAD><BODY> and ends with </BODY></HTML>. All the information you'd like to include in your Web page fits in between the <BODY> and </BODY> tags.

HTML is composed of *tags*. HTML tags are *always* enclosed in angle-brackets (< >) and are case-insensitive; that is, it doesn't matter whether you type them in upper or lower case. I almost always use upper case, because it makes the tags easier to pick out in a document, but that's just me. You can do whatever you like.

Tags typically occur in begin-end pairs. These pairs are in the form

<tag> ... </tag>

where the <tag> indicates the beginning of a tag-pair, and the </tag> indicates the end. (The three dots indicate an arbitrary amount of content between the tags.) The usual way to refer to each tag is "*tag*" for the first and "slash-*tag*" for the second, where *tag* is the actual name of the tag being discussed.

These pairs define *containers*. Any content within a container has the rules of that container applied to it. For example, the text within a "boldface container" would be boldfaced. Similarly, paragraphs are defined using a "paragraph container."

2.1.1 Document Tags

Document tags are the tags, which divide up a Web page into its basic sections, such as the header information and the part of the page, which contains the displayed text and graphics.

The first and last tags in a document should always be the HTML tags. These are the tags that tell a Web browser where the HTML in your document begins and ends. The absolute most basic of all possible Web documents is:

<HTML>

</HTML>

The HEAD tags contain all of the document's header information. Header does not mean what appears at the top of the browser window, but things like the document title and so on.

The title of the document is placed between the TITLE tags. This will appear at the top of the browser's title bar, and also appears in the history list. What is typed should probably be something, which indicates the document's contents, but it doesn't have to be. The length of the title is pretty much unlimited, but don't go overboard. Users will either sneer at or be confused by exceedingly long titles.

BODY comes after the HEAD structure. Between the BODY tags, all of the stuff that gets displayed in the browser window. All of the text, the graphics, and links, and so on -- these things occur between the BODY tags.

```
<HTML>
<HEAD>
 <TITLE>Document Title</TITLE>
</HEAD>

<BODY>

</BODY>
</HTML>
```

2.1.2 Basic Text Structures

Headings: The heading structures are most commonly used to set apart document or section titles. For example, the word "Headings" at the beginning of this section is a heading. There are six levels of headings, from Heading 1 through Heading 6. Heading 1 (H1) is "most important" and Heading 6 (H6) is "least important." By default, browsers will display the six heading levels in the same font, with the point size decreasing as the importance of the heading decreases. Here are all six HTML pairs, in descending order of importance:

```
<H1>Heading 1</H1>
<H2>Heading 2</H2>
<H3>Heading 3</H3>
<H4>Heading 4</H4>
<H5>Heading 5</H5>
<H6>Heading 6</H6>
```

These six lines, when placed into an HTML document, will simply display the six levels of headings.

Heading 1

Heading 2

Heading 3

Heading 4

Heading 5

Heading 6

Paragraphs: paragraphs are quite common in Web pages. They are one of the most basic structures in HTML. The overall structure is a page. The page is composed of a number of sections, each of which is composed of one or more paragraphs. Each paragraph is composed of words, and each word of letters.

Admittedly, this is a simplified way of looking at text, but it will do for our purposes. The furthest HTML goes down this progression is to the paragraph level. The beginning of a paragraph is marked by <P>, and the end by </P>.

Line Break: Line break is used when it is needed to end a line after a certain word but without starting a new paragraph, which is involved by using the
 tag. This forces a line break wherever you place it in the content (that is,

whatever is after the
 tag will start from the left margin of the next line on the screen.)

2.1.3 Anchors

The real point of the Web, of course, is that documents can be linked to each other, or to other types of files such as movies or sound clips, through the use of hyperlinks. These links allow authors to link documents together in intuitive ways, as opposed to traditional linear texts such as books, articles, or almost anything else printed.

The simplest possible anchor starts with <A> and ends with </A>. However, the <A> tag is never ever used by itself, because it doesn't do anything. It must be enhanced with attributes.

HREF: HREF stands for "**H**ypertext **Ref**erence" which is another way of saying, "The location of the file I want to load." Most anchors are in the form <A HREF="*URL*">, where *URL* is the location of the resource to which the link points to. The words between the open and close of the anchor would be displayed as a hyperlink.

Name: Using the NAME attribute, we can invisibly mark certain points of a document as places that can be jumped to directly, instead of loading the document and then scrolling around to find what is wanted. This is accomplished by using a *named anchor*, which is slightly different than the anchor used to create a hyperlink. Unlike HREF, the double-quotes in the NAME attribute are never optional (because of the # character).

2.1.4 Images

Besides hyperlinks, the other great advantage of the Web is the ability to integrate graphic images into a document. Some would argue that this represents one of the greatest strengths of the Web. Graphics are certainly used as heavily as hyperlinks, and represent most of the data, which is transferred.

Images are placed in Web documents using the IMG tag. This tag is empty, and therefore has no closing tag. The basic form of the image tag is <IMG>, but just like <A>, <IMG> by itself is pointless-- it will do nothing. At the very least, the browser should know where to find the image that it's supposed to place in the document.

This brings up an important point. Visually speaking, images are part of a Web document, but in reality an HTML file and any graphics it refers to are actually all separate files. In other words, one HTML file, which has five graphics within it, makes a total of six files required to make the page look right. These files are all stored on a Web server, but don't have to all been in the same exact place. (Often, server administrators will set up separate directories for pictures.)

In order to make the IMG tag work, an SRC attribute is used. SRC stands for "source," as in, "the source of this graphic." The value of SRC is the URL of the graphic to be displayed on the Web Page. Thus, a typical image tag will take the form:

<IMG SRC="*URL of graphic*">

ALT: The ALT attribute is used to define "alternate text" for an image. The value of ALT is author-defined text, enclosed in double-quotes, and ALT text can be any amount of plain text, long or short. To pick one of an infinite number of examples, a warning symbol could be marked up as follows:

<IMG SRC="warning.gif" ALT="Warning!!!">

2.2 Dynamic HTML (DHTML) and CSS

Dynamic HTML is a combination of client side scripting technology and HTML that work together to make HTML pages more interactive. For instance, with DHTML you can cause letters to changes colors when the mouse is placed over them and graphics can be animated.

Dynamic HTML (DHTML) is not any one specific technology (such as JavaScript or ActiveX). Nor is it a tag, a plug-in, or a browser. Dynamic HTML is one of the most exciting and useful things to happen to the Web in recent memory. It is a concept that has been enabled (to different extents in different browsers) by a number of technologies, including JavaScript, VBScript, the Document Object Model (DOM), layers, and Cascading style sheets (CSS).

Dynamic HTML is a powerful extension to HTML that allows for active WebPages like never before. Users now have the ability to drag-and-drop Internet objects, watch complicated animated presentations complete with sound, and have full interaction with websites. Actually, it is simply HTML that can change even after a page has been loaded into a browser. A paragraph could turn blue when the mouse moves over it, or a header could slide across the screen. Anything that can be done in HTML can be redone after the page is loaded.

These three parts make up Dynamic HTML (DHTML):

- **Client-side Scripting**

People have been using client-side scripting languages (JavaScript and VBScript in particular) to change HTML for a long time. If an image changes when you roll your mouse over it, you're looking at an example of dynamic HTML. The 4.0 browsers from both Microsoft and Netscape allow more of a page's HTML elements to be accessible from within scripting languages. The mechanism whereby page elements (or document objects) are opened to scripting languages is called the Document Object Model.

- **DOM (Document Object Model)**

In a sense, the Document Object Model is the real core of dynamic HTML. It makes HTML changeable. The DOM is the hierarchy of elements that are present in the browser at any given time. This includes environmental information such as the current date and time, browser properties such as the browser's version number, window properties such as window. Location (the page's

URL), and HTML elements such as <p> tags, divs, or tables. By exposing the DOM to scripting languages, browsers enable you to access these elements. While some elements such as the time of day can't be changed themselves, scripts to modify other elements can use them.

- **CSS (Cascading Style Sheets)**

DHTML allows content to be displayed with more design flexibility and accuracy through the use of Cascading Style Sheets (CSS). Using CSS, a standard from the World Wide Web Consortium (W3C), web authors can define fonts, margins, and line spacing for different parts of an HTML document. In addition to these stylistic improvements, CSS allows the absolute positioning of content by specifying x,y coordinates, and even a z-index, which allows different elements to overlap (see the Demo page: House Decorator example). The Microsoft implementation of DHTML also adds built-in multimedia and data objects (treated as properties of cascading style sheets) that can be controlled through scripting languages, allowing for stereo sound, on-the-fly image manipulation, and even access to server-side databases.

2.2.1 The Document Object Model

Document object models have been in browsers since the first client-side scripting language was introduced to the web in Netscape Navigator 2.0, but not all of the objects in the model were exposed to scripting. The document object model has evolved with the introduction of DHTML. Navigator 4 exposes many more objects to JavaScript, and IE4 exposes all of them (and not just to JavaScript, but to VBScript as well).

The document object model and event model combine to exercise great power over the page. For example, the text of a link can change in size and color when the mouse is moved over it, the contents of an article can expand in size when the reader selects it, and the phone number being typed into a form can be validated as each key is pressed.

2.2.1.1 Pixel Level Accuracy: Absolute Positioning Through CSS

CSS is a big part of DHTML, and absolute positioning of content is a big part of CSS. Absolute positioning means that content--text, images, and anything else that can be put in a page--can be placed at exact coordinates. Text can be overlapped, the background can be aligned with the foreground without frame and table tricks, and content can be organized within independent blocks instead of just within the constricting cells of a table.

With absolute positioning comes animation of content. Because DHTML exposes every part of the page (or most of it, in the case of Navigator 4) to scripting languages through the document object model, blocks of absolutely positioned content can be hidden, shown, and flown about the page with the help of a script. For example, a script could detect a user selecting a piece of content from a menu and highlight it. It could also move the content across the page to a more prominent position.

