

LESSON 4

Creating Dynamic Pages

OBJECTIVES

You will be able to:

- describe Dynamic HTML (DHTML).
- describe the Document Object Model (DOM) and its uses.
- identify differences in browser implementation of DHTML.
- demonstrate some uses for DHTML.

In this lesson, you will be introduced to the principles and fundamental elements of Dynamic HTML (DHTML). You will learn about Document Object Models (DOM) and how style sheets and scripting languages interact with DOM to create DHTML. You will examine and create some simple DHTML applications and explore some more sophisticated illustrations. This lesson is not intended as comprehensive instruction in DHTML. It is only intended to acquaint you with some of the potential of DHTML.

Overview of Dynamic HTML

Dynamic HTML (DHTML) is a relatively new term when it comes to creating Web pages and can have different meanings depending on who is using the term. The major browser companies, Microsoft and Netscape, have their own definitions as does the World Wide Web Consortium (the international body that oversees the HTML standard). What DHTML boils down to is greater control for the HTML author over page elements. This results in new possibilities for page layout and in Web pages which can change, based on user action and input, without having to access the server.

This is different from “regular” HTML pages, which are basically static. You can click on a hyperlink, but that simply opens another page. Once the page is opened, it will not change. The DHTML instructions, just like the HTML tags, are part of the page. This is what allows the page to change without having to access the server. Instead, events in the browser are interpreted according to instructions contained in the page’s source code. The page can actually rewrite itself and be reinterpreted by the browser for display all without being reloaded from the server. Want proof? Here’s an amusing example which, though perhaps a bit silly by itself, presents a capability with many interesting applications.

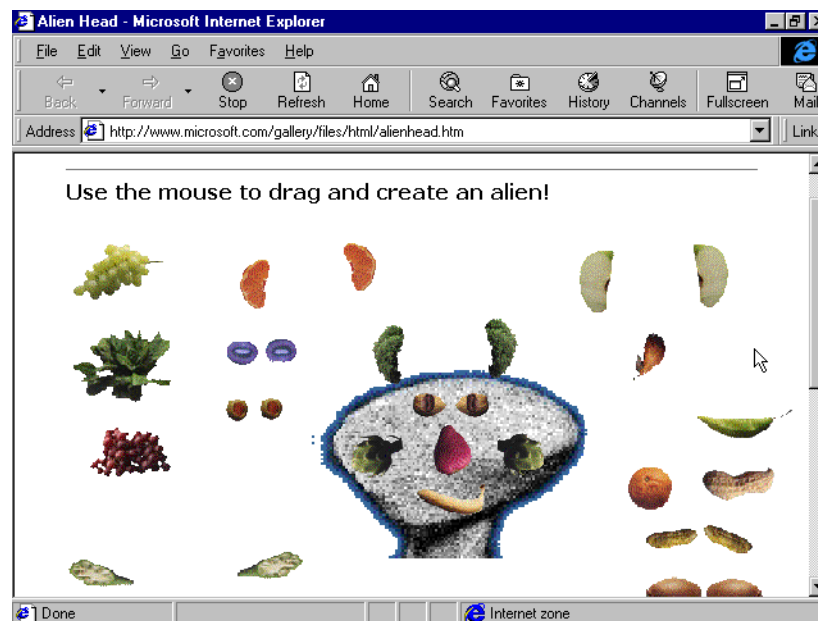


Figure 4-1: <http://www.microsoft.com/gallery/files/html/alienhead.htm>

DHTML allows you to set exactly where objects will appear in the browser window. Moreover, it will allow those objects to move in response to specific events or even be moved by the user. It will allow text or graphics on the page to be changed or even replaced completely, perhaps even by input from the user. And DHTML pages can become “aware”, remembering what a user has done or dynamically updating content from a data source. The application of these advantages requires imagination and some no small skill in scripting languages, but the result can be a much richer web environment than possible previously.

DHTML works by defining objects on the page, then applying scripts to those objects that define how the objects will act. This identification of object is known as *Document Object Model* (DOM). The scripts are created in scripting languages, such as JavaScript. The objects might be the headings, an image, a paragraph or any other page element as identified in the DOM. Because of this, style sheets are integral component of DHTML. Styles of text objects can be set according to browser capabilities or user actions. For example, entire paragraphs may be caused to appear or disappear as a result of a mouse click, thus collapsible outlines or directory trees can be created.

When DHTML-enabled browsers access a page, they “catalog” the tags on the page and when the user interacts (e.g., passes the mouse over) with an object, the script is invoked. There is a big difference, however, in how the two major browsers interpret DHTML. Netscape Navigator has a few of its own tags, such as <LAYER> that allow individual or groups of HTML elements to become an object, but does not interpret many other HTML tags as being objects. Microsoft Internet Explorer is actually much closer to the HTML 4.0 standard, which does allow most HTML tags to become objects. For example, you might apply a script to an unordered list , but because Netscape Navigator does not recognize as an object, the script is never invoked. Therefore, great care must be taken in using DHTML; you may have to provide alternative pages or risk having many users left out due to the version or brand of their browser.

As stated earlier, this course has used the Internet Explorer 4.x browser from Microsoft because it more closely adheres to the standards for style sheets and DHTML. The discrepancy between IE 4.x and Navigator 4.x is particularly extreme in the case of DHTML. Many of the pages IE 4.x can display either must be authored differently for Navigator 4.x or may not be able to be duplicated at all. Where possible, alternatives for Navigator are provided. But as no current browser interprets all of the new elements, IE 4.x has been used for this preview as it is closest to a full implementation.

There are many good resources on the WWW to help you learn about DHTML and track the latest developments. Both Netscape and Microsoft provide reference materials and development examples at the URLs listed below. Throughout this lesson, you will learn of a variety of sites which will provide more information and demonstration of applications of DHTML.

