node

Eigenschaften:

Methoden:

node: Allgemeines zur Verwendung

Das node-Objekt ist das zentrale Objekt des Document Object Models (DOM) (*node* = *Knoten*). Hintergrund ist das Modell, dass ein Auszeichnungssprachen-Dokument, egal ob in HTML oder einer anderen, XML-basierten Auszeichnungssprache geschrieben, aus einem Baum von Knoten besteht. Um sich mit diesem Modell näher zu beschäftigen, können Sie im XML-Kapitel dieses Dokuments den Abschnitt **Baumstruktur und Knoten einer XML-Datei** lesen.

Jedes Element, jedes Attribut und alle Zeichendaten stellen eigene Knoten dar. Diese Knoten bilden die Baumstruktur. Das node-Objekt stellt Eigenschaften und Methoden bereit, um auf die einzelnen Knoten zuzugreifen, egal, wie tief diese Knoten in der Baumstruktur liegen.

Das node-Objekt stellt damit die allgemeinere und für alle XML-gerechten Sprachen gültige Variante dessen dar, was die **HTML-Elementobjekte** speziell für HTML darstellen. Sie können in JavaScripts, die in HTML-Dateien notiert oder eingebunden sind, sowohl mit den HTML-Elementobjekten als auch mit dem node-Objekt arbeiten. Manches ist über die HTML-Elementobjekte bequemer zu lösen, für andere Aufgaben eignet sich wiederum das node-Objekt besser. Das node-Objekt gilt unter Puristen allerdings als das "reinere" DOM, eben weil es nicht auf HTML beschränkt ist.

Um auf die Eigenschaften und Methoden des node-Objekts zugreifen zu können, benötigen Sie einen Knoten. Um auf vorhandene Knoten im Dokument zuzugreifen, werden die Methoden des **document**-Objekts **getElementById**, **getElementsByName** und **getElementsByTagName** verwendet. Ausgehend davon können Sie die Attributknoten, Textknoten und weitere Element-Kindknoten eines Elements ansprechen.

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title>
</head><body>
<h1 id="Ueberschrift" align="center">Knoten in der <i>Baumstruktur</i></h1>
<script type="text/javascript">
Elementknoten = document.getElementById("Ueberschrift");
WertErsterKindknoten = Elementknoten.firstChild.nodeValue;
document.write("Der Wert ihres ersten Kindknotens lautet: <b>" + WertErsterKindknoten + "<\/b>");
</script>
</body></html>
```

Erläuterung:

Die Beispieldatei enthält eine Überschrift erster Ordnung mit Text, von dem ein Teil wiederum als kursiv ausgezeichnet ist. In dem JavaScript, das unterhalb davon notiert ist, wird zunächst mit `document.getElementById("Ueberschrift")` (ohne weitere Eigenschaft oder Methode dahinter) auf das `h1`-Element der Überschrift zugegriffen. Der Rückgabewert von `getElementById()` ist das Knotenobjekt der Überschrift. Der Rückgabewert wird im Beispiel in der Variablen `Elementknoten` gespeichert. Diese Variable speichert also einen gültigen Knoten des Dokuments, und auf die Variable sind daher die Eigenschaften und Methoden des `node`-Objekts anwendbar. Im Beispiel wird mit `Elementknoten.firstChild.nodeValue` der Wert des ersten Kindknotens der Überschrift ermittelt. Dessen Wert wird schließlich mit document.write() ins Dokument geschrieben.

Die Verwendung von Variablen ist nicht zwingend erforderlich. Das obige Beispiel funktioniert genauso, wenn Sie notieren:
`document.write("Der Wert ihres ersten Kindknotens lautet: <b>" +`
`document.getElementById("Ueberschrift").firstChild.nodeValue + "<\/b>");`

Der geschriebene Wert lautet im Beispiel: `Knoten in der` ... der erste Kindknoten der Überschrift ist also ihr Zeicheninhalt. Das Wort `Baumstruktur` gehört nicht dazu, da es ja durch ein `i`-Element ausgezeichnet ist, das selbst wieder einen eigenen, weiteren Kindknoten der Überschrift darstellt.

Die beiden Attribute, die im einleitenden Überschriften-Tag notiert sind, zählen übrigens **nicht** als Kindknoten. Das W3-Konsortium ist der Auffassung, dass Attribute hierarchisch gesehen keine "Unterobjekte" von Elementen sind, sondern "assoziierte Objekte". Um auf Attributknoten zuzugreifen, bietet das `node`-Objekt eigene Eigenschaften und Methoden an. Zum Verständnis ist es jedoch wichtig, dass die Attribute eines Elements nicht als dessen Kindknoten betrachtet werden, weshalb etwa eine Objekteigenschaft wie `firstChild` die Attribute übergeht.

attributes

Speichert einen Array aus verfügbaren Attributen eines Elements.

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title>
</head><body bgcolor="#FFFFCC" text="#000099">
<script type="text/javascript">
document.write("Das body-Element hat <b>" +
  document.getElementsByTagName('body')[0].attributes.length + "<\/b> Attribute");
</script>
</body></html>
```

Erläuterung:

Das Beispiel greift mit `document.getElementsByTagName('body')[0]` auf das `body`-Element der Datei zu und ermittelt über die Array-Grundeigenschaft `length` des `attributes`-Arrays die Anzahl der Attribute, die in dem Element definiert sind. Zur Kontrolle wird das Ergebnis mit `document.write` ins Dokument geschrieben. Im Beispiel ist `2` der Wert von `attributes.length`, da im einleitenden `<body>`-Tag zwei Attribute notiert sind.

