# DISTINGUISH ELEMENTS ON THE INTERNET EXPLORER AND NETSCAPE NAVIGATOR BROWSERS OF DOM STANDS FOR THE CONTROL OF THE ASP, JSP, JAVA APPLETS

**by**
**Mircea Risteiu, Gheorghe Marc, Pasculescu Dragos**

## 1. State of arts

DOM stands for Document Object Model, and allows programmers generic access - adding, deleting, and manipulating - of all styles, attributes, and elements in an Internet document. It can be accessed via any language available in the browser, including Java, JavaScript/ ECMAScript/ JScript, and VBScript (MSIE only). For explanations, the syntax used in this tutorial will be mostly of JavaScript. The DOM is supported most completely starting in IE 5 (Internet Explorer 5.x) and Gecko (NS 6, formally called NS 5- Netscape Navigator 4.7- 5.x).
Every tag, attribute, style, and piece of text is available to be accessed and manipulated via the DOM -- the possibilities are endless. This paper will try to cover some of aspects of the DOM: adding and removing tags, attributes and styles, animating existing elements, and hiding/ showing elements on a page with different behavoir in IE and NS.
Some terms have to be explaned for the context use. For instance a node is a reference to an element, its attributes, or text from the document. An element is a representation of a <TAG>. An attribute is an attribute of an element. HREF is an attribute of <A>, for example.

## 2. The DOM and browser compatibility

The DOM is currently only available in the 5.x browsers. Since Internet Explorer 5.0, and Navigator 6.0 / Mozilla are currently the only browsers with support for the DOM, browser detection is necessary in almost all circumstances. This browser detection would be as simple as:

```
if (parseInt(navigator.appVersion) >= 5)
{// has dom support }
```

Unfortunately, Microsoft set navigator.appVersion to 4.0 in Internet Explorer 5.0. They say this as a way to preserve backward compatibility with poorly written applications that checked for MSIE 4, not MSIE 4 or greater. This increases the complexity of checking for DOM support significantly. The simplest detection method is:

```
 if (parseInt(navigator.appVersion) >= 5 || navigator.appVersion.indexOf["MSIE 5"]
!= -1)  {// has dom support
}
```

This is the method of checking in all applications. Hopefully Microsoft will use the correct navigator.appVersion in MSIE 6, but if they don't this code may need to be changed. Some would say that it is not as good as it could be since it uses browser detection and not object detection, but this is used because the DOM doesn't expose any elements in IE 5 that were not in IE 4, to the best of my knowledge.

For those of you familiar with and used to the collections *document.all[]* and *document.layers[]*, it may find coding:

*var docContents = document.getElementsByTagName("*");*

**3. Adding elements to the DOM**

Adding an element to a document is very simple in any browser. The effect is different. The first step to adding to a document is to create the *node*, the next is to find where must be appended it, and the final step is to actually do the appending. A node is the representation of either a piece of text or a tag and its attributes in the DOM, and will be referred to often on this tutorial. The syntax for creating a node is very simple. To create a text node, it can be used the *createTextNode()* method of the document object. *document.createTextNode(Text)* returns a node with the given text in it. Here is an example use of it:
*var txtNode = document.createTextNode("Hello. This is a new node.")*;

