Scripting with Elements and Collections

Every HTML document consists of a combination of HTML tags and their attributes. These elements define the structure of the document and determine how the content is presented. Using the Dynamic HTML (DHTML) Object Model, you can examine and modify these elements and their modifying attributes. The following topics explain how to access the elements of the document using element collections.

The all Collection and the children Collection

An HTML document is a hierarchical construct of tags that define the contents of the document. The all [http://msdn.microsoft.com/en-us/library/ms537434(VS.85).aspx] collection on the document object represents all the elements in the document hierarchy. Each element is represented as a programmable object appearing within the collection in source order. Individual element objects are accessed by index or identifier (unique name), as shown in the following example.

Copy Code

```
<HTML>
<HEAD><TITLE>Elements: Collecting</TITLE>
<SCRIPT LANGUAGE="JScript">
function showElements() {
    var tag_names = "";
    for (i=0; i<document.all.length; i++)
        tag_names = tag_names + document.all(i).tagName + " ";
    alert("This document contains: " + tag_names);
}
</SCRIPT>
</HEAD>
<BODY onload="showElements()">
<H1>Welcome!</H1>
<P>This document is <B>very</B> short.
</BODY>
</HTML>
```

Code example: http://samples.msdn.microsoft.com/workshop/samples/author/dhtml/overview/elem_01.htm [http://samples.msdn.microsoft.com/workshop/samples/author/dhtml/overview/elem_01.htm]

When this document is loaded, the script (an event handler for the onload [http://msdn.microsoft.com/en-us/library/cc197055(VS.85).aspx] event) displays a list of the elements in the document by stepping through the **all** collection. It displays the following message.

Copy Code

```
This document contains: HTML HEAD TITLE SCRIPT BODY H1 P B
```

Each HTML element in the preceding example is represented in this message. The list does not show the end tags for the elements, because each element object represents both the start and end tags. Also, the collection does not directly indicate the hierarchy of the elements; that is, you can't tell which elements contain others. The collection lists the elements in the order in which they appear in the HTML source of the document.

In many ways, the **all** collection is similar to an array. It contains one or more items, each of the same type—in this case, element objects. You access the items using zero-based index values, or by name or identifier. The first item has index value 0, the second has 1, and so on. You can determine how many items are in the collection by using the length [http://msdn.microsoft.com/en-us/library/ms534101(VS.85).aspx] property.

Because each item in the **all** collection is an element object, you can apply properties and methods to these items. For example, you can use the tagName [http://msdn.microsoft.com/en-us/library/ms534657(VS.85).aspx] property to retrieve the HTML tag name of the element, as was done in the previous example. Similarly, you can access properties and methods of the respective element by accessing this through the document [http://msdn.microsoft.com/en-us/library/ms531073(VS.85).aspx] .**all** collection.

The **all** collection always represents the current state of the document and is automatically updated to reflect any changes made to the document. For example, if you retrieve the collection and add or remove document content that changes the HTML structure, the collection automatically reflects the new HTML content in source order.

In some cases, the **all** collection might contain more elements than are actually in the document's file. In particular, the collection always contains the html [http://msdn.microsoft.com/en-us/library/ms535255(VS.85).aspx] , head [http://msdn.microsoft.com/en-us/library/ms535252(VS.85).aspx] , TITLE [http://msdn.microsoft.com/en-us/library/ms535910(VS.85).aspx] , and BODY [http://msdn.microsoft.com/en-us/library/ms535205(VS.85).aspx] elements, even if these are not present in the source. Similarly, the collection always contains a tBody [http://msdn.microsoft.com/en-us/library/ms535902(VS.85).aspx] element for each table [http://msdn.microsoft.com/en-us/library/ms535901(VS.85).aspx] , regardless of whether **tBody** is specified in the HTML source.

The **all** collection also includes comments (!) and unknown or invalid tags. The purpose is to give you an accurate picture of the content of the document. Unknown or invalid tags are typically misspelled or misplaced tags. Knowing what and where these tags are provides an opportunity to replace them with valid tags. The **all** collection lists unknown and invalid start and end tags separately; it does not attempt to combine them into a single item.

The following example displays this message "HTML HEAD TITLE SCRIPT BODY ! P ZZZ> /ZZZ> /B".

