**DHTML Introduction**

Introduction to DHTML

**DHTML CSS**

CSS is used to style HTML elements.

**DHTML DOM**

The Document Object Model

**DHTML Events**

An incident which occurs in a particular place during a particular interval of time.

**DHTML Summary**

**Basics of DHTML**

Dynamic Hyper Text Markup Language is a technique in which HTML elements move across the page.

**Difference between Static vs. Dynamic HTML**

Static HTML means when we put HTML elements (images,paragraphs etc.) in a specific order in the source code. The browser always showed all elements in this order. styling and Positioning was done by tables, div's and such aids. If we wanted to change the positioning or order of the elements, we had to again write the HTML.

DHTML gives a way to re-organize our pages on the fly. we can take some elements out of the natural flow of the page, put them somewhere and change its position again and again by clicking a link.

The natural flow of the page is the page as the browser shows it one by one and displays them in the best possible way from the beginning to the end of the HTML document.

Now take an image, and put it somewhere on the page without regard to this natural flow. We can force the image to be in the extreme upper left corner of the page, while the rest of the content is still distributed by the natural flow.

Attach a link to that page, say "Move image to the right". As soon as the user clicks this link, a JavaScript is executed which moves the image, suppose 200 pixels to the right. We can do this at once or make the image move stealthily to its new position. In either case, the other elements on the page stay where they are, only the image moves.

**Working of DHTML**

Dynamic HTML gives authors creative control so they can manipulate any page element and change styles, content and position at any time. It provides a more dynamic experience on web pages, making them more like dynamic applications and less like static content. Dynamic HTML presents richly formatted pages and lets you interact with the content on those pages without having to download additional content from the server. it means that a page can respond immediately to user actions, such as a mouse click, without having to retrieve an entire new page from the server.

We discuss the three important components of Dynamic HTML authoring:

Positioning - precisely placing blocks of content on the page and, moving these blocks around.

Style modifications done by altering the look and display of content on the page.

Event handling is how to relate user events to changes in style and position

**Getting Started:**

DHTML is a combination of three existing technologies meshed together by the Document Object Model (DOM):

1. HTML - For creating different page elements like text, image links.

2. CSS - Cascading Style Sheets allows authors and users to attach style (e.g., fonts, spacing,) to structured documents by separating content of documents and the presentation style of documents. CSS simplifies site maintenance and Web authoring.

3. JavaScript - it allows you to accesses and dynamically controls the individual properties of both HTML and CSS

JavaScript accesses the properties of an HTML document through the DOM. The work of DOM is to represent all the attributes of HTML and Style sheets to JavaScript control. You need only to know about the DOM is what JavaScript commands it accepts. As different browsers have their slightly different versions of the DOM, so they display HTML properties differently as well as access them differently. The basic object in DHTML is a named rectangular region, whose method of declaration and JavaScript reference can differ depending on the browser involved.

**CSS is used to style HTML elements such as layering and positioning the content.**

DHTML is NOT a scripting language like Javascript, but simply a browser feature, or enhancement,that gives your browser the ability to be dynamic. What really want to learn is not DHTML itself but rather, the syntax needed to use DHTML. It is the ability of the browser to alter a web page's style and look after the page has loaded.

There are a couple of important points that you should be aware of when you are working with the DHTML client during application development.

Here's a simple demonstration. the text below change color when the mouse moves over it:

Move your mouse here

**The code looks like:**

<span id="sometext"

onMouseover="sometext.style.color='red'"

onMouseout="sometext.style.color='black'">Move your mouse here</span>

In order to get this effect, I first gained access from the element's ID. Then, through the style object, and finally the style's color property. All style properties are read/write, and are accessed in a similar manner: element id->style object->property name.

**DHTML includes CSS and JavaScript**

By means of JavaScript, DHTML is the changing of the style declarations of an HTML element. For instance,if you have a paragraph with a certain text,color defined by

p {color: #cc0000;}

you can do something like

element.style.color = '#00cc00';

as soon as this bit of script is executed the text colour changes from red to green.

Unfortunately this simple bit of DHTML is not supported by Netscape 4. In fact, Netscape 4 only supports the moving of elements across the page and making them invisible (and a few other things, but these are the most important) and visible. So we give an element a position on the page:

div {position: absolute;

top: 20px;

left: 0px;}

and then change its left:

element.style.left = 200;

Now element magically moves to 200 pixels from the left margin of the page.

**The style property**

So generally,you first go to the HTML element in Internet Explorer you want to influence like :-

document.all['tobechanged'].style.left = 200;

then change the style declarations:

element.style.styleDeclaration

If you want to change color, styleDeclaration is color, if you want to change the left, it is left. If you want to change a style declaration with a dash in it, for instance z-index, remove the dash and make the first letter after the dash a capital: zIndex. Same for margin-left => marginLeft.

