Chapter 2

The HTML/OS Language

In this chapter, you learn the HTML/OS language. You'll find the lack of data types, its English-like function names, and how it works hand-in-hand with HTML to be both powerful and refreshingly easy to learn. We start by looking at HTML/OS variables.

Variables

In HTML/OS variables carry data between program segments, HTML data-entry forms, database fields, and files. Variables have a universal format and can be created by writing instructions like `a=1`. In this case, the variable is a table consisting of a single cell. Variables can also be created with many cells. Commands that retrieve many cells, for example, include commands for loading records from databases and commands that read CSV files (delimited text files).

All variables in HTML/OS, strictly speaking, are tables of cells. The HTML/OS variable is a two-dimensional table with, at the very least, one column and one row. Cells in a variable can contain text, dates, numbers, etc. A cell containing text, for example, could be one character long or one million characters long. Either way, there's no need to define or declare the cell or the variable containing cells. You simply follow the data formatting conventions. The conventions follow.

How to Name a Variable

Variable names can contain letters, numbers, dots, and underscores. The variable names cannot begin with a number or contain special characters such as a space, a quote mark, a dollar sign, the equal sign, or a mathematical operator. Variables also cannot be the same as

functions used in the language. As a rule, it's best to use names like `abc_firstname` and `abc_lastname`. Preceding names with a prefix and an underscore achieves two goals: First it eliminates conflicts with function names, because functions in HTML/OS don't contain underscores. Second, it allows you to organize your variables better.

Assignment Rules

The equal sign character is used to assign values to variables. When a variable name is not immediately followed by `[ column,row]` or `[ column]`, the assignment replaces the content of the variable. When a cell is specified, the assignment applies to only the cell specified. For example, `x=2` and `x[ 1,1] =2` perform two different operations. The first erases any previous values of `x` and creates a one-column by one-row variable containing `2`. The second assignment puts `2` in the first cell keeping other cells, if any, the same.

Also, note that writing `[ column]` is the same as writing `[ column,1]`. In other words, when a row is not specified, it is equal to `1`.

The Default Value

The default value of any variable is a one-column by one-row variable filled with `ERROR`. This is equivalent to `NULL` in other languages. To clear a variable, set it to `ERROR`. It is a good idea to clear variables larger than 50,000 bytes if you do not need them. If you clear a variable before the end of a page, HTML/OS doesn't store it for use in successive Web pages.

Data Formatting Rules

Each cell of a variable can store text, numbers, times, and logicals. Although the variable types do not need to be declared, they do have the limits and required formats. These are listed in the sections that follow.

Text

Text is made up of sequence of eight-bit characters, provided it does not contain any `NULL` (ASCII of zero) characters. Such data may include Unicode and Binary without a `NULL`s (See BINTOTEXT in Aestiva's Knowledge Base at `http://www.aestiva.com/support`.) Text size is limited to 1 gigabyte.

Numbers

These include any positive or negative number, with or without a decimal point, up to 11 digits in length.

Dates and Times

Time is written as MM/DD/YYYY HH:MM where MM stands for month, DD stands for day, YYYY stands for year, HH stands for hours, and MM stands for minutes. When writing a time you can opt to write only the date portion, only the HH:MM portion, or both.

Note that some commands can perform date calculations only back to 1970. Also note that the date format can be relaxed. For example, only two-digit years and single-digit days or months can be used. In general however, it's best to stick to the standard format. The date format is the same whether you're writing to a variable, database, or cookie.

Logicals

Logicals are simply the words `TRUE` or `FALSE`.

Variable Indirection

To read or write to a variable whose name is stored in a second variable, precede the second variable's name with an `@` character. For example, `mydata=@my_var_name` or `@my_var_name = mydata`. This is known as variable indirection.

The Literal

Literal values should be surrounded with quote characters. The exception is when the value is a number, in which case quotes are optional. HTML/OS supports three quote characters: single-quote, double quote, and caret. Most languages support only single and double quotes. HTML/OS also supports the caret to make the storage of JavaScript and browser-side instructions easier. For example, the following instruction places a hypertext link in a variable:

```
abc_link = ^<A HREF=#onclick"window.open('http://www.yahoo.com');">Do You Yahoo?</A>^
```

Quotes in Quotes

In general, quoted text can contain quote characters, provided the quoted characters inside the text are not the same as the quoted characters that surround it.