Copy Code

```
<HTML>
<HEAD><TITLE>Elements: Collecting</TITLE>
<SCRIPT LANGUAGE="JScript">
function showElements() {
    var tag_names = "";
    for (i=0; i<document.all.length; i++)
        tag_names = tag_names + document.all(i).tagName + " ";
    alert("This document contains: " + tag_names);
}
</SCRIPT>
</HEAD>
<BODY onload="showElements()">
<!-- A comment -->
<P>This document has an <ZZZ>unknown</ZZZ> and an invalid</B> tag.
</BODY>
</HTML>
```

Code example: http://samples.msdn.microsoft.com/workshop/samples/author/dhtml/overview/elem_02.htm [http://samples.msdn.microsoft.com/workshop/samples/author/dhtml/overview/elem_02.htm]

In addition to the **all** collection on the **document** object, each individual element also exposes an **all** collection. Remember the hierarchical style of HTML—this helps to think about the **all** collection for each element. The **all** collection for an element contains all the elements contained by that element. For example, the **all** collection for the HTML element would contain everything in the source code except the HTML element (this being the only difference between the **all** collection for the HTML element and the **document**.**all** collection).

Each element also exposes a children [http://msdn.microsoft.com/en-us/library/ms537446(VS.85).aspx] collection, which contains only the elements that are direct descendants of the element in the HTML hierarchy. Another way of saying this is that the **children** collection contains only those elements whose parentElement [http://msdn.microsoft.com/en-us/library/ms534327(VS.85).aspx] property returns that element. The content of the **children** collection is undefined for overlapping elements.

The following example returns the length and content of the **all** and the **children** collections of the document.

Notice that there are only two elements that are children of the **html** tag—**head** and **BODY**. The **TITLE** and script [http://msdn.microsoft.com/en-us/library/ms535892(VS.85).aspx] elements are children of the **head** tag, but not of the **html** tag.

Copy Code

```
<HTML id=theHTML>
<HEAD>
<TITLE> Look at the all and the children collection</TITLE>
<SCRIPT>
function showme() {
alert('all: ' + window.theHTML.all.length);
for (i=0; i < theHTML.all.length;i++)
{
alert(theHTML.all[i].tagName);
}
alert('children: ' + window.theHTML.children.length);
for (i=0; i < theHTML.children.length;i++)
{
alert(theHTML.children[i].tagName);
}
}
</SCRIPT>
</HEAD>
<BODY onload=showme()>
<DIV> Some text in a DIV. This DIV will be in
the all collection of the HTML element.</DIV>
</BODY>
</HTML>
```

Code example: http://samples.msdn.microsoft.com/workshop/samples/author/dhtml/overview/elem_03.htm [http://samples.msdn.microsoft.com/workshop/samples/author/dhtml/overview/elem_03.htm]

Using Collections

The DHTML Object Model exposes several collections [http://msdn.microsoft.com/en-us/library/ms533048(VS.85).aspx] that you can use to access the elements of the document. Collections in the DHTML Object Model are similar to an array of variables created in a programming language. You can retrieve an item from the collection by providing a name or identifier for the item, or an ordinal index that specifies the position of the item in the collection. Collections indexes are zero-based, so the first item always has an index of zero. The following Microsoft JScript example displays the tag name of the p [http://msdn.microsoft.com/en-us/library/ms535878(VS.85).aspx] element, the sixth element in the collection.

Copy Code

```
<HTML>
<HEAD><TITLE>Elements: Retrieving Items</TITLE>
<SCRIPT LANGUAGE="JScript">
function showSixth() {
    var el = document.all(5);
    alert("The sixth element is: " + el.tagName);
}
</SCRIPT>
</HEAD>
<BODY onload="showSixth()">
<P>This is a very simple document.
</BODY>
</HTML>
```

Code example: http://samples.msdn.microsoft.com/workshop/samples/author/dhtml/overview/elem_04.htm [http://samples.msdn.microsoft.com/workshop/samples/author/dhtml/overview/elem_04.htm]

Because index values for collections are zero-based, the index for the last element in a collection is always one less than the length of the collection. So, a quick way to obtain access to the last element in a collection is to subtract one from its length and use the result as the index value, as shown in the following example.

Copy Code

```
var elLast = document.all(document.all.length-1);
alert("The last element in this document is: " + elLast.tagName);
```

You can also retrieve items from a collection by specifying an identifier. In this case, the element must have a valid identifier, set using the **ID** attribute. The following JScript example displays the tag name of the element that has the identifier *MyID*.

