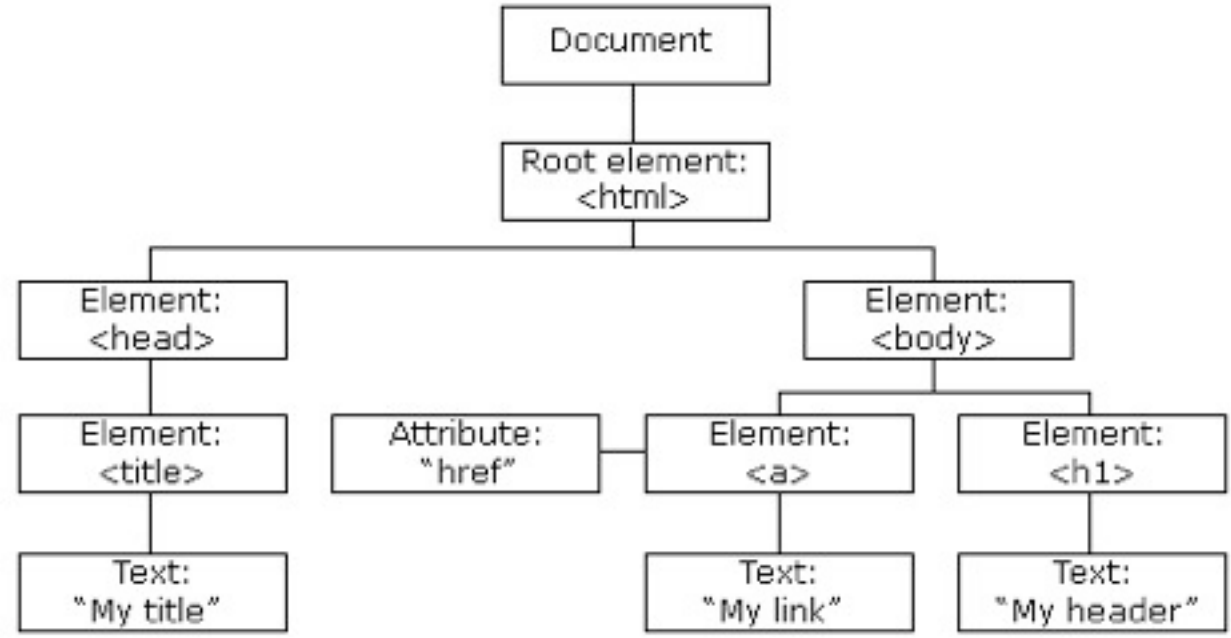
The HTML DOM (Document Object Model)

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

The **HTML DOM** model is constructed as a tree of **Objects**:

The HTML DOM Tree



With a programmable object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can react to all the events in the page

Finding HTML Elements

Often, with JavaScript, you want to manipulate HTML elements.

To do so, you have to find the elements first. There are a couple of ways to do this:

- Finding HTML elements by id
- Finding HTML elements by tag name
- Finding HTML elements by class name

Finding HTML Elements by Id

The easiest way to find HTML elements in the DOM, is by using the element id.

This example finds the element with id="intro":

Example

```
var x=document.getElementById("intro");
```

If the element is found, the method will return the element as an object (in x).

If the element is not found, x will contain null.

Finding HTML Elements by Tag Name

This example finds the element with id="main", and then finds all <p> elements inside "main":

Example

```
var x=document.getElementById("main");
var y=x.getElementsByTagName("p");
```

Finding elements by class name does not work in Internet Explorer 5,6,7, and 8.

Changing the HTML Output Stream

JavaScript can create dynamic HTML content:

Date: Mon Mar 18 2013 22:09:51 GMT+0000 (WET)

In JavaScript, document.write() can be used to write directly to the HTML output stream:

Example

```
<!DOCTYPE html>
<html>
<body>

<script>
document.write(Date());
</script>

</body>
</html>
```

Never use document.write() after the document is loaded. It will overwrite the document.

Changing HTML Content

The easiest way to modify the content of an HTML element is by using the **innerHTML** property.

To change the content of an HTML element, use this syntax:

```
document.getElementById(id).innerHTML=new HTML
```

This example changes the content of a <p> element:

Example

```
<html>
<body>

<p id="p1">Hello World!</p>

<script>
document.getElementById("p1").innerHTML="New text!";
</script>

</body>
</html>
```

This example changes the content of an <h1> element:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1 id="header">Old Header</h1>

<script>
var element=document.getElementById("header");
element.innerHTML="New Header";
</script>

</body>
</html>
```

Example explained:

- The HTML document above contains an <h1> element with id="header"
- We use the HTML DOM to get the element with id="header"
- A JavaScript changes the content (innerHTML) of that element

Changing an HTML Attribute

To change the attribute of an HTML element, use this syntax:

```
document.getElementById(id).attribute=new value
```

This example changes the src attribute of an <img> element:

Example