Navigator 4 also provides the <LAYER> tag for absolute positioning of content. IE 4, however, does not support the <LAYER> tag. Because both Navigator 4 and IE 4 support absolute positioning with CSS, and because the W3C rejected the <LAYER> tag as a standard, the preferred method of absolute positioning in both browsers is CSS.

2.2.1.2 Malleable Content: Dynamic Control of CSS Styles

With CSS styles, everything from line height to margin width to font face can be specified (and in the case of font face, if not found, degraded to the next best match). No longer are web pages limited to seven sizes of fonts; CSS allows exact point and pixel values. Internet Explorer supported CSS styles with version 3.0; Navigator adds support for them in version 4.0.

There's more to it than that though. IE 4's implementation of DHTML allows all aspects of a document's styles to be modified at any time during the

document's existence. A hypertext link could expand in size when the mouse is moved over it, items in a table of contents could light up when selected, and key words in a five thousand word article could turn bright red for easier access and more efficient browsing. Currently, Navigator 4 does not support controlling CSS styles with JavaScript.

2.2.1.3 Pages on the Fly: Dynamic Creation of Content

With absolute positioning, a document's content can be positioned and animated. With dynamic control of styles, the appearance of content can be changed at any time (IE 4 only). Now, with dynamic creation of content, content can be created and displayed without reloading or redrawing a document (either directly through the document object model in IE 4, or through hiding and showing layers in Navigator 4). Though the ability to create content "on-the-fly" existed before DHTML, it required that the entire document or frame be redrawn. With DHTML, scores in a DHTML game can be tabulated and displayed as they occur, and JavaScript clocks can tick away without being restricted to form elements.

2.2.1.4 Multimedia Medium: Microsoft's Multimedia Controls

Microsoft's implementation of DHTML comes with a host of built-in multimedia controls that enhance the visual and auditory aesthetics of any document. All of these cross-platform controls can be controlled through scripting languages, allowing them to interact with the user in interesting ways.

With the transition control, authors can make HTML documents enter or exit gracefully. The transition control comes with a plethora of preset transitions (for example, wipe, box in, and dissolve) that can be applied to the document or to any objects within the document and controlled through a script.

The animation control gives more accuracy to animations. Previously, web authors had to rely on animated GIFs (which cannot be controlled with scripting languages) to animate their pages. With the animation control, web authors can

control the frames per second, frame order, and various other properties of an animation and do so through scripts as well as through HTML. This means animations can respond to user interaction like any other object on the page.

In the audio arena, the mixer control can be used to create and play multi-channel stereo sounds. Sound files can be loaded into the mixer control, combined with other sound files, and then played on demand through scripts. In Navigator 4, on-demand scriptable audio can be played through the LiveAudio plug-in.

2.2.2 Usages and Some Examples

Here is an example, this one shows and hides the text.

```
function show() {
  eval(layerRef+'["Test"]'+styleRef+'visibility="visible"')
}
function hide() {
  eval(layerRef+'["Test"]'+styleRef+'visibility="hidden"')
}
```

The show function obviously shows the layer and the hide function hides it. As you can see I use the visibility property of the div object and set it to visible and hidden. You have maybe seen other ways to do this, but both 4.x browsers support this.

Now this script will look like:

```
<HTML>
<HEAD>
<TITLE>Hide-Show Text Demo</TITLE>
</HEAD>
<BODY>
<a href="javascript:hide('Test')">hide1</a> <a
```

```
href="javascript:show('Test')">show1</a>
<div id="Test" style="position:absolute; left:30;width:300; height:400; top:100">
```

This is the text that hides and shows, it is actually very easy to do. This is a simple example for how to use DHTML.

```
</div>
</BODY>
</HTML>
```

And then we add the complete script inside the <head> tag:

```
<script language="javascript">
<!--
//if explorer (4.x)
if (document.all) {
  layerRef='document.all'
  styleRef='.style.'
}
//else if netscape (4.x)
else if (document.layers) {
  layerRef='document.layers'
  styleRef='.'
}
else {
  location.href="not4x.html"
}
function show() {
  eval(layerRef+'["Test"]'+styleRef+'visibility="visible"')
}
function hide() {
  eval(layerRef+'["Test"]'+styleRef+'visibility="hidden"')
}
//--></script>
```

2.3 JAVA and JAVA SCRIPT

A scripting language developed by Netscape to enable Web authors to design interactive sites. Although it shares many of the features and structures of the full Java language, it was developed independently. Javascript can interact with HTML source code, enabling Web authors to spice up their sites with

dynamic content. JavaScript is endorsed by a number of software companies and is an open language that anyone can use without purchasing a license. It is supported by recent browsers from Netscape and Microsoft, though Internet Explorer supports only a subset, which Microsoft calls Jscript.

Script is another term for macro or batch file, a script is a list of commands that can be executed without user interaction. A script language is a simple programming language with which you can write scripts. Apple Computer uses the term script to refer to programs written in its HyperCard or Apple Script language.

Java is a high-level programming language developed by Sun Microsystems. Java was originally called OAK, and was designed for handheld devices and set-top boxes. Oak was unsuccessful so in 1995 Sun changed the name to Java and modified the language to take advantage of the burgeoning World Wide Web.

Java is an object-oriented language similar to C++, but simplified to eliminate language features that cause common programming errors. Java source code files (files with a .java extension) are compiled into a format called bytecode (files with a .class extension), which can then be executed by a Java interpreter. Compiled Java code can run on most computers because Java interpreters and runtime environments, known as Java Virtual Machines (VMs), exist for most operating systems, including UNIX, the Macintosh OS, and Windows. Bytecode can also be converted directly into machine language instructions by a just-in-time compiler (JIT).

Java is a general purpose programming language with a number of features that make the language well suited for use on the World Wide Web. Small Java applications are called Java applets and can be downloaded from a Web server and run on your computer by a Java-compatible Web browser, such as Netscape Navigator or Microsoft Internet Explorer.

JScript is Microsoft 's extended implementation of ECMAScript (ECMA262), an international standard based on the Netscape 's JavaScript and

Microsoft's JScript languages. JScript is implemented as a Windows Script engine. This means that it can be "plugged in" to any application that supports Windows Script, such as Internet Explorer, Active Server Pages, and Windows Script Host. It also means that any application supporting Windows Script can use multiple languages - JScript, VBScript, Perl, and others.

JScript (and the other languages) can be used for both simple tasks (such as mouseovers on Web pages) and for more complex tasks (such as updating a database with ASP or running logon scripts for Windows NT). Windows Script relies on external "object models" to carry out much of its work. For example, Internet Explorer's DOM provides objects such as 'document' and methods such as 'write()' to enable the scripting of Web pages.

There are several versions of JavaScript supported by certain browsers and browser versions. Unfortunately, this can often lead to confusion and incompatibilities. Since Netscape originally introduced JavaScript, JavaScript 1.0 was the language specification supported in Netscape Navigator 2.0. Subsequently, Navigator 3.0 supported new enhancements which comprised JavaScript 1.1. At present, Navigator 4.0 supports JavaScript 1.2.

In parallel, Microsoft attempted to support JavaScript 1.0 in their Internet Explorer 3.0 browser. Known as "Jscript," Microsoft's initial JavaScript support was unreliable and buggy. A push to standardize the language resulted in an "official" version of JavaScript sanctioned by the ECMA. Internet Explorer 4.0 includes robust support for the ECMA standardized JavaScript, which, although it shares much in common with Netscape's JavaScript 1.2, is not exactly equivalent.

While programming for any single version of JavaScript is relatively simple, writing code which functions across disparate versions, most notably Navigator 4 and MSIE 4, is one of the major challenges and topics of discussion in JavaScript programming at this time.

JavaScript code is typically embedded into an HTML document using the SCRIPT tag. You are free to embed as many scripts into a single document as you

like, using multiple SCRIPT tags. A script embedded in HTML with the SCRIPT tag uses the format:

```
<script language="JavaScript">
<!--
document.write("Hello World!");
//-->
</script>
```

The LANGUAGE attribute is optional, but recommended. You may specify that a section of code only be executed by browsers which support a particular version of JavaScript; for instance:

```
<script language="JavaScript1.2">
```

Another attribute of the SCRIPT tag, SRC, can be used to include an external file containing JavaScript code rather than code embedded into the HTML:

```
<script language="JavaScript" src="corefunctions.js">
</script>
```

The external file is simply a text file containing JavaScript code, and whose filename ends with the extension ".js". Note that although some version 3 browsers support the SRC attribute, it only functions reliably across platforms in the version 4 browsers.

Scripts can be placed inside comment fields to ensure that your JavaScript code is not displayed by old browsers that do not recognize JavaScript. The markup to begin a comment field is <!-- while you close a comment field using //-->. This practice is certainly optional, but considered good form when your page is likely to be visited by older browsers. Certainly, as older browsers fade away, this practice will likely become unnecessary.

JavaScript code, much like other programming languages, is made up of statements which serve to make assignments, compare values, and execute other sections of code. By and large, programmers will already be familiar with

JavaScript's usage of variables, operators, and statements. Below is a chart summarizing the main elements of JavaScript grammar.

Variables	Labels which refer to a changeable value. Example: *total* may be possess a value of 100.
Operators	Actors which can be used to calculate or compare values. Example: Two values may be summed using the addition operator (+); *total+tax* Example: Two values may be compared using the greater-than operator (>); *total>200*
Expressions	Any combination of variables, operators, and statements which evaluate to some result. In English parlance this might be termed a "sentence" or even a "phrase", in that grammatical elements are combined into a cogent meaning. Example: *total=100;* Example: *if (total>100)*
Statements	As in English, a statement pulls all grammatical elements together into a full thought. JavaScript statements may take the form of conditionals, loops, or object manipulations. It is good form to separate statements by semicolons, although this is only mandatory if multiple statements reside on the same line. Example: *if (total>100) {statements;} else {statements;}* Example: *while (clicks<10) {statements;}*
Objects	Containing constructs which possess a set of values, each value reflected into an individual *property* of that object. Objects are a critical concept and feature of JavaScript. A single object may contain many properties, each property which acts like a variable reflecting a certain value. JavaScript can reference a large number of "built-in" objects which refer to characteristics of a Web document. For instance, the document object contains properties which reflect the background color of the current document, its title, and many more. For a fuller explanation of the built-in objects of JavaScript, see the section on "Document Object Model".
Functions and Methods	A JavaScript function is quite similar to a "procedure" or "subroutine" in other programming languages. A function is a discrete set of statements which perform some action. It may accept incoming values (parameters), and it may return an outgoing value. A function is "called" from a JavaScript statement to perform its duty. A method is simply a function which is contained in an object. For instance, a function which closes the current window, named *close()*, is part of the window object; thus, *window.close()* is known as a method.

