Introduction to Dynamic HTML

Dynamic HTML (DHTML) is a set of innovative features originally introduced in Microsoft Internet Explorer 4.0.
By enabling authors to dynamically change the rendering and content of a Web page as the user interacts with it, DHTML enables authors to create visually compelling Web sites without the overhead of server-side programs or complicated sets of controls to achieve special effects.
With DHTML, you can easily add effects to your pages. For example, you can:

- Hide content until a given time elapses or the user interacts with the page.
- Animate text and images in your document, independently moving each element from any starting point to any ending point, following a predetermined path or one chosen by the user.
- Embed a ticker that automatically refreshes its content with the latest news, stock quotes, or other data.
- Use a **form** to capture user input, and then instantly process and respond to that data.

DHTML achieves these effects by modifying the in-memory representation of the current document and automatically reformatting it to show changes.
It does not reload the document, load a new document, or require a distant server to generate new content. Instead, it uses the user's computer to calculate and carry out changes. This means a user does not wait for text and data to complete time-consuming round trips to and from a server before seeing the results.
Advantages

i) DHTML does not require additional support from applications or embedded controls to make changes. Typically, DHTML documents are self-contained, using styles and a script to process user input and directly manipulate the HTML elements, attributes, styles, and text of the document.
ii) DHTML eliminates the shortcomings of static pages. You can create innovative Web sites, on the Internet or on an intranet, without having to sacrifice performance for interactivity.
iii) DHTML enhance the user's perception of your documents, and also improves server performance by reducing requests to the server.

Document Object Model

DHTML is not a technology in and of itself; rather, it is the product of three related and complementary technologies: HTML, Cascading Style Sheets (CSS), and script.
-To allow scripts and components to access features of HTML and CSS, the contents of the document were represented as objects in a programming model known as the Document Object Model (DOM).
The DOM API is the foundation of DHTML, providing a structured interface that allows you to access and manipulate virtually anything within the document. The HTML elements in the document are available as individual objects, meaning you can examine and modify an element and its attributes by reading and setting properties and by calling methods. The text between elements is also available through DOM properties and methods.
The DOM also provides access to user actions such as pressing a key and clicking the mouse. You can intercept and process these and other events by creating event handler functions and

routines. The event handler receives control each time a given event occurs and can carry out any appropriate action, including using the DOM to change the document.

=> view a complete listing of the objects, properties, methods, collections, and methods of the object model online.

Dynamic Styles

By using CSS, you can quickly change the appearance and formatting of elements in a document without adding or removing elements. This helps keep your documents small and the scripts that manipulate the document fast.
The object model provides programmatic access to styles. This means you can change inline styles on individual elements and change style rules using simple script-based programming. These scripts can be written in JavaScript, Microsoft JScript, or Microsoft Visual Basic Scripting Edition (VBScript).
Inline styles are CSS style assignments that have been applied to an element using the **style** attribute. You can examine and set these styles by retrieving the **style** object for an individual element. For example, to highlight the text in a heading when the user moves the mouse pointer over it, you can use the **style** object to enlarge the font and change its color, as shown in the following simple example.

Example

```
<html>
<head>
<title>Dynamic Styles</title>
<script language="JavaScript">
function doChanges(e) {
    e.style.color = "green";
    e.style.fontSize = "20px";
}
</script>
</head>
<body>
<h3 onmouseover="doChanges(this)" style="color:black;font-
size:18px">Welcome to Dynamic HTML!</h3>
<p>You can do the most amazing things with the least bit of
effort.</p>
</body>
</html>
```

The preceding example demonstrates the following:

- HTML element - In this case, the **H3** tag is the target element.
- Inline style - The element is initially displayed in black with a font size of 18px.
- Event attribute - The **onmouseover** attribute defines the action that occurs when the mouse pointer is moved over the element.

- Event handler - The function that responds to the event is declared in the **head** of the document. A DHTML object that represents the target element is passed as a function parameter using the this pointer.
- The **style** object - The **style** object contains the information that was set in the inline style when the element was defined. To change the color and font size, the function modifies the **color** and **fontSize** properties of the element. The browser immediately updates the onscreen text to display these new attribute values.

Example Two

The next example uses styles to hide a portion of the page until the user clicks the mouse. Instead of passing the target element with the this pointer, the script invokes the HTML elements by **id**.

```
<html>
<head>
<title>Dynamic Styles</title>
<script language="JavaScript">
function showMe() {
    MyHeading.style.color = "red";
    MyList.style.display = "";
}
</script>
</head>
<body onclick="showMe()">

<h3 id="MyHeading">Welcome to Dynamic HTML!</h3>
<p>You can do the most amazing things with the least bit of
effort. Just click and see!</p>

<ul id="MyList" style="display:none">
<li>Change the color, size, and typeface of text</li>
<li>Show and hide text</li>
<li>And much, much more</li>
</ul>

<p>And this is just the beginning!</p>
</body>
</html>
```

In the preceding example, note the following:

- Initially, the **display** attribute of the list is set to `none`. This causes the list to be hidden from view.
- The **onclick** event attribute is set on the **body**. The user can click anywhere on the page to trigger the event.
- The event handler invokes the target elements by **id** and clears the value of the **display** property. Instantly, the content that follows the list shifts to accommodate the new text.