Writing Commands

HTML/OS commands, known as Overlay Tags or Otags are written in one of two ways — as functions or multiword statements.

Functional Otags

`LENGTH` is an example of a functional Otag. You write the following to store the length of `abc_name` in `x`:

```
x=LENGTH(abc_name)
```

Here are more examples of functional Otags:

```
TODAY
UPPER(myname)
ADDDAYS("01/01/2000",500)
```

The parameters of functional Otags can be literal text, calculations, or expressions built with other functional Otags. Otags can have no parameters, one parameter, or many parameters. When more than one parameter is needed, they are separated from each other with a comma.

Operators

*HTML/OS supports the + (addition), - (subtraction), / (division), and * (multiplication) math operators. For example, you can write* `x = y - 1` *or* `x = x + SQRT("4")`*. The elements in math calculations can be combined with parentheses using standard math conventions. In HTML/OS the* `+` *(addition) operator pastes text together when any of the operands are non-numeric. Thus,* `"a" + "3"` *gives you* `a3`*. If you wish to paste* `1` *and* `2` *together so they produce* `12`*, use the* `&`*, the string concatenation operator in HTML/OS.*

Multiword Otags

Multiword Otags use multiple function names in a particular sequence to express an action. The `IF-THEN` statement is one example, such as:

```
IF abc_firstname= "" THEN
```

```
    error_message = "Please enter first name."

    GOTO PAGE

/IF
```

The IF-THEN statement uses the words IF, THEN, and /IF. Because these kinds of Otags require more than one predefined function name, they're known as multiword. Here are more examples of multiword Otags:

```
COPY FILE="myheader.txt" TO myheader /COPY

MAIL mytxt TO ADDRESS="bill@yahoo.com" SUBJECT="My News" /MAIL

DISPLAY FILE="sidebar.txt" /DISPLAY
```

The parameters of multiword Otags are set according to the structure of the tag. Most parameters of Otags, whether functional or multiword, can be built from complex expressions involving other functional Otags. In other words, functional Otags are nestable. Multiword Otags, on the other hand, are not necessarily nestable. IF-THEN statements, FOR loops, and WHILE statements are nestable, for example, but tags such as COPY, APPEND, and MAIL are not.

Command Expansion

Instructions can be stored in files. To insert instructions stored in a file into an Overlay, use EXPAND. For example, you can write <<a=2 EXPAND FILE="more.txt" /EXPAND>>. This expands the instructions placed in more.txt into an Overlay, running them immediately after the a=2 instruction. This capability is similar to the include concept popular in the C programming language.

Conditionals, Loops, and Functions

HTML/OS supports standard IF-THEN statements, FOR loops, WHILE loops, and FUNCTIONS. Each is described in the sections that follow.

The IF-THEN Statement

IF-THEN statements in HTML/OS begin with IF followed by a test and the word THEN. They are always terminated with an /IF. They support an optional ELSE word and one or more ELIF words. The IF-THEN statement looks as follows:

```
IF test THEN instructions
```

```
    ELIF test2 THEN instructions
    ELSE instructions
/IF
```

The tests should result in a `TRUE` or a `FALSE`. They are built from combinations of expression-operator-expression subtests combined with `AND`, `OR`, and `NOT` logicals. For example, `(A > B) AND (B > C)`. The following test operators are supported:

- `=` Equal
- `!=` Not Equal (Same as `<>`)
- `>` Greater than
- `>=` Greater than and Equal
- `<` Less than
- `<=` Less than and Equal
- `~` Begins with
- `~~` Contains

An extra equal sign may be added to force case-sensitive operation as in the following examples:

- `==` Case Sensitive Equal
- `!==` Case Sensitive Not Equal (Same as `<>=`)
- `~=` Case Sensitive Begins with
- `~~=` Case Sensitive Contains

The FOR Loop

HTML/OS supports two different kinds of `FOR` loops. The standard `FOR` loop is written as follows:

```
FOR NAME=name VALUE=value1 TO value2 STEP=value DO
  instructions
/FOR
```

The loop repeats the instructions, starting with *`value1`* and ending with *`value2`*. The `STEP` is optional. When it's not specified, a `STEP` value of `1` is assumed. Although popular in most computer languages, this kind of loop is not that popular in HTML/OS, because HTML/OS has a `FOR` loop that loops across two-dimensional variables. The second kind of loop is easier to use and is written as follows:

```
FOR NAME=name ROWNAME=rowname DO
   instructions
 /FOR
```

Unlike the standard FOR loop, all you specify here is the variable name and a temporary variable name to use inside the loop. Note that end points do not need to be calculated. Each time through the loop, *rowname* is filled with the next one-row slice of *name*.