Über den `attributes`-Array können Sie auch auf Attribute zugreifen. So liefert beispielsweise `document.getElementsByTagName('body')[0].attributes[0].nodeValue` den Wert `#FFFFCC` oder `#000099`, da `attributes[0]` eines der beiden Attribute bezeichnet, und ↓ nodeValue den zugehörigen Wert speichert.

Beachten Sie:

Die Reihenfolge, in der die Attribute im `attributes`-Array gespeichert sind, ist nicht definiert. Sie können sich nicht darauf verlassen, dass `attributes[0]` tatsächlich das erste, im HTML-Code notierte, Attribut ist. Diese Notation ist nur dazu gedacht, einfache Zählschleifen zu erlauben.

Der Internet Explorer ermittelt zwar einen Wert, jedoch nicht den richtigen. Im Beispiel wurde mit der Produktversion 5.5 statt `2` der Wert

94 ermittelt - vermutlich die Anzahl insgesamt verfügbarer Attribute für das body-Element, inklusive Event-Handler usw. Dies ist jedoch unbrauchbar.

childNodes

Speichert einen Array aus verfügbaren Kindknoten eines Knotens.

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title>
</head><body>
<p id="derText">Text mit <b>fettem Text</b> und <u>unterstrichenem Text</u></p>
<script type="text/javascript">
var Anzahl = document.getElementById("derText").childNodes.length;
var Erster = document.getElementById("derText").childNodes[0].nodeValue;
document.write("Anzahl Kindknoten: <b>" + Anzahl + "<\/b><br>");
document.write("Wert des ersten Kindknotens: <b>" + Erster + "<\/b><br>");
</script>
</body></html>
```

Erläuterung:

Das Beispiel enthält einen Textabsatz mit Zeicheninhalt und weiteren Elementen zur Formatierung. Unterhalb davon ist ein JavaScript-Bereich notiert. Dort wird mit `document.getElementById("derText")` auf das p-Element zugegriffen. Über die Array-Grundeigenschaft `length` des `childNodes`-Arrays wird die Anzahl der Kindelemente ermittelt, die das p-Element hat. Der Rückgabewert wird in der Variablen `Anzahl` gespeichert. Über `childNodes[0]` wird auf das erste Kindelement zugegriffen. Dessen Inhalt wird mit ↓ nodeValue ermittelt. Zur Kontrolle schreibt das Script die Ergebnisse mit `document.write()` ins Dokument.

Die Anzahl der Kindelemente des p-Elements beträgt übrigens 4:
Das erste Kindelement ist der Zeicheninhalt `Text mit` ,
das zweite Kindelement das b-Element,
das dritte Kindelement der Zeicheninhalt `und` ,
und das vierte Kindelement das u-Element.

Wenn ein Knoten keine Kindknoten enthält, hat `childNodes` den Wert `null`.

Beachten Sie:

Gemäß dem DOM stellt bereits ein Zeilenumbruch oder ein Leerzeichen im Quelltext zwischen Elementknoten einen eigenen Text-Kindknoten dar. Auch HTML-Kommentare sind eigene Knoten. Daran halten sich alle gängigen Browser bis auf den Internet Explorer unter Windows. Dieser ignoriert solche "Leerraum"-Zeichen zwischen Elementknoten völlig, dasselbe gilt für Kommentarknoten. Sie tauchen nicht im `childNode`-Array auf. Konqueror und Safari setzen "Leerraum"-Zeichen zwar in Textknoten um, ignorieren aber Kommentarknoten.

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title>
</head><body>
<ul id="ersteListe"><li>erster Punkt</li><li>zweiter Punkt</li></ul>
<script type="text/javascript">
var Anzahl = document.getElementById("ersteListe").childNodes.length;
document.write("<p>Erste Liste: Anzahl Kindknoten: <b>" + Anzahl + "<\/b><\/p>");
</script>
<ul id="zweiteListe">
<li>erster Punkt</li> <li>zweiter Punkt</li>
</ul>
<script type="text/javascript">
var Anzahl = document.getElementById("zweiteListe").childNodes.length;
document.write("<p>Zweite Liste: Anzahl Kindknoten: <b>" + Anzahl + "<\/b><\/p>");
</script>
</body></html>
```

Erläuterung:

Im Beispiel sind zwei Listen definiert. Beide unterscheiden sich nur dahingehend, dass sich zwischen den einzelnen Elementen der erste Liste keine Leerzeichen oder Zeilenumbrüche befinden. In der zweiten Liste dagegen wurden Leerzeichen und Zeilenumbrüche verwendet. Standardkonforme Browser geben für die erste Liste als Anzahl der Kindknoten 2 aus und für die zweite Liste 5. Die jeweiligen Leerzeichen und Zeilenumbrüche werden als eigene Kindknoten betrachtet.