Chart 2.1 Main Elements of Java Script

Variables store and retrieve data, also known as "values". A variable can refer to a value which changes or is changed. Variables are referred to by name, although the name you give them must conform to certain rules. A JavaScript identifier, or name, must start with a letter or underscore ("_"); subsequent characters can also be digits (0-9). Because JavaScript is case sensitive, letters include the characters "A" through "Z" (uppercase) and the characters "a" through "z" (lowercase). Typically, variable names are chosen to be meaningful regarding the value they hold. For example, a good variable name for containing the total price of goods orders would be *total*.

2.3.1 Scope

When you assign a new variable to an initial value, you must consider the issue of **scope**. A variable may be scoped as either *global* or *local*. A global variable may be accessed from any JavaScript on the page. A local variable may only be accessed from within the function in which it was assigned.

Commonly, you create a new global variable by simply assigning it a value:

```
newVariable=5;
```

However, if you are coding within a function and you want to create a local variable which only scopes within that function you must declare the new variable using the **var** statement:

```
function newFunction()
{ var loop=1;
  total=0;
  ...additional statements...
}
```

In the example above, the variable *loop* will be local to *newFunction()*, while *total* will be global to the entire page.

2.3.2 Type

A value, the data assigned to a variable, may consist of any sort of data. However, JavaScript considers data to fall into several possible *types*. Depending

on the type of data, certain operations may or may not be able to be performed on the values. For example, you cannot arithmetically multiply two string values. Variables can be these types:

Numbers	3 or 7.987, Integer and floating-point numbers. • Integers can be positive, 0, or negative; Integers can be expressed in decimal (base 10), hexadecimal (base 16), and octal (base 8). A decimal integer literal consists of a sequence of digits without a leading 0 (zero). A leading 0 (zero) on an integer literal indicates it is in octal; a leading 0x (or 0X) indicates hexadecimal. Hexadecimal integers can include digits (0-9) and the letters a-f and A-F. Octal integers can include only the digits 0-7. • A floating-point number can contain either a decimal point, an "e" (uppercase or lowercase), which is used to represent "ten to the power of" in scientific notation, or both. The exponent part is an "e" or "E" followed by an integer, which can be signed (preceded by "+" or "-"). A floating-point literal must have at least one digit and either a decimal point or "e" (or "E").
Booleans	True or False. The possible Boolean values are true and false. These are special values, and are not usable as 1 and 0. In a comparison, any expression that evaluates to 0 is taken to be false, and any statement that evaluates to a number other than 0 is taken to be true.
Strings	"Hello World !" Strings are delineated by single or double quotation marks. (Use single quotes to type strings that contain quotation marks.)
Objects	myObj = new Object();
Null	Not the same as zero - no value at all. A null value is one that has no value and means nothing.
Undefined	A value that is undefined is a value held by a variable after it has been created, but before a value has been assigned to it.

Chart 2.2 Types of Variables

That said, JavaScript is a loosely typed language -- you do not have to specify the data type of a variable when you declare it, and data types are converted automatically as needed during script execution. By and large, you may simply assign any type of data to any variable. The only time data-typing matters

is when you need to perform operations on the data. Certain operators behave differently depending on the type of data being deal with. For example, consider the + operator:

"5" + "10"	***yields***	"510" *(string concatenation)*
5 + 10	***yields***	15 *(arithmetic sum)*

Operators take one or more variables or values (*operands*) and return a new value; e.g. the '+' operator can add two numbers to produce a third. You use operators in expressions to relate values, whether to perform arithmetic or compare quantities. Operators are divided into several classes depending on the relation they perform.

2.3.3 Arithmetic or computational

Arithmetic operators take numerical values (either literals or variables) as their operands and return a single numerical value. The standard arithmetic operators are:

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus: the remainder after division; e.g. 10 % 3 yields 1.
++	Unary increment: this operator only takes one operand. The operand's value is increased by 1. The value returned depends on whether the ++ operator is placed before or after the operand; e.g. ++*x* will return the value of *x* following the increment whereas *x*++ will return the value of *x* prior to the increment.
--	Unary decrement: this operator only takes one operand. The operand's value is decreased by 1. The value returned depends on whether the -- operator is placed before or after the operand; e.g. --*x* will return the value of *x* following the decrement whereas *x*-- will

return the value of *x* prior to the decrement.

- Unary negation: returns the negation of operand.

Chart 2.3 Arithmetic Operations

2.3.4 Comparison

A comparison operator compares its operands and returns a logical value based on whether the comparison is true or not. The operands can be numerical or string values. When used on string values, the comparisons are based on the standard lexicographical (alphabetic) ordering.

==	"Equal to" returns true if operands are equal.
!=	"Not equal to" returns true if operands are not equal.
>	"Greater than" returns true if left operand is greater than right operand.
>=	"Greater than or equal to" returns true if left operand is greater than or equal to right operand.
<	*"Less than" returns true if left operand is less than right operand.*
<=	"Less than or equal to" returns true if left operand is less than or equal to right operand.

Chart 2.4 Comparison

2.3.5 Boolean

Boolean operators are typically used to combine multiple comparisons into a conditional expression. For example, you might want to test whether (total>100) AND (stateTax=true). A boolean operator takes two operands, each of which is a true or false value, and returns a true or false result.

&&	"And" returns true if both operands are true.

||	"Or" returns true if either operand is true.
!	"Not" returns true if the negation of the operand is true (e.g. the operand is false).

Chart 2.5 Boolean

2.3.6 String

Strings can be compared using the comparison operators. Additionally, you can concatenate strings using the + operator.

"dog" + "bert" ***yields*** "dogbert"

2.3.7 Assignment

The assignment operator (=) lets you assign a value to a variable. You can assign any value to a variable, including another variable (whose value will be assigned). Several shorthand assignment operators allow you to perform an operation and assign its result to a variable in one step.

=	Assigns the value of the righthand operand to the variable on the left. Example: total=100; Example: total=(price+tax+shipping)
+= (also -=, *=, /=)	Adds the value of the righthand operand to the lefthand variable and stores the result in the lefthand variable. Example: total+=shipping (adds value of *shipping* to *total* and assigned result to *total*)
&= (also |=)	Assigns result of (lefthand operand && righthand operand) to lefthand operand.

Chart 2.6 Assignment

2.3.8 Special

Several JavaScript operators, rarely used, fall into no particular category. These operators are summarized below.

Conditional operator **(condition) ? trueVal : falseVal**	Assigns a specified value to a variable if a condition is true, otherwise assigns an alternate value if condition is false. Example: preferredPet = (cats > dogs) ? "felines" : "canines" If (cats>dogs), *preferredPet* will be assigned the string value "felines," otherwise it will be assigned "canines".
typeof *operand*	Returns the data type of *operand*. Example -- test a variable to determine if it contains a number: if (typeof total=="number") ...

Chart 2.7 Special Operators

2.3.9 Statements

Statements define the flow of a script, known as "program flow." A statement, like a fully grammatical English sentence, is made up of smaller expressions, which, altogether, evaluate into a cogent meaning. In JavaScript, statements are organized as conditionals, loops, object manipulations, and comments.

Good practice suggests that each JavaScript statements should be terminated with a semicolon (;). This is often not strictly necessary, as a new line also serves to separate statements, but when multiple statements reside on the same line the semicolon delimiter is mandatory.

A set of statements that is surrounded by braces is called a block. Blocks of statements are used, for example, in functions and conditionals.

Normally statements are executed sequentially: $x = 1; y = 2; z = x + y;$ but this can be altered by some statements which test a condition and branch or loop according to the result.

2.3.10 Conditionals

Conditional statements direct program flow in specified directions depending upon the outcomes of specified conditions. These tests are a major influence on the order of execution in a program.

2.3.10.1 If...else

As seen in many programming languages, ***if*** the condition evaluates to true then the block of *statements1* is executed. Optionally, an ***else*** clause specifies a block of *statements2*, which are executed otherwise. You may omit the else clause if there are no statements, which need to be executed if the condition is false.

```
if (condition)
 { statements1; }

else
 { statements2; }
```

2.3.10.2 Switch (Netscape & MSIE 4)

Commonly known as a "case statement," ***switch*** matches an expression with a specified case, and executes the statements defined for that case. In essence, the switch statement is a sort of shorthand for combining many implied *if* statements together.

```
switch (expression){
  case label :
    statement;
    break;
  case label :
    statement;
```

```
    break;
  ...
  default : statement;
}
```

For example, imagine that you wanted to execute different sets of statements depending on whether *favoritePet* was "dog," "cat," or "iguana." Note that the *break;* statement prevents any cases below the match from being executed. The *default* case is matched if none of the cases match the expression.

```
switch (favoritePet){
  case "dog" :
    statements;
    break;
  case "cat" :
    statements;
    break;
  case "iguana" :
    statements;
    break;
  default : statements;
}
```

2.3.11 Loops

2.3.11.1 For

The venerable ***for*** loop repeatedly cycles through a block of statements until a test condition is false. Typically, the number of times a loop is repeated depends on a counter. The JavaScript ***for*** syntax incorporates the counter and its increments:

```
for (initial-statement; test; increment)
 { statements; }
```

The initial-statement is executed first, and once only. Commonly, this statement is used to initialize a counter variable. Then the test is applied and if it succeeds then the statements are executed. The increment is applied to the counter

variable and then the loop starts again. For instance, consider a loop, which executes 10 times:

```
for (i=0; i<10; i++)
 { statements; }
```

2.3.11.2 Do...while (Netscape & MSIE 4)

Another loop, a ***do...while*** statement executes a block of statements repeatedly until a condition becomes false. Due to its structure, this loop necessarily executes the statement at least once.

```
do
 { statements;}
while (condition)
```

2.3.11.3 While

In similar fashion as the ***do...while*** statement, the ***while*** statement executes its statement block as long as the condition is true. The main difference between ***while*** and ***do...while***, aside from the fact that only ***while*** is supported in all JavaScript versions, is that a while loop may not execute the statements even once if the condition is initially false.

```
while (condition)
 { statements; }
```

2.3.11.4 Break and Continue

Both of these statements may be used to "jump the tracks" of an iterating loop. When used within the statement block of a loop, each statement behaves slightly differently:

break	Aborts execution of the loop, drops out of loop to the next statement following the loop.
continue	Aborts *this single* iteration of the loop, returns execution to the loop control, meaning the condition specified by the loop statement. Loop may execute again if condition is still true.