For example, consider *mycolors*, a variable with a color code in column one and a color name in column two.

```
FF0000   Red
0000FF   Blue
00FF00   Green
000000   Black
FFFFFF   White
```

To convert *mycolors* into the following:

```
<option value="#FF0000">Red
<option value="#0000FF">Blue
<option value="#00FF00">Green
<option value="#000000">Black
<option value="#FFFFFF">White
```

you can write:

```
FOR NAME=mycolors ROWNAME=x DO
     DISPLAY
         '<option value="' + x[1] + '">' + x[2]
     /DISPLAY
  /FOR
```

When using the standard FOR loop, you would write:

```
j=ROWS(mycolors)
FOR NAME=i VALUE=1 TO j DO
     DISPLAY
         '<option value="' + x[1,i] + '">' + x[2,i]
```

```
    /DISPLAY
  /FOR
```

The WHILE Loop

HTML/OS supports the standard WHILE loop. The loop begins with the word WHILE followed by a test, the word DO, and one or more instructions. It is terminated with /WHILE. It's written as follows:

```
WHILE test DO
   instructions
 /WHILE
```

The *test* between the WHILE and the DO is written the same as the test used in IF-THEN statements. Also note that WHILE and FOR loops accept the use of a BREAK tag. The BREAK is placed within the instruction portion of the tags. When encountered, execution flows to the instruction immediately following the /FOR or /WHILE.

Defining New Commands with FUNCTION

In HTML/OS, you define new Otags with the FUNCTION tag as follows:

```
FUNCTION Otag(param,param,..)
   LOCALS param,param,..
 DO
   instructions
   RETURN calculation /RETURN
 /FUNCTION
```

The LOCALS and the RETURN words are optional. When a RETURN is not defined, a one-by-one variable containing nothing is returned. Parameters passed to a FUNCTION and any additional variables defined in the LOCAL section stay local to the function, meaning they exist only inside the function. If you are familiar with recursion techniques, feel free to deploy them here. The HTML/OS function can call itself to any depth.

As an example of a function definition, suppose you want to define a command called `dp_nohtml` that strips text of < and > characters, replacing them with `<` and `>` characters. Taking advantage of the search-and-replace tag called `REPLACEALL`, you could write the following:

```
FUNCTION dp_nohtml(x) DO
   x=replaceall(x,">","&gt;")
   x=replaceall(y,"<","&lt;")
   RETURN x /RETURN
 /FUNCTION
```

You now have a function `dp_nohtml(x)` that strips an input of < and > characters. The function will work on the page in which it is defined and every successive page in your user's session unless it is redefined, in which case, the new function is used.

Working with HTML Documents

In the previous sections, you learned how to write variables, functions, and statements. Now, you learn how to embed the language in HTML documents.

Specifically, language segments are placed between << and >> characters. The brackets separate your programming from the rest of the HTML document. These sections of code, known as Overlays, can be placed throughout the document.

The following Web page shows a document overlaid with programming. The `<<DISPLAY TODAY /DISPLAY>>` is an Overlay that displays today's date. The Overlay `<<x=RANDOM(100)>>` places a random number between 1 and 100 in the variable `x`. It does not display anything. The commands `DISPLAY`, `TODAY`, and `RANDOM` are examples of Overlay Tags, also known as Otags.

```
<html>
<title>Sample Document</title>
<body bgcolor=#FFFFFF>
Today is <<DISPLAY TODAY /DISPLAY>><br>
<<x=RANDOM("100")>>
My random number is <<DISPLAY x /DISPLAY>><br>
</html>
```

The document contains HTML, which is rendered by the browser, and Overlays, which are entered on the server. If you come from a PC-based programming world where the programming is rendered on the same computer, this can be a bit confusing. Web programs perform tasks on the server and also on the browser. Programming happens in two steps. In the first step, an HTML document is created. That step happens on the server. In the second step, the HTML document is rendered by a Web browser. Let's look at these two steps in detail.