Der Internet Explorer unter Windows ignoriert diese Kindknoten und erkennt erst dann einen Kindknoten mit dem Namen `#text`, wenn dieser ein entsprechendes Textzeichen oder ein erzwungenes Leerzeichen ` ` enthält.

data

Speichert Zeichendaten eines Knotens, sofern es sich um einen Textknoten handelt.

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title>
<script type="text/javascript">
function Ausgeben () {
  alert(document.getElementById("Dokumentinhalt").firstChild.data);
}
</script>
</head><body id="Dokumentinhalt" onload="Ausgeben()">
Das ist Text, der einfach so im Dokument steht.
</body></html>
```

Erläuterung:

Das Beispiel enthält zwischen `<body>` und `</body>` nichts als reinen Text, also Zeichendaten. Nachdem das Dokument geladen ist (`onload`), wird die JavaScript-Funktion `Ausgeben()` aufgerufen. Diese greift mit `document.getElementById("Dokumentinhalt")` auf das `body`-Element zu. Mit firstChild wird auf das erste Kindelement zugegriffen, und mit `data` auf dessen Inhalt. Das erste (und einzige) Kindelement des `body`-Elements ist im Beispiel der Text. Dieser wird denn auch in dem Meldungsfenster angezeigt, das die Funktion `Ausgeben()` ausgibt.

Wenn Sie den Wert der Eigenschaft `data` ändern, wird der Zeicheninhalt des Elements einfach durch den neuen Inhalt ersetzt. Wenn Sie im obigen Beispiel also notieren würden:
`document.getElementById("Dokumentinhalt").firstChild.data = "neuer Text";`
Dann würde anschließend der bisherige Text zwischen `<body>` und `</body>` dynamisch durch den neuen ersetzt.

firstChild

Speichert das Objekt des ersten Kindknotens eines Knotens.

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title>
</head><body>
<ul><li>erster Punkt</li><li>zweiter Punkt</li></ul>
<script type="text/javascript">
var ErsterPunkt = document.getElementsByTagName("ul")[0].firstChild;
document.write(ErsterPunkt.firstChild.data);
</script>
</body></html>
```

Erläuterung:

Das Beispiel enthält eine Aufzählungsliste mit zwei Listenpunkten. Unterhalb davon ist ein JavaScript-Bereich notiert. Dort wird mit `document.getElementsByTagName("ul")[0]` auf das erste `ul`-Element im Dokument zugegriffen. Mit `firstChild` wird dessen erster Kindknoten angesprochen. Im Beispiel ist dies das erste `li`-Element. In der Variablen `ErsterPunkt` ist also anschließend das Knotenobjekt des ersten `li`-Elements gespeichert, aber nicht sein Textinhalt. Dieser wird jedoch anschließend mit `document.write()` ausgegeben. Da der Textinhalt aus Sicht des `li`-Elementknotens wieder einen Unterknoten darstellt, kann mit `ErsterPunkt.firstChild.data` auf den Text des ersten Listenpunktes zugegriffen werden.

Berücksichtigen Sie beim Nachvollziehen dieses Beispieles die ↑ Besonderheit des Internet Explorers unter Windows im Umgang mit Kindknoten.

↑↓

lastChild

Speichert das Objekt des letzten Kindknotens eines Knotens.

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title>
</head><body>
<ul><li>erster Punkt</li><li>zweiter Punkt</li></ul>
<script type="text/javascript">
var LetzterPunkt = document.getElementsByTagName("ul")[0].lastChild;
document.write(LetzterPunkt.firstChild.data + " und ");
document.write(LetzterPunkt.lastChild.data);
</script>
</body></html>
```

Erläuterung:

Das Beispiel enthält eine Aufzählungsliste mit zwei Listenpunkten. Unterhalb davon ist ein JavaScript-Bereich notiert. Dort wird mit `document.getElementsByTagName("ul")[0]` auf das erste `ul`-Element im Dokument zugegriffen. Mit `lastChild` wird dessen letzter Kindknoten angesprochen. Im Beispiel ist dies das zweite und letzte `li`-Element. In der Variablen `LetzterPunkt` ist also anschließend das Knotenobjekt des zweiten `li`-Elements gespeichert, aber nicht sein Textinhalt. Dieser wird jedoch anschließend mit `document.write()` ausgegeben, und zwar gleich zweimal. Da der Textinhalt aus Sicht des `li`-Elementknotens wieder einen Unterknoten darstellt, und zwar den einzigen, kann er mit `LetzterPunkt.firstChild.data`, aber ebensogut mit `LetzterPunkt.lastChild.data` angesprochen werden.

Berücksichtigen Sie beim Nachvollziehen dieses Beispieles die ↑ Besonderheit des Internet Explorers unter Windows im Umgang mit Kindknoten.

↑↓

nextSibling

Speichert aus Sicht eines Knotens den unmittelbar nächstfolgenden Knoten im Strukturbaum. Wenn kein Knoten mehr folgt, wird `null` gespeichert.

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title>
</head><body>
<ul>
<li>erster Punkt</li>
<li>zweiter Punkt</li>
</ul>
<script type="text/javascript">
document.write("Das ul-Element hat folgende Knoten unter sich:<br>");
var Knoten = document.getElementsByTagName("ul")[0].firstChild;
while (Knoten != null) {
  document.write("Einen Knoten mit dem Namen <b>" + Knoten.nodeName + "<\/b><br>");
  Knoten = Knoten.nextSibling;
}
</script>
</body></html>
```