Note that the style property reflects the inline styles of the element. This is important if you want to read out the styles of the element. If you try to read out the color of the paragraph:

p#testP {color: #cc0000;}

<p id="testP">This is the paragraph.</p>

alert([testP].style.color);

you get an empty alert. This is because the P doesn't (yet) have an inline color style. The style defined by p#testP is not readable through the style property.

You define your styles inline, it works fine

<p style="color: #cc0000">This is the paragraph.</p>

alert([testP].style.color);

Netscape 4

The .style property is required in all browsers, except for Netscape 4

The correct element

So changing the style of a certain HTML element is not difficult, except that many things won't work in Netscape 4. However, you also have to tell the browser which HTML element you want to influence. To do this, you must invoke it by its correct element ID

[content of div]

and write the DHTML script to call on that ID:

[element with ID=tobechanged].style.left = 200;

There are two problems here. first is the proper definition of a layer in Netscape 4, and second is the differences in DOM (Document Object Model) between the various browsers.

**Layers in Netscape 4**

If you want to change the style of an element in Netscape 4, that element must have a position declaration in the style sheet. If you don't give it a position, Netscape 4 simply does not recognize the element as a layer and throws up error messages.

For example:

div#tobechanged {position: absolute;

top: 20px;

left: 0px;}

Now the DIV with ID="tobechanged" has a position and Netscape 4 recognizes it as a layer. You can also use position: relative. This means that the element is positioned by the natural flow of the page, but is nonetheless open to DHTML influence.

So take care to give your layers a position.

**DOM**

The Document Object Model(DOM) is an Application Programming Interface(API). Essentially, it provides a platform where programming languages can "talk" to HTML and XML documents. Usually the programming language that does the talking by using JavaScript.

**DOM vs DHTML Object Model**

The Document Object Model evolved from the DHTML Object Model, but one may dispute that it is more like a revolution than an evolution. The DHTML Object Model lets you update and access HTML objects individually. Each HTML tag can be manipulated and accessed via its NAME and ID attributes. Each object have its own properties, methods, and events. You can use the properties update local attributes of the object. You can use the methods to manipulate the objects and take advantage of the events to trigger pre-defined consequences.

The DOM is much more general than the DHTML Object Model.Not just for a single HTML tag it provides a model for the whole document. The DOM represents a document as a tree. Every node of the tree represents an HTML tag. The tree structure accurately describes the whole HTML document, including relationships between tags and textual entries on the page. A relationship may be of the type child, parents and sibling.

**To access an element?**

The element must have an id attribute and a scripting language is needed. JavaScript is the most compatible scripting language for browser , so we use JavaScript.

<html>

<body>

<h1 id="header">My header</h1>

<script type="text/javascript">

document.getElementById('header').style.color="red"

</script>

</body>

</html>

output of above Script which change the color:

My Header

**DHTML DOM Objects**

With the help of JavaScript, you can manipulate and access all of the HTML DOM objects.

Here is a list of Document Object Model objects and their collections, properties, methods and events.

|  |  |
| --- | --- |
| **Object** | **Description** |
| Anchor | Represents an HTML a element (a hyperlink) |
| Applet | Represents an HTML applet element. The applet element is used to place executable content on a page |
| Area | Represents an area of an image-map. An image-map is an image with clickable regions |
| Base | Represents an HTML base element |
| Basefont | Represents an HTML basefont element |
| Body | Represents the body of the document (the HTML body) |
| Button | Represents a push button on an HTML form. For each instance of an HTML <input type="button"> tag on an HTML form, a Button object is created |
| Checkbox | Represents a checkbox on an HTML form. For each instance of an HTML <input type="checkbox"> tag on an HTML form, a Checkbox object is created |
| Document | Used to access all elements in a page |
| Event | Represents the state of an event, such as the element in which the event occurred, the state of the keyboard keys, the location of the mouse, and the state of the mouse buttons |
| FileUpload | For each instance of an HTML <input type="file"> tag on a form, a FileUpload object is created |
| Form | Forms are used to prompt users for input. Represents an HTML form element |
| Frame | Represents an HTML frame |
| Frameset | Represents an HTML frameset |
| Hidden | Represents a hidden field on an HTML form. For each instance of an HTML <input type="hidden"> tag on a form, a Hidden object is created |
| History | A predefined object which can be accessed through the history property of the Window object. This object consists of an array of URLs. These URLs are all the URLs the user has visited within a browser window |
| Iframe | Represents an HTML inline-frame |
| Image | Represents an HTML img element |
| Link | Represents an HTML link element. The link element can only be used within the <head> tag |
| Location | Contains information about the current URL |
| Meta | Represents an HTML meta element |
| Navigator | Contains information about the client browser |
| Option | Represents an option in a selection list on an HTML form. For each instance of an HTML <option> tag in a selection list on a form, an Option object is created |
| Password | Represents a password field on an HTML form. For each instance of an HTML <input type="password"> tag on a form, a Password object is created |
| Radio | Represents radio buttons on an HTML form. For each instance of an HTML <input type="radio"> tag on a form, a Radio object is created |
| Reset | Represents a reset button on an HTML form. For each instance of an HTML <input type="reset"> tag on a form, a Reset object is created |
| Screen | Automatically created by the JavaScript runtime engine and it contains information about the client's display screen |
| Select | Represents a selection list on an HTML form. For each instance of an HTML <select> tag on a form, a Select object is created |
| Style | Represents an individual style statement. This object can be accessed from the document or from the elements to which that style is applied |
| Submit | Represents a submit button on an HTML form. For each instance of an HTML <input type="submit"> tag on a form, a Submit object is created |
| Table | Represents an HTML table element |
| TableData | Represents an HTML td element |
| TableHeader | Represents an HTML th element |
| TableRow | Represents an HTML tr element |
| Text | Represents a text field on an HTML form. For each instance of an HTML <input type="text"> tag on a form, a Text object is created |
| Textarea | Represents an HTML textarea element |
| Window | Corresponds to the browser window. A Window object is created automatically with every instance of a <body> or <frameset> tag |