Note When Internet Explorer encounters a tag that defines an **id** or **name**, it creates a reference to it in the global scope so that it can be easily located by script; however, this is considered non-standard behavior. To ensure the widest browser support for your DHTML, use **getElementById** to locate target elements.

Dynamic Content

With DHTML, you can change the content of the page after it is loaded. The DHTML DOM provides access to all elements in the HTML document, from creating, inserting, and deleting elements to modifying the text and attributes in individual elements.
Naturally, the dynamic and interactive features that you add to your pages might not be fully functional when viewed with a browser that does not support DHTML

Non-standard Example

Consider the following simple document, which replaces and changes elements by using a few lines of IE-specific script.

```
<html>
<head><title>Dynamic Content</title>
<script language="JavaScript">
function changeMe() {
    MyHeading.outerHTML = "<H1 ID=MyHeading>Dynamic HTML!</H1>";
    MyHeading.style.color = "green";
    MyText.innerText = "You can do the most amazing things with
the least bit of effort.";
    MyText.align = "center";
    document.body.insertAdjacentHTML("BeforeEnd", "<P
ALIGN=\"center\">Just give it a try!</P>");
}
</script>
</head>
<body onclick="changeMe()">
<h3 id="MyHeading">Welcome to Dynamic HTML!</h3>
<p id="MyText">Click anywhere on this page.</p>
</body>
</html>
```

When the user clicks on the page in the preceding example, the script does the following:

- Replaces an entire **H3** element with an **H1** using the **outerHTML** property.
- Changes the text of the paragraph using the **innerText** property.
- Inserts a new paragraph at the end of the document using the **insertAdjacentHTML** method.

Standard DOM Methods

The Document Object Model (DOM) programming model was designed to dynamically access and update the content, structure, and style of any document type. To do this, the DOM

represents the document hierarchy as a tree of nodes. Each node can be described in terms of its ancestors and descendants. Depending on where the node appears in the hierarchy, it can have children, siblings, and/or a parent.
The standard DOM methods focus more on the tree structure of the document itself, rather than on parsing and interpreting strings of HTML. The following table describes a few of the properties and methods you can use to create and manipulate content dynamically.

Method	Description
createElement	Creates a new element (node) of the specified type.
createTextNode	Creates a plain text node (no HTML).
appendChild	Adds the node to the parent element as the last child.
insertBefore	Inserts the node into the document as a child node of the parent.
replaceChild	Replaces an existing child element with a new child element.

In addition to the methods listed above, many browsers fully support the **innerHTML** property, which provides access to the HTML between the start and end tags of an element.
Although **innerHTML** is not defined by any standard, it is often simpler to use than its alternatives.

The following shows how to rewrite the nonstandard example above using a more standards-compliant DOM implementation.

```
<html>
<head><title>Dynamic Content</title>
<script language="JavaScript">
function changeMe() {
    // Replace outerHTML with createElement and replaceChild.
    var oChild = document.getElementById("MyHeading");
    var oNewChild = document.createElement('H1');
```

```
    oNewChild.id = oChild.id;
    oNewChild.innerHTML = "Dynamic HTML!";
    oNewChild.style.color = "green";
    oChild.parentNode.replaceChild(oNewChild,oChild)

    // Use innerHTML instead of innerText.
    MyText.innerHTML = "You can do the most amazing things with
the least bit of effort.";
    MyText.align = "center";

    // Change insertAdjacentHTML("BeforeEnd") to appendChild.
    var oPara = document.createElement('P');
    oPara.innerHTML = "Just give it a try!"
    oPara.align = "center";
    document.body.appendChild(oPara);
}
</script>
</head>
<body onclick="changeMe()">
<h3 id="MyHeading">Welcome to Dynamic HTML!</h3>
<p id="MyText">Click anywhere on this page.</p>
</body>
</html>
```

The preceding example is more verbose than its predecessor, but it has the advantage of working on many of the browsers in use today. The key points to remember are:

- Locate elements with **getElementById**.
- Use **createElement** to instantiate new HTML objects that you can add to the document.
- Manipulate the DOM using standard methods, such as **appendChild** and **replaceChild**.
- Use **innerHTML** rather than **innerText**.

Positioning and Animation

Positioning is the ability to place an HTML element at a specific point on a page, relative to another element or the browser window itself. By assigning **top** and **left** coordinates, you can place elements such as images, controls, and text exactly where you want them. You can also assign a **z-index** to define the order in which overlapping elements are stacked on top of one another.

Elements can be positioned using one of the following keywords:

Keyword	Description

`absolute`	The element is removed from the flow of the document and positioned with respect to its container.
`relative`	The element is offset from its container, but the space the element would have occupied in the document is preserved.
`fixed`	Windows Internet Explorer 7 and later. The element is positioned as in absolute positioning, except that it is relative to the browser window rather than the document.