Erläuterung:

Das Beispiel enthält eine Aufzählungsliste mit zwei Listenpunkten. Unterhalb davon ist ein JavaScript-Bereich notiert. Dort wird mit `document.getElementsByTagName("ul")[0].firstChild` auf das erste `ul`-Element im Dokument zugegriffen. In der nachfolgenden ≡ while-Schleife wird geprüft, ob der Knoten verschieden von `null` ist, und der ↓ nodeName des Knotens wird

ausgegeben. Anschließend ist die Variable `Knoten` mit dem nachfolgenden Knoten (`Knoten = Knoten.nextSibling;`) belegt. Wenn der Strukturbaum durchlaufen ist, ist in der Eigenschaft `nextSibling` der Wert `null` gespeichert, was zum Abbruch der Schleife führt.

Die Eigenschaft `nextSibling` arbeitet jeweils den nächsten Knoten eines Strukturbaumes ab. Sie verfolgt jedoch nicht die Kindknoten, die in einem Knoten enthalten sein können.

nodeName

Speichert den Namen eines Knotens.

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title>
</head><body><script id="dasScript" type="text/javascript">
Knoten = document.getElementById("dasScript");
var Knoten = document.body.firstChild;
document.write("Dieses Script-Element hat folgende Knotennamen: <b>" + Knoten.nodeName + "</b>")
</script>
</body></html>
```

Erläuterung:

Das Beispiel enthält im sichtbaren Dokumentbereich nichts weiter als ein JavaScript, das den Knotennamen des eigenen `script`-Elements ausgibt, also `SCRIPT`. Dazu greift das Script mit `document.body.firstChild` auf das `script`-Element zu. Dessen Knoten wird in der Variablen `Knoten` gespeichert. Mit `Knoten.nodeName` wird dann der Name dieses Knotens ermittelt.

Elementknoten und Attributknoten haben Namen, Textknoten jedoch nicht. Beim Versuch, den Namen eines Textknotens zu ermitteln, wird der Wert `#text` gespeichert.

Berücksichtigen Sie beim Nachvollziehen dieses Beispieles die Besonderheit des Internet Explorers unter Windows im Umgang mit Kindknoten.

nodeType

Speichert den Typ eines Knotens in Form einer Nummer. Das W3-Konsortium hat dazu folgende Zuordnungen festgelegt - einige davon sind XML-spezifisch:

Nummer	Knotentyp
1	Elementknoten
2	Attributknoten
3	Textknoten
4	Knoten für CDATA-Bereich
5	Knoten für Entity-Referenz
6	Knoten für Entity
7	Knoten für Verarbeitungsanweisung
8	Knoten für Kommentar
9	Dokument-Knoten
10	Dokumenttyp-Knoten
11	Dokumentfragment-Knoten
12	Knoten für Notation

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title>
</head><body>
<p align="center">ein kleiner Text</p>
```

```
<script type="text/javascript">
var Element = document.getElementsByTagName("p")[0];
var Ausrichtung = Element.getAttributeNode("align");
alert(Ausrichtung.nodeType);
</script>
</body></html>
```

Erläuterung:

Das Beispiel enthält einen Textabsatz mit einem `align`-Attribut zur Ausrichtung. Unterhalb des Textabsatzes ist ein JavaScript-Bereich notiert. Dort wird mit `document.getElementsByTagName("p")[0]` auf das `p`-Element zugegriffen. Mit `Element.getAttributeNode("align")` wird auf dessen Attributknoten zugegriffen. In der Variablen `Ausrichtung` steht anschließend das Objekt des Attributknotens. Ein Meldungsfenster gibt dann im Beispiel den Knotentyp dieses Knotens mit `Ausrichtung.nodeType` aus. Der Wert beträgt `2`, da es sich um einen Attributknoten handelt.

Beachten Sie:

Der Internet Explorer 5.x interpretiert die Eigenschaft `nodeType` zwar, in Version 5.5 aber nicht das obige Beispiel. Der Grund ist, dass er die Methode `getAttributeNode()` nicht unterstützt. Im Internet Explorer 6.0 und im Internet Explorer 5.0 Macintosh Edition ist das Beispiel dagegen nachvollziehbar.

nodeValue

Speichert den Wert oder Inhalt eines Knotens. Bei Textknoten ist dies der Text, bei Attributknoten der zugewiesene Attributwert. Bei Elementknoten hat diese Eigenschaft den Wert `null`.

