NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

HOTEL RESERVATION PROGRAM USING ASP

Graduation Project
COM 400

Student: Tahsin Erhan Önel (990739)

Supervisor: Ümit İlhan

Nicosia - 2006

ACKNOWLEDGEMENT

"First, I would like to special thank my supervisor Ümit İlhan for his helped me, my project to be successful.

Second, I would like to express my thankfulness to member of all Near East University, they gave me chance to learn more about my task that is related my future.

Third, I would like to thank my family, my all friends and other teachers to make get a vision.

Finally, I would also like to thank all my friends for their advice and support."

ABSTRACT

Over the year's computer scientist have found new techniques in computer industry. The aim of this scientist is to solve people's common problems. However a doctor examined a patient, who gives him/her prescription, a computer scientist offers the people system solution. It maybe sometimes calculator, sometimes systems robots, something give idea new business industry like e-commerce.

As we see computer is related with our life in many ways. It was divided into four main segments which are hardware, software, system analysis and network systems. My project is related mainly with, e-commerce with software systems.

First of all it has been done analysis of our topic to get optimum solution. After then it was designed whole project on the paper. Last and most important part is to load on the computer (coding etc.).

It was been using some programming languages which are Microsoft Access, Microsoft FrontPage, and ASP to load on computer. A last thing was to give a name that is Sweet Lies Hotel website and reservation using ASP.

The purpose of SLH website is to help the company who interested for e commerce.

TABLE OF CONTENTS

CHAPTER 1

1. INTRODUCTION TO ASP

My project is Hotel Reservation Program Using ASP (Active Server Page), lets start to discuss about The ASP. ASP stands for Active Server Pages, and it is Microsoft's implementation of server-side scripting. In short, this server-site scripting basically means that a script is parsed and executed by the server. When a user requests a web-page containing ASP, the web-server will parse code and send the result to the user, so the ASP code will never reach the user's browser. This is the exact opposite of JavaScript.

Server-side scripts run when a browser requests an .asp file from the Web server. ASP is called by the Web server, which processes the requested file from top to bottom and executes any script commands. It then formats a standard Web page and sends it to the browser.

You can use ASP code to do a lot of things: dynamically edit, change or add any content to a web-page, access and query databases, read or write files, connect to remote computers, create images - the only limit is your imagination. The most obvious difference from HTML files is that ASP files have the extension ".asp", but this doesn't mean that you need to separate the ASP code from the HTML code in different files; you can use one file which will include both HTML and ASP code. The web-server won't mind at all, it will parse the ASP code and forget about the HTML code. Depending on your ASP code, the web-server will output some HTML instead of the ASP code, so the web-browser will only see HTML.

This provides a higher level of security, because nobody will be able to view your ASP code, and copy it and use it on their own web-page. Furthermore, you don't need any extra components for your web-browser, because the ASP files are returned in plain HTML, so they can be viewed in any web-browser.

An ASP page is an HTML page that contains server-side scripts that are processed by the Web server before being sent to the user's browser. You can combine ASP with Extensible Markup Language (XML), Component Object Model (COM), and Hypertext Markup Language (HTML) to create powerful interactive Web sites.

After brief explanation of ASP I want to explain Hypertext Markup Language (HTML).

1.1 HTML Documents

1.1.1 What an HTML Document is?

HTML documents are plain-text (also known as ASCII) files that can be created using any text editor (e.g., Emacs or vi on UNIX machines; SimpleText on a Macintosh; Notepad on a Windows machine). You can also use word-processing software if you remember to save your document as "text only with line breaks".

1.1.2 HTML Editors

Some WYSIWYG editors are also available (e.g., Claris Home Page or Adobe PageMill, both for Windows and Macintosh). You may wish to try one of them after you learn some of the basics of HTML tagging. WYSIWYG is an acronym for "what you see is what you get"; it means that you design your HTML document visually, as if you were using a word processor, instead of writing the markup tags in a plain-text file and imagining what the resulting page will look like. It is useful to know enough HTML to code a document before you determine the usefulness of a WYSIWYG editor, in case you want to add HTML features that your editor doesn't support.

1.1.3 Tags Explained

An element is a fundamental component of the structure of a text document. Some examples of elements are heads, tables, paragraphs, and lists. Think of it this way: you use HTML tags to mark the elements of a file for your browser. Elements can contain plain text, other elements, or both.

To denote the various elements in an HTML document, you use tags. HTML tags consist of a left angle bracket (<), a tag name, and a right angle bracket (>). Tags are usually paired (e.g., <H1> and </H1>) to start and end the tag instruction. The end tag looks just like the start tag except a slash (/) precedes the text within the brackets. HTML tags are listed below.

Some elements may include an attribute, which is additional information that is included inside the start tag. For example, you can specify the alignment of images (top, middle, or bottom) by including the appropriate attribute with the image source HTML code.

1.1.4 The Minimal HTML Document

Every HTML document should contain certain standard HTML tags. Each document consists of head and body text. The head contains the title, and the body contains the actual text that is made up of paragraphs, lists, and other elements. Browsers expect specific information because they are programmed according to HTML and SGML specifications.

Required elements are shown in this sample bare-bones document:

```
<html>
<head>
<TITLE>A Simple HTML Example</TITLE>
</head>
<body>
<H1>HTML is Easy To Learn</H1>
<P>Welcome to the world of HTML.
This is the first paragraph. While short it is
still a paragraph!</P>
<P>And this is the second paragraph.</P>
</body>
</html>
```

The required elements are the <html>, <head>, <title>, and <body> tags (and their corresponding end tags). Because you should include these tags in each file, you might want to create a template file with them.

1.1.5 Teaching Tool

To see a copy of the file that your browser reads to generate the information in your current window, select View Source (or the equivalent) from the browser menu. (Most browsers have a "View" menu under which this command is listed.) The file contents, with all the HTML tags, are displayed in a new window.

This is an excellent way to see how HTML is used and to learn tips and constructs. Of course, the HTML might not be technically correct. Once you become familiar with HTML and check the many online and hard-copy references on the subject, you will learn to distinguish between "good" and "bad" HTML.

Remember that you can save a source file with the HTML codes and use it as a template for one of your Web pages or modify the format to suit your purposes.

1.2 Markup Tags

1.2.1 HTML, HEAD, TITLE and BODY

HTML: This element tells your browser that the file contains HTML-coded information. The file extension .html also indicates this an HTML document and must be used.

HEAD: The head element identifies the first part of your HTML-coded document that contains the title. The title is shown as part of your browser's window.

TITLE: The title element contains your document title and identifies its content in a global context. The title is typically displayed in the title bar at the top of the browser window, but not inside the window itself. The title is also what is displayed on someone's hotlist or bookmark list, so choose something descriptive, unique, and relatively short. A title is also used to identify your page for search engines. For example, you might include a shortened title of a book along with the chapter contents: NCSA Mosaic Guide (Windows): Installation. This tells the software name, the platform, and the chapter contents, which is more useful than simply calling the document Installation. Generally you should keep your titles to 64 characters or fewer.

BODY: The second--and largest--part of your HTML document is the body, which contains the content of your document (displayed within the text area of your browser window). The tags explained below are used within the body of your HTML document.

1.2.2 Headings

HTML has six levels of headings, numbered 1 through 6, with 1 being the largest. Headings are typically displayed in larger and/or bolder fonts than normal body text. The first heading in each document should be tagged <H1>.

The syntax of the heading element is:

<Hy>Text of heading </Hy>

where y is a number between 1 and 6 specifying the level of the heading.

Do not skip levels of headings in your document. For example, don't start with a level-one heading (<H1>) and then next use a level-three (<H3>) heading.

1.2.3 Paragraphs

Unlike documents in most word processors, carriage returns in HTML files aren't significant. In fact, any amount of whitespace -- including spaces, linefeeds, and carriage returns -- are automatically compressed into a single space when your HTML document is displayed in a browser. So you don't have to worry about how long your lines of text are. Word wrapping can occur at any point in your source file without affecting how the page will be displayed.

In the bare-bones example shown in the Minimal HTML Document section, the first paragraph is coded as

```
<P>Welcome to the world of HTML.
This is the first paragraph.
While short it is
still a paragraph!</P>
```

In the source file there is a line break between the sentences. A Web browser ignores this line break and starts a new paragraph only when it encounters another <P> tag.

```
<H1>Level-one heading</H1>
<P>Welcome to the world of HTML. This is the
first paragraph. While short it is still a
paragraph! </P> <P>And this is the second paragraph.</P>
```

To preserve readability in HTML files, put headings on separate lines, use a blank line or two where it helps identify the start of a new section, and separate paragraphs with blank lines (in addition to the <P> tags). These extra spaces will help you when you edit your files (but your browser will ignore the extra spaces because it has its own set of rules on spacing that do not depend on the spaces you put in your source file).

Using the <P> and </P> as a paragraph container means that you can center a paragraph by including the ALIGN=alignment attribute in your source file.

```
<TT><P ALIGN=CENTER></TT>
This is a centered paragraph.
[See the formatted version below.]
</P>
```

This is a centered paragraph.

It is also possible to align a paragraph to the right instead, by including the ALIGN=RIGHT attribute. ALIGN=LEFT is the default alignment; if no ALIGN attribute is included, the paragraph will be left-aligned.

1.2.4 Lists

HTML supports unnumbered, numbered, and definition lists. You can nest lists too, but use this feature sparingly because too many nested items can get difficult to follow.

1.2.4.1 Unnumbered Lists

To make an unnumbered, bulleted list,

1. Start with an opening list <UL> (for unnumbered list) tag
2. Enter the <LI> (list item) tag followed by the individual item; no closing </LI> tag is needed
3. End the entire list with a closing list </UL> tag

Below is a sample three-item list:

```
<UL>
<LI> apples
<LI> bananas
<LI> grapefruit
</UL>
```

The output is:

- apples
- bananas
- grapefruit

The <LI> items can contain multiple paragraphs. Indicate the paragraphs with the <P> paragraph tags.