Copy Code

```
<HTML>
<HEAD><TITLE>Elements: IDs as Indexes</TITLE>
<SCRIPT LANGUAGE="JScript">
function showElementWithID() {
    var el = document.all("MyID");
    alert("The element having the identifier MyID is: " + el.tagName);
}
</SCRIPT>
</HEAD>
<BODY onload="showElementWithID()">
<P ID="MyID">This is a very simple document.
</BODY>
</HTML>
```

Code example: http://samples.msdn.microsoft.com/workshop/samples/author/dhtml/overview/elem_05.htm
[http://samples.msdn.microsoft.com/workshop/samples/author/dhtml/overview/elem_05.htm]

Elements can also be accessed by name, but it is important to remember that the name attribute primarily applies only to the elements that participate in a form submission. The ID [http://msdn.microsoft.com/en-us/library/ms533880(VS.85).aspx] attribute is an identifier that applies to every element.

Be careful! While in theory ID means "unique identifier," in reality the uniqueness is not enforced. If more than one element has the same identifier, the result is a collection of elements with the same ID, rather than an individual element. The subsequent collection can only be accessed ordinally. If no element has the identifier, the result is null.

When using an identifier results in another collection, the elements in the new collection have the same order as they have in the document. To obtain access to these elements, you can use the item [http://msdn.microsoft.com/en-us/library/ms536460(VS.85).aspx] method to apply a zero-based index to the collection, as in the following example.

Copy Code

```
if (document.all("MyID").length != null)
    alert("The first element that has the identifier MyID:
              " + document.all("MyID").item(0).tagName);
```

If you already know that more than one element has the same identifier, you can skip a step and use the optional second parameter of the **item** method to retrieve an element without first retrieving the collection. The following JScript example displays the same message as the preceding example.

Copy Code

```
var el = document.all.item("MyID",0);
if (el != null)
    alert("The first element that has the identifier MyID: " + el.tagName);
```

When using scripting languages, such as JScript and Microsoft Visual Basic Scripting Edition (VBScript), you can

also access elements in collections by using the indexing syntax for that language. JScript offers three ways to index an array: an integer index enclosed in square brackets, an identifier given as a string enclosed in square brackets, and an identifier prefixed with a period. The following JScript statements show how to apply these.

Copy Code

```
var el = document.all[2];    // is the same as document.all.item(2)
var el = document.all["MyID"];   // is the same as document.all.item("MyID")
var el = document.all.MyID;      // is the same as document.all.item("MyID")
```

Again, be careful when using identifiers in this way, because the collection returns another collection if more than one element has the given identifier. Always check the return value before you use it.

You can't use identifiers to access items in an imports [http://msdn.microsoft.com/en-us/library/aa358805(VS.85).aspx] collection—the collection consists of imported style sheets rather than HTML elements, so no identifiers are available. Use integer indexes instead. Although you can use names to access items in a frames [http://msdn.microsoft.com/en-us/library/ms537459(VS.85).aspx] collection, make certain that no two windows have the same name. If you don't, only the first window with the given name will be accessible.

Creating new Collections: The tags Method

The tags [http://msdn.microsoft.com/en-us/library/ms536776(VS.85).aspx] method creates a collection of elements that have a given tag name. This method filters an existing collection and creates a new collection of a given tag name. For example, the following JScript code applies the **tags** method to the **all** collection to retrieve a new collection containing only the **table** elements in the document. It then applies a border to each table.

Copy Code

```
var doc_tables = document.all.tags("TABLE");
for (i=0; i<doc_tables.length; i++)
    doc_tables(i).border = 1;
```

The **tags** method searches for any tag name, even names that are not valid HTML. If it finds one or more elements having that name, it returns the collection. If it can't find an element having that name, it returns an empty collection. Use the **length** property to determine how many elements the collection contains. The collection is empty if its length is zero.

The **tags** method preserves the order of the elements as it creates the new collection. This means the first element in the new collection is also the first instance of the tag in the document.

Because the collection returned by the **tags** method is a subset of the **all** collection, the ordinal value that you use to access an element is different from the one you would use to access an element in the **all** collection. Sometimes it is useful to know both ordinal values. The sourceIndex [http://msdn.microsoft.com/en-us/library/ms534635(VS.85).aspx] property for the element returns the ordinal position of the element in the document's **all** collection.