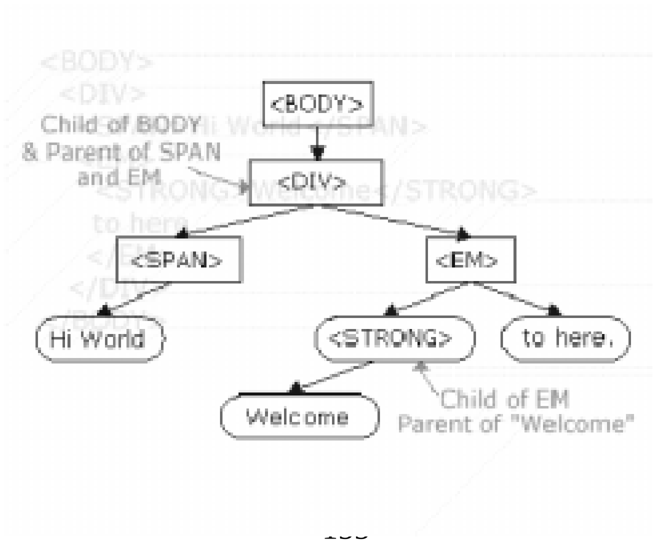
To create a new Tag we use the *createElement()* method of document. To give the new tag attributes, the *setAttribute()* method is used. For example, to create a node containing an "A" tag and then append the HREF attribute to it, we use the following:
*var link = document.createElement('a');*
*link.setAttribute('href', 'mypage.htm');*
The above example would create a link pointing to "*mypage.htm*" (<A HREF="mypage.htm">).
Once the node you wish to add has been created, you must locate where you wish to append it in the document tree. The document tree is the representation of all tags, attributes, and text (all nodes) in the Document Object Model. To locate the place in the document tree we wish to append our new node to, we have a choice. You may either assign an ID to the place in the document you wish the new node to be inserted, or navigate the document tree by use of children and parents. This will be discussed later. To use an ID to locate where we wish to append our new node we use *document.getElementById()*. This will return the object (whether it be a DIV, P, SPAN, etc..) with the given ID. For instance, to access the DIV with the ID "myDiv", we would use:
*document.getElementById("myDiv")*.

If you wish instead to access it via the hierarchy of the document, you must be familiar with the concept of a child and parent. A node is considered to be a child of any node it is nested inside of. Any node with other nodes nested inside of it is considered a parent. For instance, in all valid HTML documents, every tag but HTML should have a parent node, since everything should be nested inside of the HTML tag. The BODY tag will also almost certainly have many children, since their is usually at least one tag or piece of text inside the BODY of the document.
To access a node via the document tree, using children and parents, use *document.childNodes[]*, *document.nextSibling[]*, and *document.parentNode[]*. For example, if the document being worked with was:
*<HTML> <HEAD> <TITLE>My Document</TITLE> </HEAD> <BODY BGCOLOR="red"> <DIV ID="myDiv"></DIV> </BODY> </HTML>*
We could access "myDiv" by using:
*document.childNodes[1].childNodes[0]*"Find the main document element (HTML), and find its second child (BODY), then look for its first child (DIV)" or *document.childNodes[0]. nextSibling. childNodes[0]*
The meaning of the method is: "Find the main document element (HTML), and find its first child (HEAD), then find the next child (BODY), and find its first child (DIV)"

**4. Hiding/ showing elements**

The ability to hide and show objects cross browsers as IE or NS has been significantly changed with the ability to use in both major browsers. In order to hide and show objects in the 4.x browsers, one had to go through hoops of incompatibility.

Using the DOM the normal code is:


<BODY>
Child of BODY & Parent of SPAN and EM
<DIV>
<SPAN>
<EM>
Hi World
<STRONG>
to here.
Welcome
Child of EM Parent of "Welcome"

*<div>Hiding / Showing existing elements:</div>*

Notice how it doesn't even have an ID? Its not necessary to give it an ID, since you can access any element through the document tree (remember child and parent from the previous paragraph). To hide it, we first must locate it. You can do this using the ID or the document tree - both will work the same way. Once the object whose display we wish to manipulate has been located, you need to change its display property. You can access this property using *object.style.display*, like in the example below:

*document.childNodes[1].childNodes[1],*

*childNodes[0].style.display = "none";*

In this example, we used the document tree to locate the object we wished to change, and changed its display property value to "none". You will typically use "none" to hide an object, and "" to show an object. Those of you familiar with the Internet Explorer DOM will find this very familiar - IE uses the exact same syntax, except it has different ways to locate objects in the page. To make it easier to toggle an element's visibility, I usually use the following function:

*function HideShow(obj) { if (parseInt(navigator.appVersion) >= 5 || navigator.appVersion.indexOf["MSIE 5"] != -1) { if (obj.style.display=="none") obj.style.display=""; else obj.style.display="none"; } }*

**5. Event handling in the DOM**

One of the most significant method of DOM is the event. Event handling has been part of JavaScript since the language's inception. They refer to specific, user imitated actions within the webpage, such as the moving of your mouse over a link, the clicking on a link, or submission of a form. Thanks to event handling, our scripts are more interactive and are able to perform certain actions depending on the user's.