```
<!DOCTYPE html>
<html>
<body>

<img id="image" src="smiley.gif">

<script>
document.getElementById("image").src="landscape.jpg";
</script>

</body>
</html>
```

Changing HTML Style

To change the style of an HTML element, use this syntax:

```
document.getElementById(id).style.property=new style
```

The following example changes the style of a <p> element:

Example

```
<html>
<body>

<p id="p2">Hello World!</p>

<script>
document.getElementById("p2").style.color="blue";
```

```
</script>

<p>The paragraph above was changed by a script.</p>

</body>
</html>
```

This example changes the style of the HTML element with id="id1", when the user clicks a button:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1 id="id1">My Heading 1</h1>
<button type="button"
onclick="document.getElementById('id1').style.color='re
d'">
Click Me!</button>

</body>
</html>
```

HTML DOM Style Object Reference

HTML DOM allows JavaScript to react to HTML events.

Example

Reacting to Events

A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.

To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:

`onclick=`*`JavaScript`*

Examples of HTML events:

- When a user clicks the mouse
- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user strokes a key

In this example, the content of the <h1> element is changed when a user clicks on it:

Example

```
<!DOCTYPE html>
<html>
<body>
<h1 onclick="this.innerHTML='Ooops!'">Click on this
text!</h1>
</body>
</html>
```

Try it yourself »

In this example, a function is called from the event handler:

Example

```
<!DOCTYPE html>
```

```
<html>
<head>
<script>
function changetext(id)
{
id.innerHTML="Ooops!";
}
</script>
</head>
<body>
<h1 onclick="changetext(this)">Click on this text!</h1>
</body>
</html>
```

Try it yourself »

HTML Event Attributes

To assign events to HTML elements you can use event attributes.

Example

Assign an onclick event to a button element:

```
<button onclick="displayDate()">Try it</button>
```

Try it yourself »

In the example above, a function named *displayDate* will be executed when the button is clicked.

Assign Events Using the HTML DOM

The HTML DOM allows you to assign events to HTML elements using JavaScript:

Example

Assign an onclick event to a button element:

```
<script>
document.getElementById("myBtn").onclick=function()
{displayDate()};
</script>
```

Try it yourself »

In the example above, a function named *displayDate* is assigned to an HTML element with the id=myButn".

The function will be executed when the button is clicked.

The onload and onunload Events

The onload and onunload events are triggered when the user enters or leaves the page.

The onload event can be used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.

The onload and onunload events can be used to deal with cookies.

Example

```
<body onload="checkCookies()">
```

Try it yourself »

The onchange Event

The onchange event are often used in combination with validation of input fields.

Below is an example of how to use the onchange. The upperCase() function will be called when a user changes the content of an input field.

Example

```
<input type="text" id="fname" onchange="upperCase()">
```

Try it yourself »

The onmouseover and onmouseout Events

The onmouseover and onmouseout events can be used to trigger a function when the user mouses over, or out of, an HTML element.

Example

A simple onmouseover-onmouseout example:

Try it yourself »

The onmousedown, onmouseup and onclick Events

The onmousedown, onmouseup, and onclick events are all parts of a mouse-click. First when a mouse-button is clicked, the onmousedown event is triggered, then, when the mouse-button is released, the onmouseup event is triggered, finally, when the mouse-click is completed, the onclick event is triggered.

HTML DOM Event Object Reference

Creating New HTML Elements

To add a new element to the HTML DOM, you must create the element (element node) first, and then append it to an existing element.

Example

```
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var para=document.createElement("p");
var node=document.createTextNode("This is new.");
para.appendChild(node);

var element=document.getElementById("div1");
element.appendChild(para);
</script>
```

Try it yourself »

Example Explained

This code creates a new <p> element:

```
var para=document.createElement("p");
```

To add text to the <p> element, you must create a text node first. This code creates a text node:

```
var node=document.createTextNode("This is a new
paragraph.");
```
Then you must append the text node to the <p> element:

```
para.appendChild(node);
```
Finally you must append the new element to an existing element.

This code finds an existing element:

```
var element=document.getElementById("div1");
```
This code appends the new element to the existing element:

```
element.appendChild(para);
```

Removing Existing HTML Elements

To remove an HTML element, you must know the parent of the element:

Example

```
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>
<script>
var parent=document.getElementById("div1");
var child=document.getElementById("p1");
parent.removeChild(child);
</script>
```

Try it yourself »

Example Explained

This HTML document contains a element with two child nodes (two <p> elements):

```
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>
```

Find the element with id="div1":

```
var parent=document.getElementById("div1");
```

Find the <p> element with id="p1":

```
var child=document.getElementById("p1");
```

Remove the child from the parent:

```
parent.removeChild(child);
```

It would be nice to be able to remove an element without referring to the parent.
But sorry. The DOM needs to know both the element you want to remove, and its parent.

Here is a common workaround: Find the child you want to remove, and use its parentNode property to find the parent:

```
var child=document.getElementById("p1");
child.parentNode.removeChild(child);
```

Full HTML DOM Tutorial