Accessing Element Properties

Many element properties are named-value pairs that correspond to the HTML attributes of the element and to other values associated with the element. You can get the value of a property to determine how the corresponding attribute is set, and in many cases change the value of this property to dynamically affect the element. This is the key concept behind manipulating dynamic styles and dynamic content. Accessing the properties that let you change styles and content is done by accessing the element object through the collections exposed in the DHTML Object Model. For example, you can center an H1 [http://msdn.microsoft.com/en-us/library/ms535253(VS.85).aspx] element by accessing the **H1** element through the **all** collection on the document and then setting its align [http://msdn.microsoft.com/en-us/library/ms533066(VS.85).aspx] property to "center", the same value you would use with the **ALIGN** attribute.

Copy Code

```
var coll = document.all.tags("H1");
```

```
if (coll.length>0)
    coll(0).align="center";
```

Similarly, you can change the image file name for an img [http://msdn.microsoft.com/en-us/library/ms535259(VS.85).aspx] element by setting its src [http://msdn.microsoft.com/en-us/library/ms534643(VS.85).aspx] property.

Copy Code

```
var coll = document.all.tags("IMG");
if (coll.length>0)
    coll(0).src="newimage.gif";
```

Most properties have the same name and take the same values as the corresponding attribute. This means most properties take strings, enumerated values, or numeric values. If a property name is different from its corresponding attribute, it is usually fairly close to the attribute name. For example, the className [http://msdn.microsoft.com/en-us/library/ms533560(VS.85).aspx] property corresponds to the **CLASS** attribute. Name changes are done to avoid conflicts with keywords in common scripting languages, or to ensure that property names don't contain characters that are invalid in common scripting languages.

A few properties represent subobjects. For example, the style [http://msdn.microsoft.com/en-us/library/ms536498(VS.85).aspx] object corresponds to the **STYLE** attribute and represents a subobject that provides access to all the Cascading Style Sheets (CSS) attributes that can be applied to the element as an inline style. The **style** object itself takes properties, which you can set to alter the look and formatting of the element. The following example uses the **style** object to change the CSS color attribute of all **H1** elements to green.

Copy Code

```
var coll = document.all.tags("H1");
for (i=0; i<coll.length; i++)
    coll[i].style.color = "green";
```

Some properties do not correspond to element attributes. These properties typically provide additional information about the element that isn't available through attributes. For example, the **tagName** property specifies the HTML tag name of the element, and the **sourceIndex** property specifies the position of the element in the document's **all** collection.

Although you can get and set the values of many properties, some properties are read-only, meaning you can get the values but you cannot set or change them. For example, the **tagName** property is a read-only property—changing the tag name of an element is not permitted.

Sometimes an HTML document has invalid attributes, or invalid values assigned to attributes. If an invalid value is assigned to an attribute, the corresponding property is always set to its default value. In the following example, "middle" is not a valid value for the **ALIGN** attribute, so the document displays the message "left", which is the default value for the align [http://msdn.microsoft.com/en-us/library/ms533069(VS.85).aspx] property.

Copy Code

```
<HTML>
<HEAD><TITLE>Elements: Using Properties</TITLE>
<SCRIPT LANGUAGE="JScript">
function showAlignment() {
    var coll = document.all.tags("H1");
    if (coll.length>0)
        alert("The alignment for the first heading is " + coll(0).align);
}
</SCRIPT>
</HEAD>
<BODY onload="showAlignment()">
<H1 ALIGN="center">Welcome!</H1>
```

```
<P>This document is <B>very</B> short.
</BODY>
</HTML>
```

Code example: http://samples.msdn.microsoft.com/workshop/samples/author/dhtml/overview/elem_06.htm
[http://samples.msdn.microsoft.com/workshop/samples/author/dhtml/overview/elem_06.htm]

The getAttribute [http://msdn.microsoft.com/en-us/library/ms536429(VS.85).aspx] , setAttribute [http://msdn.microsoft.com/en-us/library/ms536739(VS.85).aspx] , and removeAttribute [http://msdn.microsoft.com/en-us/library/ms536696(VS.85).aspx] methods give you access to the attributes of an element without using properties. You supply an attribute name, and these methods either return or set the current value of the attribute, or they remove the attribute.

For example, you can get and set the current value of the **ALIGN** attribute (rather than retrieve the **align** property) by using **getAttribute** and **setAttribute**, as shown in the following example. The document displays the current value of the attribute ("left"), then changes that value to "center", centering the **H1** element.