1.2.4.2 Numbered Lists

A numbered list (also called an ordered list, from which the tag name derives) is identical to an unnumbered list, except it uses <OL> instead of <UL>. The items are tagged using the same <LI> tag. The following HTML code:

<OL>
<LI> oranges
<LI> peaches
<LI> grapes
</OL>

produces this formatted output:

1. oranges
2. peaches
3. grapes

1.2.4.3 Definitions Lists

A definition list (coded as <DL>) usually consists of alternating a definition term (coded as <DT>) and a definition definition (coded as <DD>). Web browsers generally format the definition on a new line and indent it.

The following is an example of a definition list:

<DL>
<DT> NCSA
<DD> NCSA, the National Center for Supercomputing
Applications, is located on the campus of the
University of Illinois at Urbana-Champaign.
<DT> Cornell Theory Center
<DD> CTC is located on the campus of Cornell
University in Ithaca, New York.
</DL>

The output looks like:

NCSA

NCSA, the National Center for Supercomputing Applications, is located on the campus of the University of Illinois at Urbana-Champaign.

Cornell Theory Center

CTC is located on the campus of Cornell University in Ithaca, New York.

The <DT> and <DD> entries can contain multiple paragraphs (indicated by <P> paragraph tags), lists, or other definition information.

The COMPACT attribute can be used routinely in case your definition terms are very short. If, for example, you are showing some computer options, the options may fit on the same line as the start of the definition.

```
<DL COMPACT>
<DT> -i
<DD>invokes NCSA Mosaic for Microsoft Windows
 using the initialization file defined in the path
<DT> -k
<DD>invokes NCSA Mosaic for Microsoft Windows in
 kiosk mode
</DL>
```

The output looks like:

-i

invokes NCSA Mosaic for Microsoft Windows using the initialization file defined in the path.

-k

invokes NCSA Mosaic for Microsoft Windows in kiosk mode.

1.2.4.4 Nested Lists

Lists can be nested. You can also have a number of paragraphs, each containing a nested list, in a single list item.

Here is a sample nested list:

```
<UL>
<LI> A few New England states:
  <UL>
  <LI> Vermont
```

```
<LI> New Hampshire
<LI> Maine
</UL>
<LI> Two Midwestern states:
<UL>
<LI> Michigan
<LI> Indiana
</UL>
</UL>
```

The nested list is displayed as

- A few New England states:
  - Vermont
  - New Hampshire
  - Maine
- Two Midwestern states:
  - Michigan
  - Indiana

1.2.5 Preformatted Text

Use the<PRE> tag (which stands for "preformatted") to generate text in a fixed-width font. This tag also makes spaces, new lines, and tabs significant -- multiple spaces are displayed as multiple spaces, and lines break in the same locations as in the source HTML file. This is useful for program listings, among other things. For example, the following lines:

```
<PRE>
#!/bin/csh
cd $SCR
cfs get mysrc.f:mycfsdir/mysrc.f
cfs get myinfile:mycfsdir/myinfile
fc -02 -o mya.out mysrc.f
mya.out
cfs save myoutfile:mycfsdir/myoutfile
rm *
```

```
</PRE>
```

display as:

```
#!/bin/csh
cd $SCR
cfs get mysrc.f:mycfsdir/mysrc.f
cfs get myinfile:mycfsdir/myinfile
fc -02 -o mya.out mysrc.f
mya.out
cfs save myoutfile:mycfsdir/myoutfile
rm *
```

The <PRE> tag can be used with an optional WIDTH attribute that specifies the maximum number of characters for a line. WIDTH also signals your browser to choose an appropriate font and indentation for the text.

Hyperlinks can be used within <PRE> sections. You should avoid using other HTML tags within <PRE> sections, however.

1.2.6 Extended Qoutations

Use the <BLOCKQUOTE> tag to include lengthy quotations in a separate block on the screen. Most browsers generally change the margins for the quotation to separate it from surrounding text.

In the example:

```
<P>Omit needless words.</P>
<BLOCKQUOTE>
<P>Vigorous writing is concise. A sentence should
contain no unnecessary words, a paragraph no unnecessary
sentences, for the same reason that a drawing should have
no unnecessary lines and a machine no unnecessary parts.
</P>
<P>--William Strunk, Jr., 1918 </P>
</BLOCKQUOTE>
```

the result is:

Omit needless words.

Vigorous writing is concise. A sentence should contain no unnecessary words, a paragraph no unnecessary sentences, for the same reason that a drawing should have no unnecessary lines and a machine no unnecessary parts.

--William Strunk, Jr., 1918

1.2.7 Forced Line Breaks/Postal Addresses

The
 tag forces a line break with no extra (white) space between lines. Using <P> elements for short lines of text such as postal addresses results in unwanted additional white space. For example, with:

```
National Center for Supercomputing Applications<BR>
605 East Springfield Avenue<BR>
Champaign, Illinois 61820-5518<BR>
```

The output is:
National Center for Supercomputing Applications
605 East Springfield Avenue
Champaign, Illinois 61820-5518

1.2.8 Horizontal Rules

The <HR> tag produces a horizontal line the width of the browser window. A horizontal rule is useful to separate major sections of your document.
You can vary a rule's size (thickness) and width (the percentage of the window covered by the rule). Experiment with the settings until you are satisfied with the presentation. For example:

```
<HR SIZE=4 WIDTH="50%">
```
displays as:

1.3 Character Formatting

HTML has two types of styles for individual words or sentences: logical and physical. Logical styles tag text according to its meaning, while physical styles indicate the specific appearance of a section. For example, in the preceding sentence, the words "logical styles" was tagged as "emphasis." The same effect (formatting those words in italics) could have been achieved via a different tag that tells your browser to "put these words in italics."

1.3.1 Logical Versus Physical Styles

If physical and logical styles produce the same result on the screen, why are there both?

In the ideal SGML universe, content is divorced from presentation. Thus SGML tags a level-one heading as a level-one heading, but does not specify that the level-one heading should be, for instance, 24-point bold Times centered. The advantage of this approach (it's similar in concept to style sheets in many word processors) is that if you decide to change level-one headings to be 20-point left-justified Helvetica, all you have to do is change the definition of the level-one heading in your Web browser. Indeed, many browsers today let you define how you want the various HTML tags rendered on-screen using what are called cascading style sheets, or CSS. CSS is more advanced than HTML, though, and will not be covered in this Primer. (You can learn more about CSS at the World Wide Web Consortium CSS site.)

Another advantage of logical tags is that they help enforce consistency in your documents. It's easier to tag something as <H1> than to remember that level-one headings are 24-point bold Times centered or whatever. For example, consider the <STRONG> tag. Most browsers render it in bold text. However, it is possible that a reader would prefer that these sections be displayed in red instead. (This is possible using a local cascading style sheet on the reader's own computer.) Logical styles offer this flexibility.

Of course, if you want something to be displayed in italics (for example) and do not want a browser's setting to display it differently, you should use physical styles. Physical styles, therefore, offer consistency in that something you tag a certain way will always be displayed that way for readers of your document.

Logical Styles

<DFN>

for a word being defined. Typically displayed in italics. (NCSA Mosaic is a World Wide Web browser.)

<EM>

for emphasis. Typically displayed in italics. (Consultants cannot reset your password unless you call the help line.)

<CITE>

for titles of books, films, etc. Typically displayed in italics. (A Beginner's Guide to HTML)

<CODE>

for computer code. Displayed in a fixed-width font. (The <stdio.h> header file)

<KBD>

for user keyboard entry. Typically displayed in plain fixed-width font. (Enter passwd to change your password.)

<SAMP>

for a sequence of literal characters. Displayed in a fixed-width font. (Segmentation fault: Core dumped.)

<STRONG>

for strong emphasis. Typically displayed in bold. (NOTE: Always check your links.)

<VAR>

for a variable, where you will replace the variable with specific information. Typically displayed in italics. (rm filename deletes the file.)

Physical Styles

<B>

bold text

<I>

italic text

<TT>

typewriter text, e.g. fixed-width font.

1.3.2 Escape Sequences

Character entities have two functions:

- Escaping special characters
- Displaying other characters not available in the plain ASCII character set (primarily characters with diacritical marks)

Three ASCII characters--the left angle bracket (<), the right angle bracket (>), and the ampersand (&)--have special meanings in HTML and therefore cannot be used "as is" in text. (The angle brackets are used to indicate the beginning and end of HTML tags, and the ampersand is used to indicate the beginning of an escape sequence.) Double quote marks may be used as-is but a character entity may also be used (").

To use one of the three characters in an HTML document, you must enter its escape sequence instead:

<
the escape sequence for <
>
the escape sequence for >
&
the escape sequence for &

Additional escape sequences support accented characters, such as:

ö
a lowercase o with an umlaut: ö
ñ
a lowercase n with a tilde: ñ
È
an uppercase E with a grave accent: È

1.4 Linking

The chief power of HTML comes from its ability to link text and/or an image to another document or section of a document. A browser highlights the identified text or image with color and/or underlines to indicate that it is a hypertext link (often shortened to hyperlink or just link).

HTML's single hypertext-related tag is <A>, which stands for anchor. To include an anchor in your document:

1. start the anchor with <A (include a space after the A)
2. specify the document you're linking to by entering the parameter HREF="filename" followed by a closing right angle bracket (>)
3. enter the text that will serve as the hypertext link in the current document
4. enter the ending anchor tag: </A> (no space is needed before the end anchor tag)

Here is a sample hypertext reference in a file called US.html:

```
<A HREF="MaineStats.html">Maine</A>
```

This entry makes the word Maine the hyperlink to the document MaineStats.html, which is in the same directory as the first document.

1.4.1 Relative Pathnames Versus Absolute Pathnames

You can link to documents in other directories by specifying the relative path from the current document to the linked document. For example, a link to a file NYStats.html located in the subdirectory AtlanticStates would be:

```
<A HREF="AtlanticStates/NYStats.html">New York</A>
```

These are called relative links because you are specifying the path to the linked file relative to the location of the current file. You can also use the absolute pathname (the complete URL) of the file, but relative links are more efficient in accessing a server. They also have the advantage of making your documents more "portable" -- for instance, you can create several web pages in a single folder on your local computer, using relative links to hyperlink one page to another, and then upload the entire folder of web pages to your web server. The pages on the server will then link to other pages on the server, and the copies on your hard drive will still point to the other pages stored there.