- <http://www.microsoft.com/workshop/author/dhtml/>
- <http://developer.netscape.com/one/dynhtml/index.html>

The Document Object Model (DOM)

The concept behind a Document Object Model (DOM) is relatively simple. It defines the different parts of a document and describes the relationship between these parts. If it doesn't immediately seem clear that a document even has parts, it may help to think of it this way. A document is made up of paragraphs. Those paragraphs are made up of sentences. The sentences are made up of words. You could even carry it all the way out to the letters which make up the words. Each of these parts might be considered as objects in the document. In this example, the relationships are clear. The words are part of the sentence. The sentence could then be called the *parent* of the words, each one being a *child* of that sentence. The sentence is then a child of the paragraph which contains it. The various paragraphs are therefore the children of the parent document. Each child object belongs to some parent object.

By its very nature, HTML identifies parts of a document. There is a HEAD and a BODY. In the HEAD may be a TITLE. In the BODY may be headings, paragraphs, tables, forms, lists and more. Not only does HTML identify these parts, but it also names them through the use of the HTML tags. Further, many of these HTML tags have attributes like the BGCOLOR attribute of the BODY tag. The background color can be thought of as a property of the BODY. In HTML, to create a red background for some DocumentX, the naming structure looks like this:

```
<BODY BGCOLOR= "red">
```

What a DOM does is create a way to address these objects and their properties. Under a DOM, the same objects and properties in DocumentX might be written this way:

```
documentx.body.bgcolor = red
```

In this example, object "documentx" has a child "body" which has a property "bgcolor" which has been set to a value of "red". Clearly, it seems that we could change the background color by setting this value to "blue".

But if HTML can already identify objects and set their properties, why create another system? Because since such a system can be understood by scripting languages, like JavaScript, you are now able to new levels of intelligence and interactivity through the user's browser. Plain old HTML now becomes Dynamic HTML! The DOM is therefore vital to the creation of Dynamic HTML.

The implementation of a DOM in a web browser actually first occurred in 1995 with Netscape Navigator 2.0 to support the introduction of LiveScript/JavaScript. Its only application at the time appeared to be the validation of form data. When a form was submitted, fields could be checked to see if they contained the right kind of data. This DOM identified the form as an object as well as the child objects inside the form, such as the radio buttons or text fields.

Work on the creation of style sheets in the HTML standard encouraged thinking about the different ways to identify and name elements in a document. You may remember that Netscape did not originally support the CSS standards. Therefore, two different DOMs emerged. The most obvious example is the use by Netscape of an object called

LAYER to group objects together in a document, which is not recognized by current HTML standards. Other browsers are using DIV, an accepted HTML element for this purpose. This and other differences in the DOMs mean that many pages are not compatible with both major browsers.

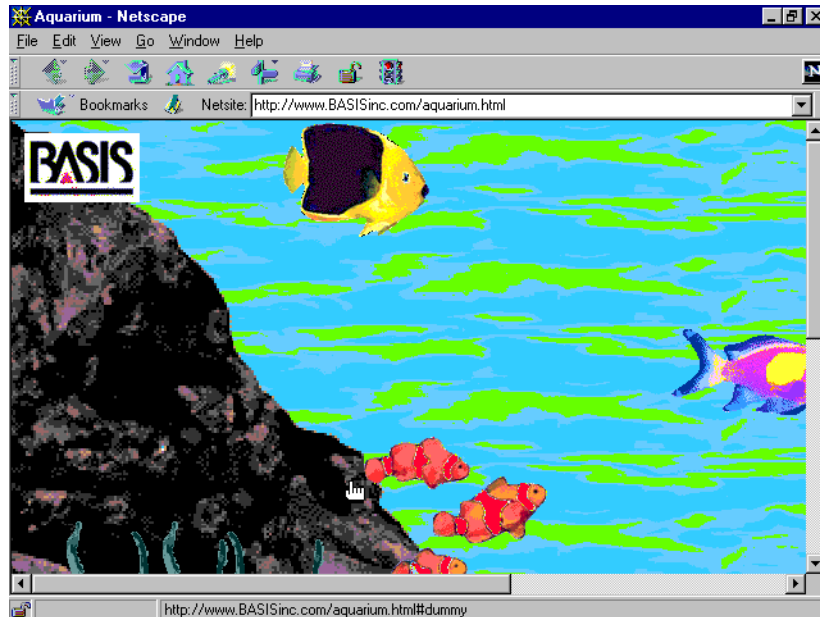


Figure 4-2: <http://www.BASISinc.com/aquarium.html> in Navigator

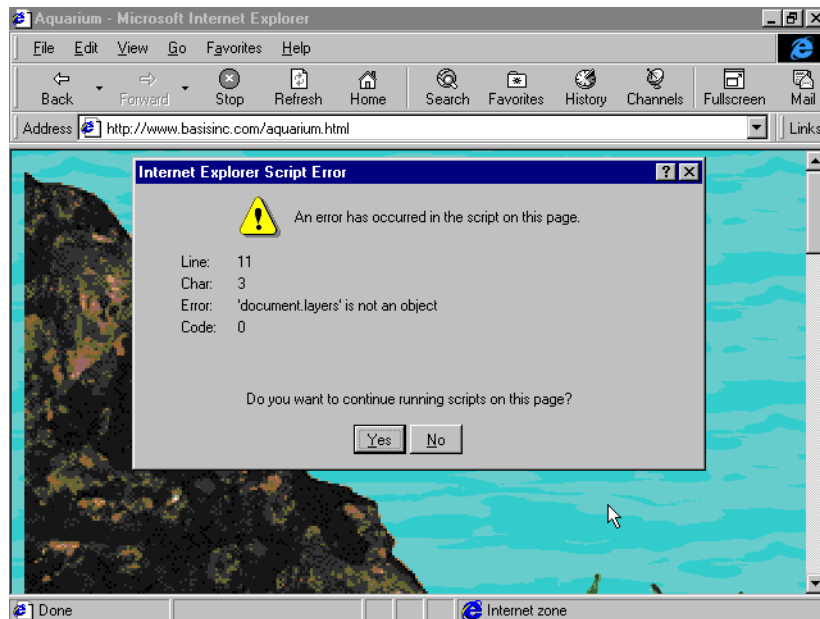


Figure 4-3: Internet Explorer Doesn't Recognize "Layers"

The good news is that Internet Explorer comes close to supporting the full complement of HTML 4.0 elements. The same can be said for IE's support of JavaScript. (It's also the only major browser at this writing to support any version of the newly proposed Extensible Markup Language, know as XML.) Therefore, it is the preferred browser for DHTML. The bad news is that there are a lot of other browsers out there, so implementing DHTML is not practical in settings where you are trying to reach a broad audience.