2.3.12 Comments

Despite the fact that comments are purely optional, they can easily be a crucial part of your program. Comments can explain the action, like a color commentary, which can be a great help in understanding the code. Whether as a teaching tool or to simply remind yourself what the code does, comments are best sprinkled liberally throughout a program. Remember, comments are for humans, so write them that way.

Comments can also be used for debugging -- you can comment "out" sections of code to prevent them from being executed. In doing so you may learn more about why a certain problem is occurring in your program.

Because JavaScript must ignore comments, there is an appropriate syntax for demarcating text as a comment. For single line comments, simply precede the line with two backslashes. For multi-line comment blocks, begin the comment with /* and close with */.

```
//A lonely ol' single line comment
/* A dense thicket of commentary, spanning many captivating lines
of explanation and intrigue. */
```

2.3.13 Functions

A function groups together a set of statements under a named subroutine. This allows you to conveniently "call" the function whenever its action is required. Functions are a fundamental building block of most JavaScript programs, so you'll become quite familiar with their use. Before you can call on a function, of course, you must first create it. We can break down the use of functions, then, into two logical categories: defining functions and calling functions.

2.3.13.1 Defining Functions

The function definition is a statement, which describes the function: its name, any values (known as "arguments"), which it accepts incoming, and the statements of which the function is comprised.

```
function funcName(argument1,argument2,etc)
{ statements; }
```

A function doesn't necessarily require arguments, in which case you need only write out the parenthesis; e.g. funcName(). If you do specify arguments, those arguments will be variables within the function body (the statements which make up the function). The initial values of those variables will be any values passed on by the function call.

Generally it's best to define the functions for a page in the HEAD portion of a document. Since the HEAD is loaded first, this guarantees that functions are loaded before the user has a chance to do anything that might call a function. Alternately, some programmers place all of their functions into a separate file, and include them in a page using the SRC attribute of the SCRIPT tag. Either way, the key is to load the function definitions before any code is executed.

Consider, for example, a simple function, which outputs an argument to the Web page, as a bold and blinking message:

```
function boldblink(message)
{ document.write("<blink><strong>"+message+"</strong></blink>"); }
```

Some functions may return a value to the calling expression. The following function accepts two arguments, x and y, and returns the result of x raised to the y power:

```
function raiseP(x,y)
{ total=1;
 for (j=0; j<y; j++)
  { total*=x; }
 return total; //result of x raised to y power
}
```

2.3.13.2 Calling Functions

A function waits in the wings until it is called onto the stage. You call a function simply by specifying its name followed by a parenthetical list of arguments, if any:

```
clearPage();
boldblink("Call me gaudy!");
```

Functions, which return a result, should be called from within an expression:

```
total=raiseP(2,8);
if (raiseP(tax,2)<100) ...
```

An object is a "package" of data; a collection of properties (variables) and methods (functions) all classed under a single name. For example, imagine that there was an object named *car*. We could say that the *car* object possesses several properties: make, model, year, and color, for example. We might even say that car possesses some methods: go(), stop(), and reverse(). Although *car* is obviously fictional, you can see that its properties and methods all relate to a common theme.

In JavaScript you may create your own objects for storing data. More commonly, though, you will use the many "built-in" objects, which allow you to work with, manipulate, and access the Web page and Web browser. This set of pre-existing objects is known as the "Document Object Model".

2.3.14 Document Object Model

Often referred to as the *DOM*, this object model is a hierarchy of all objects "built in" to JavaScript. Most of these objects are directly related to characteristics of the Web page or browser. The reason we qualify the term "built in" is because the DOM is technically separate from JavaScript itself. That is, the JavaScript language specification, standardized by the ECMA, does not actually specify the nature or specifics of the DOM. Consequently, Netscape and Microsoft have developed their own individual DOM's, which are not entirely compatible. Additionally, the DOM stands apart from JavaScript because other scripting languages could theoretically access it as well.

In summary, then, what we refer to as "JavaScript" is actually made up of JavaScript, the language, and the DOM, or object model, which JavaScript can access. In a future WDVL article we will take a closer look at the DOM and its current and future role.

Below is a graphical chart (Figure 2.1) illustrating a high-level view of Netscape's DOM. Microsoft's DOM is actually a superset of Netscape's, and so the chart below actually represents a subset of Microsoft's own DOM.

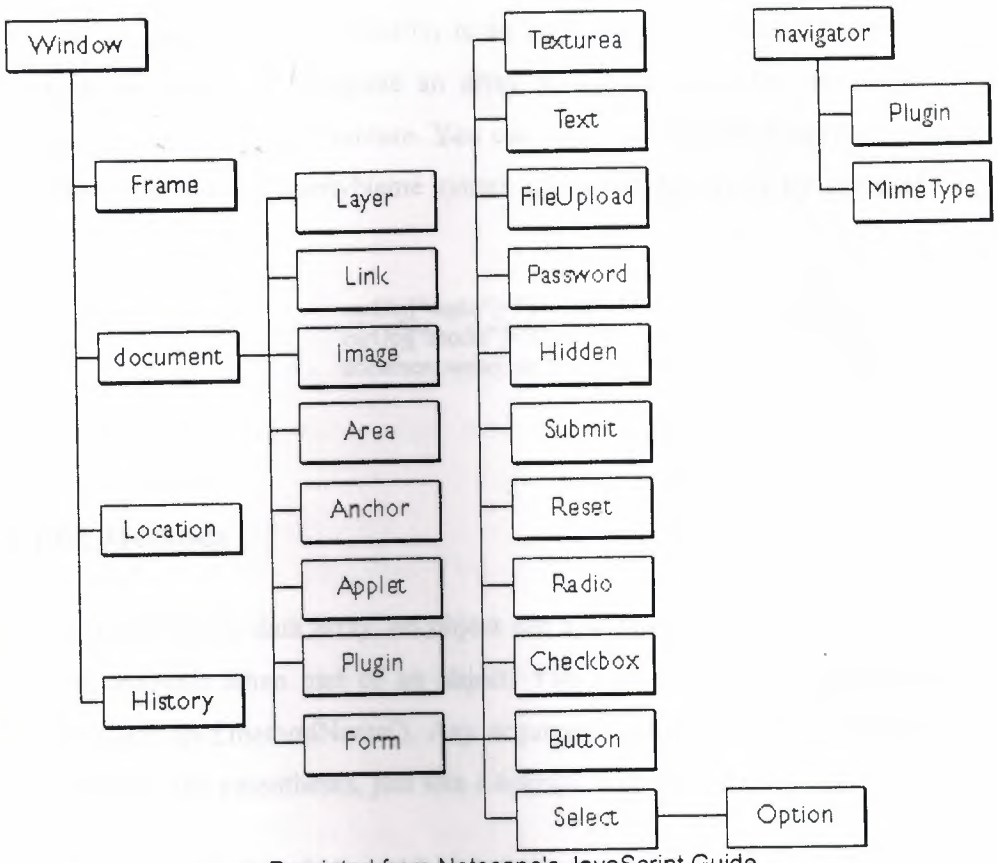

Figure 2.1 High Level View

2.3.14.1 Properties

Access the properties of an object with a simple notation: objectName.propertyName. Both the object name and property name are case sensitive, so watch your typing. Because a property is essentially a variable, you can create new properties by simply assigning it a value. Assuming, for instance, that carObj already exists (we'll learn to create a new object shortly), you can give it properties named make, model, and year as follows:

```
carObj.make="Toyota";
carObj.model="Camry";
carObj.year=1990;
document.write(carObj.year);
```

A JavaScript object, basically, is an array. If you're familiar with other languages you probably recognize an array as a collection of values residing within a single named data structure. You can access an object's properties either using the objectName.propertyName syntax illustrated above, or by using array syntax:

```
carObj["make"]="Toyota";
carObj["model"]="Camry";
document.write(carObj["year"]);
```

2.3.14.2 Methods

Unlike a basic data array, an object can also contain functions, which are known as *methods* when part of an object. You call a method using the basic syntax: objectName.methodName(). Any arguments required for the method are passed between the parentheses, just like a normal function call.

For example, the window object possesses a method named *close()*, which simply closes the specified browser window:

```
window.close();
```

2.3.15 Summary

It can be said that a JavaScript program is a series of statements, which work together towards a particular goal, and are made up of component grammatical elements, such as expressions and operators. Because JavaScript is a programming language invented "for the Web," it is oriented towards the specific needs of Web developers. The set of pre-built objects largely reflect characteristics of the Web page, allowing your JavaScript program to manipulate, modify, and react to the Web page.

Interactivity is the driving force behind JavaScript, and so most JavaScript programs are launched by actions, which occur on the Web page, often by the

user. In doing so, JavaScript's purpose is to nudge Web pages away from static displays of data towards applications, which can process and react.

2.4 VB SCRIPT

VBScript, Microsoft's Visual Basic Scripting Edition, is a scaled down version of Visual Basic. While it doesn't offer the functionality of Visual Basic, it does provide a powerful, easy to learn tool that can be used to add interaction to the web pages.