In general, you should use relative links whenever possible because:

1. it's easier to move a group of documents to another location (because the relative path names will still be valid)
2. it's more efficient connecting to the server

3. there is less to type

However, use absolute pathnames when linking to documents that are not directly related. For example, consider a group of documents that comprise a user manual. Links within this group should be relative links. Links to other documents (perhaps a reference to related software) should use absolute pathnames instead. This way if you move the user manual to a different directory, none of the links would have to be updated.

1.4.2 URL's

The World Wide Web uses Uniform Resource Locators (URLs) to specify the location of files on other servers. A URL includes the type of resource being accessed (e.g., Web, gopher, FTP), the address of the server, and the location of the file. The syntax is:

scheme://host.domain [:port]/path/ filename

where scheme is one of

file

a file on your local system

ftp

a file on an anonymous FTP server

http

a file on a World Wide Web server

gopher

a file on a Gopher server

WAIS

a file on a WAIS server

news

a Usenet newsgroup

telnet

a connection to a Telnet-based service

The port number can generally be omitted. (That means unless someone tells you otherwise, leave it out.)

For example, to include a link to this primer in your document, enter:

<A HREF="http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimer.html">
NCSA's Beginner's Guide to HTML</A>

This entry makes the text NCSA's Beginner's Guide to HTML a hyperlink to this document.

1.4.3 Links to Specific Sections

Anchors can also be used to move a reader to a particular section in a document (either the same or a different document) rather than to the top, which is the default. This type of an anchor is commonly called a named anchor because to create the links, you insert HTML names within the document.

You can also link to a specific section in another document. That information is presented first because understanding that helps you understand linking within one document.

1.4.4 Links Between Sections of Different Documents

Suppose you want to set a link from document A (documentA.html) to a specific section in another document (MaineStats.html).

Enter the HTML coding for a link to a named anchor:

documentA.html:

```
In addition to the many state parks, Maine is also home to
<a href="MaineStats.html#ANP">Acadia National Park</a>.
```

Think of the characters after the hash (#) mark as a tab within the MaineStats.html file. This tab tells your browser what should be displayed at the top of the window when the link is activated. In other words, the first line in your browser window should be the Acadia National Park heading.

Next, create the named anchor (in this example "ANP") in MaineStats.html:

```
<H2><A NAME="ANP">Acadia National Park</a></H2>
```

With both of these elements in place, you can bring a reader directly to the Acadia reference in MaineStats.html.

1.4.5 Links to Specific Sections within the Current Document

The technique is the same except the filename is omitted.

For example, to link to the ANP anchor from within MaineStats, enter:

```
...More information about
<A HREF="#ANP">Acadia National Park</a>
is available elsewhere in this document.
```

Be sure to include the <A NAME=> tag at the place in your document where you want the link to jump to (<A NAME="ANP">Acadia National Park).

Named anchors are particularly useful when you think readers will print a document in its entirety or when you have a lot of short information you want to place online in one file.

1.4.6 Mailto

You can make it easy for a reader to send electronic mail to a specific person or mail alias by including the mailto attribute in a hyperlink. The format is:

```
<A HREF="mailto:emailinfo@host">Name</a>
```

For example, enter:

```
<A HREF="mailto:pubs@ncsa.uiuc.edu">
NCSA Publications Group</a>
```

to create a mail window that is already configured to open a mail window for the NCSA Publications Group alias.

1.5 Inline Images

Most Web browsers can display inline images (that is, images next to text) that are in X Bitmap (XBM), GIF, or JPEG format. Other image formats are also being incorporated into Web browsers [e.g., the Portable Network Graphic (PNG) format]. Each image takes additional time to download and slows down the initial display of a document. Carefully select your images and the number of images in a document.

To include an inline image, enter:

<IMG SRC=ImageName>

where ImageName is the URL of the image file.

The syntax for <IMG SRC> URLs is identical to that used in an anchor HREF. If the image file is a GIF file, then the filename part of ImageName must end with .gif. Filenames of X Bitmap images must end with .xbm; JPEG image files must end with .jpg or .jpeg; and Portable Network Graphic files must end with .png.

1.5.1 Image Size Attributes

You should include two other attributes on <IMG> tags to tell your browser the size of the images it is downloading with the text. The HEIGHT and WIDTH attributes let your browser set aside the appropriate space (in pixels) for the images as it downloads the rest of the file. (You can get the pixel size from your image-processing software, such as Adobe Photoshop. Some browsers will also display the dimensions of an image file in the title bar if the image is viewed by itself without an enclosing HTML document.)

For example, to include a self portrait image in a file along with the portrait's dimensions, enter:

<IMG SRC=SelfPortrait.gif HEIGHT=100 WIDTH=65>

1.5.2 Aligning Images

You have some flexibility when displaying images. You can have images separated from text and aligned to the left or right or centered. Or you can have an image aligned with text. Try several possibilities to see how your information looks best.

1.5.2.1 Aligning Text with an Image

By default the bottom of an image is aligned with the following text, as shown in this paragraph. You can align images to the top or center of a paragraph using the ALIGN= attributes TOP and CENTER.

This text is aligned with the top of the image (<IMG SRC = "BarHotlist.gif" ALT="[HOTLIST]" ALIGN=TOP>). Notice how the browser aligns only one line and then jumps to the bottom of the image for the rest of the text.

And this text is centered on the image (<IMG SRC = "BarHotlist.gif" ALT="[HOTLIST]" ALIGN=CENTER>). Again, only one line of text is centered; the rest is below the image.

1.5.2.2 Images without Text

To display an image without any associated text (e.g., your organization's logo), make it a separate paragraph. Use the paragraph ALIGN= attribute to center the image or adjust it to the right side of the window as shown below:

```
<p ALIGN=CENTER>
<IMG SRC = "BarHotlist.gif" ALT="[HOTLIST]">
</p>
```

The image is centered; this paragraph starts below it and left justified.

1.5.2.3 Alternate Text For Images

Some World Wide Web browsers -- primarily the text-only browsers such as Lynx -- cannot display images. Some users turn off image loading even if their software can display images (especially if they are using a modem or have a slow connection). HTML provides a mechanism to tell readers what they are missing on your pages if they can't load images.

The ALT attribute lets you specify text to be displayed instead of an image. For example:

```
<IMG SRC="UpArrow.gif" ALT="Up">
```

where UpArrow.gif is the picture of an upward pointing arrow. With graphics-capable viewers that have image-loading turned on, you see the up arrow graphic. With a text-only browser or if image-loading is turned off, the word Up is shown in your window in place of the image.

You should try to include alternate text for each image you use in your document, which is a courtesy for your readers -- or, for users who might be visually impaired, a necessity.

1.5.2.4 Images as Hyperlinks

Inline images can be used as hyperlinks just like plain text. The following HTML code:

<A HREF="hotlist.html"><IMG SRC="BarHotlist.gif" ALT="[HOTLIST]"></A>

The BORDER attribute can also be set to non-zero values, whether or not the image is used as a hyperlink. In this case, the border will appear using the default text color for the web page. For instance, if you wanted to give your image a plain black border to help it stand out on the page, you might try this:

<IMG SRC="BarHotlist.gif" BORDER=6 ALT="[HOTLIST]">

1.5.3 Background Graphics

Newer versions of Web browsers can load an image and use it as a background when displaying a page. Some people like background images and some don't. In general, if you want to include a background, make sure your text can be read easily when displayed on top of the image.

Background images can be a texture (linen finished paper, for example) or an image of an object (a logo possibly). You create the background image as you do any image.

However you only have to create a small piece of the image. Using a feature called tiling, a browser takes the image and repeats it across and down to fill your browser window. In sum you generate one image, and the browser replicates it enough times to fill your window. This action is automatic when you use the background tag shown below.

The tag to include a background image is included in the <BODY> statement as an attribute:

<BODY BACKGROUND="filename.gif">

1.5.4 Background Color

By default browsers display text in black on a gray background. However, you can change both elements if you want. Some HTML authors select a background color and coordinate it with a change in the color of the text.

Always preview changes like this to make sure your pages are readable. (For example, many people find red text on a black background difficult to read!) In general, try to avoid using high-contrast images or images that use the color of your text anywhere within the graphic.

You change the color of text, links, visited links, and active links (links that are currently being clicked on) using further attributes of the <BODY> tag. For example:

```
<BODY BGCOLOR="#000000" TEXT="#FFFFFF" LINK="#9690CC">
```

This creates a window with a black background (BGCOLOR), white text (TEXT), and silvery hyperlinks (LINK).

The six-digit number and letter combinations represent colors by giving their RGB (red, green, blue) value. The six digits are actually three two-digit numbers in sequence, representing the amount of red, green, or blue as a hexadecimal value in the range 00-FF. For example, 000000 is black (no color at all), FF0000 is bright red, 0000FF is bright blue, and FFFFFF is white (fully saturated with all three colors).

1.5.5 External Images, Sounds, and Animations

You may want to have an image open as a separate document when a user activates a link on either a word or a smaller, inline version of the image included in your document. This is called an external image, and it is useful if you do not wish to slow down the loading of the main document with large inline images.

To include a reference to an external image, enter:

```
<A HREF="MyImage.gif">link anchor</A>
```

You can also use a smaller image as a link to a larger image. Enter:

```
<A HREF="LargerImage.gif"><IMG SRC="SmallImage.gif"></A>
```

The reader sees the SmallImage.gif image and clicks on it to open the LargerImage.gif file.

Use the same syntax for links to external animations and sounds. The only difference is the file extension of the linked file. For example,

<A HREF="AdamsRib.mov">link anchor</A>

specifies a link to a QuickTime movie. Some common file types and their extensions are:

plain text
.txt
HTML document
.html
GIF image
.gif
TIFF image
.tiff
X Bitmap image
.xbm
JPEG image
.jpg or .jpeg
PostScript file
.ps
AIFF sound file
.aiff
AU sound file
.au
WAV sound file
.wav
QuickTime movie
.mov
MPEG movie
.mpeg or .mpg

Keep in mind your intended audience and their access to software.

1.6 Tables

Before HTML tags for tables were finalized, authors had to carefully format their tabular information within <PRE> tags, counting spaces and previewing their output. Tables are very useful for presentation of tabular information as well as a boon to creative HTML authors who use the table tags to present their regular Web pages.