There is a brief, but informative, introduction to DOM at the WebCoder site. It is also a fine example of DHTML. As you move through the tutorial, you will view a number of apparently unique screens. However, all of this content is contained in one HTML documents. Individual page elements are shown, hidden and moved as the user moves through the information.



Exercise 4-1: DOM Tutorial

In this exercise, you will view a DOM tutorial both to illustrate DHTML in action and to review the fundamentals of the Document Object Model.

1. If you have an Internet connection, open the document located at <http://www.webcoder.com/howto/15/index.html> in your web browser or the location as provided by your instructor.
2. Follow the instructions as presented.

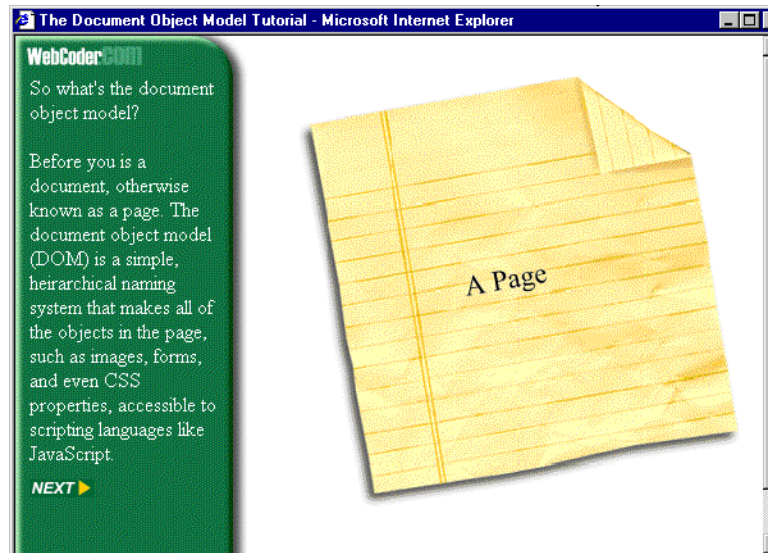


Figure 4-4: <http://www.webcoder.com/howto/15/index.html>

DHTML Syntax

The first step in creating DHTML applications is to identify and name the objects to be included in a page. Of course, you probably will not identify every page element. You'll only specify those needed for your purposes.

With HTML 4.0, there is now a new attribute, ID, for almost every tag. Through the ID attribute, you may give each individual HTML element a unique name, creating the objects used in DHTML applications. In the case of textual content, you may use the style and class names you used with Cascading Style Sheets (CSS) to distinguish one block of text from another. Therefore, anything on a page can be identified and named.

Here are a few examples. The first names a form field as "T1". The second uses the ID attribute of the DIV tag to name a block of text as "maintext". The third uses the CLASS name from a style definition to name a block of text as "Item".

```
<INPUT id=T1 type=text style="width: 400">
```

```
<DIV id=maintext>
```

Gestalt Systems, Inc. is a computer training company headquartered in the Dulles Technology Corridor, just outside of Washington, D.C., that provides information technology education and training services to end-users, technical staff, and business executives.

```
</DIV>
```

```
<SPAN class=Item>Home</SPAN>
```

Once identified and named, you can perform operations on these objects, replacing them, moving them or setting their properties. This requires the use of the hierarchy of objects as described in the Document Object Model (DOM) as described above. In response to a user moving a cursor over a word, you might set the color of a named block of text to red. But in order to do more than simply react to events in the browser, the scripting languages, like JavaScript become vital. Now you have the ability to evaluate the user's input and respond accordingly. These are the building blocks of some very complex web-based applications.

Creating and Using DHTML

There are three building blocks of DHTML: styles sheets, scripting languages and the DOM as implemented by the browser. What you can create depends on your skill and imagination. The only real limit is the implement of these three components by the browser. Unfortunately, those limitations are still severe.

The following exercises will demonstrate solutions based on CSS1 plus JavaScript and the DOM of the Internet Explorer 4.x browser. You will build some simple applications and examine some more complex examples. You will also get to examine some applications as implemented in alternative versions for Navigator 4.x and Internet Explorer 4.x browsers. As you will see, DHTML is certainly worthy of a course of its own, but you should get a glimpse of the possibilities.



Exercise 4-1: Replacing Text

In this exercise, you will complete an application that solicits input from a user and replaces existing text with that input.

1. Open the **replace.htm** document in your text editor.
2. Type the text as shown below to create the content to be replaced.

```
<H2>Replacing Text with User Input</H2>
<HR>
```

```
<DIV id=ReplaceMe >What you type in the field below will replace this
sentence.</DIV>
```

```
<BR>
```

```
<INPUT id=T1 type=text style="width: 400">
```

```
<P>
```

```
<INPUT type=button value="Click me to change"
onclick="ReplaceMe.innerHTML = T1.value">
```

3. Save the changes.
4. Open the **replace.htm** document in your web browser.
5. Type a phrase or sentence of your choice into the field.
6. Click once on the **Click me to change** button.