The DOM of IE5+ /NS6+ provides expanded methods and flexibility (relative to older browsers) for capturing events. In this part, we try to explain event handling in the DOM, and the differing support for it in IE5+ and NS6+. There are two usual ways of assigning event handlers

Let's first review (for most of us, at least) the 2 common and conventional ways of setting up an event handler- via HTML, or scripting. In both cases, a function or code is attached at the end, which is executed when the handler detects the specified event.

**5.1 By using HTML, and attributes**

We can define an event handler directly inside the relevant HTML tag, by embedding it as a attribute. A piece of JavaScript is also included to tell the browser to perform something when the event occurs. For example,

```
<a href="http://freewarejava.com" onMouseover="window.status='Click here for
Java applets';return true" onMouseout="window.status=''">Freewarejava.com</a>
```

The meaning is: here the event handler (onMouseover) is directly added inside the desired element (A), along with the JavaScript to execute.

**5.2 By scripting**

It can be also assigned and set up event handlers to elements using scripting, and inside own script. This allows for the event handlers to be dynamically set up, without having to mess around with the HTML codes on the page. When setting up event handlers for an element directly inside your script, the code to execute for the events must be defined inside a function. Just look at the below, which does the same thing as above, but with the event handler defined using scripting:

```
<a ID="test" href="http://freewarejava.com">Freewarejava.com</a>
<script>
function changestatus(){
window.status="Click here for Java applets" return true
function changebackstatus(){
window.status='' }
document.getElementById("test").onmouseover=changestatus
document.getElementById("test").onmouseout=changebackstatus
</script>
```

Notice how we attached the two functions (strongly necessary in NS) to execute for the two events- the function names without the parenthesis. This is called a reference call to the function. When assigning a function to an event via scripting, always use a function call. If you were to include the parenthesis of the function inside the definition, an error will be generated.

**6. The DOM event flow**

The DOM introduces a new concept for detecting events and assigning corresponding event handlers to react to them. Naturally, it also supports the 2 conventional techniques discussed earlier. In order to fully understand and appreciate this new method, we must first understand how events are handled in the DOM.

**6.1 A round trip ticket- event capture and event bubbling**

The DOM interprets all user action very differently from the user. Let's take the simple user action of moving the mouse over a link as an example. To the DOM, the following events in fact took place, and in that order, for the three instances:
*1) You moved your mouse over the document 2) You moved your mouse over any tags containing the target <A> tag 3) You moved your mouse over the <A> tag in question*

And more:
*4) You moved your mouse over any tags containing the target <A> tag 5) You moved your mouse over the document. (the end).*

Using a diagram, technically the events are:
Mouse over document -->Mouse over any containment tags of <A> –>Mouse over destination <A> --> Mouse over any containment tags of <A> --> Mouse over document. The above "roundtrip" model is used by the DOM to interpret most events. We call the event as it travels to the intended element (part in blue) *event capture*, and as it travels back *event bubble*.

Using object detection to sniff out different browsers
So how does object detection relate to browser detection? Well, since different browsers support different objects, you can use OD to actually discern what browser the surfer is using, by carefully selecting which objects you test. For example, since only NS 3+ and IE 4+ support the images object, by testing "document.images", you can screen out these browsers. NS 4 exclusively supports the layers object, so "document.layers" tests whether the surfer is using NS 4

**REFERENCES**

[1] xxx- DHTML and CSS for the World Wide Web, http://www.webbedenvironments.com/ dhtml/code/DHTML_CSSCode/
[2] M. Risteiu- Comunicare in Internet, “1 Decembrie 1918” University of Alba Iulia, 2001
[3] xxx- Images in IE vs NS?, http://www.experts-exchange.com/ Web/Web_Languages/HTML/Q_20224388.html

**Authors:** *Mircea Risteiu- “1 Decembrie 1918”University of Alba Iulia, Gheorghe Marc- S.C. SICOMED S.A., Pasculescu Dragos- University of Petrosani*