Think of your tabular information in light of the coding explained below. A table has heads where you explain what the columns/rows include, rows for information, cells for each item. In the following table, the first column contains the header information, each row explains an HTML table tag, and each cell contains a paired tag or an explanation of the tag's function.

Table Elements	
Element	**Description**
<TABLE> ... </TABLE>	defines a table in HTML. If the BORDER attribute is present, your browser displays the table with a border.
<CAPTION> ... </CAPTION>	defines the caption for the title of the table. The default position of the title is centered at the top of the table. The attribute ALIGN=BOTTOM can be used to position the caption below the table. **NOTE:** Any kind of markup tag can be used in the caption.
<TR> ... </TR>	specifies a table row within a table. You may define default attributes for the entire row: ALIGN (LEFT, CENTER, RIGHT) and/or VALIGN (TOP, MIDDLE, BOTTOM). See Table Attributes at the end of this table for more information.
<TH> ... </TH>	defines a table header cell. By default the text in this cell is bold and centered. Table header cells may contain other attributes to determine the characteristics of the cell and/or its contents. See Table Attributes at the end of this table for more information.
<TD> ... </TD>	defines a table data cell. By default the text in this cell is aligned left and centered vertically. Table data cells may contain other attributes to determine the characteristics of the cell and/or its contents. See Table Attributes at the end of this table for more information.

Table Attributes	
NOTE: Attributes defined within <TH> ... </TH> or <TD> ... </TD> cells override the default alignment set in a <TR> ... </TR>.	
Attribute	**Description**
ALIGN (LEFT, CENTER, RIGHT)	Horizontal alignment of a cell.
VALIGN (TOP, MIDDLE, BOTTOM)	Vertical alignment of a cell.
COLSPAN=*n*	The number (*n*) of columns a cell spans.
ROWSPAN=*n*	The number (*n*) of rows a cell spans.
NOWRAP	Turn off word wrapping within a cell.

General Table Format

The general format of a table looks like this:

<TABLE>
<!-- start of table definition -->

<CAPTION> caption contents </CAPTION>
<!-- caption definition -->

<TR>
<!-- start of header row definition -->
  <TH> first header cell contents </TH>
  <TH> last header cell contents </TH>
</TR>
<!-- end of header row definition -->

<TR>
<!-- start of first row definition -->
  <TD> first row, first cell contents </TD>
  <TD> first row, last cell contents </TD>

```
</TR>
<!-- end of first row definition -->

<TR>
<!-- start of last row definition -->
   <TD> last row, first cell contents </TD>
   <TD> last row, last cell contents </TD>
</TR>
<!-- end of last row definition -->

</TABLE>
<!-- end of table definition -->
```

You can cut-and-paste the above code into your own HTML documents, adding new rows or cells as necessary. The above example looks like this when rendered in a browser.

The <TABLE> and </TABLE> tags must surround the entire table definition. The first item inside the table is the CAPTION, which is optional. Then you can have any number of rows defined by the <TR> and </TR> tags. Within a row you can have any number of cells defined by the <TD>...</TD> or <TH>...</TH> tags. Each row of a table is, essentially, formatted independently of the rows above and below it. This lets you easily display tables like the one above with a single cell, such as Table Attributes, spanning columns of the table.

1.6.1 Tables for Nontabular Information

Some HTML authors use tables to present nontabular information. For example, because links can be included in table cells, some authors use a table with no borders to create "one" image from separate images. Browsers that can display tables properly show the various images seamlessly, making the created image seem like an image map (one image with hyperlinked quadrants).

Using table borders with images can create an impressive display as well. Experiment and see what you like.

1.7 Fill-out Forms

Web forms let a reader return information to a Web server for some action. For example, suppose you collect names and email addresses so you can email some information to people who request it. For each person who enters his or her name and address, you need some information to be sent and the respondent's particulars added to a data base.

This processing of incoming data is usually handled by a script or program written in Perl or another language that manipulates text, files, and information. If you cannot write a program or script for your incoming information, you need to find someone who can do this for you.

The forms themselves are not hard to code. They follow the same constructs as other HTML tags. What could be difficult is the program or script that takes the information submitted in a form and processes it. Because of the need for specialized scripts to handle the incoming form information, fill-out forms are not discussed in this primer.

CHAPTER 2

2. INSIDE OF ASP (Active Server Page)

2.1 ASP Installation

No matter if you want to use ASP just to learn web-programming, or you want to create a major dynamic web-site, you must use a web-server that supports the ASP technology. There are two approaches to this. Either you choose to install Microsoft's Personal Web Server (PWS) or Internet Information Services (IIS) on your own PC, or you will have to find a web-hosting company running IIS which will host your web-site. If you're just looking to check out what ASP is all about, forget about the web-hosting company. Installing PWS or IIS on your computer is not such a big deal, and you don't have to be a highly qualified programmer to do this, some basic computer skills will do just fine.

First of all, you should know that there are different installation methods according to your Windows version. Even if you can install PWS on Windows 95, this operating system is really old and unstable, so you should forget about it. You should have at least Windows 98 if you want to use ASP, but Windows 2000 is highly recommended.

To run ASP on Windows 98, you have to install PWS from the Windows 98 CD. Explore the CD, and you will find the PWS in the Add-ons folder. Run setup.exe, and after the installation is done, you will find a folder called "Inetpub" on your hard-drive. Inside, there is another folder called "wwwroot", which holds all the files of your local web-server. Give it a try. Copy a HTML file in "wwwroot", and then open your web-browser and type in the address: "http://localhost/my_file.html". Don't forget to actually replace "my_file.html" with the name of your HTML file you copied. The same thing you must do when you want to use an ASP page.

To install ASP on Windows NT, you will have to download "Windows NT 4.0 Option Pack from Microsoft", because PWS is not included in Windows NT. This is not the case for Windows 2000. Click on Start, and select Settings, then Control Panel. Double-click Add/Remove Programs, then select Add/Remove Windows Components. A wizard window will show up on your screen, where you will see "Internet Information Services". Check that item, then click OK. Windows will then install IIS, and create the "Inetpub" folder. From now on, you can use IIS the same way you can

use PWS on Windows 98. Check the previous section for additional details. But unlike Windows 98, you will see that the installation program has added a new icon on your taskbar - the IIS symbol. You should click the Start button that appears, so IIS can start.

Windows XP differs a little bit. You might know that there are two versions - Windows XP Home Edition and Windows XP Professional. Unfortunately, IIS is only available for Windows XP Professional. To install it, take a look at the previous section - the installation process of IIS on Windows XP Professional is identical to installing IIS on Windows 2000. You can start or stop the IIS web-server by going into the Control Panel, then the Administrative Tools. You will find the "IIS Manager" item, which you can double-click to view its properties.

Regardless of the operating system that you're using, don't forget to test the web-server after you have installed it.

2.2 First ASP Script

You can write ASP scripts the same way you write HTML code. You don't need expensive tools or professional programming environment; you only need the most basic text editor in the world - Notepad - which can be found in any Windows version. Just like a HTML document, ASP files are made out of plain text. So let's give it a try. Click on Start, point to Programs, Accessories, than click on Notepad. Presto! Now type in the following text:

```
<html>
<body>
<%
Response.Write("Hello World!")
%>
</body>
</html>
```

That's it! Save the file to your "Inetpub\wwwroot\" folder, and give it an easy to remember name - "hello.asp". ASP files use the ".asp" extension, so remember to include this. To take a look at the output of the file, point your web-browser to "http://localhost/hello.asp". Now try to view the source of the document - right click

somewhere on the page, and then click "View Source". You will not see the ASP source code, because your web-browser didn't receive it.

This way you can figure out for yourself that the web-server parses the text within the ASP tags - "<% (...) %>" - and leaves the rest of the HTML code untouched. While the previous example was made out of both HTML and ASP code, you can choose to use only ASP code:

```
<%
Response.Write("<html>")
Response.Write("<body>")
Response.Write("Hello World!")
Response.Write("</body>")
Response.Write("</html>")
%>
```

Save this file under the name "hello2.asp", in the same folder. Again, if you try to view its source after you have opened the file with your browser - "http://localhost/hello2.asp" - you will not see any ASP code. In fact, you will see the same HTML code as the previous "hello.asp".

There are two approaches to ASP scripting: VBScript or JScript. VBScript is the default scripting language, so you don't have to inform ASP that you're using it. But if you want to use JScript as the default language, you must insert a language specification at the top of the page:

```
<%@ language="javascript"%>
<html>
<body>
<%
Response.Write("Hello World!");
%>
</body>
</html>
```

While this might look the same at first, it's a complete different language. But the most important thing about it is that, unlike VBScript, JavaScript is CaSe SeNsItIvE. This means "Write" is not the same as "write", so you should really pay attention to your code.

The VBScript and JScript (which is Microsoft's implementation of JavaScript) languages are included in ASP, so you don't have to install any extra components. On the other hand, if you want to use another scripting language - for example Perl, REXX or Python - you will have to a script engine to handle that specific language.

2.3 Jscript Syntax

The JScript code is written in text format like most other programming languages. It consists of blocks of statements, that, when combined together, create a script. You can find most programming tools within such a block: variables, expressions, calls to functions, and immediate data references such as strings and numbers (also called "literals").

A statement is like a sentence in English, and consists of one or more expressions, operators, keywords, etc. Statements are separated by semicolons ";", and, in most cases, a statement is written using a single line of text. Nevertheless, you can write a statement over two or more lines, most of all to improve the accessibility of it all. It's one thing to have write a statement using a really long line of text, where you must scroll a lot to actually "see it", and it's another to have the same statement written on multiple lines.

In the previous JScript we wrote, you can notice the semicolon at the end of the statement.

```
Response.Write("Hello World!");
```

Sometimes, you need to use multiple statements as if they were one. To do this, you must surround all these statements with bracers "{...}"; this is referred to as a block of statements. So even if JScript expects one statement, you can insert a whole block. There are some exceptions to this rule, and some of the most important are the headers of some functions such as "for" and "while". One more thing you might need to know is that, even though a statement ends with a semicolon, this rule does not apply to a block of statements.