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title>
<script type="text/javascript">
function TextAendern () {
  document.getElementById("derText").firstChild.nodeValue = document.Formular.neuerText.value;
}
</script>
</head><body>
<p id="derText">Hier steht ein Text</p>
<form name="Formular" action="" onsubmit="TextAendern(); return false">
<input type="text" size="40" name="neuerText">
<input type="submit" value=" OK ">
</form>
</body></html>
```

Erläuterung:

In dem Beispiel ist ein Textabsatz notiert und unterhalb davon ein Formular mit einem Eingabefeld und einem Klick-Button. Beim Absenden des Formulars wird die Funktion `TextAendern()` aufgerufen, die im Dateikopf notiert ist. Diese Funktion greift mit `document.getElementById("derText")` auf das `p`-Element zu, weiterhin mit firstChild auf den ersten Kindknoten dieses Elements, also den Textinhalt, und weist dessen Eigenschaft `nodeValue` den Inhalt aus dem Eingabefeld des Formulars zu.

Beachten Sie:

Das Beispiel zeigt, dass `nodeValue` eine ähnliche Funktionalität hat wie das die Eigenschaft innerText beim klassischen DHTML nach Microsoft-Syntax. Dennoch gibt es Unterschiede: Wenn beispielsweise notiert wäre:
`<p>Text mit <b>fettem Text</b></p>`
Dann würde `firstChild.nodeValue` aus Sicht des `p`-Elements nur den Wert `Text mit` liefern und auch nur diesen Teil ändern können, da dahinter durch das innere `b`-Element ein neuer Knoten beginnt.
Eine direkte Entsprechung zu innerHTML gibt es erst recht nicht im DOM. "Inneres HTML" muss im DOM mit Hilfe geeigneter Methoden wie document.createElement(), document.createAttribute() und document.createTextNode() erzeugt werden.

parentNode

Speichert den Elternknoten eines Knotens.

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title>
</head><body>
<ul>
<li>ein Punkt</li>
<li>ein zweiter</li>
</ul>
<script type="text/javascript">
alert(document.getElementsByTagName("li")[0].parentNode.parentNode.tagName);
</script>
</body></html>
```

Erläuterung:

Das Beispiel enthält eine Aufzählungsliste. Unterhalb davon ist ein JavaScript-Bereich notiert. Dort wird der Name des Großelternelements des ersten `li`-Elements in einem Meldungsfenster ausgegeben. Dazu wird mit `document.getElementsByTagName("li")[0]` auf das erste `li`-Element zugegriffen. Das erste `parentNode` dahinter greift auf dessen Elternelement zu, und das zweite `parentNode` auf das Elternelement des Elternelements. Von diesem Element wird mit `tagName` der Name des Elementes ermittelt und ausgegeben. Das Attribut `tagName` kennen alle Knoten der Art *Element*.
Ausgegeben wird im Beispiel `BODY`.

previousSibling

Speichert aus Sicht eines Knotens den unmittelbar vorhergehenden Knoten im Strukturbaum. Wenn kein Knoten vorausgeht, wird `null` gespeichert.

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title></head>
<body id="Dokumentinhalt"
onload="alert(document.getElementById('Dokumentinhalt').previousSibling.nodeName)">
</body></html>
```

Erläuterung:

Das Beispiel gibt, nachdem das Dokument geladen ist (`onload`) in einem Meldungsfenster den Knotennamen des Vorgängerknotens vom `body`-Element aus. Dazu wird mit `document.getElementById('Dokumentinhalt')` auf das `body`-Element zugegriffen.
Ausgegeben wird dann `HEAD`, da das `head`-Element im Sinne des Strukturbaums das unmittelbare Vorgängerelement des `body`-Elements ist.

Berücksichtigen Sie beim Nachvollziehen dieses Beispieles die Besonderheit des Internet Explorers unter Windows im Umgang mit Kindknoten.

appendChild()

Hängt einen zuvor neu erzeugten Knoten in die bestehende Knotenstruktur ein, und zwar so, dass er als letztes Kindelement eines anzugebenden Knotens eingefügt wird.

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title></head>
<body>
<ol id="Liste">
<li>Element</li>
```

```
</ol>
<script type="text/javascript">
document.getElementById("Liste").removeChild(document.getElementById("Liste").firstChild);

for (var i = 0; i < 10; i++) {
  var newLI = document.createElement("li");
  var liNummer = i + 1;
  var newLIText = document.createTextNode("Das ist Listeneintrag Nummer " + liNummer);
  document.getElementById("Liste").appendChild(newLI);
  document.getElementsByTagName("li")[i].appendChild(newLIText);
}
</script>
</body></html>
```

Erläuterung:

Das Beispiel füllt eine nummerierte Liste automatisch mit Daten. Unmittelbar nach dem Einlesen der nummerierten Liste folgt ein JavaScript-Bereich. Zuerst wird mit der Methode removeChild() das vorhandene Listenelement entfernt. Anschließend ist eine for-Schleife notiert, die 10 mal durchlaufen wird. Bei jedem Schleifendurchgang wird zunächst mit document.createElement() ein neues Element vom Typ `li` erzeugt. Dann wird eine Variable `liNummer` auf einen Wert gesetzt, der um 1 höher ist als der des Schleifenzählers `i`. Diese Variable wird in der nachfolgenden Anweisung benutzt, bei der mit document.createTextNode() ein neuer Textknoten erzeugt wird. Anschließend folgen - immer noch innerhalb der `for`-Schleife - zwei `appendChild()`-Aufrufe. Die erste der Anweisungen greift mit `document.getElementById("Liste")` auf das `ol`-Element zu und führt dazu, dass diesem ein neues Kindelement am Ende hinzugefügt wird. Angehängt wird der zuvor neu erzeugte Elementknoten `newLI`, der ja ein neues `li`-Element speichert. Beim zweiten Aufruf wird der Schleifenzähler `i` benutzt, um mit `document.getElementsByTagName("li")[i]` auf das gerade neu erzeugte `li`-Element zuzugreifen. Ihm wird mit `appendChild()` der zuvor erzeugte Textknoten als Kindelement hinzugefügt. Auf diese Weise füllt sich die Liste bei jedem Schleifendurchlauf um ein neues `li`-Element mitsamt Zeicheninhalt.

appendData()

Fügt einem Textknoten oder dem Wert eines Attributknotens am Ende Daten hinzu, ohne die bestehenden Daten zu überschreiben.