Copy Code

```
<HTML>
<HEAD><TITLE>Elements: Using Methods</TITLE>
<SCRIPT LANGUAGE="JScript">
function showAndSetAlignment() {
    alert(MyHeading.getAttribute("align"));
    MyHeading.setAttribute("align","center");
}
</SCRIPT>
</HEAD>
<BODY onload="showAndSetAlignment()">
<H1 ID=MyHeading ALIGN="left">Welcome!</H1>
<P>This a short document.
</BODY>
</HTML>
```

Code example: http://samples.msdn.microsoft.com/workshop/samples/author/dhtml/overview/elem_07.htm
[http://samples.msdn.microsoft.com/workshop/samples/author/dhtml/overview/elem_07.htm]

The attribute methods are especially useful when working with unrecognized attributes and invalid values. For example, if an attribute has an invalid value, the **getAttribute** method retrieves that exact value rather than the default value for the attribute. This means you can use the method to examine what is really in the HTML document and make decisions based on that. The following example uses the **getAttribute** and **setAttribute** methods to repair the alignment of the **H1** element ("middle" is not a valid value).

Copy Code

```
<HTML>
<HEAD><TITLE>Elements: Invalid Values</TITLE>
<SCRIPT LANGUAGE="JScript">
function checkValues() {
    if (MyHeading.getAttribute("align")=="middle")
        MyHeading.setAttribute("align","center");
    if (MyHeading.getAttribute("xyz")!="123")
        MyHeading.setAttribute("xyz","123");
}
</SCRIPT>
</HEAD>
<BODY onload="checkValues()">
<H1 ID=MyHeading XyZ="123" ALIGN="middle">Welcome!</H1>
<P>This is a short document.
</BODY>
</HTML>
```

Code example: http://samples.msdn.microsoft.com/workshop/samples/author/dhtml/overview/elem_08.htm [http://samples.msdn.microsoft.com/workshop/samples/author/dhtml/overview/elem_08.htm]

By default, the **getAttribute** and **setAttribute** attribute methods are case insensitive when looking for an unknown attribute. So, in the preceding example, even though "xyz" isn't identical to "XyZ," these methods get and set the attribute anyway. You can force these methods to search for an exact match by setting the optional case-sensitive parameter to true. For example, the following code, if placed in the preceding example, displays "Not found", because "xyz" does not match "XyZ".

Copy Code

```
if (document.all.MyHeading.getAttribute("xyz", true)==null)
    alert("Not found");
```

If multiple matches are found for a case-sensitive usage of these methods, the actual item returned is not guaranteed to be consistent across platforms.

Examining the Element Hierarchy

The elements of an HTML document form a hierarchy in which some elements contain others. The **html** element, at the top of the hierarchy, contains all other elements in the document. The **BODY** element, contained by **html**, represents the document. Block elements, such as **p**, contain text and inline elements, which can themselves contain text and inline elements, and so on.

You can examine this structural hierarchy using the contains [http://msdn.microsoft.com/en-us/library/ms536377(VS.85).aspx] method and the **parentElement** property. The **all** collection and the **children** collection for each element can also help determine the document structure.

The **contains** method returns either

```
true
```

or

```
false
```

to indicate whether one element contains another. You apply the method to an element and supply another element as the parameter. For example, the following example displays the message "True!", because the **BODY** element contains **p**.

Copy Code

```
<HTML>
<HEAD><TITLE>Elements: Hierarchy</TITLE>
<SCRIPT LANGUAGE="JScript">
function showContains() {
    var el = document.all.tags("P").item(0);
    if (document.body.contains(el))
        alert("True!");
    else
        alert("False!");
}
</SCRIPT>
</HEAD>
<BODY onload="showContains()">
<P>This is a very simple document.
</BODY>
</HTML>
```

Code example: http://samples.msdn.microsoft.com/workshop/samples/author/dhtml/overview/elem_09.htm [http://samples.msdn.microsoft.com/workshop/samples/author/dhtml/overview/elem_09.htm]

Note In the preceding example, the body [http://msdn.microsoft.com/en-us/library/ms535205(VS.85).aspx] property of the **document** object is used to retrieve the **body** object.

The **parentElement** property always identifies the next element in the hierarchy that contains the element. You can combine the **contains** method and **parentElement** in a more complicated script to determine the hierarchy of the elements document. The following example uses these to determine the hierarchy, then it displays a message that shows the hierarchy.

