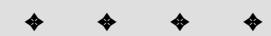
CHAPTER 27

Table and List Objects



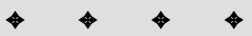
In This Chapter

Modifying table cell content

Adding and deleting table rows

TABLE, CAPTION, TBODY, TFOOT, THEAD, COL, COLGROUP, TH, TR, and TD element objects

OL, UL, LI, and DL list element objects



Tables are incredibly popular HTML constructions. When you consider that a lot of CGI programs search SQL databases and display data gathered from SQL tables, it's not unusual to find the table concept carried over from data storage to data display. Spreadsheet programs certainly put the notion of tabular display into the minds of most computer users.

One of the truly beneficial properties of tables in HTML is that they pack a lot of page organization and alignment punch in just a few tags and attributes. Even if you're not a graphics designer or a dedicated HTML jockey, you can get rows and columns of text and images to line up perfectly on the page. This behavior also lures many page designers to sculpt elaborately detailed pages out of what appear to be positioned elements. Earlier browsers didn't offer positioning facilities, so borderless tables were torqued into performing all kinds of placement tricks with the help of precisely sized, transparent images creating the illusion of white space between carefully placed elements. If you use some of the WYSIWYG authoring tools for HTML pages, you may not realize how much table-related HTML code is generated for you as you use the tool to drag an image to a particular location on the page.

Someone probably could write an entire book on the HTML aspects of tables by themselves, especially when taking into account the variability of rendering that can occur. But that's not the task at hand. The first part of this chapter focuses on the scriptable aspects of TABLE element objects and the shopping list of elements that support tables. All of these objects became scriptable objects in browsers starting with IE4 and NN6. Later in the chapter, I discuss element objects that create formatted lists in pages.

The Table Object Family Hierarchy

The repertoire of table-related elements expanded a bit with the HTML 4.0 specification, and the W3C DOM built upon that foundation. While most of this discussion is best left to HTML texts, the structure of a full-fledged table and the relationships among the elements — particularly the parent-child relationships — may affect your scripting and event handling.

You are probably well familiar with the most basic table structure that predates HTML 4.0. Such a table (in a 2×2 layout) can have the following form:

```
<TABLE>
    <TR>
        <TD></TD>
        <TD></TD>
    </TR>
    <TR>
        <TD></TD>
        <TD></TD>
    </TR>
</TABLE>
```

If you want to place a row of cells at the top of each column such that the contents of the cells act as headers for each column, then add such a row as follows:

```
<TABLE>
    <TR>
        <TH></TH>
        <TH></TH>
    </TR>
    <TR>
        <TD></TD>
        <TD></TD>
    </TR>
    <TR>
        <TD></TD>
        <TD></TD>
    </TR>
</TABLE>
```

You can also include a caption associated with the table. Its tag goes immediately after the TABLE element's start tag:

```
<TABLE>
    <CAPTION></CAPTION>
    <TR>
        <TH></TH>
        <TH></TH>
    </TR>
    <TR>
```

```
        <TD></TD>
        <TD></TD>
    </TR>
    <TR>
        <TD></TD>
        <TD></TD>
    </TR>
</TABLE>
```

In line with its emphasis on providing contextual tags, HTML 4.0 includes three tags that enable you to define groups of table rows according to whether they are the header, body, or footer of the table (THEAD, TBODY, and TFOOT elements, respectively). A table footer, for example, can display column totals. The only seemingly illogical rule about these elements is that you should define the TFOOT element and its row contents before the TBODY element(s) in the table. Even with this source code placement, the TFOOT row appears at the bottom of the table.

Some browsers produce visual dividers between these sections (IE5+ for Windows does a nice job of this). Moreover, you can have multiple TBODY sections within a table. Some browsers render dividers between these TBODY sections (again, IE5+ for Windows does it well). Regardless of the built-in divider support, these contextual groupings also enable you to assign style sheets to HTML tag selectors rather than having to dream up a scheme of class and ID names tied to style sheet rules. Building upon the skeletal table shown thus far, you add the THEAD and TBODY elements like this:

```
<TABLE>
    <CAPTION></CAPTION>
    <THEAD>
        <TR>
            <TH></TH>
            <TH></TH>
        </TR>
    </THEAD>
    <TBODY>
        <TR>
            <TD></TD>
            <TD></TD>
        </TR>
        <TR>
            <TD></TD>
            <TD></TD>
        </TR>
    </TBODY>
</TABLE>
```

That's the extent of table-oriented HTML containers. The remaining two elements, COLGROUP and COL, provide a different "slice" of the table for style sheets and other visual groupings. One of the most obvious purposes of these two elements is to assign a width or other style to all cells in a particular column or group of columns. You can also use these elements to group adjacent columns so that

dividers are drawn between groups of columns — if the browser (such as IE5+ for Windows) supports dividers between column groups — without specifying global table borders. You can see an example of the HTML for a complex table in the HTML 4.0 specification (`http://www.w3.org/TR/REC-html40/struct/tables.html#h-11.5`). Elsewhere on that same page, you can find the formal specification for all table-related tags and attributes as defined by the W3C.

Populating table cells

Source material for a table's content can come from many different places. Most of the tables you see on the Web are hard-coded in the HTML. That is to say, the content of the table is fixed inside a static HTML file on the server.

But tables may also convey content from live databases or content that changes more frequently than the Web site's author manually updates other content. The source and your Web development infrastructure (not to mention your technical skills) dictate other avenues for populating tables.

After hard-coded HTML files, the next most common way to generate tables is through server-based CGI programs. These programs (written in Perl, C, and many other languages, including server-side JavaScript on those few servers that support it) generally compose a query for the database and then repackage the data returned from the database into HTML-formatted pages.

A more client-side-oriented approach is to let JavaScript apply the `document.write()` method to compose the table's tags as the page loads. Data for the cells can come from JavaScript arrays defined at the beginning of the document or defined in external `.js` library files that are linked in as the page loads. In the newest browsers, the data may come from blocks of XML-formatted data stuffed into the document. These solutions can work in situations where you need to update the table data periodically, but the table delivered to the client does not reflect the instantaneous state of a database. For example, a daily batch program on a server can capture the day's sales totals and write out a `.js` text file to a known place on the server. The file consists entirely of JavaScript array definitions. When the HTML page loads, the current `.js` file is automatically loaded into the page, and `document.write()` statements compose the table's HTML from the data supplied in the arrays. While the script that assembles the HTML for the tables might appear formidable to a nonscripter, a nonscripter can also manually update the array data by following a template format supplied by the programmer.

Finally, if your page visitors run IE4+ for Windows (only), you can take advantage of a Windows-specific technology called *data binding*. Data binding invokes the powers of one or more ActiveX controls that come with the IE browser. These objects (collectively called Data Source Objects) let HTML pages access ODBC databases (as well as some formatted text files). As the page loads, the table fills with data pulled live from the database. You can see an example of data binding in Chapter 15 under the description of the data binding property: `dataFld`. The HTML file carries tags for only one row of cells, but data binding fills in the rest of the rows and cells.

Modifying table cell content

You can modify the HTML content of a table cell directly in IE4+ and NN6+. Some tricks with positioned elements in NN4 can, under some circumstances, make it appear to the user as if the table content is being modified.

By far, the most compatible way to modify a table cell's content in IE4+ and NN6+ is via the TD element's `innerHTML` property (a Microsoft invention that is not sanctioned by the W3C DOM Level 2 but is supported in NN6). Even if the content is simply text that is to inherit the style format of the surrounding TD element, you can still use the `innerHTML` property. If the size of the new content affects the dimensions of the cell's column width or row height, the browser reflows the rest of the table content around the new content.

If you prefer to follow the W3C DOM form of modifying an element's content (for IE5+ and NN6), then you can generate the new content via the `document.createElement()` sequence and assign that new content to the cell by way of the TD element's `replaceChild()` method.

The situation for NN4 is quite gnarled because the content you replace must be within its own layer (either a LAYER element or positioned container element, such as a DIV or SPAN). No matter how you create the layer in your HTML, you must overcome the problem that a layer floats in its own plane and must be positioned precisely where the table cell is. Table cells are not objects in NN4, so you must create a positioning context in the cell by first creating a relative-positioned layer that can contain nothing more than an "invisible" nonbreaking space character (` `). The layer displaying the content must be absolute-positioned with respect to that relative-positioned layer. Nesting of layers in NN4 causes headaches, especially when scripts reference the deeply nested content — content that is, essentially, an HTML document inside the nested layer.

Listing 27-1 shows a synthesis of different techniques to effect cell content replacement, including script code branches that emulate the appearance of replacement in NN4. The table represents only one line of what might be an order form for several products. As the user makes a selection of the quantity, the extended total is displayed in the rightmost column.

You can find the key features of the NN4 implementation in the script that dynamically writes the table cell content within the HTML as the page loads. The cell begins with a relative-positioned SPAN element. This SPAN is positioned at the top left of the table cell, as planned. That spot now is the positioning context for the absolute-positioned SPAN nested inside it. This second span is the layer whose document contains the displayed content. The content, itself, is yet another SPAN element because it simplifies the application of a style sheet rule (to display the total in red) when you replace the content. Because a newly written NN4 layer does not inherit the style sheet of its next outermost layer, you must apply the style as part of the new content.