**DHTML Events**

An incident which occurs in a particular place during a particular interval of time.

**EVENT HANDLING**

Events are represented by an event object as a member variable of the window object, such as window.event or whether as a parameter to your function

Event handlers are Javascript code that are not include inside the <script> tags, but rather, inside the html tags, that execute Javascript when some events fire, such as pressing a button, moving your mouse over a link, submitting a form, etc.

The basic syntax of these event handlers is : name\_of\_handler="JavaScript code here"

For example:

<a onclick="alert('Hello World!')" href="javascript:;" class="link">Click Here</a>

The above code produces the following output:

Click Here

Event Functions:

**onclick:**

The onclick: Event is used to invoke Javascript upon clicking a link or form boxes as in the below example:

<form name="myalertform">  
<input onclick="alert('hello World')" type="button" value="Click me" name="test" class="link">  
</form>

The above code produces the following output:   
F:\ki$hore\DHTML\1.JPG

Clicking the above button invokes the alert function. The event handler is added within the event that it handles, like the the form button event tag.

onload:

The onload: function invokes Javascript after the page has finished loading. The below example loads a webpage, then closes it after the image has finished downloading.

**onload**

Displays an alert box when the page has finished loading.

**onmouseover:**

The onmouseover: function is used to invoke Javascript when the mouse passes over a link.

<a onmouseover="status='You activated this message when you moused over the link!';return true" onmouseout="status=' '" href="javascript:;" class="link">

Mouse over this link and look the status bar message.

</a>

The above code produces the following output:

Mouse over this link and look the status bar message.

**Common Event Handlers:**

**Given below is a list of common Javascript event handlers:**

|  |  |
| --- | --- |
| **Event** | **Occurs when...** |
| onabort | a user aborts page loading |
| onblur | a user leaves an object |
| onchange | a user changes the value of an object |
| onclick | a user clicks on an object |
| ondblclick | a user double-clicks on an object |
| onfocus | a user makes an object active |
| onkeydown | a keyboard key is on its way down |
| onkeypress | a keyboard key is pressed |
| onkeyup | a keyboard key is released |
| onload | a page is finished loading. **Note:** In Netscape this event occurs during the loading of a page! |
| onmousedown | a user presses a mouse-button |
| onmousemove | a cursor moves on an object |
| onmouseover | a cursor moves over an object |
| onmouseout | a cursor moves off an object |
| onmouseup | a user releases a mouse-button |
| onreset | a user resets a form |
| onselect | a user selects content on a page |
| onsubmit | a user submits a form |
| onunload | a user closes a page |

|  |
| --- |
| Here below the parameters in which they can be used |

|  |  |
| --- | --- |
| **Handlers** onAbort onBlur onClick onChange onError onFocus onLoad onMouseover onMouseout onReset onSelect onSubmit onUnload | **Can be used with these tags:** images windows, all form elements, frames buttons, radio buttons,checkboxes, submit buttons, links text fields, textareas, select lists windows, images windows, frames, and all form elements body, images areas, links links forms text fields, textareas submit button body |

**Summary**

By Combining HTML,a Document Object Model (DOM) , CSS-positioning and JavaScript most people refer to as "DHTML". One thing DHTML can do is to Repeat the movement several times and you'll get animation and move content around in the browser window. With JavaScript's capabilities of continued function calls with specific delays DHTML is an easy method of animated movement at a predefined pace.

DHTML uses both Javascript and the DOM to provide the script programmer with a great deal of power and flexibility in creating attractive, interesting and dynamic Web pages. There's a lot to cover in this area, and this tutorial has provided you with only an introduction to the details of DHTML. If you decide to move beyond these basics you will find plenty of information available on the Web and in books.

Top of Form

Bottom of Form