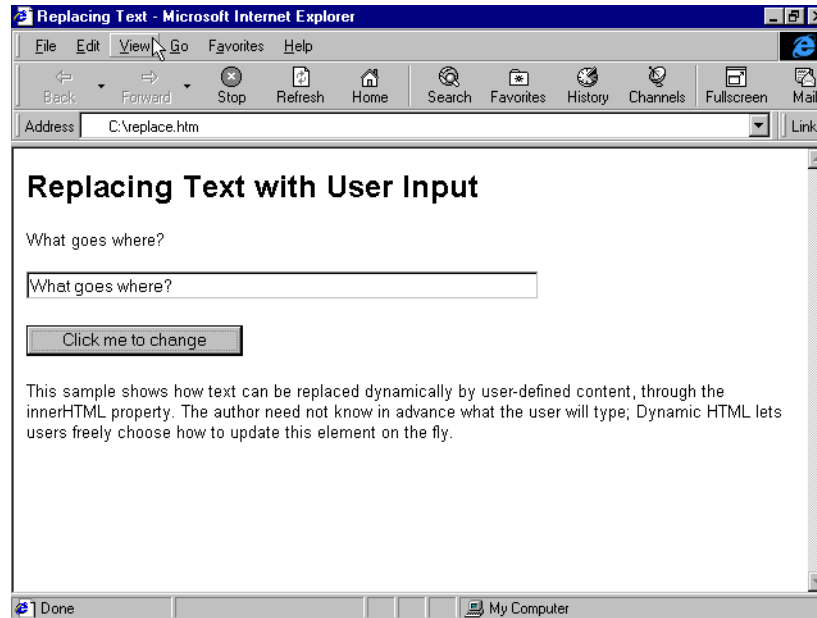


Figure 4-5: Replacing text

To reset the field, simply refresh or reload the document.

This application has three parts. The part you created establishes the target or placeholder for the text the user will type. It uses the DIV tag to create a named object of the document through the ID attribute. This object will now be recognized by the browser by the name "ReplaceMe".

<DIV id=ReplaceMe >What you type in the field below will replace this sentence.</DIV>

The second part of the application creates an object for user input. In this case, the INPUT tag is used outside of its usual function as part of a form. The type is set to provide a text field and the size of that field is set with a STYLE attribute. Most importantly, this field is named "T1", again using the ID attribute.

<INPUT id=T1 type=text style="width: 400">

Finally, the action of the application is provided through the use of another form INPUT, in this case a button. The button is augmented with a little JavaScript statement to handle the user event, a mouse click. Using a property called "innerHTML", all the content between the HTML tags of the DIV object called "ReplaceMe" will be replaced with the "value" of the object called T1 when the button is clicked. In this case, the value of that object is the text the user typed. The equation is read as "object.property=value" or in plain English, "The property of this object should be set to this value."

<INPUT type=button value="Click me to change" onclick="ReplaceMe.innerHTML = T1.value">

Though this is obviously not a very useful application, it does illustrate your ability to create objects, response to user actions and change page elements through the setting of the properties of these objects.



Exercise 4-2: Simple Text Animation (Rollovers)

In this exercise, you will create a simple text animation called a “rollover.” You will use STYLE class definitions to define alternative views of text elements based on user actions.

1. Open the **rollover1.htm** document in your web browser.
2. Move your mouse over the three items in the list.

You should observe no changes.

3. Open the rollover1.htm document in your text editor.

Note the style definitions at the top of this document.

4. Add the text as shown below to each of the list items.

```
<H1>Learn more about our company</H1>
<UL>
<LI>
<A
  HREF = "#"
  CLASS = "off"
  onMouseOver = "this.className ='on';"
  onMouseOut = "this.className = 'off';">Customers
</A>
<LI>
<A
  HREF = "#"
  CLASS = "off"
  onMouseOver = "this.className ='on';"
  onMouseOut = "this.className = 'off';">Products
</A>
<LI>
<A
  HREF = "#"
  CLASS = "off"
  onMouseOver = "this.className ='on';"
  onMouseOut = "this.className = 'off';">Technologies
</A>
</UL>
```

We have presented the HTML with each of the tag attributes on separate lines only for the purposes of clarity. As with any HTML text, the line breaks as shown have no impact.

5. Save the changes.
6. Reload the **rollover1.htm** document in your web browser and again move your mouse over the list items.

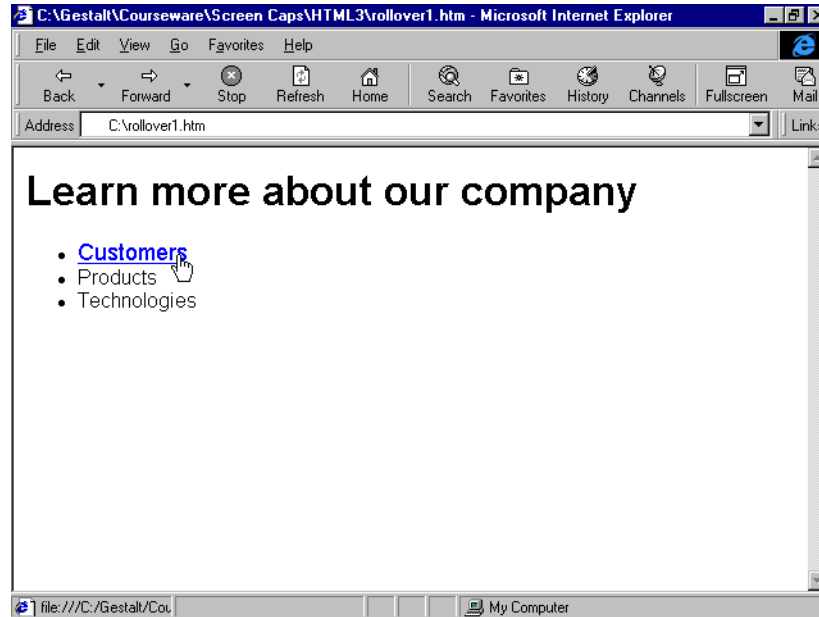


Figure 4-6: A simple Rollover example

This application is quite easy. It uses STYLE definitions, a naming syntax from the DOM and another piece of JavaScript to handle the user action. The STYLE definitions are ordinary CSS and define two classes, ".on" and ".off". (Again, this format is used only so that you can see the elements used more easily.)

```
.on {  
    font-size: 18;  
    text-decoration: underline;  
    color: blue;  
}  
  
.off {  
    font-size: 16;  
    color: black;  
    text-decoration: none;  
}
```

The second part of the application applies the styles to an object, in this case a hyperlink. By default when the document is loaded, the CLASS name is "off". The effect is triggered by the cursor over the hyperlink, changing the style as the cursor moves on and off the hyperlink by resetting the CLASS name of the style.

```
onMouseOver = "this.className ='on';"  
onMouseOut = "this.className = 'off';"
```

"this" is the generic name for any object holding the focus of the current event, in this case, the hyperlink which the cursor is moving over. "className" represents a property of the object which is then set to "on" or "off". Note the semi-colon after 'on' and 'off'. This punctuation is important as the end of a statement in JavaScript.