The initial SPAN content contains a series of nonbreaking space characters that force NN4 to open space for eventual replacement content. Recall that an NN4 page

does not reflow the page to accommodate resized content. This means that whatever you intend to insert in the table cell can be no larger than the original space allocated for it.

Although the page shown in Listing 27-1 consists of only one row of data, the scripts and naming conventions are intended to be carried out among multiple rows. The product name appears in several object names and IDs in each row, and the scripts count on the convention being followed throughout. In fact, the regularity of the namings can allow the content for a table's row to form a script function that is invoked for each table row. The product code name can be passed as the parameter, and all object names and IDs can be assembled in that function. The regularity of table content often lends itself to script-generated construction.

For NN4, when the table gets complicated, you will have more success defining the absolute-positioned elements outside of the table entirely. They should be defined on their own at the bottom of the BODY. You can still position them with respect to the relative-positioned elements in the table, but all such layers are now only one level deep within the main document.

Listing 27-1: **Replacing Table Cell Content**

```
<HTML>
<HEAD>
<TITLE>Modifying Table Cell Content</TITLE>
<STYLE TYPE="text/css">
.absoluteWrap {position:absolute}
.relativeWrap {position:relative}
.total {color:red}
</STYLE>

<SCRIPT LANGUAGE="JavaScript">
var Ver4 = parseInt(navigator.appVersion) == 4
var Ver4Up = parseInt(navigator.appVersion) >= 4
var Nav4 = ((navigator.appName == "Netscape") && Ver4)
var modifiable

// calculate and display a row's total
function showTotal(qtyList) {
    var qty = qtyList.options[qtyList.selectedIndex].value
    var prodID = qtyList.name
    var total = "US$" +
        (qty * parseFloat(qtyList.form.elements[prodID + "Price"].value))
    var newCellHTML = "<SPAN CLASS='total'>" + total + "</SPAN>"

    if(Nav4) {
        document.layers[prodID + "TotalWrapper"].document.layers[prodID +
            "Total"].document.write(newCellHTML)
        document.layers[prodID + "TotalWrapper"].document.layers[prodID +
            "Total"].document.close()
    } else if (modifiable) {
```

```
        if (document.all) {
            document.all(prodID + "Total").innerHTML = newCellHTML
        } else {
            document.getElementById(prodID + "Total").innerHTML = newCellHTML
        }
    }
}

// initialize global flag for browsers capable of modifiable content
function init() {
    modifiable  = (Ver4Up && document.body && document.body.innerHTML)
}

// display content for all products (e.g., in case of Back navigation)
function showAllTotals(form) {
    for (var i = 0; i < form.elements.length; i++) {
        if (form.elements[i].type == "select-one") {
            showTotal(form.elements[i])
        }
    }
}
</SCRIPT>
</HEAD>

<BODY onLoad="init(); showAllTotals(document.orderForm)">
<H1>Modifying Table Cell Content</H1>
<HR>
<FORM NAME="orderForm">
<TABLE BORDER=1>
<COLGROUP WIDTH=150>
<COLGROUP WIDTH=100>
<COLGROUP WIDTH=50>
<COLGROUP WIDTH=100
<TR>
    <TH>Product Description</TH>
    <TH>Price Each</TH>
    <TH>Quantity</TH>
    <TH>Total</TH>
</TR>
<TR>
    <TD>Wonder Widget 9000</TD>
    <TD>US$125.00</TD>
    <TD><SELECT NAME="ww9000" onChange="showTotal(this)">
        <OPTION VALUE="0">0
        <OPTION VALUE="1">1
        <OPTION VALUE="2">2
        <OPTION VALUE="3">3
        </SELECT>
        <INPUT TYPE="hidden" NAME="ww9000Price" VALUE="125.00"></TD>
    <TD>
    <SCRIPT LANGUAGE="JavaScript">
    if (Nav4) {
```

Continued

Listing 27-1 *(continued)*

```
        var placeHolder = "        "
        placeHolder += "       "
        document.write("<SPAN ID='ww9000TotalWrapper' CLASS='relativeWrap'>")
        document.write("<SPAN ID='ww9000Total' CLASS='absoluteWrap'></SPAN>")
        document.write("<SPAN>" + placeHolder + "</SPAN></SPAN>")
    } else {
        document.write("<SPAN ID='ww9000Total' CLASS='relativeWrap'>" +
          "<P> </P></SPAN>")
    }
    </SCRIPT>
    </TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

Modifying table rows

In IE4+ and NN6+, all table-related elements are full-fledged objects within the browser's object model. This means that you are free to use your choice of DOM element modification techniques on the row and column makeup of a table. But due to the frequent complexity of tables and all of their nested elements, the code required to manage a table can balloon in size. To the rescue come some methods that enable you to add and remove rows and cells from a table. Despite minor differences in the implementations of these methods across DOMs, the syntax exhibits sufficient unanimity to allow one set of code to work on both browsers — especially for adding elements to a table.

Table 27-1 provides a quick summary of the key methods used to add or remove elements within a TABLE, a table section (THEAD, TBODY, or TFOOT), and a row (TR). For simple tables (in other words, those that do not define THEAD or TFOOT segments), you can work exclusively with the row modification methods of the TABLE element object (and then the cell modification methods of the rows within the TABLE element). The reason for the duplication of the row methods in the table section objects is that instead of having to worry about row index numbers lining up among the combined total of head, body, and foot rows, you can treat each segment as a distinct unit. For example, if you want to add a row just to the beginning of the TFOOT section, you can use the `insertRow()` method for the TFOOT element object and not have to count up the TR elements and perform arithmetic to arrive at the desired row number. Instead, simply use the `insertRow()` method on the TFOOT element object and supply the method with parameters that ensure the row is inserted as the first row of the element.

IE5 for the Macintosh offers unpredictable results when inserting rows of a table via these methods. The browser does behave when modifying the HTML elements by accumulating the HTML for a row as a string and then adding the row to the table via IE DOM methods such as `insertAdjacentHTML()`. If your pages must modify the composition of tables after the page loads—and your audience includes IE5/Mac users—then use the element and node insertion techniques rather than the methods shown in Table 27-1 and techniques described next.

Table 27-1 IE4+ and NN6 Table Modification Methods

TABLE	*THEAD, TBODY, TFOOT*	*TR*
`insertRow()`	`insertRow()`	`insertCell()`
`deleteRow()`	`deleteRow()`	`deleteCell()`
`createTHead()`		
`deleteTHead()`		
`createTFoot()`		
`deleteTFoot()`		
`createCaption()`		
`deleteCaption()`		

The basic sequence for inserting a row into a table entails the following steps:

1. Invoke `insertRow()` and capture the returned reference to the new, unpopulated row.
2. Use the reference to the row to invoke `insertCell()` for each cell in the row, capturing the returned reference to each new, unpopulated cell.
3. Assign values to properties of the cell, including its content.

The following code fragment appends a new row to a table (`myTABLE`) and supplies information for the two cells in that row:

```
// parameter of -1 appends to table
// (you can use document.all.myTABLE.insertRow(-1) for IE4+ only)
var newRow = document.getElementById("myTABLE").insertRow(-1)
// parameter of 0 inserts at first cell position
var newCell = newRow.insertCell(0)
newCell.innerHTML = "Mighty Widget 2000"
// parameter of 1 inserts at second cell position
newCell = newRow.insertCell(1)
newCell.innerHTML = "Release Date TBA"
```

A key point to note about this sequence is that the `insertRow()` and `insertCell()` methods do their jobs before any content is handed over to the table. In other words, you first create the HTML space for the content and then add the content.

Listing 27-2 presents a living environment that adds and removes THEAD, TR, and TFOOT elements to an empty table in the HTML. The only subelement inside the TABLE element is a TBODY element, which directs the insertion of table rows so as not to disturb any existing THEAD or TFOOT elements. You can also see how to add or remove a caption from a table via caption-specific methods.

The first release version of NN6 does not behave well when scripts excessively modify tables. After some scripted changes, the browser reflows the page while ignoring TABLE element attributes, such as `CELLSPACING`.

Each table row consists of the hours, minutes, seconds, and milliseconds of a time stamp generated when you add the row. The color of any freshly added row in the TBODY is a darker color than the normal TBODY rows. This is so you can see what happens when you specify an index value to the `insertRow()` method. Some of the code here concerns itself with enabling and disabling form controls and updating SELECT elements, so don't be deterred by the length of Listing 27-2.

Listing 27-2: **Inserting/Removing Row Elements**

```
<HTML>
<HEAD>
<TITLE>Modifying Table Cell Content</TITLE>
<STYLE TYPE="text/css">
THEAD {background-color:lightyellow; font-weight:bold}
TFOOT {background-color:lightgreen; font-weight:bold}
#myTABLE {background-color:bisque}
</STYLE>