The server side—When a Web page is run, the HTML/OS engine reads the HTML document from top to bottom, passing over HTML and Overlays. This is known as the server side. When HTML/OS finds an Overlay, it runs it. The Overlay may change that part of the document, fill a variable with a value, or perform other actions. When the HTML/OS engine finds HTML outside of an Overlay, it uses it to fill in the rest of the document. When it's done, HTML/OS takes the resulting document and sends it to the user requesting it.

The browser side—Now the browser side starts. Here the HTML and JavaScript, Java applets, Flash, or client-side plug-ins are processed. This step produces the user screen. Because Overlays are interpreted on the Web server, the documents reaching the browser have no Overlays in them. They're pure HTML. It's why HTML/OS programming works, is robust in that it works across all kinds of browsers, and can be reliably used to program Web products. This robustness does not exist in browser-side languages, such as HTML. There you need to be more careful. See the note titled "Get an Attitude."

Get an Attitude

It is important to note the difference in attitude you must have when programming the server side versus the browser side. The server-side programming is not browser-dependent and contains a well-defined instruction set. You are free to exercise the programming to its full extent. This could not be more untrue of the browser side. You often have little or no control over the browser used, the version used, the configuration of the browser used, or the memory and hardware on which the browser runs. You can't test these factors thoroughly, either. Also, client-side languages, such as JavaScript, change. Using anything more than basic JavaScript puts your product at risk when new browsers are introduced. The attitude you must take when programming on the browser is one of extreme caution. It's best to keep client-side programming to a minimum.

Getting back to the server side, Overlays can be placed in different parts of a document. Depending on how they are placed, they'll run before the page is run, while the page is run,

or when they are specifically requested by a user clicking a hypertext link or submitting an HTML form.

The first Overlay in a page, provided it is above an `<HTML>` tag, is called an Underlay. It is run prior to the page being built. The next Overlay or any Overlay in the display portion of an HTML document (between the `<html>` and `</html>` tags of an HTML document) are sometimes called Inlays. They run when the page is being built. If the Overlay is placed after the `</HTML>` part of the page, below the display part of the page, the Overlay is run upon request. These Overlays are called On-click Overlays, because they don't run until they're called. Also, unlike the Underlays and Inlays, On-click Overlays are named. They always begin with `<<overlay` followed by the name of the Overlay. On-click Overlays run when a hypertext link or HTML form that references that Overlay is clicked.

The following code example shows a document with an Underlay, an Inlay, and an On-click Overlay:

```
<<IF ISLOGGEDIN="TRUE" THEN GOTO "menu.html" /IF >>
<HTML>
This is my web page.<BR>
Today's date is: << DISPLAY TODAY /DISPLAY >><BR>
Type password to enter my site;
<FORM METHOD=POST ACTION=letmein>
<INPUT TYPE=PASSWORD SIZE=15 NAME=mypassword>
<INPUT TYPE=SUBMIT NAME=mybutton VALUE="LOGIN">
</FORM>
</HTML>
<<OVERLAY letmein
IF mypassword="12345" THEN
ISLOGGEDIN="TRUE"
GOTO "menu.html"
ELSE GOTO PAGE
/IF
>>
```

The Underlay checks to see whether the user has logged in using the variable `ISLOGGEDIN`, which would equal `TRUE` had the user already logged in. Then an Inlay displays today's date. Below that, an HTML form is linked to the On-click Overlay called `letmein`. When

the submit button is clicked, the Overlay is called. It checks the password entered. If it's equal to `12345`, `ISLOGGEDIN` is set to `TRUE` and the user is sent to `menu.html`.

Some Rules Concerning Overlays

- *You can have at most, one Underlay on a Web page, but as many Inlays and On-click Overlays as you want. If you have more than one Underlay, the second is treated as an Inlay.*

- *HTML/OS commands run in any kind of Overlay. However, there are two noteworthy exceptions: Tags that display HTML in the page can be used only when you're in the display portion of the page—in other words, in Inlays. Second, the* `GOTO` *tag, a tag that redirects users from one page to another, can be used only in Underlays and in On-click Overlays. It does not make sense to redirect programming to another page while a page is in the middle of being displayed, so redirection is not allowed in Inlays.*