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title>
<script type="text/javascript">
function ergaenzen () {
  var Rest = document.createTextNode("vollkommen!");
  document.getElementById("Absatz").firstChild.appendData(Rest.nodeValue);
}
</script>
</head>
<body>
<p id="Absatz">Ich bin ja so un</p>
<p><a href="javascript:ergaenzen()">un - was?</a></p>
</body></html>
```

Erläuterung:

Das Beispiel enthält einen Absatz mit nicht ganz sinnigem Text und unterhalb davon einen Verweis. Beim Anklicken des Verweises wird die Funktion `ergaenzen()` aufgerufen, die im Dateikopf notiert ist. Diese Funktion erzeugt zunächst mit document.createTextNode() einen neuen Textknoten. Anschließend greift sie mit `document.getElementById("Absatz").firstChild` auf den Knoten zu, der den Zeicheninhalt des Textabsatzes mit dem unfertigen Text darstellt, und fügt dort mit `appendData()` den Wert des neu erzeugten Textknotens (`Rest.nodeValue`) hinzu. Aus dem Text `ich bin ja so un` wird also `ich bin ja so unvollkommen!`

Beachten Sie:

Der Internet Explorer 5.x unterstützt diese Methode noch nicht. Sie können sich damit behelfen, auf den Inhalt eines Textknotens zuzugreifen und dann mit `nodeValue += "Text"` Daten hinzuzufügen. Im Internet Explorer 6.0 und im Internet Explorer 5.0 Macintosh Edition wird die Methode dagegen unterstützt.

cloneNode()

Erstellt eine identische Kopie eines Knotens, je nach Wunsch mit oder ohne zugehörige Unterknotenstruktur.

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title></head>
<body>
<span id="Dolly">Dolly </span>
<script type="text/javascript">
Dolly2 = document.getElementById("Dolly").cloneNode(true);
document.getElementById("Dolly").firstChild.nodeValue += Dolly2.firstChild.nodeValue;
</script>
</body></html>
```

Erläuterung:

Das Beispiel enthält einen in einem `span`-Element notierten Text `Dolly` . Unterhalb davon ist ein JavaScript-Bereich notiert. Dort wird mit `document.getElementById("Dolly")` auf das `span`-Element zugegriffen. Dieses wird mit `cloneNode()` kopiert, und der Rückgabewert wird in der Variablen `Dolly2` gespeichert. Der Rückgabewert ist eine identische Kopie des Elementknotens des `span`-Elements. Als Parameter wird der Methode im Beispiel `true` übergeben. Das bedeutet, dass auch der Textinhalt des Elements mitkopiert wird. Um nur das Element ohne den Inhalt zu klonen, müssen Sie `false` übergeben.
Im Beispiel wird anschließend mit `document.getElementById("Dolly").firstChild.nodeValue` der Inhalt des `span`-Elements angesprochen und mittels Zeichenkettenoperation um den Wert des Kindelements des Klons erweitert, auf den mit `Dolly2.firstChild.nodeValue` zugegriffen wird. Am Ende steht also `Dolly Dolly` als Inhalt in dem `span`-Element.

deleteData()

Löscht Daten eines Textknotens oder den Wert eines Attributknotens.