<SCRIPT LANGUAGE="JavaScript">
var theTable, theTableBody
function init() {
    theTable = (document.all) ? document.all.myTABLE :
        document.getElementById("myTABLE")
    theTableBody = theTable.tBodies[0]
}
function appendRow(form) {
    insertTableRow(form, -1)
}

function addRow(form) {
    insertTableRow(form, form.insertIndex.value)
}

function insertTableRow(form, where) {
    var now = new Date()
    var nowData = [now.getHours(), now.getMinutes(), now.getSeconds(),
```

```
        now.getMilliseconds()]
    clearBGColors()
    var newCell
    var newRow = theTableBody.insertRow(where)
    for (var i = 0; i < nowData.length; i++) {
        newCell = newRow.insertCell(i)
        newCell.innerHTML = nowData[i]
        newCell.style.backgroundColor = "salmon"
    }
    updateRowCounters(form)
}

function removeRow(form) {
    theTableBody.deleteRow(form.deleteIndex.value)
    updateRowCounters(form)
}

function insertTHEAD(form) {
    var THEADData = ["Hours","Minutes","Seconds","Milliseconds"]
    var newCell
    var newTHEAD = theTable.createTHead()
    newTHEAD.id = "myTHEAD"
    var newRow = newTHEAD.insertRow(-1)
    for (var i = 0; i < THEADData.length; i++) {
        newCell = newRow.insertCell(i)
        newCell.innerHTML = THEADData[i]
    }
    updateRowCounters(form)
    form.addTHEAD.disabled = true
    form.deleteTHEAD.disabled = false
}

function removeTHEAD(form) {
    theTable.deleteTHead()
    updateRowCounters(form)
    form.addTHEAD.disabled = false
    form.deleteTHEAD.disabled = true
}

function insertTFOOT(form) {
    var TFOOTData = ["Hours","Minutes","Seconds","Milliseconds"]
    var newCell
    var newTFOOT = theTable.createTFoot()
    newTFOOT.id = "myTFOOT"
    var newRow = newTFOOT.insertRow(-1)
    for (var i = 0; i < TFOOTData.length; i++) {
        newCell = newRow.insertCell(i)
        newCell.innerHTML = TFOOTData[i]
    }
    updateRowCounters(form)
    form.addTFOOT.disabled = true
    form.deleteTFOOT.disabled = false
}
```

Continued

Listing 27-2 *(continued)*

```
function removeTFOOT(form) {
    theTable.deleteTFoot()
    updateRowCounters(form)
    form.addTFOOT.disabled = false
    form.deleteTFOOT.disabled = true
}

function insertCaption(form) {
    var captionData = form.captionText.value
    var newCaption = theTable.createCaption()
    newCaption.innerHTML = captionData
    form.addCaption.disabled = true
    form.deleteCaption.disabled = false
}

function removeCaption(form) {
    theTable.deleteCaption()
    form.addCaption.disabled = false
    form.deleteCaption.disabled = true
}

// housekeeping functions
function updateRowCounters(form) {
    var sel1 = form.insertIndex
    var sel2 = form.deleteIndex
    sel1.options.length = 0
    sel2.options.length = 0
    for (var i = 0; i < theTableBody.rows.length; i++) {
        sel1.options[i] = new Option(i, i)
        sel2.options[i] = new Option(i, i)
    }
    form.removeRowBtn.disabled = (i==0)
}

function clearBGColors() {
    for (var i = 0; i < theTableBody.rows.length; i++) {
        for (var j = 0; j < theTableBody.rows[i].cells.length; j++) {
            theTableBody.rows[i].cells[j].style.backgroundColor = ""
        }
    }
}

</SCRIPT>
</HEAD>

<BODY onLoad="init()">
<H1>Modifying Tables</H1>
<HR>
<FORM NAME="controls">
<FIELDSET>
<LEGEND>Add/Remove Rows</LEGEND>
```

```
<TABLE WIDTH="100%" CELLSPACING=20><TR>
<TD><INPUT TYPE="button" VALUE="Append 1 Row"
    onClick="appendRow(this.form)"></TD>
<TD><INPUT TYPE="button" VALUE="Insert 1 Row" onClick="addRow(this.form)"> at
index:
    <SELECT NAME="insertIndex">
        <OPTION VALUE="0">0
    </SELECT></TD>
<TD><INPUT TYPE="button" NAME="removeRowBtn" VALUE="Delete 1 Row" DISABLED
    onClick="removeRow(this.form)"> at index:
    <SELECT NAME="deleteIndex">
        <OPTION VALUE="0">0
    </SELECT></TD>
</TR>
</TABLE>
</FIELDSET>
<FIELDSET>
<LEGEND>Add/Remove THEAD and TFOOT</LEGEND>
<TABLE WIDTH="100%" CELLSPACING=20><TR>
<TD><INPUT TYPE="button" NAME="addTHEAD" VALUE="Insert THEAD"
    onClick="insertTHEAD(this.form)"><BR>
    <INPUT TYPE="button" NAME="deleteTHEAD" VALUE="Remove THEAD" DISABLED
        onClick="removeTHEAD(this.form)">
</TD>
<TD><INPUT TYPE="button" NAME="addTFOOT" VALUE="Insert TFOOT"
    onClick="insertTFOOT(this.form)"><BR>
    <INPUT TYPE="button" NAME="deleteTFOOT" VALUE="Remove TFOOT" DISABLED
        onClick="removeTFOOT(this.form)">
</TD>
</TR>
</TABLE>
</FIELDSET>
<FIELDSET>
<LEGEND>Add/Remove Caption</LEGEND>
<TABLE WIDTH="100%" CELLSPACING=20><TR>
<TD><INPUT TYPE="button" NAME="addCaption" VALUE="Add Caption"
    onClick="insertCaption(this.form)"></TD>
<TD>Text: <INPUT TYPE="text" NAME="captionText" SIZE=40 VALUE="Sample Caption">
<TD><INPUT TYPE="button" NAME="deleteCaption" VALUE="Delete Caption" DISABLED
    onClick="removeCaption(this.form)"></TD>
</TR>
</TABLE>
</FIELDSET>
</FORM>
<HR>
<TABLE ID="myTABLE" CELLPADDING=10 BORDER=1>
<TBODY>
</TABLE>
</BODY>
</HTML>
```

Modifying table columns

Unlike the table row-oriented elements, such as TBODY, the COL and COLGROUP elements are not containers of cells. Instead, these elements serve as directives for the rendering of columns within a table. But through scripting, you can add or remove one or more columns from a table on the fly. There is no magic to it; you simply insert or delete the same-indexed cell from every row of the table.

Listing 27-3 demonstrates adding and removing a left-hand column of a table. The table presents the four longest rivers in Africa, and the new column provides the numeric ranking. Thanks to the regularity of this table, the values for the rankings can be calculated dynamically. Note, too, that the `className` property of each new table cell is set to a class that has a style sheet rule defined for it. Instead of inheriting the style of the table, the cells obey the more specific background color and font weight rules defined for the cells. (The early release of NN6 does not render the enabling and disabling of the buttons in this example correctly, but the buttons operate as intended.)

Listing 27-3: **Modifying Table Columns**

```
<HTML>
<HEAD>
<TITLE>Modifying Table Columns</TITLE>
<STYLE TYPE="text/css">
THEAD {background-color:lightyellow; font-weight:bold}
.rankCells {background-color:lightgreen; font-weight:bold}
#myTABLE {background-color:bisque}
</STYLE>

<SCRIPT LANGUAGE="JavaScript">
var theTable, theTableBody
function init() {
    theTable = (document.all) ? document.all.myTABLE :
        document.getElementById("myTABLE")
    theTableBody = theTable.tBodies[0]
}

function insertColumn(form) {
    var oneRow, newCell, rank
    if (theTable.tHead) {
        oneRow = theTable.tHead.rows[0]
        newCell = oneRow.insertCell(0)
        newCell.innerHTML = "Ranking"
    }
    rank = 1
    for (var i = 0; i < theTableBody.rows.length; i++) {
        oneRow = theTableBody.rows[i]
        newCell = oneRow.insertCell(0)
        newCell.className = "rankCells"
        newCell.innerHTML = rank++
```

```
    }
    form.addColumn.disabled = true
    form.removeColumn.disabled = false
}

function deleteColumn(form) {
    var oneRow
    if (theTable.tHead) {
        oneRow = theTable.tHead.rows[0]
        oneRow.deleteCell(0)
    }
    for (var i = 0; i < theTableBody.rows.length; i++) {
        oneRow = theTableBody.rows[i]
        oneRow.deleteCell(0)
    }
    form.addColumn.disabled = false
    form.removeColumn.disabled = true
}
</SCRIPT>
</HEAD>