Copy Code

```
<HTML>
<HEAD><TITLE>Elements: Hierarchy</TITLE>
<SCRIPT LANGUAGE="JScript">
function showHierarchy() {
    var depth = 0;
    var msg = document.all(0).tagName;
    for (i=1; i<document.all.length; i++) {
        if (document.all(i-1).contains(document.all(i))==true) {
            depth = depth + 1;
        } else {
            var elParent = document.all(i-1).parentElement;
            for ( ; depth>0; depth--) {
                if (elParent.contains(document.all(i))==true)
                    break;
                elParent = elParent.parentElement;
            }
        }
        msg = msg + "\n";
        for (j=1; j<=depth; j++)
            msg = msg + "  ";
        msg = msg + document.all(i).tagName;
    }
    alert("This document contains:\n" + msg);
}
</SCRIPT>
</HEAD>
<BODY onload="showHierarchy()">
<H1>Welcome!</H1>
<P>This document is <B>very</B> short.
</BODY>
</HTML>
```

Code example: http://samples.msdn.microsoft.com/workshop/samples/author/dhtml/overview/elem_10.htm [http://samples.msdn.microsoft.com/workshop/samples/author/dhtml/overview/elem_10.htm]

Getting an Element's Position and Dimensions

You can determine the location, width, and height of an element using the offsetLeft [http://msdn.microsoft.com/en-us/library/ms534200(VS.85).aspx] , offsetTop [http://msdn.microsoft.com/en-us/library/ms534303(VS.85).aspx] , offsetHeight [http://msdn.microsoft.com/en-us/library/ms534199(VS.85).aspx] , and offsetWidth [http://msdn.microsoft.com/en-us/library/ms534304(VS.85).aspx] properties. These numeric properties specify the physical coordinates and dimensions of the element relative to the element's offset parent. The following example is a simple clock that adjusts the size of its readout to fit the current width and height of the document body.

Copy Code

```
<HTML>
<HEAD><TITLE>A Simple Clock</TITLE>
<SCRIPT LANGUAGE="JScript">
function startClock() {
    window.setInterval("Clock_Tick()", 1000);
    Clock_Tick();
}

var ratio = 4;
```

```
function Clock_Tick()
{
    var s = Date();
    var t = s.substring(11,19);
    var doc_height = document.body.offsetHeight;
    var doc_width = document.body.offsetWidth;

    if ((doc_height*ratio)>doc_width)
        doc_height = doc_width / ratio;
    MyTime.innerText = t;
    MyTime.style.fontSize = doc_height;
}
</SCRIPT>
</HEAD>
<BODY onload="startClock()">
<P ID="MyTime"> </P>
</BODY>
</HTML>
```

Code example: http://samples.msdn.microsoft.com/workshop/samples/author/dhtml/overview/elem_11.htm [http://samples.msdn.microsoft.com/workshop/samples/author/dhtml/overview/elem_11.htm]

The **offsetLeft** and **offsetTop** property values are relative to the element specified by the offsetParent [http://msdn.microsoft.com/en-us/library/ms534302(VS.85).aspx] property for the element. Most of the time the property returns **body**. In the following example, even though the td [http://msdn.microsoft.com/en-us/library/ms535903(VS.85).aspx] element appears to the far right in the document, its position is (0,0) because its offset parent is the tr [http://msdn.microsoft.com/en-us/library/ms535911(VS.85).aspx] element, not the document body.

Copy Code

```
<HTML>
<HEAD><TITLE>Elements: Positions</TITLE>
<SCRIPT LANGUAGE="JScript">
function showPosition() {
    var el = MyID;
    alert("The TD element is at (" + el.offsetLeft + "," + el.offsetTop + ")\n" +
        "The offset parent is " + el.offsetParent.tagName );
}
</SCRIPT>
</HEAD>
<BODY onload="showPosition()">
<P>This document contains a right-aligned table.
<TABLE BORDER=1 ALIGN=right>
<TR><TD ID=MyID>This is a small table.
</TABLE>
</BODY>
</HTML>
```

Code example: http://samples.msdn.microsoft.com/workshop/samples/author/dhtml/overview/elem_12.htm [http://samples.msdn.microsoft.com/workshop/samples/author/dhtml/overview/elem_12.htm]

Scrolling Elements into View

Another useful method is scrollIntoView [http://msdn.microsoft.com/en-us/library/ms536730(VS.85).aspx] . This method scrolls the element into view within the window, placing it at either the top or bottom of the window. The method is useful for immediately showing the user the result of some action, without requiring the user to manually scroll the document to find the result. The following example underlines the content of the fifth paragraph and scrolls it into view at the top of the window.

Copy Code

```
var coll = document.all.tags("P");
if (coll.length>=5) {
    coll(4).style.textDecoration = "underline";
    coll(4).scrollIntoView(true);
}
```

Tags:

Community Content