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title>
<script type="text/javascript">
function loeschen () {
  var AnzahlZeichen = document.getElementsByTagName("p")[0].firstChild.nodeValue.length;
  document.getElementsByTagName("p")[0].firstChild.deleteData(0, AnzahlZeichen);
}
</script>
</head>
<body>
<p>Die Beredtsamkeit an sich ist die Tugend des Redens
und die Lust des Schweigens an seinem Gegenteil.
Die Beredtsamkeit ist also ...
<a href="javascript:loeschen()">och nee, besser nicht!</a></p>
</body></html>
```

Erläuterung:

Das Beispiel enthält einen Textabsatz mit Text und einem Verweis. Beim Anklicken des Verweises wird die Funktion `loeschen()` aufgerufen, die im Dateikopf notiert ist. Diese Funktion ermittelt zunächst durch Zugriff auf den Zeicheninhalt des Absatzes (`document.getElementsByTagName("p")[0].firstChild.nodeValue`) mit der Methode string.length dessen Zeichenanzahl. Der Wert wird für die folgende Anweisung benötigt. Dort wird wieder mit `document.getElementsByTagName("p")[0].firstChild` auf den Zeicheninhalt des Absatzes zugegriffen. Mit `deleteData()` wird der Inhalt gelöscht.
Die Methode `deleteData()` erwartet zwei Parameter:
1. das Zeichen, ab dem gelöscht werden soll (`0` steht für "ab dem ersten Zeichen"),
2. wie viele Zeichen gelöscht werden sollen (im Beispiel wird dazu die Variable `AnzahlZeichen` übergeben, in der die zuvor ermittelte Anzahl der Zeichen im Textknoten gespeichert ist).
Der Verweis im Beispiel bleibt übrigens nach dem Löschen stehen, weil das `a`-Element des Verweises schon wieder einen neuen Kindknoten darstellt.

Beachten Sie:

Der Internet Explorer 5.x unterstützt diese Methode noch nicht. Um die gesamten Zeichendaten eines Textknotens zu löschen, können Sie sich damit behelfen, auf den Knoten zuzugreifen und der Eigenschaft `nodeValue` den Wert `""` (leere Zeichenkette) zuzuweisen. Im

Internet Explorer 6.0 und im Internet Explorer 5.0 Macintosh Edition wird die Methode dagegen unterstützt.

 getAttribute()

Ermittelt den Wert eines bestimmten Attributs in einem Element.

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title>
<script type="text/javascript">
function Anzeigen (attr) {
  alert(document.body.getAttribute(attr));
}
</script></head>
<body bgcolor="#FFFFCC" text="#E00000" link="#0000E0" alink="#000080" vlink="#000000">
<a href="javascript:Anzeigen('bgcolor')">Hintergrundfarbe?</a><br>
<a href="javascript:Anzeigen('text')">Textfarbe?</a><br>
<a href="javascript:Anzeigen('link')">Linkfarbe noch nicht besuchte Seiten?</a><br>
<a href="javascript:Anzeigen('vlink')">Linkfarbe besuchte Seiten?</a><br>
<a href="javascript:Anzeigen('alink')">Linkfarbe aktivierte Links?</a>
</body></html>
```

Erläuterung:

Das Beispiel enthält mehrere Verweise. Alle Verweise rufen beim Anklicken die Funktion `Anzeigen()` auf, die im Dateikopf notiert ist. Übergeben wird der Funktion der gewünschte Attributname. Die Funktion greift mit `document.body` auf das `body`-Element zu. Mit `getAttribute()` lassen sich dann Attributwerte des einleitenden `<body>`-Tags ermitteln. Die Methode erwartet den Namen des gewünschten Attributs und liefert dessen Wert zurück. Im Beispiel bekommt sie jeweils den der Funktion übergebenen Parameter `attr` weitergereicht.

 getAttributeNode()

"Holt" einen Attributknoten. Liefert das Knotenobjekt des gewünschten Attributs zurück.

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title>
<script type="text/javascript">
function anpassen () {
  var CSSKnoten = document.getElementsByTagName("h1")[0].getAttributeNode("style");
  for (var i = 0; i < document.getElementsByTagName("p").length; i++) {
    var CSSKnotenNeu = CSSKnoten.cloneNode(true);
    document.getElementsByTagName("p")[i].setAttributeNode(CSSKnotenNeu);
  }
}
</script></head>
<body>
<h1 style="font-family:Tahoma; font-size:200%; color:red">Rot und Groß!</h1>
<p>ein Absatz</p>
<p>und noch einer</p>
<p>und einer mit einem <a href="javascript:anpassen()">Link zum Anpassen</a></p>
</body></html>
```

Erläuterung:

Das Beispiel enthält eine Überschrift mit einem `style`-Attribut sowie drei Textabsätze. Der letzte davon enthält einen Verweis, bei dessen Anklicken die Funktion `anpassen()` aufgerufen wird, die im Dateikopf notiert ist. Diese Funktion holt sich mit `document.getElementsByTagName("h1")[0].getAttributeNode("style")` das Knotenobjekt des `style`-Elements aus der Überschrift. Der Rückgabewert, also das Attributknoten-Objekt, wird in der Variablen `CSSKnoten` gespeichert. Anschließend wird in einer for-Schleife auf alle `p`-Elemente des Dokuments zugegriffen. Bei jedem Schleifendurchlauf wird mit ↑ cloneNode() eine Kopie des Attributknotens erzeugt und mit ↓ setAttributeNode() dem jeweils aktuellen `p`-Element hinzugefügt. Die `p`-Elemente "erben" auf diese

Weise die CSS-Eigenschaften der Überschrift.

Beachten Sie:

Die Methode `getAttributeNode()` werden Sie selten brauchen. In den meisten Fällen können Sie ↑ **getAttribute()** verwenden, das sich einfacher handhaben lässt und auch vom Internet Explorer besser unterstützt wird.

Opera unterstützt die Methode `getAttributeNode()` bereits in Version 7. Dieser Browser hat jedoch spezielle Probleme beim Auslesen von `style`-Attributknoten. Im Beispiel enthält `CSSKnoten` zwar einen `style`-Attributknoten, dieser hat aber keinen Attributwert, sodass die Schleife leere Attribute setzt. Erst Opera 8 interpretiert das Beispiel korrekt.

↑↓

getElementsByTagName()

Diese Methode entspricht der **gleichnamigen Methode beim document-Objekt**, bezieht sich jedoch nicht auf das ganze Dokument, sondern nur auf einen bestimmten Elementknoten. Sie liefert einen Array mit allen Elementknoten eines bestimmten Namens zurück, die sich innerhalb des Elements befinden. Erwartet als Parameter den Namen der gesuchten Elemente als String (z.B. `"h1"` oder `"a"`).

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title></head>
<body>
<h2>Erster Bereich</h2>
<p>erster Absatz</p>
<p>zweiter Absatz</p>

<div id="bereich">
<h2>Zweiter Bereich</h2>
<p>dritter Absatz</p>
<p>vierter Absatz</p>
</div>

<h2>Dritter Bereich</h2>
<p>erster Absatz</p>
<p>zweiter Absatz</p>

<script type="text/javascript">
var bereich = document.getElementById("bereich");
var absaetze = bereich.getElementsByTagName("p");
var ueberschriften = bereich.getElementsByTagName("h2");

for (var i = 0; i < ueberschriften.length; i++) {
  ueberschriften[i].style.color = "blue";
}

for (i = 0; i < absaetze.length; i++) {
  absaetze[i].style.color = "red";
}
</script>
</body></html>
```