<BODY onLoad="init()">
<H1>Modifying Table Columns</H1>
<HR>
<FORM NAME="controls">
<FIELDSET>
<LEGEND>Add/Remove Left Column</LEGEND>
<TABLE WIDTH="100%" CELLSPACING=20><TR>
<TD><INPUT TYPE="button" NAME="addColumn" VALUE="Insert Left Column"
    onClick="insertColumn(this.form)"></TD>
<TD><INPUT TYPE="button" NAME="removeColumn" VALUE="Remove Left Column"
    DISABLED onClick="deleteColumn(this.form)"></TD>
</TR>
</TABLE>
</FIELDSET>
</TABLE>
</FIELDSET>
</FORM>
<HR>
<TABLE ID="myTABLE" CELLPADDING=5 BORDER=1>
<THEAD ID="myTHEAD">
<TR>
    <TD>River<TD>Outflow<TD>Miles<TD>Kilometers
</TR>
</THEAD>
<TBODY>
<TR>
    <TD>Nile<TD>Mediterranean<TD>4160<TD>6700
</TR>
<TR>
    <TD>Congo<TD>Atlantic Ocean<TD>2900<TD>4670
</TR>
<TR>
```

Continued

Listing 27-3 *(continued)*

```
    <TD>Niger<TD>Atlantic Ocean<TD>2600<TD>4180
</TR>
<TR>
    <TD>Zambezi<TD>Indian Ocean<TD>1700<TD>2740
</TR>
</TABLE>
</BODY>
</HTML>
```

W3C DOM table object classes

If you ever read the W3C DOM Level 2 specification, notice that the objects defined for tables do not align themselves fully with the actual elements defined in the HTML 4.0 specification. That's not to say the DOM scoffs at the HTML spec; rather, the needs of a DOM with respect to tables differ a bit. For example, as far as the W3C DOM is concerned, the THEAD, TBODY, and TFOOT are all regarded as table sections and are thus known as `HTMLTableSectionElement` objects. In other words, in the W3C DOM, there is no particular distinction among the types of table section elements. They're all lumped together, and they bear the same properties and methods. With its strong adherence to the W3C DOM, the NN6 DOM sticks to the W3C DOM object constructions.

When you work in both the IE and W3C DOMs at the same time, it's helpful to know the relationships between the object naming conventions used in each. Table 27-2 provides a quick cross-reference between the object types in both DOMs. None of terminology in Table 27-2 affects the way scripts construct references to elements or the way elements are nested within one another. The containment hierarchy is driven by the HTML element containment — and that remains the same regardless of DOM exposure.

Table 27-2 **Table Object Classifications**

W3C DOM (NN6)	*IE4+ and HTML*
`HTMLTableElement`	TABLE
`HTMLTableCaptionElement`	CAPTION
`HTMLTableColElement`	COL, COLGROUP
`HTMLTableSectionElement`	TBODY, TFOOT, THEAD
`HTMLTableRowElement`	TR
`HTMLTableCellElement`	TD, TH

While the following object-specific discussions list the objects according to their HTML tag name, I group these objects according to the W3C DOM classifications because element objects that share a classification also share the same properties, methods, and event handlers.

TABLE Element Object

For HTML element properties, methods, and event handlers, see Chapter 15.

Properties	*Methods*	*Event Handlers*
align	createCaption()	onScroll
background	createTFoot()	
bgColor	createTHead()	
border	deleteCaption()	
borderColor	deleteRow()	
borderColorDark	deleteTFoot()	
borderColorLight	deleteTHead()	
caption	firstPage()	
cellPadding	insertRow()	
cellSpacing	lastPage()	
cells	moveRow()	
cols	nextPage()	
datePageSize	previousPage()	
frame	refresh()	
height		
rows		
rules		
summary		
tBodies		
tFoot		
tHead		
width		

Syntax

Accessing TABLE element object properties and methods:

```
(IE4+)        [window.]document.all.elemID.property | method([parameters])
(IE5+/NN6)    [window.]document.getElementById("elemID").property |
              method([parameters])
```

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

About this object

The TABLE element object is the outermost container of table-related information. The HTML element has a large number of attributes, most of which are echoed by their counterpart properties in the object model. You rarely will modify these properties if the values are set in the tag's attributes. However, if you construct a new TABLE element object for insertion into the page, use these properties to assign values to the equivalents of the element's attributes.

A number of additional properties return collections of cell, row, and row section objects; still more properties return references to other, singular objects within the table (such as the CAPTION element object). For example, if your script needs to iterate through all rows within just the TBODY elements (in other words, without affecting the rows in the THEAD element), your script can perform a nested `for` loop to access each row:

```
var oneTBody, oneRow
for (var i = 0; i < tableRef.tBodies.length; i++) {
    oneTBody = tableRef.tBodies[i]
    for (var j = 0; j < oneTBody.rows.length; j++) {
        oneRow = oneTBody.rows[j]
        // more stuff working on each row
    }
}
```

For a simple table that does not define table row sections, you can iterate through the `rows` collection property of a TABLE element object. You can even access cells directly; but it may be easier to keep track of cells in a loop by going through them row by row (via the `cells` property of each TR element object).

A large number of methods enable you to modify the structure of a table (as described earlier in this chapter), but they primarily work with rows. Column modifications require a different approach, as also demonstrated earlier.

Properties

align

Value: String (`center`, `left`, `right`) Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

The `align` property controls the horizontal alignment of the table with respect to the next outermost container that provides positioning context. Most typically, the next outermost positioning container is the BODY element. Modifications to this property on an existing table cause the surrounding content to reflow on the page. Be sure you test the consequences of any modification with a variety of browser window sizes.

Example on the CD-ROM

Related Item: `style.align` property.

background

Value: URL String Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Only IE4+ makes a provision for assigning a background image to a table, and the `background` property controls that value. You can swap out an image by assigning a new URL to the `background` property. The image appears in front of any background color assigned to the table. Thus, you can assign attributes for both characteristics so that there is at least a background color (and an image for IE users).

Example on the CD-ROM

Related Item: `IMG.src` property.

bgColor

Value: Color Value String Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

The `bgColor` attribute controls the background color of a table (the `BGCOLOR` attribute). Colors assigned to the entire table are overridden if colors are assigned to row, row groups, or cells within the table. If you set the `bgColor` property, the `backgroundColor` style property is not affected. Assign values in any acceptable color string format, such as hexadecimal triplets (for example, "`#FCFC00`") or the generally recognized plain-language names (for example, "`cornflowerblue`").

Example on the CD-ROM

Related Item: `style.backgroundColor` property.

border

Value: Integer Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

The `border` property controls the thickness of the table's borders. Values indicate the number of pixels thick the border should be. A value of zero removes all visible borders surrounding the table. Different browsers render table cell borders differently depending on background colors and other visual attributes of tables and table elements. Be sure to verify the appearance on as many browsers and operating systems as possible.

Example on the CD-ROM

Related Item: `borderColor` property.

borderColor
borderColorDark
borderColorLight

Value: Color Value String Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

IE4+ provides attributes and corresponding properties to control the border colors of a table. When table borders have enough thickness to display a three-dimensional raised look, the appearance is created by generating two dark and two light edges (simulating a light source coming from the upper-left or lower-right corner). If you want to do a better job of specifying the color combinations for the light and dark edges, you can control them individually via the `borderColorLight` and `borderColorDark` properties, respectively. You can assign colors in any valid color value (hexadecimal triplet or plain-language name); but when you read the property, the value is returned as a hexadecimal triplet (for example, `"#008000"`).

Example on the CD-ROM

Related Item: `TD.borderColor` property.

caption

Value: CAPTION element object reference Read/Write (see text)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

The `caption` property returns a reference to the CAPTION element object that is nested inside the current table. If there is no CAPTION element, the value is `null`. You can use this property as a shortcut reference to the CAPTION element if you

need to read or modify that element's properties. The property is read/write in NN6, provided you create a valid CAPTION element object and assign that new object to the `caption` property.

Example on the CD-ROM

Related Item: CAPTION element object.

cellPadding
cellSpacing

Value: Integer Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

The `cellPadding` property is a table-wide specification for the blank space inserted between the edge of a table cell and the content of the cell. One value affects the padding on all four sides. The effect of cell padding is especially apparent when there are borders between cells; in this case, the padding provides welcome breathing space between the border and content. The `cellSpacing` property influences the thickness of borders between cells. If no visible borders are present between cells in a table, you can usually set either `CELLPADDING` or `CELLSPACING` to provide the desired blank space between cells.

Example on the CD-ROM

Related Item: `border` property.

cells

Value: Array Read-Only

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

The cells property (not implemented in IE5/Mac) returns an array (collection) of all TD and TH element objects within the entire table. From the perspective of the TABLE element object, this "view" encompasses all cells — whether they are inside a table row segment (for example, a THEAD) or in a freestanding row. In the W3C DOM (and NN6), the cells collection is accessible only as a property of a TR object. However, a rows collection is available from all table container elements, thus enabling you to iterate through all cells of all rows.

Example on the CD-ROM

Related Items: rows, TR.cells properties.

cols

Value: Integer Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

The cols property represents the IE-specific COLS attribute for TABLE elements. Specifying this attribute should speed table rendering. If you don't specify the attribute explicitly in your HTML, the property has a value of zero — the property does not tell you dynamically the size of your table. Although this property is read/write, you cannot use this property to add or remove columns from a table. Instead, use the table modification methods discussed later in this section.

Related Item: rows property.

dataPageSize

Value: Integer Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

When using IE4+ data binding to obtain table data from a data source, there may be more rows or data (records) than you wish to display in one table. If so, you can

define the number of rows (records) that constitutes a "page" of data within the table. With this limit installed for the table, you can then use the `firstPage()`, `previousPage()`, `nextPage()`, and `lastPage()` methods to access another page relative to the currently viewed page. While you usually establish this value via the `DATAPAGESIZE` attribute of the TABLE element, you can adjust it later via the `dataPageSize` property to show more or fewer records per "page" in the table.

Example on the CD-ROM

Related Items: `dataSrc`, `dataFld` properties; `firstPage()`, `lastPage()`, `nextPage()`, `previousPage()` methods.

frame

Value: String Constant Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

The `frame` property enables you to control which side or sides of the table's border should be displayed. Values for this property can be any of a fixed set of string constants. Table 27-3 lists the acceptable values. Hiding or showing table border edges under script control can have an effect on the layout and placement of both the table and surrounding elements.

Table 27-3 **Table frame Property Values**

Value	*Description*
`above`	Top edge only
`below`	Bottom edge only
`border`	All four sides (same as box)
`box`	All four sides (same as border)
`hsides`	Horizontal (top and bottom) edges only

Value	*Description*
lhs	Left-hand side edge only
rhs	Right-hand side edge only
void	No borders
vsides	Vertical (left and right) edges only

Example (with Listing 27-4) on the CD-ROM

Related Items: border, borderColor, rules properties.

height
width

Value: Integer or Length String Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

The height (IE4+) and width (IE4+/NN6+) properties represent the HEIGHT and WIDTH attributes assigned to the TABLE element. If no values are assigned to the element in the tag, the properties do not reveal the rendered size of the table (use the offsetHeight and offsetWidth properties for that information). Values for these properties can be integers representing pixel dimensions or strings containing percentage values, just like the attribute values. Scripts can shrink the dimensions of a table no smaller than the minimum space required to render the cell content. Notice that only the width property is W3C DOM-sanctioned (as well as the corresponding property in the HTML 4.0 specification).

Example on the CD-ROM

Related Items: offsetHeight, offsetWidth properties.

rows

Value: Array of Row Objects — Read-Only

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

The `rows` property returns an array (collection) of TR element objects in the current table. This array includes rows in the THEAD, TBODY, and TFOOT row sections if the table is segmented. You can use the `rows` property to create a cross-browser script that accesses each cell of a table. Such a nested `for` loop looks like the following:

```
var oneCell
for (var i = 0; i < tableRef.rows.length; i++) {
    for (var j = 0; j < tableRef.rows[i].cells.length; j++) {
        oneCell = tableRef.rows[i].cells[j]
        // more statements working with the cell
    }
}
```

If you want to limit the scope of the `rows` property to rows within a row segment (for example, just in the TBODY), you can access this property for any of the three types of row segment objects.

Example on the CD-ROM

Related Items: `TBODY.rows`, `TFOOT.rows`, `THEAD.rows` properties.

rules

Value: String Constant — Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

In contrast to the `frame` property, the `rules` property governs the display of borders between cells. Values for this property can be any of a fixed set of string constants. Table 27-4 lists the acceptable values. Hiding or showing table cell border

edges under script control can have an effect on the layout and placement of both the table and surrounding elements. Early versions of NN6 may not render scripted changes to the `rules` property, but reading or writing the property does not cause errors.

Table 27-4 **Table rules Property Values**

Value	*Description*
`all`	Borders around every cell
`cols`	Vertical borders between columns
`groups`	Vertical borders between column groups; horizontal borders between row groups
`none`	No borders between cells
`rows`	Horizontal borders between row groups

Example (with Listing 27-5) on the CD-ROM

Related Items: `border`, `borderColor`, `frame` properties.

summary

Value: String Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

The `summary` property represents the HTML 4.0 `SUMMARY` attribute. The text assigned to this attribute is intended for use by browsers that present a page's content through nonvisual means. For example, a browser equipped to use speech synthesis to read the page aloud can use the text of the summary to describe the table for the user.

Related Item: `caption` property.

tBodies

Value: Array of TBODY element objects — Read-Only

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

The `tBodies` property returns an array of all TBODY elements in the table. Even if you don't specify a TBODY element, every table contains an implied TBODY element. Thus, to access a batch of rows of a simple table other than the THEAD and TFOOT sections, you can use the `tBodies[0]` array notation. From there, you can get the rows of the table body section via the `rows` property. This property is not available in IE4/Mac.

Example on the CD-ROM

Related Items: `tFoot`, `tHead` properties.

tFoot
tHead

Value: Row segment element object — Read/Write (see text)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Each table can have (at most) one TFOOT and one THEAD element. If you specify one of these for the table, the `tFoot` and `tHead` properties, respectively, return references to those element objects. These properties are read-only in IE, but NN6 enables you to assign valid TFOOT and THEAD element objects to these properties in order to insert or replace the elements in the current table. The process for doing this is similar to the sequence described in the `caption` property. For either of these two elements, however, you have to construct the desired number of table cell objects (and row objects if you want multiple rows) for the newly created row

segment object. See the discussions of these two objects for details on accessing rows and cells of the segments.

Related Items: TBODY, TFOOT, THEAD objects.

width

See height.

Methods

createCaption()
deleteCaption()

Returns: Reference to new CAPTION element object; Nothing.

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

The `createCaption()` and `deleteCaption()` convenience methods enable you to add or remove a CAPTION element object from the current table. When you create a new caption, the action simply inserts the equivalent of a blank CAPTION element tag into the TABLE element (this may not, however, be reflected in the source view of the page). You must populate the CAPTION element with text or HTML before it appears on the page. Because the method returns a reference to the newly created object, you can use that reference to assign content to its `innerHTML` property or you can append a child text node.

Because a table can have only one CAPTION element nested within, the `deleteCaption()` method belongs to the TABLE element object. The method returns no value.

Example

See Listing 27-2 for an example of creating, inserting, and removing a CAPTION element object from a table.

Related Item: CAPTION element object.

```
createTFoot()
createTHead()
deleteTFoot()
deleteTHead()
```

Returns: Element references (create methods); Nothing.

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			(✓)	✓	✓

These four methods enable you to add or remove TFOOT and THEAD table row section objects. When you create a THEAD or TFOOT element, the methods return references to the newly inserted elements. But, as with `createCaption()`, these methods do nothing to display content. Instead, use the returned references to populate the row(s) of the header and footer with cells. Regardless of the number of rows associated with a THEAD or TFOOT element, the `deleteTFoot()` and `deleteTHead()` methods remove all associated rows and return no values.

While these methods are available in IE4, you may not have complete write access to the properties of the objects returned by the creation methods. For example, you may not be able to assign a value to the `id` property of the TFOOT or THEAD element returned by their respective creation methods.

Example

See Listing 27-2 for an example of creating, inserting, and removing TFOOT and THEAD elements object from a table.

Related Items: TFOOT, THEAD element objects.

```
deleteRow(rowIndex)
insertRow(rowIndex)
```

Returns: Nothing; Reference to newly created row.

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

The `insertRow()` and `deleteRow()` convenience methods assist in adding TR elements to, and removing them from, a TABLE element. Inserting a row does little more than the equivalent of inserting a pair of empty TR element tags into the HTML (although you may not see them in the source view of the page). It is up to the rest of your scripts to assign properties to the row and populate it with new cells (see the `insertCell()` method of the TR element object).

Attributes for both methods are zero-based index numbers. In the case of `insertRow()`, the number indicates the row *before* which the new row is to be inserted. To append the row to the end of the table, use `-1` as a shortcut parameter. To delete a row, use the index value for that row. Be aware that if you intend to employ `deleteRow()` to remove all rows from a table (presumably to repopulate the table with a new set), the most efficient way is to use a `while` loop that continues to remove the first row until there are no more:

```
while (tableRef.rows.length > 0) {
    tableRef.deleteRow(0)
}
```

Example

See Listing 27-2 for examples of inserting and deleting table rows.

Related Item: `TD.insertCell()` method.

`firstPage()`
`lastPage()`

Returns: Nothing.

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

For tables that are bound to external data sources via IE4+ data binding, the `firstPage()` and `lastPage()` methods zoom to the first and last pages of the data, respectively. You must specify the table's data page size for the Data Source Object to know how many records to assign to a "page" of data. Note that while related methods — `nextPage()` and `previousPage()` — are available in IE4, these two methods were available in IE5 first.

Related Items: `dataPageSize`, `dataSrc`, `dataFld` properties; `nextPage()`, `previousPage()` methods.

moveRow(*sourceRowIndex, destinationRowIndex*)

Returns: Row element object.

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

The IE5+ `moveRow()` convenience method enables you to move a row from one position to another within the same table. Both parameters are integer index values. The first parameter is the index of the row you want to move; the second is the index of the row to where you want to move the row. Because no movement takes place when the method is invoked, the removal of the source row does not impact the index count of the destination row. But after the method executes, the row that was in the destination row is now pushed down one row. This method returns a reference to the moved row.

You can accomplish this same functionality in W3C DOM compatible syntax (for both IE5+ and NN6+) via the `replaceChild()` method of the TABLE element.

Example on the CD-ROM

Related Item: `replaceChild()` method.

nextPage()
previousPage()

Returns: Nothing.

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

For tables that are bound to external data sources via IE4+ data binding, the `nextPage()` and `previousPage()` methods jump ahead and back one page of the data, respectively. You must specify the table's data page size for the Data Source Object to know how many records to assign to a "page" of data. Typically, navigational buttons associated with the table invoke these methods.

Related Items: `dataPageSize`, `dataSrc`, `dataFld` properties; `firstPage()`, `lastPage()` methods.

`refresh()`

Returns: Nothing.

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

For tables that are bound to external data sources via IE4+ data binding, the `refresh()` method retrieves the current data source data for display in the table. A script can use `setTimeout()` to invoke a function that calls this method at an interval of your desire. If you frequently update the database associated with the table, this method can help keep the table up to date without requiring the client to download the entire page (and perhaps run into cache conflicts).

Related Items: `dataPageSize`, `dataSrc`, `dataFld` properties.

TBODY, TFOOT, and THEAD Element Objects

For HTML element properties, methods, and event handlers, see Chapter 15.

Properties	*Methods*	*Event Handlers*
`align`†	`deleteRow()`†	
`bgColor`†	`insertRow()`†	
`ch`	`moveRow()`†	
`chOff`		
`rows`†		
`vAlign`		

†See TABLE element object.

Syntax

Accessing TBODY, TFOOT, and THEAD element object properties and methods:

```
(IE4+)     [window.]document.all.elemID.property | method([parameters])
(IE5+/NN6) [window.]document.getElementById("elemID").property |
           method([parameters])
```

Accessing TBODY element object properties and methods:

```
(IE4+)     [window.]document.all.tableID.tBodies[i].property |
           method([parameters])
(IE5+/NN6) [window.]document.getElementById("tableID").tBodies[i].property |
           method([parameters])
```

Accessing TFOOT element object properties and methods:

```
(IE4+)     [window.]document.all.tableID.tFoot.property | method([parameters])
(IE5+/NN6) [window.]document.getElementById("tableID").tFoot.property |
           method([parameters])
```

Accessing THEAD element object properties and methods:

```
(IE4+)     [window.]document.all.tableID.tHead.property | method([parameters])
(IE5+/NN6) [window.]document.getElementById("tableID").tHead.property |
           method([parameters])
```

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			(✓)	✓	✓

About these objects

Each of these element objects represents a row grouping within a TABLE element (an `HTMLTableSectionElement` in the syntax of the W3C DOM specification). A table can have only one THEAD and one TFOOT, but it can have as many TBODY elements as your table organization requires.

These elements share many properties and methods with the TABLE element in that they all contain rows. The benefit of defining table segments is apparent if you use table rules (see the `TABLE.rules` property earlier in this chapter) and if you wish to limit the scope of row activities only to rows within one segment. For instance, if your table has a THEAD that is to remain static, your scripts can merrily loop through the rows of only the TBODY section without coming anywhere near the row(s) in the THEAD.

None of these elements are available in IE4 for the Macintosh.

Properties

ch
chOff

Value: One-Character String Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

The `ch` and `chOff` properties are defined for NN6, but they may be serving as placeholders for future implementation. These properties represent the optional `CHAR` and `CHAROFF` attributes of table row section elements in the HTML 4.0 specification. If these are implemented in a future browser, they will help align cell content within a column or column group similar to the way word processers allow for formatting features such as decimal tabs. For details on these attributes, see `http://www.w3.org/TR/REC-html40/struct/tables.html#adef-char`.

Related Items: COL, COLGROUP objects.

vAlign

Value: String Constant Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Providing the cell-oriented `vAlign` property for a table row section enables you to specify a vertical alignment to apply to all cells within that section rather than specify the `VALIGN` attribute for each TD element. By default, browsers render cell content with a middle vertical alignment within the cell. If you want to modify the setting for an existing table section (or assign it to a new one you create), the values must be one of the following string constants: `baseline`, `bottom`, `middle`, or `top`.

Example on the CD-ROM

Related Item: `TD.vAlign` property.

CAPTION Element Object

For HTML element properties, methods, and event handlers, see Chapter 15.

Properties	*Methods*	*Event Handlers*
`align`†		
`vAlign`††		

†See TABLE element object.

††See TBODY element object.

Syntax

Accessing CAPTION element object properties and methods:

```
(IE4+)     [window.]document.all.elemID.property | method([parameters])
(IE5+/NN6) [window.]document.getElementById("elemID").property |
           method([parameters])
```

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

About this object

A CAPTION element is a simple HTML container whose only prerequisite is that it must be nested inside a TABLE element. That nesting allows the TABLE element object to control insertion and removal of a CAPTION element at will. You can modify the content of a CAPTION element just like you do any HTML element (in DOMs that allow such modification). You can see an example of how the TABLE element object uses some of its methods to create and remove a CAPTION element in Listing 27-2.

The only properties that lift the CAPTION element object above a mere contextual element (described in Chapter 15) are `vAlign` (IE4+) and the W3C DOM-sanctioned `align` (IE4+ and NN6+). I describe these properties and their values for other objects in this chapter.

COL and COLGROUP Element Objects

For HTML element properties, methods, and event handlers, see Chapter 15.

Properties	*Methods*	*Event Handlers*
align†		
ch††		
chOff††		
span		
vAlign††		
width		

†See TABLE element object.

††See TBODY element object.

Syntax

Accessing COL and COLGROUP element object properties and methods:

```
(IE4+)     [window.]document.all.elemID.property | method([parameters])
(IE5+/NN6) [window.]document.getElementById("elemID").property |
           method([parameters])
```

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

About these objects

The purpose of the COL and COLGROUP elements is to allow cells within one or more columns to be treated as a single entity for purposes of style sheet and other style-related control. In other words, if you want one column of a table to be all boldface, you can assign that style sheet rule to the COL element that encompasses that column. All cells within that column inherit the style sheet rule definition.

Having two different element names allows for the nesting of column groups, which can come in handy for complex tables. For instance, consider a table that reports the forecasted and actual sales for a list of products across four quarters of a year. The left column of the table stands alone with the product item numbers. To the right is one large grouping of eight columns that encompasses the four pairs of forecasted/actual sales pairs. All eight columns of cells are to be formatted with a particular font style to help differentiate the pairs of columns for each quarter. You also want to assign a different background color. Therefore, you designate each pair of columns as its own subgroup within the eight-column master grouping. The COLGROUP and COL tags for this nine-column table are as follows:

```
<COL ID="productIDs">
<COLGROUP ID="fiscalYear" SPAN="8" WIDTH="40">
    <COL ID="Q1" SPAN="2">
    <COL ID="Q2" SPAN="2">
    <COL ID="Q3" SPAN="2">
    <COL ID="Q4" SPAN="2">
</COLGROUP>
```

Up in the HEAD section of this document are style sheet rules similar to the following:

```
<STYLE TYPE="text/css">
#productIDs {font-weight:bold}
#fiscalYear {font-family: Courier, "Courier New", monospace}
#Q1 {background-color: lightyellow}
#Q2 {background-color: pink}
#Q3 {background-color: lightblue}
#Q4 {background-color: lightgreen}
</STYLE>
```

The HTML code for the column groups demonstrates the two key attributes: `SPAN` and `WIDTH`. Both of these attributes are reflected as properties of the objects, and I describe them in the following section. Notice, however, that COL and COLGROUP elements act cumulatively and in source code order to define the column groups for the table. In other words, if the style of the left-hand column is not important, the table still requires the initial one-column COL element before the eight-column COLGROUP element. Otherwise, the browser makes the first eight columns the column group. Therefore, it is a good idea to account for every column with COL and/or COLGROUP elements if you intend to use any column grouping in your table.

From a scripter's point of view, you are more likely to modify styles for a column or column group than you are to alter properties such as `span` or `width`. But, if your scripts generate new tables, you may create new COL or COLGROUP elements whose properties you definitely should initialize with values.

Properties

span

Value: Integer Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

The `span` property represents the number of columns that the column group should encompass. Don't confuse this property with the `colSpan` property of TD and TH elements. A COL or COLGROUP `span` does not have any impact on the rendering or combination of multiple cells into one. It simply draws an imaginary lasso around as many columns as are specified, signifying that these columns can be treated as a group for style purposes (and also for drawing of divider rules, if you set the table's `rules` property to `groups`).

Example on the CD-ROM

Related Item: `width` property.

width

Value: Length String Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

The only reason the `width` property is highlighted for these objects is that the property (and corresponding attribute) impacts the width of table cells inside the scope of the column grouping. For example, if you assign a width of 50 pixels to a COLGROUP whose `SPAN` attribute is set to `3`, all cells in all three columns inherit the 50-pixel width specification. For more details on the values acceptable to this property, see the `TABLE.width` property description earlier in this chapter.

Related Item: `TABLE.width` property.

TR Element Object

For HTML element properties, methods, and event handlers, see Chapter 15.

Properties	*Methods*	*Event Handlers*
align†	deleteCell()	
bgColor†	insertCell()	
borderColor†		
borderColorDark†		
borderColorLight†		
cells		
ch††		
chOff††		
height		
rowIndex		
sectionRowIndex		
vAlign††		

†See TABLE element object.

††See TBODY element object.

Syntax

Accessing TR element object properties and methods:

```
(IE4+)     [window.]document.all.elemID.property | method([parameters])
(IE4+)     [window.]document.all.tableID.rows[i].property | method([parameters])
(IE4+)     [window.]document.all.tableRowSectionID.rows[i].property |
           method([parameters])
(IE5+/NN6) [window.]document.getElementById("elemID"). property |
           method([parameters])
(IE5+/NN6) [window.]document.getElementById("tableID").rows[i].property |
           method([parameters])
(IE5+/NN6) [window.]document.getElementById("tableRowSectionID")
           .rows[i].property | method([parameters])
```

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

About this object

Table rows are important objects within the complex nesting of table-related elements and objects. When a table represents server database data, one row usually equals one record. And, although you can employ scripting to add columns to a table, the more common table modifications are to add or delete rows — hence the presence of the TABLE element object's `insertRow()` and `deleteRow()` methods.

The primary job of the TR element is to act as a container for TD elements. All the cells in a row inherit some attributes and properties that you apply to that row. An array of cell objects is available for iteration via `for` loops. A TR element object, therefore, also has methods that insert and remove individual cells in that row.

The number of columns in a row is determined by the number of TD elements or, more specifically, by the number of columns that the cells intend to span. One row can have four TD elements, while the next row can have only two TD elements — each of which is defined to occupy two columns. The row of the table with the most TD elements and column reservations determines the column width for the entire table.

Of the properties just listed, the ones related to border color are available in IE4+ only. In IE4+, the border is drawn around each cell of the row rather than the entire row. The HTML 4.0 specification (and the W3C DOM Level 2 specification by extension) does not recognize border colors for rows alone, nor are style sheet border rules inherited by the cell children of a row. However, you can define borders for individual cells or classes of cells.

Properties

cells

Value: Array of TD element objects Read-Only

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

The `cells` property returns an array (collection) of TD element objects nested inside the current TR object. The `length` property of this array indicates the number of actual TD elements in the row, which may not be the number of columns if one or more cells occupy multiple columns.

Use the `cells` property in `for` loops to iterate through all cells within a row. Assuming your script has a reference to a single row, the loop should look like the following:

```
for (var i = 0; i < rowRef.cells.length; i++) {
    oneCell = rowRef.cells[i]
    // more statements working with the cell
}
```

Example on the CD-ROM

Related Items: `TABLE.rows`, `TD.cellIndex` properties.

height

Value: Integer or Length String — Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

IE4+ enables page authors to predefine a height for a table row; this attribute is echoed by the `height` property. The value can be a number of pixels or a percentage length value. Note that this property does not reveal the rendered height of the row unless you explicitly set the attribute in the HTML. To get the actual height (in IE4+ and NN6+), use the `offsetHeight` property. You cannot adjust the `height` property to be smaller than the table normally renders the row.

Example on the CD-ROM

Related Item: `offsetHeight` property (Chapter 15).

rowIndex
sectionRowIndex

Value: Integer Read-Only

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Each row occupies a position within the collection of rows in the table as well as within the collection of rows for a table section (THEAD, TBODY, or TFOOT). The `rowIndex` property returns the zero-based index value of the row inside the `rows` collection for the entire table, regardless of table section composition. In contrast, the `sectionRowIndex` property returns the zero-based index value of the row inside its row section container. If the table has no row sections defined for it, a single, all-encompassing TBODY element is assumed; in this case, the `sectionRowIndex` and `rowIndex` values are equal.

These properties serve in functions that are passed a reference to a row. However, the functions might also need to know the position of the row within the table or section. While there is no TR object property that returns a reference to the next outermost table row section or the table itself, the `parent` and parent's `parent` elements, respectively, can reference these objects.

Example on the CD-ROM

Related Items: `TABLE.rows`, `TBODY.rows`, `TFOOT.rows`, `THEAD.rows` properties.

Methods

deleteCell(*cellIndex*)
insertCell(*cellIndex*)

Returns: Nothing; Reference to New Cell.

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

The act of inserting a row into a table is not complete until you also insert cells into the row. The `insertCell()` method does just that, with a parameter indicating the zero-based index of the cell's position among other cells in the row. A value of `-1` appends the cell to the end of existing cells in the row.

When you invoke the `insertCell()` method, it returns a reference to the new cell. This gives you the opportunity to adjust other properties of that cell before moving onto the next cell. For example, if you want to insert a cell that has a column span of `2`, you adjust the `colSpan` property of the cell whose reference just returned, as in the following:

```
var oneCell = tableRowRef.insertCell(-1)
oneCell.colSpan = 2
```

Scripts that add rows and cells must make sure that they add the identical number of cells (or cell column spaces) from one row to the next. Otherwise, you have an unbalanced table with ugly blank spaces where you probably don't want them.

To remove a cell from a row, use the `deleteCell()` method. The parameter is a zero-based index value of the cell you want to remove. If all you want to do is replace the content of a cell, apply the new content to the `innerHTML` property of the TD element. This is smoother and safer than deleting and reinserting a cell because any execution error that occurs in the process results in an unbalanced table. Finally, to rid yourself of all cells in a row, use the `deleteRow()` method of the TABLE and table row section element objects.

Example

See Listing 27-2 for an example of inserting cells during the row insertion process.

Related Item: `TABLE.insertRow()` method.

TD and TH Element Objects

For HTML element properties, methods, and event handlers, see Chapter 15.

Properties	*Methods*	*Event Handlers*
`abbr`		
`align†`		
`axis`		
`background†`		

Properties	*Methods*	*Event Handlers*
bgColor†		
borderColor†		
borderColorDark†		
borderColorLight†		
cellIndex		
ch††		
chOff††		
colSpan		
headers		
height		
noWrap		
rowSpan		
vAlign††		
width		

†See TABLE element object.

††See TBODY element object.

Syntax

Accessing TD and TH element object properties and methods:

```
(IE4+)     [window.]document.all.elemID.property | method([parameters])
(IE4+)     [window.]document.all.tableID.cells[i].property |
           method([parameters])
(IE4+)     [window.]document.all.tableRowSectionID.cells[i].property |
           method([parameters])
(IE4+)     [window.]document.all.tableRowID.cells[i].property |
           method([parameters])
(IE5+/NN6) [window.]document.getElementById("elemID"). property |
           method([parameters])
(IE5+/NN6) [window.]document.getElementById("tableID").cells[i].property |
           method([parameters])
(IE5+/NN6) [window.]document.getElementById("tableRowSectionID")
           .cells[i].property | method([parameters])
(IE5+/NN6) [window.]document.getElementById("tableRowID").rows[i].property |
           method([parameters])
```

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

About these objects

TD (table data) and TH (table header) elements create cells within a table. By common convention, a TH element is rendered in today's browsers with a distinctive style — usually with a bold font and center alignment. A table cell is as deeply nested as you can get with table-related elements.

Properties of cells that are delivered in the HTML of the page are rarely modified (with the exception of the `innerHTML` property). But you still need full access to properties of cells if your scripts add rows to a table dynamically. After creating each blank table cell object, your scripts can adjust `colSpan`, `rowSpan`, `noWrap`, and other properties that influence the characteristics of that cell within the table.

See the beginning of this chapter for discussions and examples of how to add rows of cells and modify cell content under script control.

Properties

abbr
axis
headers

Value: See Text Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

These three properties are defined for table cell element objects in the W3C DOM and NN6. They all represent attributes for these elements in the HTML 4.0 specification. The purposes of these attributes and properties are geared toward browsers that provide alternate means of rendering content, such as through speech synthesis. While these properties are definitely valid for NN6, they have no practical effect. Perhaps other versions of browsers built upon the same Mozilla engine as

NN6 will use these attributes to good effect. For general application, however, you can ignore these properties — but also avoid using them as data storage spaces while a page loads. Consider them reserved for future use.

cellIndex

Value: Integer Read-Only

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

The `cellIndex` property returns an integer indicating the zero-based count of the current cell within its row. Thus, if a script is passed a reference to a cell, the `cellIndex` property reveals its position within the row. Inserting or deleting cells in the row at lower index values influences the `cellIndex` value after the alteration.

Example on the CD-ROM

Related Item: `TR.rowIndex` property.

colSpan
rowSpan

Value: Integer Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

The `colSpan` and `rowSpan` properties represent the `COLSPAN` and `ROWSPAN` attributes of table cell elements. Assign values to these properties only when you are creating new table rows and cells — and you are firm in your table cell design. If you fail to assign the correct values to either of these properties, your table cell alignment will get out of whack. Modifying these property values on an existing

table is extremely risky unless you are performing other cell manipulation to maintain the balance of rows and columns. Values for both properties are integers greater than or equal to 1.

Example on the CD-ROM

Related Item: `COL.span` property.

height
width

Value: Integer and Length String — Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Table cells may be specified to be larger than their default rendered size. This usually happens in the `HEIGHT` and `WIDTH` attributes of the cell. Settings of the `WIDTH` attribute of a COL or COLGROUP element (IE4+ and NN6+) may also govern the width of a cell. A cell's height can be inherited from the `HEIGHT` attribute setting of a table row or row section (IE4+). Both `HEIGHT` and `WIDTH` attributes are deprecated in HTML 4.0 in favor of the `height` and `width` style sheet attributes. That said, the `height` and `width` properties of a table cell echo only the settings of the explicit attributes in the cell's tag. If a style sheet in the element tag governs a cell's dimensions, then visit the cell object's `style` property to determine the dimensions. Explicit attributes override style sheet rules.

Values for these two properties are length values. These can be pixel integers or percentage values as strings. Attempts to set the sizes smaller than their default rendered size results in a cell of default size. Also be aware that enlarging a cell affects the width of the entire column and/or height of the entire row occupied by that cell.

Example on the CD-ROM

Related Items: `COL.width`, `TR.height` properties.

noWrap

Value: Boolean Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

The default behavior of a table cell is to wrap text lines within the cell if the text would extend beyond the right edge of the cell as calculated from the width of the entire table. But you can force the table to be wider to accommodate the text in an unwrapped line of text by setting the `noWrap` property (or `NOWRAP` attribute) of the cell to `true`. The `NOWRAP` attribute is deprecated in HTML 4.0.

Example on the CD-ROM

rowSpan

See `colSpan`.

width

See `height`.

OL Element Object

For HTML element properties, methods, and event handlers, see Chapter 15.

Properties	*Methods*	*Event Handlers*
`compact`		
`start`		
`type`		

Syntax

Accessing OL element object properties and methods:

```
(IE4+)     [window.]document.all.elemID.property | method([parameters])
(IE5+/NN6) [window.]document.getElementById("elemID"). property |
           method([parameters])
```

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

About this object

The OL (ordered list) element is a container of LI (list item) elements. An *ordered list* means that the list items have a sequence and are preceded by a number or letter to signify the position within the sequence. The few element-specific attributes are being deprecated in favor of style sheet definitions. For the sake of backward compatibility with existing content, however, it is likely that many future generations of browsers will continue to support these deprecated attributes. These attributes are therefore available as properties of the element object.

Most of the special appearance of a list (notably indentation) is handled automatically by the browser's interpretation of how an ordered list should look. You have control over the numbering or lettering schemes and the starting point for those sequences.

Properties

compact

Value: Boolean Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Although the properties are defined for the browsers just shown (not IE4/Mac, however), the `compact` property (and the deprecated attribute it echoes) has no impact on the density of the listing.

start

Value: Integer Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

The `start` property governs which number or letter begins the sequence of leading characters for nested LI items. If the `TYPE` attribute specifies numbers, then the corresponding number is used; if it specifies letters, then the letter of the alphabet corresponding to the number becomes as the starting character. You can change the numbering in the middle of a sequence via the `LI.value` property.

It is an extremely rare case that requires you to modify this property for an existing OL element. But if your script is creating a new element for a segment of ordered list items that has some other content intervening from an earlier OL element, you can use the property to assign a starting value to the OL group.

Example on the CD-ROM

Related Items: `type`, `LI.value` properties.

type

Value: String Constant Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

An OL element can use any of five different numbering schemes. Each scheme has a type code, whose value you can use for the `type` property. The following table shows the property values and examples:

Value	*Example*
`A`	`A, B, C, ...`
`a`	`a, b, c, ...`

Value	*Example*
I	I, II, III, ...
i	i, ii, iii, ...
1	1, 2, 3, ...

The default value is 1. You are free to adjust the property after the table has rendered, and you can even stipulate a different type for specific LI elements nested inside (see the LI.type property). If you want to have further nesting with a different numbering scheme, you can nest the OL elements and specify the desired type for each nesting level, as shown in the following HTML example:

```
<OL TYPE="A">
    <LI>One
    <LI>Two
    <LI>Three
        <OL TYPE="a">
            <LI>Sub One
            <LI>Sub Two
            <LI>Sub Three
        </OL>
    <LI>Four
</OL>
```

Indenting the HTML is optional, but it may help you to keep the nesting straight.

Example on the CD-ROM

Related Items: start, UL.type, LI.type properties.

UL Element Object

For HTML element properties, methods, and event handlers, see Chapter 15.

Properties	*Methods*	*Event Handlers*
compact†		
type		

†See OL Element Object.

Syntax

Accessing UL element object properties and methods:

```
(IE4+)     [window.]document.all.elemID.property | method([parameters])
(IE5+/NN6) [window.]document.getElementById("elemID").property |
           method([parameters])
```

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

About this object

The UL (unordered list) element is a container of LI (list item) elements. An *unordered list* means that the list items have no sequence and are preceded by symbols that don't signify any particular order. The few element-specific attributes are being deprecated in favor of style sheet definitions. For the sake of backward compatibility with existing content, however, it is likely that many future generations of browsers will continue to support these deprecated attributes. These attributes are therefore available as properties of the element object.

Most of the special appearance of a list (notably indentation) is handled automatically by the browser's interpretation of how an ordered list should look. You have control over the three possible characters that precede each item.

Properties

type

Value: String Constant Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

A UL element can use any of three different leading characters. Each character type has a type code whose value you can employ for the `type` property. Property values

are `circle`, `disc`, and `square`. The difference between a `circle` and `disc` is that the `circle` is unfilled, while the `disc` is solid. The default value is `disc`.

Example on the CD-ROM

Related Items: `OL.type`, `UL.type` properties.

LI Element Object

For HTML element properties, methods, and event handlers, see Chapter 15.

Properties	*Methods*	*Event Handlers*
`type`		
`value`		

Syntax

Accessing LI element object properties and methods:

```
(IE4+)     [window.]document.all.elemID.property | method([parameters])
(IE5+/NN6) [window.]document.getElementById("elemID").property |
           method([parameters])
```

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

About this object

An LI (list item) element contains the HTML that is displayed for each item within an OL or UL list. Note that you can put any HTML you want inside a list item, including images. Attributes and properties of this element enable you to override the specifications declared in the OL or UL containers (except in IE/Mac).

Properties

type

Value: String Constant Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Because either an OL or UL container can own an LI element, the `type` property accepts any of the values that you assign to the `type` properties of both the OL and UL element objects. See the `OL.type` and `UL.type` properties earlier in this chapter for lists of those values.

Exercise caution, however, if you attempt to mix and match types. For example, if you try to set the `LI.type` property of an LI element to `circle` inside an OL element, the results vary from browser to browser. NN6, for example, follows your command; however, IE may display some other characters.

Example on the CD-ROM

Related Items: `OL.type`, `UL.type` properties.

value

Value: Integer Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

The `value` property governs which number or letter is used for the current list item inside an ordered list. Employ this attribute and property to override the natural progression. Because these sequence characters can be letters, numbers, or Roman numerals, the integer you specify for this property is converted to the numbering scheme in force by the LI or OL element's `type` property.

Example on the CD-ROM

Related Item: `OL.start` property.

DL, DT, and DD Element Objects

For HTML element properties, methods, and event handlers, see Chapter 15.

Properties	*Methods*	*Event Handlers*
`compact`†		

†See OL Element Object.

Syntax

Accessing DL, DT, and DD element object properties and methods:

```
(IE4+)        [window.] document.all.elemID.property | method([parameters])
(IE5+/NN6+)   [window.] document.getElementById("elemID"). property |
              method([parameters])
```

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

About these objects

Three elements — DL, DT, and DD — provide context and (optionally) formatting for definitions in a document. The DL element is the outer wrapper signifying a definition list. Each definition term should be inside a DT element, while the definition description should be in the nested DD element. The HTML for a simple definition list has the following structure:

```
<DL>
    <DT>First term
    <DD>First term's definition
    <DT>Second  term
    <DD>Second  term's definition
</DL>
```

While there are no specific requirements for rendering definition lists by convention, the term and description are usually on different lines with the description indented.

All three of these elements are treated as element objects, sharing the same properties, methods, and event handlers of generic element objects. The only one of the three that has anything special is the DL element, which has a `compact` property. IE4+ for Windows does respond to this attribute and property by putting the description and term on the same line if the term is shorter than the usual indentation space of the description.

DIR and MENU Element Objects

For HTML element properties, methods, and event handlers, see Chapter 15.

Properties	*Methods*	*Event Handlers*
`compact`†		

†See OL Element Object.

Syntax

Accessing DIR and MENU element object properties and methods:

```
(IE4+)     [window.]document.all.elemID.property | method([parameters])
(IE5+/NN6) [window.]document.getElementById("elemID"). property |
           method([parameters])
```

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

About these objects

The DIR and MENU elements are treated in modern browsers as if they were UL elements for unordered lists of items. Both elements are deprecated in HTML 4.0; yet, because they are acknowledged in that standard, they are also acknowledged in the W3C DOM (and the IE DOM, too). Originally intended to assist in creating single and double columns of text (long since supplanted by tables), usage of these elements has fallen out of favor and is discouraged.

✦ ✦ ✦