Here is another example of using a change of styles as a way of animating text. In this case, the text is again changed by resetting the CLASS name. In addition, the scripting handles a little graphic animation created by the swapping of GIF files.



Exercise 4-3: Scripting a Rollover Event

In this exercise, you will test a more complicated rollover example and examine the scripting required to create it.

1. Open the **rollover2.htm** document in your web browser.
2. Move the cursor over the five blue elements to observe the effects.

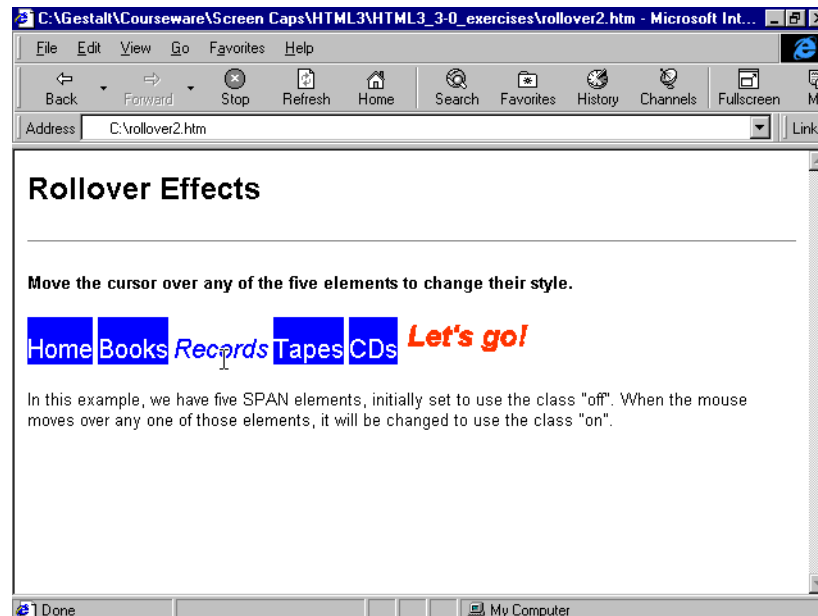


Figure 4-7: A rollover effect with text and graphics

3. From the **V**iew menu, choose **S**ource or open the **rollover2.htm** document in your text editor.

```
<html>
<head>
<style>
.off {
  font-size: 20;
  font-style: normal;
  background-color: blue;
  color: white
}
.on {
  font-size: 20;
  font-style: italic;
  background-color: white;
  color: blue
}
</style>
</head>
<body bgcolor="#FFFFFF">
<FONT FACE="arial,Helvetica" SIZE=2>
```

Styles are defined here with the CLASS names "on" and "off".

```
<div id=Rollover>
<H2>Rollover Effects</H2>
<hr>
<br>
<B>Move the cursor over any of the five elements to change their
style.</B><P>
```

————— An object is created using the DIV tag to capture user actions within this part of the page.

```
<span class=off>Home</span>
<span class=off>Books</span>
<span class=off>Records</span>
<span class=off>Tapes</span>
<span class=off>CDs</span>

</div>
```

————— Individual objects are created with the SPAN tag

————— This IMG tag becomes an object with the name "graphic".

```
<br>
```

This source code continues on the next page

[SOME TEXT DELETED]

<script>

```
function rollon() {
  if (window.event.srcElement.className == "off") {
    window.event.srcElement.className = "on";
    graphic.src="colr_go.gif";
  }
}
```

A "rollon" function is created to reset the CLASS name for the SPAN text and set the source of object "graphic" as named in the IMG tag to a different GIF file. The same is done below for "rolloff".

Rollover.onmouseover = rollon;

If a mouseover event occurs with the object "Rollover", it takes the name "rollon". The same is done below for "rolloff".

```
function rolloff() {
  if (window.event.srcElement.className == "on") {
    window.event.srcElement.className = "off";
    graphic.src="grey_go.gif";
  }
}
```

Rollover.onmouseout = rolloff;

</script>

</body>

</html>

To make things a little easier to see, this example does not create hyperlinks out of these SPAN elements. You can do so by simply enclosing the SPAN element in a hyperlink tag. Here's what that might look like:

```
<A HREF= "some_page.htm"><SPAN class=off>Home</SPAN></A>
```

Adding Scripting Logic

Having created applications which are capable of responding to user actions, you can go farther by adding some logic to your scripts. This will enable you to evaluate user actions and more specific responses. Again, you will use the DOM to identify objects to be manipulated based on instructions from the scripts.



Exercise 4-4: A Multiple Choice Question

In this exercise, you will create the logic required in a script to evaluate a user's response and provide feedback.

1. Open the **onclick.htm** document in your text editor.
2. Examine the objects created as part of the TABLE.

3. Type the source text as shown below to create the script for this application.

```
</TABLE>

<SCRIPT LANGUAGE=JavaScript>
var srcElement
function answer(){
srcElement = window.event.srcElement
    if (srcElement.id == "Athens") {
        response.src= "wrong.gif";
    } else {
        response.src= "right.gif";
    }
}
}

</SCRIPT>

</BODY>
</HTML>
```

Check the punctuation and capitalization carefully.