Positioning with CSS

Positioning is a component of CSS. This means that you set the position of an element by setting the appropriate CSS attributes for that element. The following example shows how to set the absolute position of an image.

```
<html>
<head>
  <title>Positioning</title>
</head>
<body>
<h3>Welcome to Dynamic HTML!</h3>
<p>With positioning, you can place images exactly where you want them,
even behind text and other images.</P>
<img style="position:absolute;top:0;left:0;z-index:-1" src="clouds.gif">
</body>
</html>
```

In the preceding example:

- The image is placed at the document's top left corner by setting **top** and **left** to zero.
- The image is placed behind the text on the page by setting the **z-index** attribute to -1.

Positioning with Script

Because the DOM provides access to styles and style sheets, you can set and change the position of any element as simply as you can set and change its color. This makes it especially easy to change the position of elements based on how the user is viewing the document, and even to animate the elements. For animation, all you need is to modify slightly the position of an element on some interval. The following example presents an image that glides across the page and comes to rest at the left margin.

```
<html>
<head>
<title>Dynamic Positioning</title>
<script language="JavaScript">
var id;
function StartGlide()
{
    Banner.style.pixelLeft = document.body.offsetWidth;
    Banner.style.visibility = "visible";
    id = window.setInterval(Glide,50);
}
function Glide()
{
    Banner.style.pixelLeft -= 10;
    if (Banner.style.pixelLeft <= 0) {
        Banner.style.pixelLeft = 0;
        window.clearInterval(id);
    }
}
</script>
</head>
<body onload="StartGlide()">
<h3>Welcome to Dynamic HTML!</h3>
<p>With dynamic positioning, you can move images anywhere in the document
even while the user views the document.</p>
<img id="Banner" STYLE="visibility:hidden;position:absolute;z-index:-1"
SRC="eightball.gif" />
</body>
</html>
```

In the preceding example:

- The image is absolutely positioned and initially invisible (**visibility** set to `hidden`).
- The **StartGlide** function unhides the image, sets its position of the image to the far-right edge of the page, and begins calling **Glide** with an **interval** of 50 milliseconds.
- The **Glide** function moves the image to the left by 10 pixels, and when the image is finally at the left edge, it cancels the interval.

Dynamic positioning has many uses in consumer and business applications. By combining dynamic styles, positioning, transparent images, and transparent Microsoft ActiveX controls, you can present a rich set of animation effects.

Data Binding

Data binding is a DHTML feature that lets you easily bind individual elements in your document to data from another source, such as a database or comma-delimited text file.

-When the document is loaded, the data is automatically retrieved from the source and formatted and displayed within the element.

Uses

a) *To automatically and dynamically generate tables in your document;* you can bind a **table** element to a data source. When the document is viewed, a new row is created in the table for each record retrieved from the source, and the cells of each row are filled with text and data from the fields of the record.
-the user can view the page while new rows are created in the table (dynamic generation).
-after all the table data is present, you can manipulate (sort or filter) the data without requiring the server to send additional data. The table is simply regenerated, using the previously retrieved data to fill the new rows and cells of the table.
b) *To bind one or more elements in the document to specific fields of a given record*. When the page is viewed, the elements are filled with text and data from the fields in that record, sometimes called the "current" record. A simple example is a form letter in which the name, e-mail address, and other details about an individual are filled from a database. To adapt the letter for a given individual, you simply specify which record should be the current record. No other changes to the letter are needed.
c) *To bind the fields in a form to fields in a record.* Not only can the user view the content of the record, but the user can also change that content by changing the settings and values of the form. The user can then submit these changes so that the new data is uploaded to the source—for example, to the HTTP server or database.

To provide data binding in your documents, you must add a data source object (DSO) to your document. This invisible object is simply an ActiveX control or Java applet that knows how to communicate with the data source. The following example shows how easy it is to bind a table to a DSO. When viewed, this example displays the first three fields from all the comma-delimited records of the file sampdata.csv in a clear, easy-to-read table.

```
<html>
<head><title>DataURL Example</title></head>
<body>
<object classid="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83" ID="sampdata">
   <param name="DataURL" value="sampdata.csv">
   <param name="UseHeader" value="True">
</object>
<table border="1" datasrc="#sampdata">
<thead><tr><th>A</th><th>B</th><th>C</th></thead>
<tbody>
<tr>
    <td><span datafld="a"></span></td>
    <td><span datafld="b"></span></td>
    <td align="right"><span datafld="c"></span></td>
</tbody>
</table>
</body>
</html>
```

To populate the table from a DSO, the example does the following:

- Instantiate the DSO and load the dataset with an **OBJECT** tag.
- Binds to the data source by a **DATASRC** attribute on the **table** element.
- Binds to fields in the recordset with **DATAFLD** attributes on each table cell. Rows inside the **TBODY** element are repeated for each record.