Commenting your source code helps a lot. A comment is some text that is not taken into consideration by JScript when it parses the text, so it's there only for the user's "eyes". It helps a lot, as I was saying, because there are times when you want to get back to the script you just wrote, and you need to know what a script or a part of it does without actually browsing through its source code. You might also want to write down to-do's, tips about improvement, etc.

There are two ways you can use comments in JScript: single line comments and multi-line comments. Single-line comments are useful for small comments, when you need to add a to-do or small notice. To define a single line comment, you must add a pair of forward slashes "//", then the comment itself. Multi-line comments are useful for when writing details information about how the code works, copyright information etc. A multi-line comment begins with a forward slash followed by an asterisk "/*", and it ends with the opposite, the asterisk followed by the forward slash "*/". Here are some examples of both types:

```
/* This is a demo script
It is free of charge, and you can modify it as much as you want.
*/
dbName = "database"; //the name of your database
dbHost = "localhost"; //the address of your database server
//to-do: add more info here
```

2.4 Java Script Data Types

There three main (primitive) data types in JScript: string, number and boolean. Like in any other language, strings are collection of characters (letters, digits, signs, blanks, etc.) strung together. The string type is used mainly to represent text, and the string values must be enclosed in matching pairs of single or double quotation marks.

When it comes to numbers, Jscript doesn't differentiate integers and floating point values, like other languages do. Integer values can be positive whole numbers, negative whole numbers, and 0. You can represent these numbers in base 10 (decimal), base 8 (octal), and base 16 (hexadecimal), but most numbers in JScript are written in decimal. If you want to specify an octal integer, you should add a leading zero in front of the number containing digits from 0 to 7. If you add 8 or 9, the number will be

interpreted as a decimal number. To represent hexadecimal integers, you must add the characters "0x" in front of the number. Hexadecimal numbers can contain digits from 0 through 9, and letters A through F.

Floating-point values are whole numbers with a decimal portion. You can also express these values using a scientific notation: the character "e" is used to represent "ten to the power of". You should know that numbers beginning with "0x" and "00" who also contain a decimal point will generate an error, because floating-point values are available only for decimal values, and not also octal or hexadecimal values.

While a number and a string can contain lots of values, the boolean data type can only contain two: true or false. This is because a Boolean value is a truth-value, and it expresses whether an expression evaluates to true of false.

There are two special data types in JScript: "null" and "undefined". The null data type can only hold one value: null; this data type cannot be used as the name of a function or a variable. Data containing null is interpreted as containing "no value" or "no object". This means that is doesn't hold a valid number, string, boolean, array or object; so if you want to erase the contents of a variable, without deleting it, you can always assign it the null value.

The undefined data type is used instead of display some error messages, for example when some data has been declared, but no value was assigned to it.

2.5 Variables in Script

A programming language cannot function without variables. Variables are used to store, retrieve, and also manipulate the value it contains. Based on that value, a variable can contain a string, a number, a boolean value etc. To use a variable, you must first declare it, which means that some memory is allocated to store the variable, so you can refer to it later in your script. You can use the "var" statement to define a variable, and you can choose to initialize also initialize the variable. If you don't initialize a variable in the var statement, then it will be assigned the value "undefined".

```
var money; //a simple declaration
var first_number, second_number; //multiple declarations in one var keyword
var name1 = "Jack", name2 = "Laura"; //multiple declarations and initializations at the same time
```

You can always declare a variable without using the var keyword, and then assign a value to it:

price = 1500; // The variable price is declared implicitly.

Take a little time to assign the variable a meaningful name so you know what it holds. Because Jscript is a case-sensitive language, a variable name such as "MyName" is different from "myName". There are some rules in using variable names. First, remember that the first character of the variable must be a letter or the underscore character "_". While a number cannot be used as the first character, the following characters can be numbers, letters, and underscores.

Also, you should not assign a variable the same name as a reserved word.

As you may have noticed, you don't have to also declare the variable type. In other languages, this is a very important step, but JScript is very flexible from this point of view. Its variables have a type corresponding to the type of value they contain. The first benefit of this flexible feature is that you can treat a variable as if it were of another type. This means that if you try to "add" a number to a string, then the number will be converted to a string and then both variables will be concatenated. This process is called coercing. This way, adding a number or a boolean to a string will coerce the result into a string; the result of adding a number to a boolean variable will coerce the result to a number.

2.6 Operators & Expressions in ASP / Jscript

You are already familiar with some operators in JScript, even if you've never used a programming language before. In fact, you've already used some of them in the previous chapters, and they didn't look unusual at all to you.

The arithmetic operators are quite what you expect them to be, similar to the ones used in the basic arithmetic you learned in the first grade:

+ for addition
- for subtraction
* for multiplication
/ for division
% for modulus

++ increment

-- decrement

The increment and decrement operators may seem a little stranger at first sight: they are used to increase, respectively decrease a variable by one. So, when you write

x = 1

x++

this means that x equals 2

You will see a lot more of these operators when we discuss the conditional structures and the loops below.

Note that the "+" sign can also be used as a string operator - for instance, when you need to put two strings of text together. Begin by assigning the text to a variable, and then use the "+" operator to bring them together:

```
text = "The capital city is ";
capital_city = "Washington";
text_complete = text + capital_city;
```

Now the text_complete variable contains the string "The capital city is Washington". Make sure you include the blank spaces in the string, or, if you want to use them separately, place them between quotation marks:

```
text = text1 + " " + text2 + " " + text3;
```

JScript also has assignment operators. You've used one of them to assign values to variables, and it probably "felt" quite normal:

```
x = 5;
```

meaning that you've assigned the value 5 to the variable x.

JScript also uses the compound assignment operators, which assign a value after performing the designed operation: x += y is the same as x = x + y. If both x and y are numeric or boolean, then they will be added, is both are strings, or only one of them is a string, they will be concatenated.

In the same way, you have:

x -= y the same as x = x - y

x += y the same as x = x * y

x /= y the same as x = x / y

x %= y the same as x = x % y

When you need to find out if x is equal to 5, you'll use one of the comparison operators: ==. So, x == 5 will return either "true" or "false". The comparison operators, all of whom return the values "true" or "false", as the following:

== for "is equal to"

!= for "is not equal to"

> for "is greater than"

< for "is less than"

>= for "is greater than or equal to"

<= for "is less than or equal to"

Finally, you have the logical operators. Again, you've probably seen these ones before as well:

! for "not"

&& for "end"

|| for "or"

Also like in regular arithmetic, parentheses are used to alter the order in which the operations are performed, meaning that the operations between brackets have the priority.

Considering the variables:

x = 1

y = 2

the comparison

x == y

will return "false", but the comparison

!(x == y) will return "true.

Considering the same variables,

(x == 1 || y == 5) returns true.

Translated into plain English, this sounds like this: If x is equal to 1, and y is equal to 2, than the statement "Either x equals one OR y equals 5" is true. Learning a programming language is a lot like learning a foreign language (a bit easier, though), so, when you tell the computer do to something, make sure you know exactly what you want to say, and then translate it into the respective programming language. Usually, there is more than one way for telling the computer to do the same thing, same is, in the English language, there are several sentences with the same meaning, and you decide which one you want to use.

So, when you "translate" from English into JScript, "if both x equals 1 AND y equals 5 " will look like (x == 1 && y == 5), and will return "false" (both in plain English and in JScript).

2.7 Arrays in Jscript

While variables are used for storing single values (a string, a number, etc.), arrays special variables that can hold for multiple values in the same variable. It's pretty difficult to store 50 names of different people in 50 variables, so this is the main purpose of a variable: it allows you to store as many elements as you want in it. You've already used arrays in the loops created before, and they didn't puzzle you, because they are quite logical and easy to understand.

Because you have such many elements, there must be a very quick way to access them. Well, you should know that there is this thing called indexing, which means that each item has an associated index. JScript indexes an array's elements starting from zero and increments the element's index with each new addition, this way you can always find out the last index of an element: it's the total number of elements minus one.

A second way to index an array is by using strings. These types of array are often referred to as associative arrays, because they have a string associated with each element. This is really useful when you're using small arrays and you want to know exactly what each element is all about. The index, either numeric or string, is always enclosed by brackets "[]".

There are two types arrays in Jscript: typed arrays and object arrays. The typed array has a base data type, so this means that each elements of the array must be of the

same type (for example, all elements are strings). You can declare a typed array using the "new" operator, for instance:

```
var country_capitals = new Array(4)
```

This creates a new array, with four elements. You can do the same, by specifying each element:

```
var country_capitals = new Array("Washington", "Paris", "London", "Moscow")
```

In order to refer to one of the elements of the array, you'll have to use the name of the array, and the index number. Remember that the first index number is 0. So, when you want to assign data to the elements of the array, you have to use the following syntax:

```
country_capitals[0] = "Washington"
country_capitals[1] = "Paris"
country_capitals[2] = "London"
country_capitals[3] = "Moscow"
```

When you need to, you can retrieve the data stored in the elements of the array, with this syntax:

```
town = country_capitals[0]
```

There is a wide variety of uses for the arrays and the data they store. In web pages, you will often see the elements of an array displayed as tables. Also, you can use them for a basic word search, or a filter operation.

Properties can be attributed to an array, by using the syntax: object.property_name. There are three properties for the arrays: prototype (which ads properties to the array), length (for the number of elements) and constructor (with the function that created the prototype of the object).

There are several methods you can use in connection with the arrays, through which they perform the respective actions. The syntax is object.method_name().

One such function you will often find is concat() which joins two arrays (or more, as the case may be). The returned result will be a new array, of course. Another method is pus("element"), which ads the respective element (or more than one) at the end of the given array. In this case, the returned value is the new length of the array.

The names of these methods are quite intuitive, so in many cases you'll understand what they do when you'll see them - reverse(), for instance, reverses the order of the elements of the array, while sort(), well, it sorts the elements. You should spend some time looking at these methods - you don't need to learn them by heart or anything, but it would be useful to know that they exist, so you may use them when you need them.