4. Save the changes.
5. Open the **onclick.htm** document in your web browser.

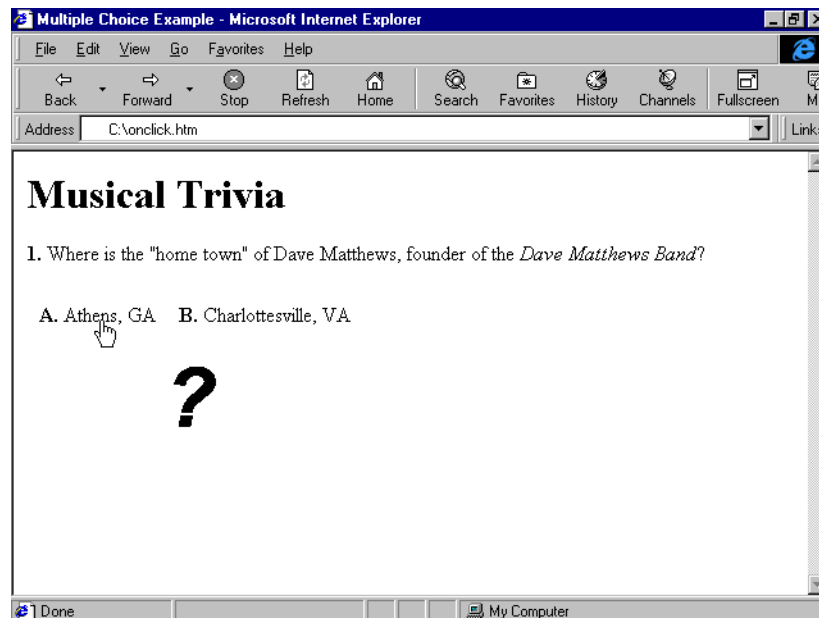


Figure 4-8: A multiple choice question

6. Test the application.

This application shows a simple branching function. First, the answer choices are created and named as objects. In this case, the table cells become the objects with a unique name, "Athens" or "Char". If there is a mouseclick inside the cell, a function, which will be defined in the script is activated. An extra guide for the user has been provided by setting the cursor through a style definition for the cells.

```
<TD ID=Athens onclick="answer();" STYLE="cursor:hand"> <B> A.</B>
Athens, GA </TD>
<TD ID=Char onclick="answer();" STYLE="cursor:hand"><B> B.</B>
Charlottesville, VA </TD>
```

Next, an object to contain the feedback is created by creating and naming an IMG.

```
<IMG ID=response SRC="q-mark.gif">
```

Through the function that is created, when the user makes a choice, the browser checks for the ID of that object. Depending on that ID, the question mark graphic is replaced with one providing feedback to the user by resetting the source of the "response" object. First, a variable is "declared", then given a value. This is a standard way of creating author defined elements in programming and scripting. In this case, its only purpose is to shorten and to simplify the naming of objects.

```
var srcElement
function answer(){
  srcElement = window.event.srcElement
```

Now, the function checks the ID of the object. If it is "Athens", then it instructs the browser to use the "wrong.gif" as the source of the "response" object. Otherwise, the browser is to use the "right.gif".

```
    if (srcElement.id == "Athens") {
      response.src= "wrong.gif";
    } else {
      response.src= "right.gif";
    }
  }
}
```

Note that if you had more answer choices, you could set the function to check for each specific possibility. Also, you could replace a block of text as shown in an earlier exercise. Or you could send the user to another page or multimedia file in response by replacing the "response.src=" statement with something like the following:

```
window.open("http://www.gestalt-sys.com");
```

Clearly, you are now able to create "intelligent" applications that do not rely on server-side processing. Of course, they will rely on your knowledge of a scripting language like JavaScript. The next example shows a slightly more complex application of this same scripting logic.


```

<DIV id=Out1d style="display:None">
  &nbsp;&nbsp;&nbsp;<IMG src="folder.gif" id=Out2 class=Outline
style="cursor: hand">&nbsp;&nbsp;&nbsp;Item 1.1<br>
  <DIV id=Out2d style="display:None">
    &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<IMG src="bullet1.gif">&nbsp;&nbsp;&nbsp;Item
1.1.1<br>
    &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<IMG src="bullet1.gif">&nbsp;&nbsp;&nbsp;Item
1.1.2<br>
  </DIV>
  &nbsp;&nbsp;&nbsp;<IMG src="bullet1.gif">&nbsp;&nbsp;&nbsp;Item 1.2<br>
</DIV>

```

The Item 1.1 level of the outline is created with the source text shaded above. Another DIV tag is used and given a name, "Out1d". Again, because of the assigned style, this section of the outline will not be seen when the document is loaded. In addition, an IMG tag is created that contains the folder icon that designates a collapsible section of the outline.

```

<IMG src="folder.gif" id=Out2 class=Outline style="cursor: hand">

```

The IMG tag has an ID of "Out2" which is intended to correspond with the name of the first DIV tag, which was "Out2d". Again, because of the assigned style, this section of the outline will not be seen when the document is loaded. Further, the IMG has been assigned a CLASS name, though this CLASS is not identified as a style as in previous examples. It is simply another way of naming the object. Finally, a style is assigned to create a cursor to prompt the user that this is a clickable location.

```

<DIV id=Out0>
  <IMG src="folder.gif" id=Out1 class=Outline style="cursor:
hand">&nbsp;&nbsp;&nbsp;Item 1<br>
  <DIV id=Out1d style="display:None">
    &nbsp;&nbsp;&nbsp;<IMG src="folder.gif" id=Out2 class=Outline
style="cursor: hand">&nbsp;&nbsp;&nbsp;Item 1.1<br>
    <DIV id=Out2d style="display:None">
      &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<IMG src="bullet1.gif">&nbsp;&nbsp;&nbsp;Item
1.1.1<br>
      &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<IMG src="bullet1.gif">&nbsp;&nbsp;&nbsp;Item
1.1.2<br>
    </DIV>
    &nbsp;&nbsp;&nbsp;<IMG src="bullet1.gif">&nbsp;&nbsp;&nbsp;Item 1.2<br>
  </DIV>
  <IMG src="bullet1.gif">&nbsp;&nbsp;&nbsp;Item 2<br>
</DIV>

```

Now the last, "top" level of the outline with Item 1 is added. The naming convention for the objects is the same as before. The IMG before a DIV has a corresponding name, so each section of the outline has a name and a named icon associated with it.