- *An Overlay with a single calculation is a shorthand way of displaying the results of the same calculation with a* `DISPLAY` *command. For example, writing* `<<TODAY>>` *or* `<<a+1>>` *is the same as writing* `<<DISPLAY TODAY /DISPLAY>>` *or* `<<DISPLAY a+1 /DISPLAY>>`*, respectively.*

- *Overlays cannot contain other Overlays. For example, you cannot write* `<< a=1 b=2 c=<<a+b>> d=4>>`*. Instead, you would write:* `<<a=1 b=2 c=a+b d=4>>`*.*

- *HTML can contain Overlays. For example, it's okay to write:* `<img src=<<DISPLAY myimage /DISPLAY>> border=0>` *or* `<img src=<<myimage>> border=0>`*. (See the third rule.)*

Hypertext and Form Links

The way you link to another Web pages depends on how you write the URL in a hypertext link or HTML form. Depending on how you write the URL, you may link to another Web page, execute an On-click Overlay at the bottom of the page, or exit HTML/OS entirely.

Linking to Other Pages in HTML/OS

To link to a page in `HTML/OS`, specify the page name. For example, `<A HREF="/abc.html">` or `<FORM METHOD=POST ACTION="/abc.html">`. The pages must end with `HTML` or `HTM`.

Linking to On-click Overlays

To link to an On-click Overlay at the bottom of the page, specify the name of the On-click Overlay. For example, `<A HREF="abc">` or `<FORM METHOD=POST ACTION=abc>`. On-click Overlay names must not contain any dots.

Links That Exit HTML/OS

To exit HTML/OS and link to a URL, begin the URL with `http://`. This tells HTML/OS you wish to exit the HTML/OS environment, leaving behind your variables as you move to your next destination. Alternatively, you can exchange the `HREF` or `ACTION` tags in the hypertext and form statements with `EXITHREF` and `EXITACTION` tags, respectively. This is another way to tell HTML/OS to exit the environment.

Linking to Pages on Other Copies of HTML/OS

To move to a Web page located on a different copy of HTML/OS, precede the name with its network drive name and a colon, such as `X:/cart.html`. This tells HTML/OS to stay in the HTML/OS environment and keep all variables, but move to the document in the designated copy of HTML/OS. Note that linking to other pages is allowed only if both copies of HTML/OS allow access to each other. This is set up in the Control Panel of each copy of HTML/OS.

Links with Name-Value Pairs

Hypertext links that stay inside HTML/OS can have a name-value pair specified in them. For example, `<A HREF="/abc.html" NAME=X VALUE=3>`. When the link is clicked, the variable `X` is set to `3`.

Only one name-value pair can be attached to any link. To pass multiple parameters in a link, place the different values in a single variable and split up the variables in an On-click Overlay at the bottom of the page. For example, you can write `<A HREF=splitit NAME=X VALUE="3|22|5">`. The On-click Overlay can be written as `<<Overlay Splitit x=split(x,"|") a=x[1] b=x[2] c=x[3] GOTO "/abc.html">>`. The `LISTTOROW` tag converts the pipe-delimited list into a three-column by one-row table. The successive assignments store cells in variables. Clicking the link sets `a`, `b`, and `c` to `3`, `22`, and `5` respectively and sends the user to the page `/abc.html`.

Secure Links

To link using Secure Socket Layer (SSL), make sure SSL is set up in your Control Panel, and then add the word `SECURE` to your `HREF` or `FORM` link, for example, `<A HREF="/abc.html" SECURE>` or `<FORM METHOD=POST ACTION=abc SECURE>`.

Forms and Their Variables

HTML/OS variables automatically set values of variables in HTML form components. For example, writing the following displays a pull-down menu with Alaska preselected:

```
<<mystate="AK">>
<select name=mystate>
<option value="AL">Alabama
<option value="AK">Alaska
<option value="AR">Arkansas
</select name=mystate>
```

There is no need to use the HTML form tag `SELECTED` to set the default. For example, writing, `<input type=text NAME=myname size=15>` displays an input box with the value of the variable `myname`. There is no need to set the `VALUE` in the form component. In HTML/OS, form elements reflect the variable environment.

As an example, consider a large HTML form. You may want to split it across multiple HTML documents. In HTML/OS, you split the HTML form into multiple forms, placing the smaller forms on different Web pages. As a user navigates between these pages, data the user enters stays intact. No special programming is required.