VBScript brings professional programming techniques to HTML web documents. With VBScript, documents and applications can be created, which are previously could only have been made available as a desktop program written with something like Visual Basic. It gives the ability to interact with and manipulate HTML documents directly from the browser. With VBScript, even interacting with and manipulating the browser itself can be done, sending it instructions from the VBScript program, and pulling in its variables for own use.

Specifically, VBScript was created by Microsoft to use either as a client-side scripting language for the Microsoft Internet Explorer (versions 3.0 and later) or as a server-side scripting language with the Microsoft Internet Information Server (versions 3.0 and later). A primary advantage for using the server-side approach is that the VBScript is processed by the server before it is transmitted to the client. Therefore, the client only receives an HTML page and we do not have to concern ourselves as to whether the browser can interpret the VBScript. In contrast, by using the client-side approach, purposely, the workload is transferred to the browser in order to reduce the workload of the server. Unfortunately, older or non-Microsoft browsers may not be able to correctly interpret and display the transmitted file. In addition to this, the source code is exposed to the browser user. On the brighter side, a client-side program can produce a more-responsive application, since user input can be processed on the client machine, and not sent back to the server for processing.

The true importance of VBScript is that it is the default language of Active Server Pages (ASP).

ASP is an exciting technology from Microsoft that is of significant value to developers. ASP extends standard HTML by adding built-in objects and server-side scripting, and by allowing access to databases and other server-side ActiveX components. This will be explained later in this chapter.

For many Web-application developers, VBScript may very well be the most important programming language.

VBScript Version 5.0 was released in 1999. Certainly, the most important new feature of Version 5.0 is the ability to use the **Class** statement to create your own class objects. Other new features of interest include the **Timer** function, the **With** statement, and regular expression searching using the **RegExp** and **Match** objects.

Recently, Microsoft renamed VBScript Version 5.0 to Version 5.5 to signify that it is part of the Windows Script Version 5.5 package.

Figure 2.2 shows a web page that displays the date on which this particular HTML file was last modified, and also shows the referrer of this page, or the URL of the document whose hyperlink was followed to reach this page. It is impossible to display either of these items of information with standard HTML (unless, of course, you "hardcode" them into your HTML document, in which case you're probably not really displaying the date the file was last modified or the hyperlink by which it was reached). Instead, HTML intermixed with VBScript was used to produce this page.

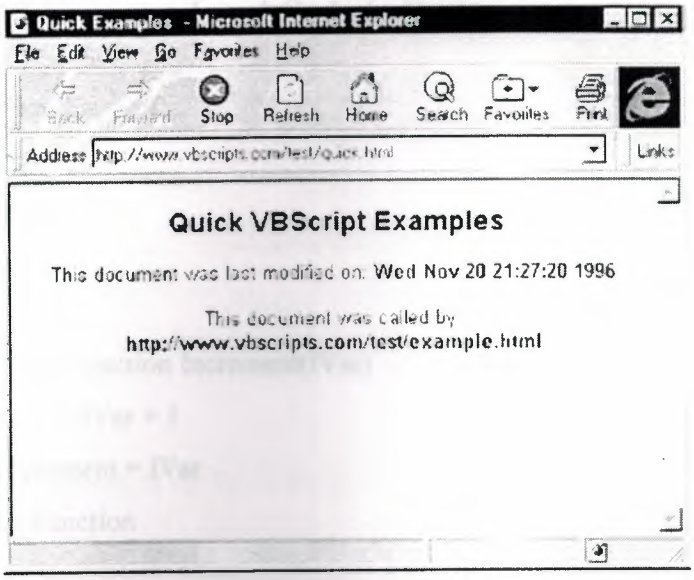

Figure 2.2 Diplaying Date

Some of the main uses of VBScript are:

- Reference and manipulate document objects
- Reference and manipulate the browser
- Reference the contents of another loaded document or documents
- Create a document "on the fly" from the browser
- Store, reference, and manipulate data input by the user
- Store, reference, and manipulate data downloaded from the server
- Perform calculations on data
- Display messages to the user
- Access cookies easily
- Reference and manipulate a wide range of "add-on" components, both ActiveX controls and Java applets
- Display two-dimensional HTML

In order to write VBScript, you have to know how to structure your code so that your scripts and programs execute properly. Each of the different types of VBScript that you write has different rules regarding its structure.

2.4.1 EXAMPLE: Script-Level Code in WSH

```
Option Explicit

Dim x
x = 10

Private Function Increment(lVar)
  lVar = lVar + 1
  Increment = lVar
End Function

Private Function Decrement(lVar)
  lVar = lVar - 1
  Decrement = lVar
End Function

Dim sMsg
sMsg = "The current value of x is " & x & vbCrLf

Dim y
y = 20
If x = 0 Then x = 10

sMsg = sMsg & "Value returned by Increment: " & Increment(x) &
vbCrLf
sMsg = sMsg & "Value returned by Increment: " & Increment(x) &
vbCrLf
sMsg = sMsg & "Value returned by Decrement: " & Decrement(x)
& vbCrLf
sMsg = sMsg & "The value of x is now " & x & vbCrLf
sMsg = sMsg & "The value of y is " & y & vbCrLf

MsgBox sMsg
```

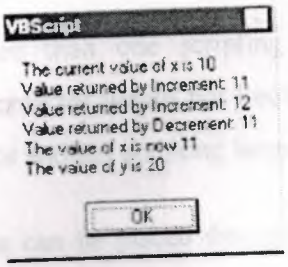

Figure 2.3 Example

2.4.2 Adding VBScript to Web Pages

Scripting languages, like JavaScript and VBScript, are designed as an extension to HTML. The web browser receives scripts along with the rest of the web document. It is the browser's responsibility to parse and process the scripts. HTML was extended to include a tag that is used to incorporate scripts into HTML-the *<SCRIPT>* tag.

2.4.2.1 The <SCRIPT> Tag

Scripts are added into web pages within a pair of *<SCRIPT>* tags. The *<SCRIPT>* tag signifies the start of the script section, while *</SCRIPT>* marks the end. An example of this is shown below:

```
<HTML>

<HEAD>

<TITLE>Working With VBScript</TITLE>

<SCRIPT LANGUAGE="VBScript">

 MsgBox "Welcome to my Web page!"

</SCRIPT>
```

The beginning *<SCRIPT>* tag includes a *LANGUAGE* argument that indicates the scripting language that will be used. The *LANGUAGE* argument is

required because there is more than one scripting language. Without the *LANGUAGE* argument, a web browser would not know if the text between the tags were JavaScript, VBScript or another scripting language.

While technically scripts can be placed throughout an HTML document using pairs of <*SCRIPT*> tags, typically scripts are often found at either the top or bottom of a Web document. This provides for easy reference and maintenance.

Although, functions and procedures (or subroutines) are central to modern programming. Dividing the script into subroutines helps to maintain and write programs by segregating related code into smaller, manageable sections. It also helps to reduce the number of lines of code to write by allowing reusing the same subroutine or function many times in different situations and from different parts of the program.

2.4.3 Defining Subroutines: The Sub . . . End Sub Construct

The Sub...End Sub construct is used to define a *subroutine*; that is, a procedure that performs some operation but does not return a value to its calling program. Blocks of code defined as subroutines with the Sub...End Sub construct can be called in two ways:

1- Automatically

Some subroutines provide the means by which an object interfaces with the script. For instance, when a class defined with the Class...End Class construct is initialized, its Initialize event, if one has been defined, is executed automatically. For subroutines of this type, the routine's name can be constructed in only one way, as follows:

Sub *objectname_event*

For example, *Sub Class_Initialize* is a valid name of a subroutine. This type of subroutine is known as an *event handler* or an *event procedure*.

2- Referring to it by name

A subroutine can be executed at any time by referring to it by name in another part of the script. While it is possible to execute event procedures in this way, this method is most commonly used to execute *custom subroutines*. Custom subroutines are constructed to perform particular tasks within a program, and can be assigned virtually any name. They allow you to place code that's commonly used or that is shared by more than one part of a program in a single place and no need to duplicate the same code through the application.

2.4.3.1 Subroutine Names

There are several very straightforward rules to remember when giving names to subroutines:

- The name can contain any alphabetical or numeric characters and the underscore character.
- The name cannot start with a numeric character.
- The name cannot contain any spaces. Use the underscore character to separate words to make them easier to read.

For example:

```
Sub 123MySub( ) ' Illegal
Sub My Sub Routine( ) ' Illegal
```

both contain illegal subroutine names. However:

```
Sub MySub123( ) ' Legal
Sub MySubRoutine( ) ' Legal
```

are legal subroutine names.

2.4.3.1 Example : Using a Custom Subroutine to Share Code

```
Sub cmdButton1_OnClick
   Call ShowAlertBox(cmdButton1.Value)
End Sub

Sub cmdButton2_OnClick
   ShowAlertBox cmdButton2.Value
End Sub

Sub cmdButton3_OnClick
   ShowAlertBox cmdButton3.Value
End Sub

Sub ShowAlertBox(strButtonValue)
   dim strMessage
   strMessage = "This is to let you know" & vbCrLf
   strMessage = strMessage & "you just pressed the button" & vbCrLf
   strMessage = strMessage & "marked " & strButtonValue
   Alert strMessage
End Sub
```

2.5 Common Gateway Interface (CGI)

Common Gateway Interface is a method to allow programmers to write programs that send and received data from websites. CGI is commonly used with HTML forms that allow users to submit data via the form. One problem with CGI is that it must start a new process for every user request, this can quickly slow down busy servers, and consequently CGI is not considered a very scalable solution. However, CGI can be implemented on a larger range of server software.

The Common Gateway Interface (CGI) specification lets Web servers execute other programs and incorporate their output into the text, graphics, and

audio sent to a Web browser. The server and the CGI program work together to enhance and customize the World Wide Web's capabilities.