2.8 Conditional Structure in Jscript

Conditional structures are one the most important feature in any programming language. Jscript implements the if...else statement, just like a lot of other languages. The process is simple: the if structure evaluates an expression to a truth value (true or false); if the expression is evaluated to "true", then some certain statements are called afterwards, otherwise, if the user provides an else statement, Jscript parses the statements after "else". This is how you declare such a structure:

```
if (expression)
statements_if;
[else
statements_else];
```

We placed the "else" part in brackets because it's optional, so you can use "if" without "else". Here are some examples of both structures:

```
if (number == 1) //tests if the variable "number" holds the value "1"
{
response = "The number is one"; //this is executed if the expression is true
}
else
{
response = "The number is not one"; //this is executed if the expression is false
```

```
}

if (color == "white") //tests if the variable "color" holds the value "white"
color = "black"; //if so, let's change the white color to black
```

If you have an if...else structure with only one statement in each part (if and else), then you might find it easier to use the conditional operator "?"...":". This way, you will reduce the amount of written text.

```
expression ? statement_true : statement_false;
```

As you can see, we first type the expression, then the question mark. If the expression is evaluated to true, the statement after the question mark is parsed. Otherwise, the statement after ":" is parsed. You should know that you must have both operators for the whole thing to work; so if you don't use "else" in your structure, stick to using the standard if structure.

```
(number = 1) ? response = "The number is one" : response = "The number is not one";
```

Jscript also provides a way to execute more than two block of statements based on more than two values of a single expression. This can be accomplished by using the switch statement:

```
switch (expression)
{
case value_1:
statements_1
case value_2:
statements_2
...
[ default:
statements_default]

}
```

"switch" tests the expression for all the specified values, then executes the statements after it has encountered the first value that is equal to the expression's result. If it doesn't find any, then it executes the default statement, if there is one. If no values match the expression's result, and there is no "default" statement specified, then nothing is executed. After it has started to run the first block of statements it continues to run also the other statements, even if they don't "belong" to the value which was equal to the expression's result. To stop it, you can use a "break" statement:

```
switch (color)
{
case "black":
html_color = "000000";
break;
case "white":
html_color = "FFFFFF";
break;
case "red":
html_color = "FF0000";
break;
default:
html_color = "FFFFFF";
}
```

2.9 Loop Structures in Jscript

There are a lot of times when you would want to execute more or less the same statements for a large number of times. You can do this though the use of loops. The concept of looping is based on the fact that a value is changed every time the loop executes an iteration - which is a single execution of the loop. The loops is typically controlled by a test of a variable, and it is that variable's value that should change in the iteration, because if we don't do nothing, be will have an infinite loop which does the same thing over and over again, and it never stops. That's not good, and it may be the answer why your script hangs for a long time and it produces no result.

There are four types of loops in JScript: for loops, for...in loops, while loops, do...while loops. The easiest one is the for statement. It specifies a counter variable which it will use for a test condition, and an action that updates the counter before each iteration.

```
for (variable_initialization; expression; increment)
  statements
```

"for" executes the statements as long as the specified expression is evaluated to "true". The following example fills an array with numbers from 0 to 9:

```
var numbers = new Array();
for (i = 0; i < 10; i++) //as long as the variable "i" is smaller than 10...
{
numbers[i] = i;
}
```

A special kind of loop structure is the for...in loop. Jscript uses it for stepping through all the elements of an array, or all the user-defined properties of an object. The syntax is slightly different from the original:

```
for (variable in [object | array])
  statements
```

But if you want a more open approach to loops, you can always use the while structure. There is a slight difference between for loop and while loop - when you use while, you must do all the counter variable initialization and variation yourself. "while" will only accept a expression, and if the expression is evaluated to "true", then it will execute the statements you provide. Because while loops don't have an explicit built-in counter variable, they are more exposed to infinite loops, so always remember to change the counter-variable's value inside the loop! Here is the same loop example as before, but we use "while" instead of "for":

```
var numbers = new Array();
```

```
i = 0; //we have to do the initialization ourselves
while(i < 10)

{
numbers[i] = i;
i++; //don't forget to alter the counter-variable's value, or else you'll get an infinite loop
}
```

Similar to the while loop is the do...while loop, which actually performs the test after it executes the loop. This guarantees that the loop is executed at least once, after which it may or may not continue, based on the evaluation of the expression provided with while:

```
var numbers = new Array();
i = 0;
do
{
numbers[i] = i; //this is parsed at least once...
i++; //this too
}
while (i < 10);
```

There are times when you encounter special values within a loop, and you may want to simple exit the loop, or continue to the next iteration. JScript provides two such statements: "break" and "continue". The break statement immediately stops the execution of the loop, while the continue statement stop the current iteration only, and starts the next one. No matter which statement you choose to use, the previous statements before break or continue are always executed, but the following ones will not be parsed.

You can use break and continue with any of the "for", "while" or "do...while" loops.

```
var numbers = new Array();
for (i = 0; i < 10; i++) //as long as the variable "i" is smaller than 10...
```

```
{
if(i == 5)
continue; //let's skip the number 5
numbers[i] = i;
}
```

2.10 Function in Script

When dealing with a large script, you will very often need to write additional "sub-scripts" that are called by the main script several times. This will save you a lot of time from rewriting the same piece of code every time you need it in the main script. Such "sub-scripts" are called functions. Their main purpose is to take some values, do some actions based on those values, and then return a new result. This way, you won't have to rewrite the pieces of code again and again, but merely call a function that implements that code.

Most of the times you will need to pass information to a function; you can do this by enclosing the information in parentheses after the function's name. These arguments - which is the name of the passed information - are separated by commas. But there are times when you won't need to specify any arguments - you will still need to add the parentheses after the name of the function, but nothing else within them.
There are two types of functions: built-in functions (functions that are built into the language) and user-defined functions (functions that are defined by the user). There are a lot of built-in functions, each one having its purpose, for example the eval() function that takes a string as an argument and evaluates the expression the string holds:

```
result = eval("(5 * 5) + 3"); //assigns the value 28 to the variable "result"
```

But the most important thing about functions is creating them on your own. To define a function you must first use the function statement and insert the function's name and arguments, then you must add the actual code. You can use "return" to assign a value to the function's result.

```
function make_sum(a, b, c) //don't add a semicolon in the end!
{
result = a + b + c;
return result;
}
```

As you can see, we assign the sum of the arguments a, b and c to the variable result, then we return the value this variable holds.

The result variable is a local variable, which means that it is used only within this function. Local variables are created and then destroyed every time the function is called, and they cannot be accessed by other functions outside it.

2.11 Object Oriented Programming in ASP

The Object-Oriented Programming was one of the biggest breakthroughs programmers ever made. It's a different approach to programming, which will make your life a lot easier when you deal with long programs. You don't use the ASP because you want to write "Hello, world!" on a single web page. Instead, you will use it to create complex web sites, such as pages for electronic commerce, or polls and quizzes, or in order to manipulate a complex database. In these cases, you'll have to deal with many items and operations, and countless repetitions become boring and tiresome. Instead of writing the same code lines over and over again, you can now use the object-oriented programming.

This is not a new program, but rather, a new way of thinking. If you've spent some time studying HTML and DHTML, you're probably used to it, but if you've never met this concept, you should spend some time to make sure you understand it correctly. Let's take one very simple example. Let's say you have a car, which, of course, is an object. Now, this car has some properties that differentiate it from other cars, like the number of doors, the speed, the color and so on. Besides, it also performs a number of actions: it accelerates, decelerates, stops. Well, in a programmer's language, the car is called "object", the properties are called, well "properties" or "variables" and the functions and actions are called "methods". And that's all there is to it.

Some objects are built-in, but others you'll need to define for yourself. In JScript, the built-in objects include string, date, math, and array. We will discuss them in detail in due time.

Methods are those operations that modify the object, by manipulating the variables. In our example, you manipulate the "speed" variable of the car object by using the acceleration and deceleration methods. Only an object's methods should modify its variables, you cannot do it directly. If you need to change the variable

"color", for instance, you must have a method, called "painting", that will help you do that.

You will find a full list of JScript objects and their properties and methods in the JScript reference, available on the net. The general syntax is:

```
object.property_name
object.method_name()
```

The String object is the one that works with text. One of its properties is called length, and it returns the number of characters of the respective string. One of its methods is called substring(), and it subtracts a string, contained between the specified values. For instance, substring(5,12) will return a string formed by all the characters between the 5th and the 12th character of the original string.

```
var str = "This is a demo script. You can modify it as much as you like. Copyright softwareprojects 2004"
document.write(str.length) // This will return the value 93
document.write(str.substring(0, 23)) // This will return the string This is a demo script.
document.write(str.toUpperCase()) // This will return the string THIS IS A DEMO SCRIPT
document.write(str.toLowerCase()) // This will return the string this is a demo script
```

Use these examples in your script and see what happens. As you can see, the names of the objects, their properties and their methods are quite easy to figure out - you don't need a lot of schooling to understand what a method called "toUpperCase()" does. Note that this method does not require any parameters - there is noting more to define for it, but the brackets are still needed.

Now that we've figured this one out, let's move on to the date object. You can assign the date to a variable, much in the same way as you assign a number or a string:

```
var x = new Date()
```

You can write down any parameters you want between brackets:

```
var x = new Date("January 1, 2004")
var x = new Date("January 1, 2004 12:00:00")
```

If you don't write any parameters, the script will simply use the date from the respective machine. The two main methods for this object are getDate(), which returns the date from an objects, and, for our variable x, looks like this:

x.getDate()

and setDate(), which sets the date of the month in an object, and looks like this:

x.setDate()

There are various different variations of the two, such as getDay(), getHours(), getMinutes(), setMonths(), setSeconds(). I think you got the idea. Experiment with the Date object a little. It's easy, simple and you see it used very often, on many web pages. This is the perfect object to help you get used to the syntax and the use of parameters. The Math object deals with the basic mathematical constants and functions. Same as the rest of the objects discussed in this chapter, the Math object is already there, you don't need to create it, and you may find it very useful on occasions. It has functions such as PI, which returns the PI, naturally, and E, which returns the base of a natural logarithm. (If you can't remember what the natural base of a logarithm is, then give it a try and see if you can figure it out.) The methods are pretty neat, if you want to make some mathematical games, you have abs(x), which returns the absolute value of x, cox(x) - the cosine of x, round(x), which rounds x to the nearest integer, and random(), which returns a random number between 0 and 1.