Now you move to the script. First, you name the function, then declare some variables and assign a value to one of them. As before, these variables are principally for convenience in naming objects.

```
<script language=JavaScript>  
function clickHandler() {  
    var targetId, srcElement, targetElement;  
    srcElement = window.event.srcElement;
```

The “clickHandler” function is designed to evaluate a mouseclick by the user. At the bottom of the script, it is assigned to handle any click anywhere in the document.

```
document.onclick = clickHandler;
```

Next, the function evaluates the object that has been clicked. If its CLASS name is “Outline”, then a value is set for the “targetID” variable.

```
if (srcElement.className == "Outline") {  
    targetId = srcElement.id + "d";
```

What is it the user has been prompted to click? The folder icons all have a CLASS name of “Outline” and their ID is “Out” plus a number. This ID is the “srcElement.id” mentioned in the script. If you take this ID, “Out1” for example, and add the letter “d” to it, you now have the ID of a corresponding DIV section, namely “Out1d” in this instance. Now the last variable, “targetElement” can be set.

```
targetElement = document.all(targetId);
```

Here the “targetElement” is any instance of an object with the ID of “Out1d”. Now the logic of the script can get to work. If the display property of the object is set to none, then the display is reset to the display, which in IE 4.x is the default. In addition to the named DIV section becoming visible, the source of the original source element, the folder icon is now set to a new folder icon to indicate the open section, “ofolder.gif”.

```
if (targetElement.style.display == "none") {  
    targetElement.style.display = "";  
    srcElement.src = "ofolder.gif";
```

In the event that the element is already displayed, the script sets the value back to “none” and changes the icon back to the original folder.

```
targetElement.style.display = "none";  
srcElement.src = "folder.gif";
```

The work of the script is finished. You can create an outline with as many sections as you wish without any more scripting merely by creating this naming convention for the clickable objects and the corresponding text objects.

This solution for collapsible outlines only works in Internet Explorer 4.x; however, when viewed in other browsers the entire list should be visible. The collapsing action does not occur, but users should see all of the content.

More Examples

Let's turn to some examples created by authors and designers who needed to create "real world" problems. As you examine these applications, you will see a great variety is approaches to design and logical problems coupled with equally great imagination.



Exercise: 4-6 Some Working Applications

In this lesson, you will examine applications and scripts from a variety of working applications.

1. Open the **sites.htm** document in your web browser.
2. Click to link to the **RuleWeb site**.

Mr. Rule currently is the senior web designer for Discovery. This archive is a collection of solutions he's gathered and developed to support his work.

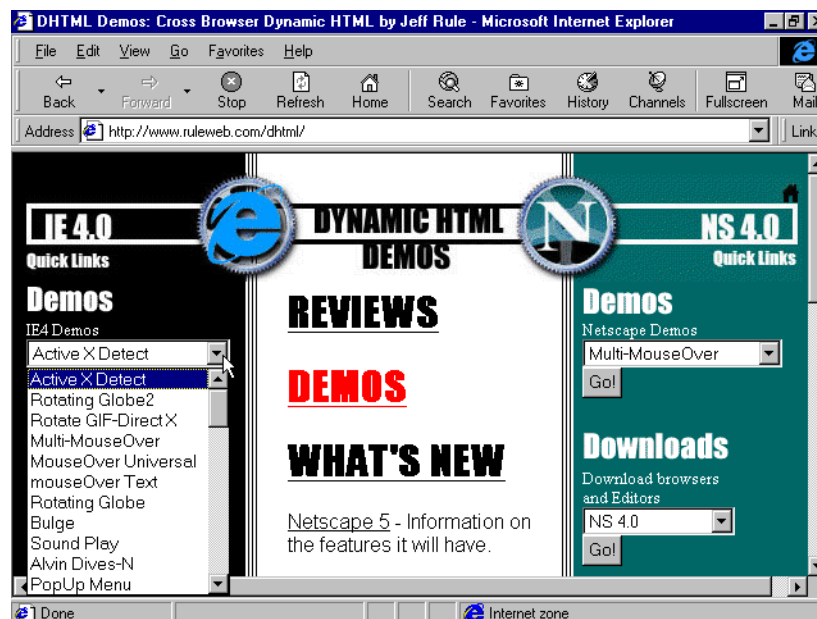


Figure 4-10: The RuleWeb DHTML Archive

The transitions you may see while visiting this site are a result of the special MS FrontPage98 extensions and require a server with those extensions loaded.

3. Click once on the IE 4.0 Demos drop-down and **choose PopOut Menu-N**, then click the **Go!** button.

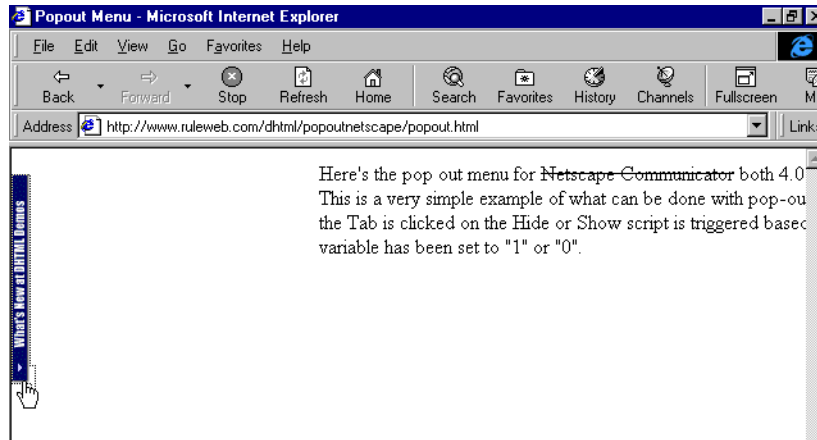


Figure 4-11: Example of a Pop-Out Tab

4. Click on the dark rectangular box to display and hide the text field.

If you view the source of this page, you will find a method for hiding and showing an object by changing the style of that object, similar to what you have done in previous exercises.