By providing a standard interface, the CGI specification lets developers use a wide variety of programming tools. CGI programs work the magic behind processing forms, looking up records in a database, sending e-mail, building on-the-fly page counters, and dozens of other activities. Without CGI, a Web server can offer only static documents and links to other pages or servers. With CGI, the Web comes alive-it becomes interactive, informative, and useful. CGI can also be a lot of fun!

2.5.1 CGI Beyond the World Wide Web and HTML

Browsers and Web servers communicate by using the Hypertext Transfer Protocol (HTTP). Tim Berners-Lee at CERN developed the World Wide Web using HTTP and one other incredibly useful concept: the Universal Resource Locator (URL). The URL is an addressing scheme that lets browsers know where to go, how to get there, and what to do after they reach the destination. Technically, a URL is a form of Universal Resource Identifier (URI) used to access an object using existing Internet protocols. Because this book deals only with existing protocols, all URLs will be called URLs, not worrying about the technical hair-splitting. URLs are defined by RFC 1630.

Fortunately, most browsers keep a local copy, called a *cache*, of recently accessed documents. When the browser notices that it's about to re-fetch something already in the cache, it just supplies the information from the cache rather than contact the server again. This alleviates a great deal of network traffic

Your Web browser doesn't know much about the documents it asks for. It just submits the URL and finds out what it's getting when the answer comes back. The server supplies certain codes, using the Multipurpose Internet Mail Extensions (MIME) specifications, to tell the browser what's what. This is how your browser knows to display a graphic but save a .ZIP file to disk. Most Web documents are Hypertext Markup Language (HTML): just plain text with

embedded instructions for formatting and displaying. By itself, the server is only smart enough to send documents and to tell the browser what kind of documents they are. But the server also knows one key thing: How to launch other programs. When a server sees that a URL points to a file, it sends back the contents of that file. When the URL points to a program, however, the server fires up the program. The server then sends back the program's output as if it were a file.

Well, for one thing, a CGI program can read and write data files (a Web server can only read them) and produce different results each time it runs. This is how page counters work. Each time the page counter is called, it hunts up the previous count from information stored on the server, increments it by one, and creates a .GIF or .JPG on the fly as its output. The server sends the graphics data back to the browser just as if it were a real file living somewhere on the server.

2.5.2 How CGI Works

A CGI program isn't anything special by itself. That is, it doesn't do magic tricks or require a genius to create it. In fact, most CGI programs are fairly simple things, written in C or Perl (two popular programming languages).

CGI programs are often called *scripts* because the first CGI programs were written using UNIX shell scripts (bash or sh) and Perl. Perl is an interpreted language, somewhat like a DOS batch file, but much more powerful. When a Perl program is executed, the Perl instructions are interpreted and compiled into machine instructions right then. In this sense, a Perl program is a script for the interpreter to follow, much as Shakespeare's *Hamlet* is a script for actors to follow.

Other languages, like C, are compiled ahead of time, and the resultant executable isn't normally called a script. Compiled programs usually run faster but often are more complicated to program and certainly harder to modify.

In the CGI world, however, interpreted and compiled programs are called *scripts*.

Before the server launches the script, it prepares a number of *environment variables* representing the current state of the server, who is asking for the information, and so on. The environment variables given to a script are exactly like normal environment variables, except that they can't set them from the command line. They're created on the fly and last only until that particular script is finished. Each script gets its own unique set of variables. In fact, a busy server often has many scripts executing at once, each with its own environment.

Also, depending on how the script is invoked, the server may pass information another way, too. Although each server handles things a little differently, and although Windows servers often have other methods available, the CGI specification calls for the server to use STDOUT (Standard Output) to pass information to the script.

2.5.2.1 Standard Input and Output

STDIN and *STDOUT* are mnemonics for *Standard Input* and *Standard Output*, two predefined stream/file handles. Each process inherits these two handles already open. Command-line programs that write to the screen usually do so by writing to STDOUT. If the input to a program is redirected, really STDIN is been redirected. If the output of a program is redirected, really STDOUT is redirected. This mechanism is what allows pipes to work.

For Web servers, STDOUT is the feed leading to the script's STDIN. The script's STDOUT feeds back to the server's STDIN, making a complete route. From the script's point of view, STDIN is what comes from the server, and STDOUT is where it writes its output. Beyond that, the script doesn't need to worry about what's being redirected where. The server uses its STDOUT when invoking a CGI program with the POST method. For the GET method, the server doesn't use STDOUT. In both cases, however, the server expects the CGI script to return its information via the script's STDOUT.

This standard works well in the text-based UNIX environment where all processes have access to STDIN and STDOUT. In the Windows and Windows NT environments, however, STDIN and STDOUT are available only to non-

graphical (console-mode) programs. To complicate matters further, NT creates a different sort of STDIN and STDOUT for 32-bit programs than it does for 16-bit programs. Because most Web servers are 32-bit services under NT, this means that CGI scripts have to be 32-bit console-mode programs. That leaves popular languages such as Visual Basic and Delphi out in the cold. One popular NT server, the freeware HTTPS from EMWAC, can talk only to CGI programs this way. Fortunately, there are several ways around this problem.

2.5.3 Where CGI Scripts Live

Just like any other file on a server, CGI scripts have to live somewhere. Depending on the server, CGI scripts may have to live all in one special directory.

Typically-whether required by the server or not-Webmasters, a special case of the system administrator disease, put all the scripts in one place. This directory is usually part of the Web server's tree, often just one level beneath the Web server's root. By far the most common directory name is CGI-BIN, a tradition that got started by the earliest servers to support CGI: servers that (believe it or not) hard-coded the directory name. UNIX hacks will like the BIN part, but because the files are rarely named *.bin and often aren't in binary format anyway, the rest of the world roll their eyes and shrug. Today, servers usually allow specifying the name of the directory and often support multiple CGI directories for multiple virtual servers (that is, one physical server that pretends to be many different ones, each with its own directory tree).

Suppose that the UNIX Web server is installed so that the fully qualified path name is /usr/bin/https/Webroot. The CGI-BIN directory would then be /usr/bin/https/Webroot/cgi-bin. That's where the Webmaster, puts the files. From the Web server's point of view, /usr/bin/https/Webroot is the directory tree's root, so it would be possible to refer to a file there called index.html with a URL of /index.html. A script called myscript.pl living in the CGI-BIN directory would be referred to as /cgi-bin/myscript.pl.

On a Windows or NT server, much the same thing happens. The server might be installed in c:\winnt35\system32\https, with a server root of d:\Webroot. It is possible to refer to the file default.htm in the server root as /default.htm, never minding that its real location is d:\Webroot\default.htm. If the CGI directory is d:\Webroot\scripts, we would refer to a script called myscript.exe as /scripts/myscript.exe.

2.5.4 CGI Server Requirements

CGI scripts, by their very nature, place an extra burden on the Web server. They're separate programs, which means the server process must spawn a new task for every CGI script that's executed. The server can't just launch the program and then sit around waiting for the response-chances are good that others are asking for URLs in the meantime. So the new task must operate asynchronously, and the server has to monitor the task to see when it's done.

The overhead of spawning a task and waiting for it to complete is usually minimal, but the task itself will use system resources-memory and disk-and also will consume processor time slices. Even so, any server that can't run two programs at a time isn't much of a server. But the other URLs being satisfied while the program is running.

What if there are a dozen, or a hundred, of them, and what if most of them are also CGI scripts? A popular site can easily garner dozens of hits almost simultaneously. If the server tries to satisfy all of them, and each one takes up memory, disk, and processor time, we can quickly bog the server down so far that it becomes worthless.

There's also the matter of file contention. Not only are the various processes (CGI scripts, the server itself, plus whatever else may be running) vying for processor time and memory, they may be trying to access the same files. For example, a guest book script may be displaying the guest book to three browsers while updating it with the input from a fourth. (There's nothing to keep the multiple scripts running from being the same script multiple times.) The mechanisms for ensuring a file is available-locking it while writing and releasing it when done-all take time: system OS time and simple computation time. Making a script foolproof this way also makes the script bigger and more complex, meaning longer load times and longer execution times.

Therefore, to run CGI scripts the server's capacity should be known, the site has to planned and monitor performance on an ongoing basis. These requirements are based on the software being running on the server, what CGI scripts being used and what kind of traffic the server sees.

2.6 Active Server Pages (ASP)

Active Server Pages (ASP) is a compile-free, text based scripting environment for creating dynamic web sites. It allows a developer to quickly and easily create dynamic database-driven web sites, intranets and extranets.

To write an ASP script all we need is a text editor, Notepad will do. To run ASP we need an installation of Internet Information Server (IIS) or a version of personal web server. ASP can connect to all popular databases but most ASP programmers start out with Access or SQL Server. There are also ASP environments available for other operating systems such as Chilisoft ASP, which will run on operating systems such as UNIX.

ASP first became available to developers in October 96 with the release of the public beta for Internet Information Server 3 (IIS). Up until this point, ASP has been known by the project name "Denali". Microsoft followed up with ASP 2 in August 1997 as part of IIS 4 and with IIS 5 and Windows 2000, ASP is now at version 3.

Microsoft has developed a new replacement for what many people call "Classic ASP" called ASP.NET. as part of their huge .NET initiative, but many people and companies will need to keep their classic asp systems for a while even once ASP.NET is officially released.

ASP.NET is a revolutionary programming framework that enables the rapid development of powerful web applications and services. Part of the Microsoft .NET Platform, it provides the easiest and most scalable way to build, deploy and run web applications that can target any browser or device.

Chapter 3

RESERVING A BOOK FROM LIBRARY (USING ASP)

In this chapter, I will introduce an example of using Active Server Pages (ASP) to reserve a book on the web from a certain library, for instance the Near East University Library.

To perform this application, it is needed to create a database that contains certain variables and connect it to the web page. The ASP can work with any popular database including Access and SQL, as it was mentioned before.

3.1 Creating the Database

The database will be called (studentdb.dbf) and it will contain variables that are related to the student registered in the university's database files, such as the student name, student number, the name of the book reserved and the department (Figure 3.1).

I will use Access 2000 to create the database file, and then connect it to the web page. The database file will be located in the server of the library, which is also connected to the web page using specific commands in ASP.