2.12 Expanding classes in ASP

Remember when we first introduced the objects, we used the example of a car, which is an object. But the car belongs to a class of objects, the vehicles. This is what happens to the programming objects as well.

You should think of classes as general blueprints for the respective objects. In other words, the classes contain types of objects, in a very wide sense of the word. A class of objects has all the properties and all the methods of the object, but a class is not an object. You should think of the class as a sort of definition of the actual object.

After spending so much time to define objects and their properties and methods, it may seem a waste of time to think about classes as well. But the very first time you'll have to deal with a really big project, you'll be thankful that these classes were invented.

Keep in mind that you won't see classes at work, they are just blueprints, you use the objects defined, not the class itself.
For defining a new class, you use the class statement:

```
class ExampleClass
```

When you need an instance of the class, you have to use the new operator:

```
var x: ExampleClass = new ExampleClass
```

Classes have fields, which are a lot like properties, methods and constructors. A constructor is a method with the same name as the class, which runs when an instance of the class is created, with the new operator.

By default, all the members of the class are publicly accessible, but you can also make certain fields protected or private. Private fields can only be accessed by the other members of the same class, while protected fields can only be accessed by other members of the same class or members of the derived classes. This is called encapsulation, and it is one of the most powerful features of the object-oriented programming. By using it, a piece of code can read and use variables from your class, but it can't modify them, and this may be vital in certain instances. Also, you can use encapsulation to constrain the class members to do something, or to prevent them from doing something. If you define the class correctly from the very beginning, you can save a lot of time later on, and considerably reduce the chance of errors and mistakes.

One other major advantage of using classes is that it allows breaking down the script into modules. In this way, several programmers or several teams of programmers can work on the same script, without having to coordinate themselves perfectly. This is also useful for smaller scripts - if you want to ad another module later on, or if the original programmer left and somebody else needs to take over.

Last but not least, the classes have the so-called "inheritance" characteristic. This means that you can create a class based on another one, using the keyword extends, like this:

```
class ExampleClass extends DerivedClass
```

Now, our original class, ExampleClass, is now called base class, while the new one is called the derived class, and it takes its functionality from the definition of the base class. This means that you use a derived class when you want it to have all the properties and methods of its base class, plus something else. Once again, this little trick helps you avoid writing long lines of code over and over again, when you need something to perform basically the same function, with minor adjustments.
Inheritance is a really powerful tool. You should think of it as a relation of a child who "inherits" features from his parents, but is not the identical copy of the parent. Also, you should consider that you can use the inheritance feature from one "generation" to another as many times you need, to create derived classes from ones that originally derived, and so on.

If, at this point, you find classes and inheritance a bit confusing, you should spend some time reading and deciphering some examples of scripts already made. You will need quite a lot of practice to use them properly, but keep in mind that this is the core structure of any object-oriented scripting, very popular and very much in demand these days. So, both in web programming and in classic programming you are sure to run into objects and classes again.

2.13 Response and request objects in ASP

Let's remember the very first ASP script written, the one that looked like this:

```
Response.Write("Hello World!")
```

Now, try to guess what response and write() are. Of course, response is a built-in object of ASP, while write() is one of its methods. You can introduce HTML tags in the string, it will be interpreted correctly:

```
Response.Write("<b> Hello World! </b>")
```

The response object can do things far more powerful than write text and ad HTML code to your page. As the name implies, its function is to send a response to the user, an output from the server. Now, this brings us one huge step closer to the initial goal of learning ASP, that is, to allow the creation of dynamic, interactive web pages.

For instance, one property for the response object is called ContentType. By default this is set to text/html, but you can change it to anything else, such as image, or a specific type of image, like JPEG or GIF, or a certain application. Let's take a look at the following example:

```
<% Response.ContentType="application/vnd.ms-excel" %>
<html>
<title> Excel Table </title>
<body>
<table>
<tr>
<td> Washington
<td> Paris
<td> London
<td> Moscow
</tr>
</table>
</body>
</html>
```

As you can see, this is a simple HTML table, but the user will see it as an Excel table, as long as this piece of software is installed on the respective computer. If it's not installed, then, well, it wasn't such a great idea. You can use this feature to constrain the user to viewer to use a certain program, or to constrain the content to accept only a certain type of input.

You've seen the message "404 Not Found" hundreds of times on the Internet. It is caused by a property of the response object, called Status. It is often found in a conditional statement, such as If the ip address is . then Response.Status="404 Not Found" or "401 Unauthorized".

Another property is Expires, which establishes a period of time (expressed in minutes) to keep a page cached on a browser before it expires. There is also a version of this property called ExpiresAbsolute, which does the same, but sets a specific date and time for the page to expire. If you don't want the page to be cached at all, you can set the Expires property to a negative value.

```
Response.Expires = -1
```

The Buffer property decides whether to buffer the content of the page - meaning that the server will not allow the user to load the content of the page until it processes all the server scripts, or until it runs the methods Flush or End. If you want to use this method, you need to place it before the actual HTML code, and set it to "true". For the earlier versions of IIS (before 5.0), this was set by default to "false", but in the 5.0 and later versions, the default is "true". This is useful if you want the browser to finish performing a loop before showing anything to the user.

We've already met two methods of the response object, End and Flush. The End method puts an end to the script, and returns the result processed until the respective moment. The Flush method sends the HMTL content that has already been buffered without any delay (without waiting for the rest of the content to be buffered as well).
One method that is quite common is the Redirect, looking like this:

```
Response.Redirect("http://www.softwareprojects.org")
```

Now let's take a look at the request object, used to obtain information from the user. It is really valuable in connection with forms, as shown in the following chapter. It has a property called TotalBytes, which returns the number of bytes sent by the client's request. This is read-only, as you cannot modify this value:

```
X = Request.TotalBytes
```

After this, you can use the method called BinaryRead(a), to retrieve the data the user sent to the server as part of a POST request. The data will be stored in safe array. You need to specify the parameter a, which is how many bites you want the method to read from the user (value usually equal to the X variable returned by the TotalBytes property above).

Besides all these, the Request object also has collections, including ServerVariables (containing all the server variables), QueryString (with all the variables in a HTTP query string), forms and cookies, which are the most important, and will be discussed in the following chapters.

2.14 File manipulation in ASP

When you have to work with complex directors and file structures, you need objects and methods to manipulate them, and ASP has several such built-in options.
In order to work with the files and folders from your web server, you have the FileSystemObject. It can manipulate files, folders, directory paths and retrieve information about them. The drives you work with can be the hard disk, or removable disks, such as a CDRom or a Floppy Disk unit, and even a network drive.

Before you start using this object, keep in mind that it's not 100% secure. After all, the applications created modify the files existing on your server, and may cause quite major safety glitches, so make sure you use passwords to protect your applications. Also, try not to make them public. For the same reasons, some (quite a few) web hosting companies do not support this object, so check with your host to see if you are allowed to use it.

The FileSystemObject has a property called Drives, which returns the drives available for the computer:

```
var fso;
fso = new ActiveXObject("Scripting.FileSystemObject");
drives = fso.Drives;
```

Note that the result is not a single object, but a collection of drives. If you want to display all of them, you can use a for loop. You can also check to see if a file exists on the server:

```
var fso;
fso = new ActiveXObject("Scripting.FileSystemObject");
if(fso.FileExists("c:\windows\1.txt"))
{
Response.Write("Found file!");
}
else
{
Response.Write("Files does not exist.");
```

}

This little piece of script checks if file called "1.txt" exists in the windows folder on the server, and prints the results. Note that, when you are done, you have to explicitly remove the file from the memory of the server by using the keyword "nothing".

There are more methods available for the FileSystemObject. We will list here some of them, with their respective parameters, but remember that they can be dangerous, if not used correctly, and you may delete or "misplace" important files on your server for good:

DeleteFolder (name, force) - deletes the designated folder. The name indicates the name of the folder, while force indicates its behavior towards read-only folders and files. If the force attribute is set to true, than these files and folders will also be deleted.
MoveFolder(source, destination) - moves the folder from the source path to the destination path

BuildPath(path, name) - adds the file or folder indicated by the name to the respective path

GetParentFolderName(name) - pretty obvious, returns the name of the parent folder

Much in the same way, there are methods for working with files, which work exactly like those described above, for the folders:

CopyFile(source, destination, overwrite)
DeleteFile(name, force)
FileExists(name)
GetFile(name)
MoveFile(source, destination)

In addition, you can create and open a file:
CreateTextFile (name, overwrite) - creates a new text file with the given name. The overwrite parameter, set to true, means that it will overwrite a file with the same name, if this already exists.
OpenTextFile(name, create, format) - not only opens the file, but also returns at TextStream object that you can use to access the file.

As you probably figured it out by now, there are methods for working with Drives as well, in the same intuitive manner:

DriveExists (name) - returns the value true if the designated driver exists, and the false value if it doesn't

GetDrive(name) - returns a reference to the specified drive object

You can understand now why leaving these applications unprotected can be dangerous. At the same time, you will be faced with situations when many users have to access files or folders from your web server, and maybe even all of them at the same time. We'll see how you can handle that.

2.15 Cookies in ASP

When you have many visitors, you need to use Cookies to store information in your visitors' computers. Basically, what you do is send some bits of information to your clients, and then request it back, when you need to use it.

If you run a search on the Windows Help about cookies, you'll see that it's quite secretive about what they are and how you can set your computer to accept or reject them. As you will see, there are some users who have strong feelings about Cookies, and consider them an intrusion on their privacy. Some complex web sites, such as portals or e-commerce sites, can't work without them.

In order to create a Cookie, you need to place the command Response.Cookies before your <html> tag. You can also assign properties to a Cookie, such as the date when you want it to expire.

```
<%
Response.Cookies("MyFirstCookie") = "Test";
%>
```

This creates a cookie, named MyFirstCookie, with the value Test assigned to it, and which expires on the 1st of January 2005. If you want the cookie to expire as soon as your visitor leaves, the Expires property gets the value 1. Cookies that are valid only until the browser window is closed are called in-memory cookies, while those saved on the client's disk are called disk-based cookies.