Indexed and Automatic Fill

Names used in HTML forms can address specific cells in arrays. For example, you can write `<INPUT TYPE=TEXT NAME=myvar[ 2,3] >`. When the form is submitted, column 2, row 3 of the variable `myvar` is set. This is known as the Indexed Fill of a variable.

You can also write to arrays by specifying input variables repeatedly. For example, if you were to write the line `<INPUT TYPE=TEXT NAME=myvar>` three times in an HTML form, when the form is submitted, the data placed in the three input components would be saved to `myvar`. `Myvar[ 1,1]` would get the first input. `Myvar[ 1,2]` would get the second. And `Myvar[ 1,3]` would get the third. This is known as the Automatic Fill of a variable.

Launching Applications

To launch an HTML/OS application, you need a special link that launches the first page of your application. This link is known as a Start-link. When it's clicked, a new user session is

created, the Aestiva HTML/OS engine is launched, and the first page of the application is processed. Start-links are written as follows:

```
HTTPSERVICE://DOMAIN/CGI/ENGINE/FIRSTPAGE
```

Each item in uppercase should be replaced with its corresponding value. The `HTTPSERVICE` is where an `http` or `https` should be written. `DOMAIN` is where your domain name should be written, for example, `www.mysite.com`. `CGI` is where your cgi-alias should appear, for example, `cgi-bin`.

The `ENGINE` is the name of the HTML/OS engine. It is typically `start.exe` on Windows systems and `start.cgi` on others. Finally, the `FIRSTPAGE` is the HTML/OS name of the page you want to access, for example, `/apps/mypage.html`.

Your entire Start-link, other than the `FIRSTPAGE` part at the end, is called a Start-path. Start-paths do not change after a site is set up. The URL to HTML/OS pages from outside of HTML/OS is the page name in HTML/OS preceded by your Start-path. (See the accompanying note, "What's Your Start-path.")

What's Your Start-path?

Don't know your Start-path? Here are three ways to find it:

- *Look at the URL to your login screen. The Start-path is that URL minus the `/login.html` portion at the end.*
- *Click the diamond of any icon on your HTML/OS desktop. Next to the security option, you'll see a link called Start-link. Click it. It will show you the Start-link to that application. The link, minus the HTML/OS page at the end, is your Start-path.*
- *Run the following instruction in an HTML/OS page:* `DISPLAY SYSSTARTLINK(page) /DISPLAY` *. This gives you the Start-link to the current page. The URL minus the page name is your Start-path.*

Passing Variables in Start-links

To pass variables to an HTML/OS application, you can put them in the Start-link URL. To do that you use standard HTML techniques. You place `NAME-VALUE` pairs at the end of the Start-link's URL, as follows:

```
http://www.mysite.com/cgi/start.cgi/mypage.html&var1=a&var2=b
```

The URL, when enabled, launches the page `mypage.html` and passes into it the two variables, `var1` and `var2`, with the respective values `a` and `b`.

Allowing Access to Applications and Start-link Variables

To test a Start-link, enter it in the address line of your browser. If you are denied access, you have not enabled a Start-link. HTML/OS uses Security-by-default. Start-links need to be enabled before they can be used. To enable a Start-link, use your Control Panel. Under Security, you'll find an option called Start-link Allow. Add the Web page specified after your Start-path. Allow the list and try the URL again. Now it should access the page.

The variables used in Start-links need to be enabled. Otherwise, the values are ignored. To enable the use of a variable, use your Control Panel. Under Security, you'll find an option called Variable Allow. Add the variable names you want to use. Now the variables can be used in Start-links.

Application Forking

Sometimes you want to launch an application with a new user session that no longer interacts with the current user session, but begins with the original session's variables. This situation often occurs when you are popping up windows.

To accomplish this, you fork the new application. A `FORK` is a new user session that inherits the current user session. For example, `<A FORK=”/mypopup.html” TARGET=”NewWin>Popup Window</A>` is a link that forks the application `/mypopup.html`.

A `FORK` link looks like an `HREF` link, with the `HREF` replaced with the word `FORK`. Activity in forked user sessions has no effect on the parent application until data is passed back to the parent.

Chapter Summary

This chapter reviewed the HTML/OS language and how it integrates with HTML. The language is easy to learn, but different from others in that it's a computer language made specifically for the Web.

The next chapter looks at databases. Together, these two chapters give you the fundamentals needed to progress through the rest of this book.