5. Use your Back button to return to the RuleWeb DHTML Demo page.
6. Click once on the IE 4.0 Demos drop-down and **choose Skeleton Game-N**, then click the **Go!** button.

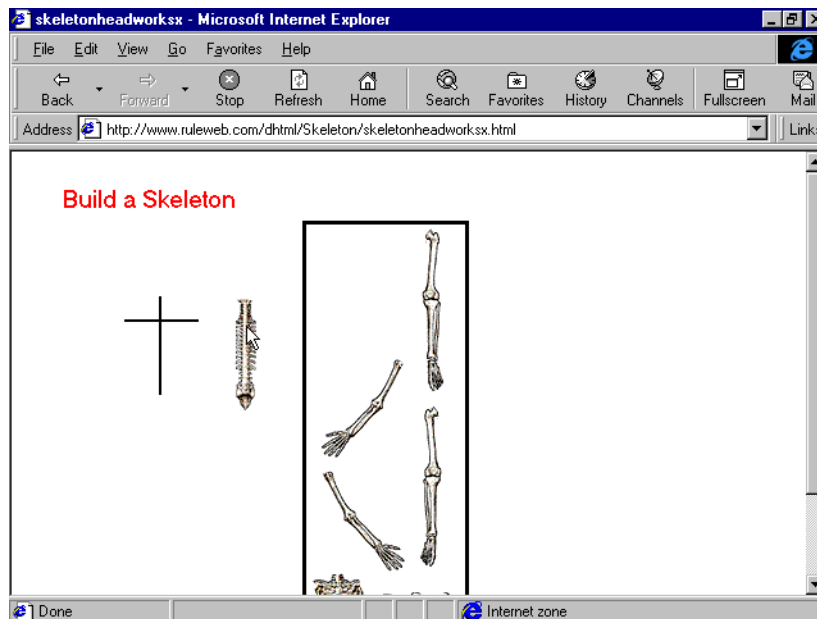


Figure 4-12: Drag and Drop Example

7. Drag and drop the bones to create a properly aligned skeleton.
8. Use your Back button to return to the RuleWeb DHTML Demo page.

9. Explore some of the other demo applications.
10. Click the Netscape 4.0 drop-down to note that there are some corresponding applications under the Netscape side of the DHTML Demo page.
11. Return to the **sites.htm** document in your web browser.
12. Click to link to the **HTMLGuru site**.
13. Follow the instructions as they appear.

Be patient. Depending on the speed of your Internet connection, this site may take a little extra time to load.

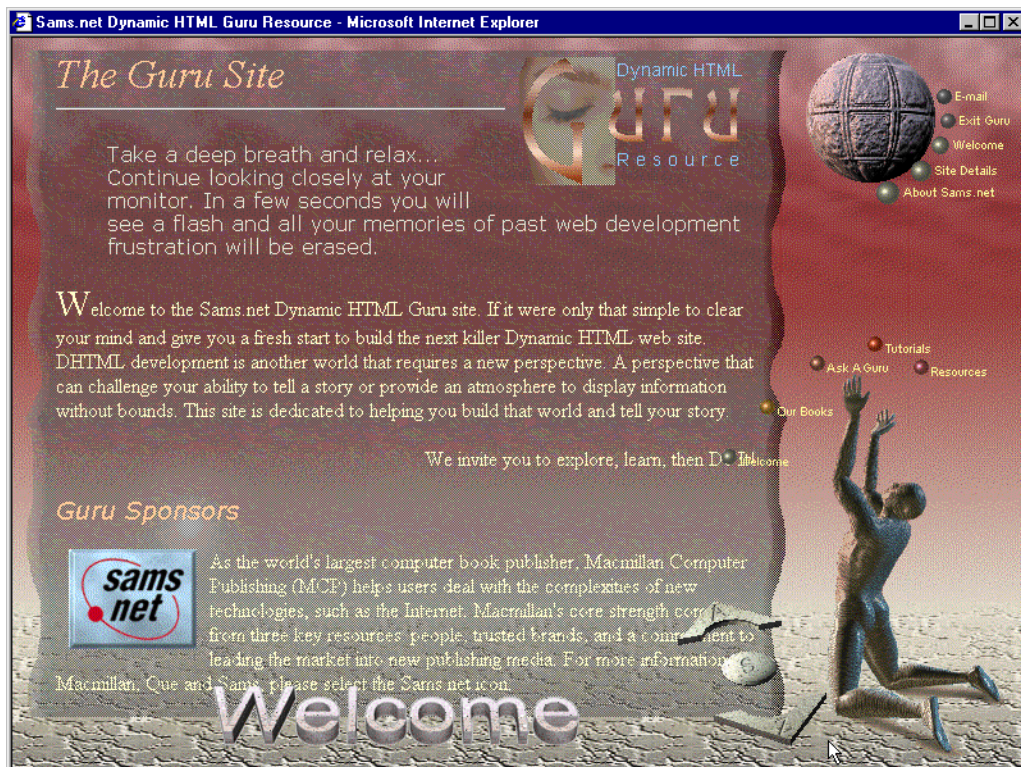


Figure 4-13: An Ambitious DHTML Site

This graphic is different from others in this course because this site can only be viewed properly if the resolution of your monitor is set to 800x600 or larger.

14. Test this application by clicking on the various active elements.
15. When finished close this browser window and return to the **site.htm** document.
16. With the instructor's permission, explore some of the other resources listed.

Review

In this lesson, you learned about Dynamic HTML and how it is changing the way Web pages are being designed. You have learned about Document Object Models (DOM) and how style sheets and scripting languages interact with DOM to create DHTML. You examined and created some simple DHTML applications and explored some more sophisticated illustrations. You should now be able to evaluate the benefits and difficulties of using DHTML in your web design.

The URLs for the demonstration sites used in this lesson are shown below:

<http://www.microsoft.com/gallery/files/html/alienhead.htm> (IE only)

<http://www.BASISinc.com/aquarium.html> (Netscape only)

<http://www.ruleweb.com/dhtml/> (both)

<http://www.htmlguru.com/>