In order to retrieve the value of the cookie, you will use the command Request.Cookies:

```
<%
x = Request.Cookies("MyFirstCookie");
Response.Write("MyFirstCookie has the value of " + x)
%>
```

The output will be "MyFirstCookie has the value of Test"

Using cookies, your page will be able to "identify" the visitors, allowing the offer of personalized content. If a user sends his name, through a form, a cookie can store it and use it later on. It can store other values, such as what links the user accessed, what item he chose from a list, and so on.

A cookie can contain more than one value, in which case these values are called keys:

```
Response.Cookies("MyFirstCookie")("Key1") = "Japan";
Response.Cookies("MyFirstCookie")("Key2") = "China";
Response.Cookies("MyFirstCookie")("Key3") = "Austrlia";
```

When you need to reference the values in the cookie with keys, you have to use the key value:

```
Response.Write(Request.Cookies("MyFirstCookie") ("Key1"));
Response.Write(Request.Cookies("MyFirstCookie") ("Key2"));
Response.Write(Request.Cookies("MyFirstCookie") ("Key3"));
```

You can use the HasKeys property in order to check if one cookie has keys or not.

Not all browsers support cookies, and you need a way to avoid this problem. One way of doing so is by using forms. Forms, used in HTML, allow you to get some input from the user. In combination with the Request object from ASP, they can be used for a variety of ends. Let's say you want to create a form where the user inputs a name and age.

```
<form method="post" action="noCookies.asp">
Name: <input type="text" name="name" value="">
```

```
Age: <input type="text" name="age" value="">
<input type="submit" value="Submit">
</form>
```

This puts the forms on your page, and now you need the noCookies.asp file:

```
<%
name = Request.Form("name");
age = Request.Form("age");
Response.Write("Your name is " + name + " and your age is " + age);
%>
```

Another way of avoiding browsers that have to cookies is to put your parameters directly into the URL. ASP does not have a built-in method to determine whether a browser has the cookies enabled or not - there are other ways to deal with that problem, but we will not cover them now.

Finally, one last thing about cookies: when you create one, it automatically saves the path of the site that created it. In this way, when the user returns to the site, the cookie follows its path back to the server. You can change this path when you want, something useful one you want one cookie to correspond to two or more applications:

```
Response.Cookies("MyFirstCookie").
Path="http://server/softwareprojects/application/";
```

2.16 Sessions in ASP

The Session object helps identify the user for a longer period of time. Think of what happens when you log in into an Internet-based mail account. You have to input your username and password only once - this starts your session - and then you can read and send email for as long as you want. When you hit the "Log out" button, or you close the browser window, your session comes to and, and the application doesn't know who you are anymore.

In order to solve the identification issue, ASP creates a unique cookie for each user - a sort of passport, and we've seen how it's done. The Session object stores the information about the users. Each user has a different Session object, and the server destroys this object when the session expires. The values contained in the Session object

can be accessed by all the pages of your ASP web site - otherwise the user would have to input the data on every page. The fact that all applications share the same cookies may raise some securities issue. Keep in mind that ASP offers some encryption options.

The Session object has two collections: Contents and StaticObjects.

In the Contents collection, you will find all the items appended to the session by a script command: Session.Contents (Key). The "Key" parameter, the name of the item to retrieve, is required. The StaticObjects collection contains all the items appended to the session with the HTML <object> tag.

The SessionID property returns the unique ID, generated by the server for each user. There is also a Timeout property, which sets or returns the timeout for the respective session, in minutes. The session will end after that number of minutes, unless the user refreshes the page, or requests a new page.

The Session object has a method called Abandon, which kills the user's session. When you use it, you'll see that it acts only after it executed the entire script on the respective page. There is also a method called Contents.Remove, which deletes items from the Contents. You can indicate the items to be deleted by their names or by their indexes.

The Session object has two events: Session_OnStart and Session_onEnd - which occur, of course, when a session is created and when a session ends (no matter the reason why it is brought to an end). Both of the events are placed in the Global.asa file. The Global.asa file is optional. The objects, methods and variables you declare in the Global.asa file are accessed by every page of your ASP application - so, if you have objects you need to use on every page, this is a great way of saving some time and many keystrokes. You can have only one Global.asa file for an application, and you must store it in the root directory of your application, or else the pages won't know where to look for it. You don't use <% %> in the Global.asa file - instead, you will have to place everything in the HTML <script> </script> tag. If you need to create new objects, you can do so here, using the <object> </object> tag.

2.17 Database Manipulation in ASP

You started learning ASP because you need to create a big, complex site, interactive and easy to update - otherwise, you can use the HTML and save all this trouble. So, if you have a lot of information, you need to store it someplace, and order it - so you'll end up putting it in a database.

ASP works well with databases, in fact, it was designed with this purpose in mind. You can modify the data in the database, and your web site will modify by itself, quickly and without messing up the rest of the application. For ASP, you can use any database you want, but, as it is Microsoft technology, it would probably be better to use Microsoft Access, which is not only fully compatible with ASP, but also easy to use, intuitive and it comes with every Microsoft Office CD. So, if you have Word and Excel on your computer, you have Access as well. Spend some time to get familiar with it. You will need to organize your data carefully, since the database will represent the "skeleton" of your entire application. If you want to, you can use MS SQL Server, Oracle, MySQL Databases or any other type of database you are familiar with, it won't make a big difference.

Then, you need to connect the database, and there are two methods to do with: with or without DNS (Data Source Name). If your site is hosted by a company (as is usually happens), you have to contact them and ask them to set up the DNS for you (you won't take them by surprise with this request, I'm sure they've heard of it before). You will have to tell them where your database is located, and the name chosen for your DNS.

In order to make the interaction with the database easier, Microsoft has created a group of objects - also known as the ADO technology (ActiveX Data Objects). There is a Connection Object, for the connection with your database, and a Recordset Object, for getting the data out of the database. These objects act just like the one we've seen before, and they have properties and methods too. For instance, the Connection object has methods such as Open, Close, Execute, and properties such as ConnectionTimeOut, State, Provider, Version. As you can see, they are in plain English, it's quite easy to understand what each of them does.

Besides using the RecordSet object, another method of adding records to the database is by using SQL statements. SQL stands for Structured Query Language and it represents the standard language that deals with databases. It is also quite intuitive, for instance, the statement you will be using in order to add a record is "insert". You will also find statements such as "select", "delete" and "update". With the "insert" statement, you will need to tell it the name of the database, the fields, or columns where you want the data inserted, and the actual data you want to insert, like this:

```
insert into database_name (column01, column02) values ("value01", "value02")
```

A database can store anything. You can ad pictures, HTML forms, anything else you see on the Internet today. It is important to keep them well organized, do not mix data that don't belong together, or are not of the same time, and pay close attention to the relations among them. The best design you can create for your page will not help the users, if they have problems accessing the content.

Once you start using databases, you actually use the ASP for the purpose it was designed. As your scripts get bigger and more complex, they are also a bit more difficult to handle. You can write a code separately, and then include it where you need it. For this, you have the #include directive, used for elements that you want to keep separately, for better organization, or that you want to re-use on several pages. The syntax is:

<!--#include file ="file_name"-->

2.18 Debugging & efficiency in ASP

All programmers have mistakes in their scripts, it's impossible not to make some. There are several methods to detect and correct them. Microsoft offers a Script Debugger which can be a useful tool, particularly if you've used Microsoft technology all over (ASP and Access, and the JScript or VB languages).

Note that you have to enable the Debugger, before you can start using it. It will allow you to do different things, such as to run only parts of your code, to see if they're working, or to check the evolution of the variables as the script develops. With this tool you can insert breakpoints, which will stop your script from running at a certain moment, so you can have a closer look of how it behaved until then. You cannot make modifications in your script directly in the Debugger, you need to open the editor in which you wrote the code lines initially, and to modify your code where it is.

When your code is not working, you should check for syntax mistakes, as they are the most common, and the most annoying. It can be anything, a typing mistake, a misspelled word, missing quotation marks, and so on. In most cases, if your script doesn't run at all, then you probably have a very simple, basic mistake.

If your script seems to runs smoothly, but its results are incorrect, then you probably have a logic mistake. These are a little trickier to handle, and it often means that your approach was mistaken from the very beginning, and you will have a lot of

modifications to make. Check your "loops" and "ifs", one of the most common logical mistakes is found in the conditions for the "ifs".

There are also run-time errors. This means that the script found an operation which is impossible to perform, and so it has to stop. You need to correct that error before your script continues working.

IIS5 also offers an object called ASPError. You can only use it with the method called Server.GetLastError(), and it will give you data about the problems encountered with the script. Its properties include Source (which returns the actual code line which caused the error), File (the name of the file that generated the error), Description (yes, this is actually a brief description of the error), Column (the column position within the file that generated the error) and Category (which attempts to tell you what type of error it was, if it was caused by an ASP script or by an object, and so on). If the error is ASP-related, you can also use the property ASPDescription, which returns a more detailed description of the error.

The ASPError object can help you set up a centralized way of dealing with errors. Sometimes you can't be there to correct the error, or maybe you want the script to send you an e-mail with the description of the error. This tool is very useful when you are dealing with large scripts and when you want more than one person to handle the errors. You can also use it to create some custom-made, user friendly error messages.

CONCLUSION

Perhaps the most obvious application for the ASP is that of setting up e-commerce sites, or web sites for newspapers and magazines, because it is so easy to handle security issues, multiple users and quick updates. You can also organize a download center, or web-based file manager, you can insert a poll or a virtual voting booth on your site, a guestbook, you can insert and control banners and ads, handle the statistics about the number of people visiting or even create your own online games.

REFERENCES

[1] http://www.w3schools.com/asp/default.asp

[2] http://www.programlama.com/sys/c2html/view.php3?DocID=2584

[3] http://www.programlama.com/sys/c2html/view.php3?DocID=1878

[4] http://www.dynamicdrive.com/dynamicindex6/popcalendar.htm

[5] http://www.w3schools.com/html/html_intro.asp

[6] http://www.softwareprojects.org/asp-tutorial-01.htm

[7] http://www.maththinking.com/boat/booksIndex.html