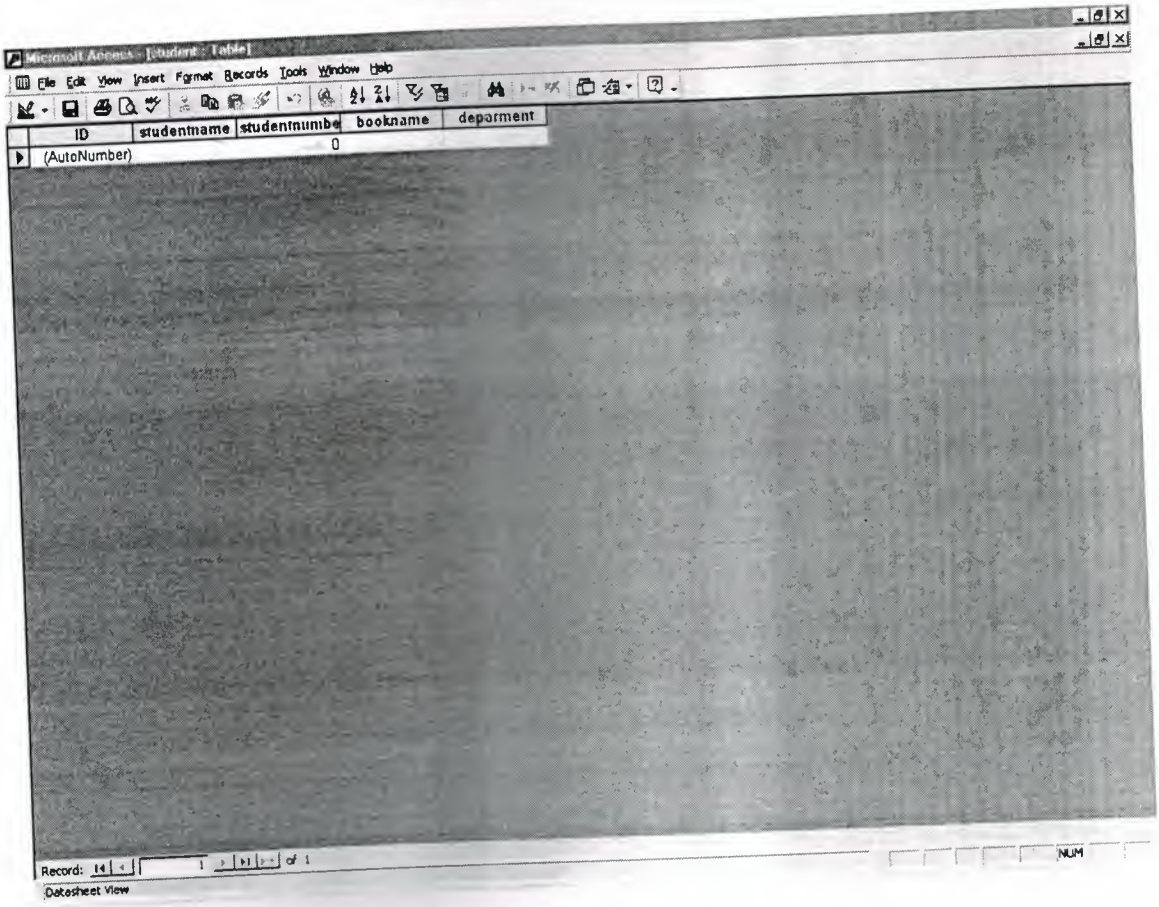

Figure 3.1 Database layout

3.2 The Page's Code

The code of the page is written using HTML and ASP so that the page would be able to communicate with the server. I used Dreamwaver program to design the page (Figure 3.2).

The first page that will appear is going to display a list of the books' names that can be reserved. These books' names will be retrieved from the database located in the library's server.

We connect to the data base with the following code:

```
<%
Dim objConn

'create an ADO connection object
```

```
Set objConn = Server.CreateObject("ADODB.Connection")

'open the connection to the database
'sqlservername = the name of the SQL server (if used)
'accessdb = the path to the access db from this script
'bookname=the bookname to be retreived

ObjConn. Open "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=" &
Server.MapPath ("studentdb")
%>
```

The code of the list page will be:

```
<html>

<head>
<meta http-equiv="Content-Language" content="en-us">
<meta name="GENERATOR" content="Microsoft FrontPage 5.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<meta http-equiv="Content-Type" content="text/html; charset=windows-
1252">
<title>THE LIBRARY</title>
</head>

<body>

<form method="POST" action="--WEBBOT-SELF--">
<!--webbot bot="SaveResults" u-file="_private/form_results.csv" s-
format="TEXT/CSV" s-label-fields="TRUE" --><p>
 </p>
<p>    <font color="#FF3300"><b> THE
LIBRARY</b></font></p>
<p><font color="#0000FF">SELECT YOUR BOOK FROM THE
CURRENTLY AVAILABLE LIST:</font></p>
<p><input type="radio" value="V1" checked name="R1">Book name
1        
```

```
<input type="radio" name="R1" value="V15">Book
name8           
     
</p>
<p><input type="radio" name="R1" value="V2">Book name
2        
<input type="radio" name="R1" value="V9">Book name9</p>
<p><input type="radio" name="R1" value="V3">Book name
3        
<input type="radio" name="R1" value="V10"> Book name10</p>
<p><input type="radio" name="R1" value="V4">Book name
4        
<input type="radio" name="R1" value="V11">Book name11</p>
<p><input type="radio" name="R1" value="V5">Book name
5        
<input type="radio" name="R1" value="V12">Book name12</p>
<p><input type="radio" name="R1" value="V6">Book name
6        
<input type="radio" name="R1" value="V13">Book name13</p>
<p><input type="radio" name="R1" value="V7">Book name
7        
<input type="radio" name="R1" value="V14">Book name14</p>
<p> </p>

<p>           &
nbsp;         <input type="submit"
value="Continue" action="reserve.asp" method="POST" name="B1"></p>
<p> </p>
<p> </p>
<p> </p>
<p> </p>
<p> </p>
<p> </p>
<p> </p>
```

```
<p> </p>
<p> </p>
<p> </p>
<p> </p>
<p> </p>
</form>

</body>

</html>
```

The layout of the index.html page (listing page) will be as shown in figure 3.2.

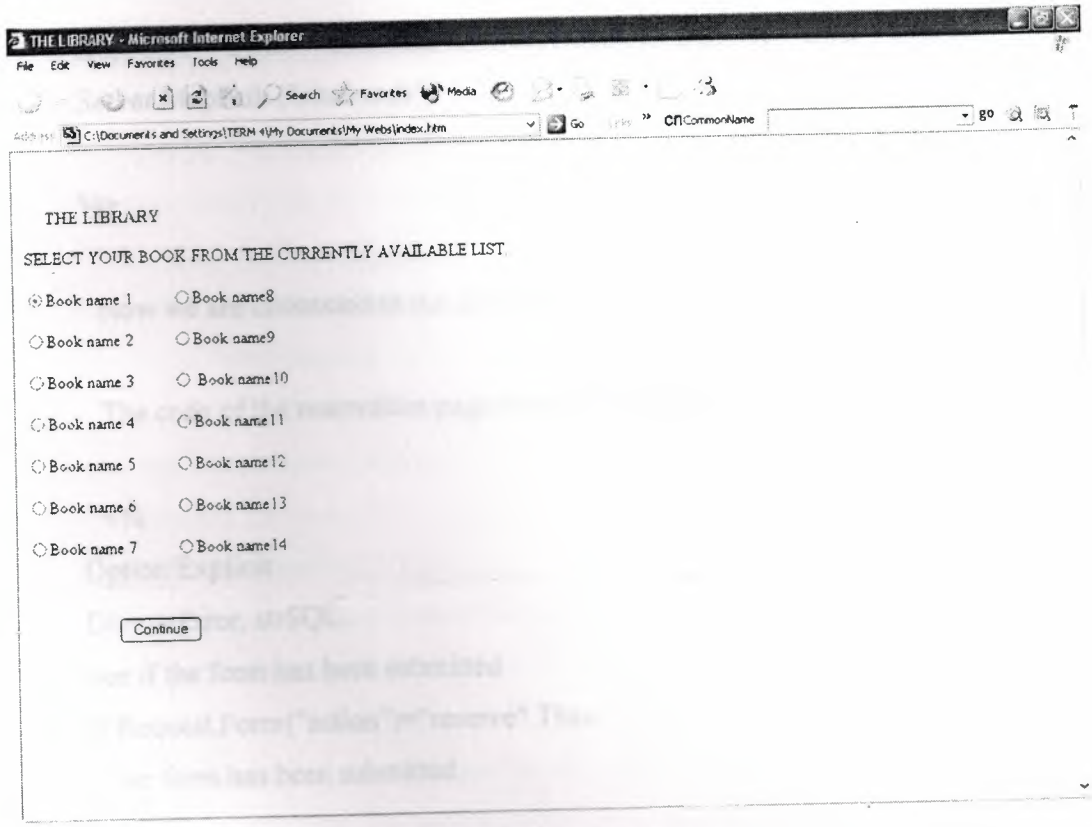

Figure 3.2 Listing Page Layout

When pressing the (Continue) button, the Reserving page will appear. There, we will connect again to the database. To implement this progress is as follows:

```
<%
Dim objConn

'create an ADO connection object
Set objConn = Server.CreateObject("ADODB.Connection")

'open the connection to the database
'sqlservername = the name of the SQL server (if used)
'accessdb = the path to the access db from this script
'studentname = student name to connect to the db
'studentnumber = student number to connect to the db
'department= department to connect to the db

ObjConn. Open "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=" &
Server.MapPath ("studentdb")

%>
```

Now we are connected to the database.

The code of the reservation page (reserve. asp) will be as follows:

```
<%
Option Explicit
Dim strError, strSQL
'see if the form has been submitted
If Request.Form("action")="reserve" Then
  'the form has been submitted

  '// validate the form

  'check if a student name has been entered
  If Request.Form("studentname") = "" Then _
     strError = strError & "- Please enter your name<br>" & vbNewLine
```

```
'check if the number has been entered
If Request.Form("studentnumber") = "" Then _
   strError = strError & "- Please enter your number<br>" & vbNewLine

'// check if an error has occured
If strError = "" Then
  'continue
  'include database connection code
  %>
  <!--#include file="inc-dbconnection.asp"-->
  <%
  On Error Resume Next

  '// create the SQL
  strSQL = "INSERT INTO members
([studentname],[studentnumber],[department]) VALUES " & _
    "('" & fixQuotes(Request.Form("studentname")) & "','" & _
    fixQuotes(Request.Form("studentnumber")) & "','" & _
    fixQuotes(Request.Form("department")) & "')"
  '// run the SQL
  objConn.Execute strSQL
  '// check for an error
  If Err.Number = 222 Then
    strError = "- You Entered a wromg name<br>" & vbNewLine
  Else Err.Number <> 0 Then
    strError = "- An error occured. " & Err.Number & " : " & _
      Err.Description & "<br>" & vbNewLine
    Response.End
  End If

  'restore standard error handling
  On Error Goto 0
```

```
  End If
  If strError <> "" Then
    'output the error message
    'add extra HTML...
    strError = "<p><font color=""#FF0000"">The following errors
occured:" & _
      "</font><br>" & vbNewLine & strError
  End If
End If

Function fixQuotes(strData)
  fixQuotes = Replace(strData,"'","''")
End Function
%>
<html>
<head>
<title>RESERVE YOUR BOOK-NEAR EAST UNIVERSITY</title>
<script language="JavaScript">
<!--
function MM_reloadPage(init) {  //reloads the window if Nav4 resized
  if (init==true) with (navigator) {if
((appName=="Netscape")&&(parseInt(appVersion)==4)) {
    document.MM_pgW=innerWidth; document.MM_pgH=innerHeight;
onresize=MM_reloadPage; }}
  else if (innerWidth!=document.MM_pgW ||
innerHeight!=document.MM_pgH) location.reload();
}
MM_reloadPage(true);
// -->
</script>
</head>
<body bgcolor="#FF9933">
<div id="Layer1" style="position:absolute; left:382px; top:165px;
width:266px; height:41px; z-index:1">
```

```
 <h1 align="center">THE LIBRARY</h1>
</div>
<div id="Layer1" style="position:absolute; left:94px; top:247px;
width:400px; height:28px; z-index:1">
 <p><font face="Times New Roman, Times, serif"><i><u><b><font
color="#0000FF">Please
   fill out the following form to reserve your desired
book:</font></b></u></i></font></p>
</div>
<%=strError%>
<div id="Layer2" style="position:absolute; left:213px; top:68px;
width:627px; height:52px; z-index:2"><font size="+4"
color="#990000">NEAR
 EAST UNIVERSITY</font></div>
<div id="Layer3" style="position:absolute; left:205px; top:700px;
width:482px; height:19px; z-index:3">
 <p align="center"><i><font face="Georgia, Times New Roman, Times,
serif" size="1">All
   rights reserved &copy; 2002 by Mohamemed J.
Mohammed</font></i></p>
</div>
<form action="reserve.asp" method="POST">
 <div align="center">
  <div id="Layer1" style="position:absolute; left:245px; top:509px;
width:400px; height:118px; z-index:1">
    <input type="hidden" name="action" value="reserve">
    <table border="0">
     <tr>
      <td height="29"><b>Student Name</b></td>
      <td height="29">
<%=Server.HTMLEncode(Request.Form("studentname"))%>
       <input type="text" name="studentname" value="">
      </td>
     </tr>
```

```
<tr>
 <td><b>Student Number</b></td>
 <td>
     <%=Server.HTMLEncode(Request.Form("studentnumber"))%>
  <input type="text" maxlength=20 name="studentnumber">

 </td>
</tr>
<tr>
 <td><b>Department</b> </td>
 <td>
  <div align="left">

<%=Server.HTMLEncode(Request.Form("department"))%>
     <input type="text" maxlength=20 name="department">

  </div>
 </td>
</tr>
<tr>
 <td> </td>
 <td>
  <input type="submit" value="Complete Reservation">
 </td>
</tr>
</table>
</div>
</div>
<div align="center"></div>
<div id="Layer1" style="position:absolute; left:220px; top:309px;
width:400px; height:118px; z-index:1">
 <p><b><font color="#800000">For how long do you want to reserve the
book?!</font></b></p>
 <p>
```

```
<input type="radio" name="radiobutton" value="radiobutton">
<b>1-3 Days</b></p>
<p>
<input type="radio" name="radiobutton" value="radiobutton">
<b>3-5 Days</b></p>
<p>
<input type="radio" name="radiobutton" value="radiobutton">
<b>One week</b></p>
<p> </p>
</div>
</form>
</body>
</html>
```

When the code is implemented, the web page we look as in figure 3.3

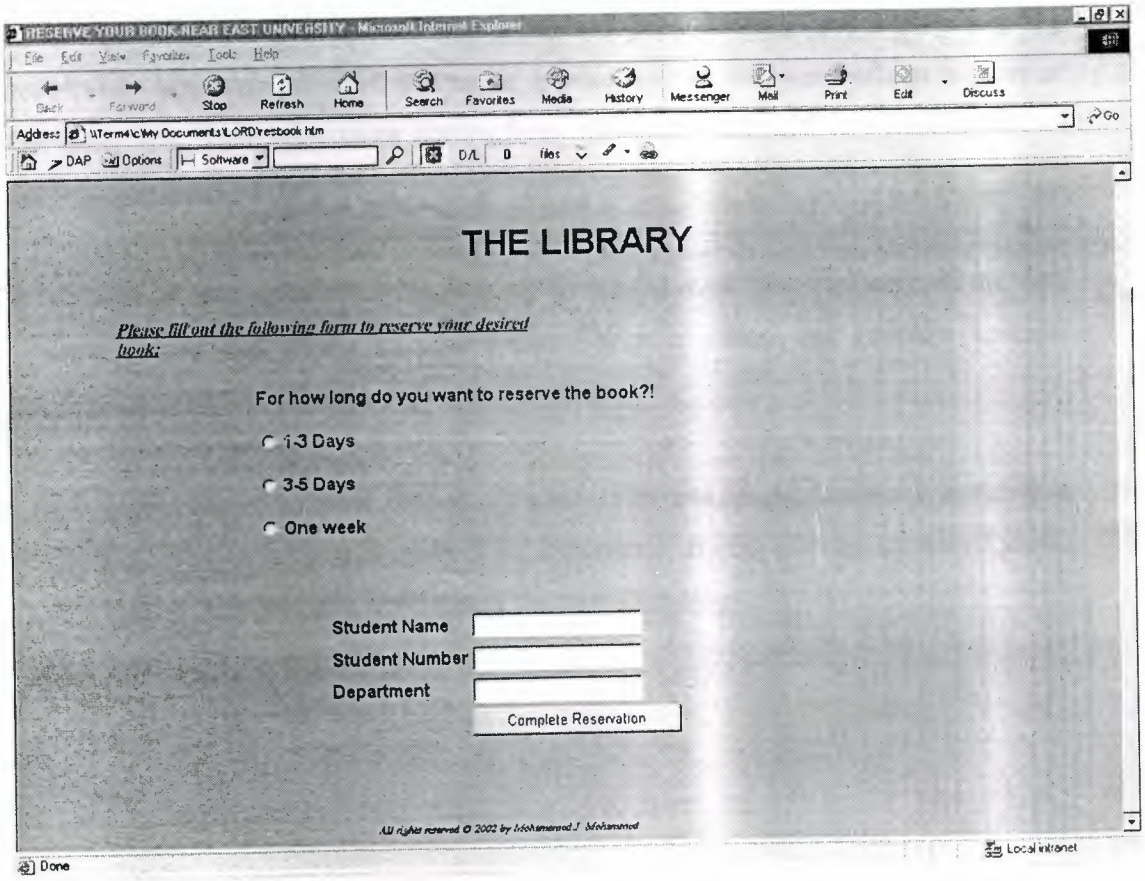

Figure 3.3 Web page layout

CONCLUSION

Web technology is strong on interactivity, but low on automation. Electronic commerce on the Web is primarily driven manually via a browser. In order to achieve business-to-business integration organizations have resorted to proprietary protocols. The many-to-many nature of Web commerce demands a standard for automated integration.

This proposal defines the infrastructure necessary for Web resources to be described as functional interfaces that can be invoked directly from business applications written in languages such as Java, C/C++, COBOL, and Visual Basic.

By capturing details such as input parameters, service URLs, and data extraction methods for output parameters, WIDL enables automation of interactions normally performed manually via a browser.

In a general view, nowadays, Internet programming and web designing technology is a very important and useful application that helps and affects in a lot of our daily life. This importance comes from the Internet it self in providing useful information in all our life fields.

Specifically, Active Server Pages (ASP) is the most powerful page on the web because it has the ability to communicate with a server and perform specific applications with the database saved in that server.

ASP technology was first in October 1996 and its in version 3 now. It is continuing to be developed because of its importance. By the coming couple of years, ASP will lead the world to a new type of Internet technology.

REFERENCES

[1] Paul Lomax, Learning VB Script, 1st Edition.

[2] Aaron Weiss, Java Script Tutorial for Programmers.

[3] A. Keyton Weissigner, ASP in a Nutshell, A Desktop Quick Reference.

[4] Scott Mitchell, Designing Active Server Pages.

[5] Louis Rosenfeld & Peter Morville, Information Architecture for The World Wide Web.

[6] Paul Lomax, Matt Childs & Ron Petrusha, VB Script in a Nutshell.

[7] Shelly Powers, Developing ASP Components.

[8] http://www.examples.oreilly.com/devaspcom2/chapters/

[9] http://www.web.oreilly.com.

[10] http://www.oreilly.com.

[11] http://www.webopedia.internet.com.

[12] http://www.cwru.edu.