Erläuterung:

Im Beispiel sind verschiedene Überschriften und Absätze notiert, von denen eine Überschrift und zwei Absätze in einem `div`-Element mit der ID `bereich` liegen. Im Folgenden sollen alle Überschriften und Absätze innerhalb dieses `div`-Elements angesprochen und umformatiert werden.

Zunächst wird über **document.getElementById()** das `div`-Element angesprochen und der zurückgegebene Elementknoten in der Variable `bereich` gespeichert Dadurch können wir später mehrmals darauf zuzugreifen, ohne `getElementById()` mehrmals ausführen zu muss. In den nächsten beiden Zeilen wird die Methode `getElementsByTagName()` des `div`-Elementknotens aufgerufen, einmal mit `"p"` und einmal mit `"h2"`. Diese Aufrufe geben jeweils Arrays mit allen `p`- bzw. `h2`-Elementen innerhalb des `div`-Elements zurück. Diese Arrays werden der Einfachheit halber in den Variablen `absaetze` und `ueberschriften` gespeichert, da später mehrfach auf sie zugegriffen wird.

Zuletzt werden beide Arrays mit **for-Schleifen** durchlaufen. Über `ueberschriften[i]` bzw. `absaetze[i]` werden die einzelnen Elementknoten im Array angesprochen. Mithilfe des **style-Objekts** wird die **Textfarbe** bei der Überschrift auf blau und bei den Absätzen auf rot gesetzt.

↑↓

hasChildNodes()

Ermittelt, ob ein Knoten Kindknoten unter sich hat. Gibt den booleschen Wert `true` zurück, wenn ja, und `false`, wenn nein.

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title></head>
<body>
<p></p>
<p>mit Inhalt</p>
<p></p>
<p>mit Inhalt</p>
<script type="text/javascript">
for (var i = 0; i < document.getElementsByTagName("p").length; i++)
  if (document.getElementsByTagName("p")[i].hasChildNodes())
    document.write((i + 1) + ". p-Element hat einen Inhalt!<br>");
</script>
</body></html>
```

Erläuterung:

Das Beispiel enthält vier Textabsätze. Zwei davon haben einen Zeicheninhalt, zwei andere sind leer. Unterhalb der Textabsätze ist ein JavaScript notiert. Dieses greift in einer for-Schleife der Reihe nach mit `document.getElementsByTagName("p")` auf alle `p`-Elemente des Dokuments zu. Dabei wird mit `if(document.getElementsByTagName("p")[i].hasChildNodes())` abgefragt, ob das jeweils aktuell in der Schleife behandelte `p`-Element Kindknoten hat. Wenn ja, wird mit `document.write()` ins Dokument geschrieben, beim wie vielten Element ein Kindknoten gefunden wurde. Im Beispiel ist das beim zweiten und vierten Element der Fall, da diese beiden Elemente Textknoten enthalten.

insertBefore()

Fügt innerhalb eines Knotens einen Kindknoten vor einem anderen Kindknoten ein.

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title></head>
<body>
<p id="derText">Text <i id="derKursiveText">und mit kursivem Text</i></p>
<script type="text/javascript">
var neuB = document.createElement("b");
var neuBText = document.createTextNode("mit fettem Text ");
neuB.appendChild(neuBText);
document.getElementById("derText").insertBefore(neuB, document.getElementById("derKursiveText"));
</script>
</body></html>
```

Erläuterung:

Das Beispiel enthält einen Textabsatz mit Text und einem Kindelement `<i>...</i>`. Unterhalb des Absatzes ist ein JavaScript-Bereich notiert. Darin wird zunächst mit document.createElement() ein neues Element des Typs `b` für fetten Text erzeugt. Anschließend wird mit document.createTextNode() ein Textknoten erzeugt, der dann mit appendChild() als Inhalt des neu erzeugten `b`-Elements eingefügt wird. Der ganze Komplex aus `b`-Element mit Inhalt ist dann in der Variablen `neuB` gespeichert. Mit `document.getElementById("derText")` greift das Script sodann auf das `p`-Element zu und fügt mit `insertBefore()` das neue Element vor dem anderen Kindelement ein, das die Id `"derKursiveText"` hat. Am Ende lautet der komplette Inhalt des `p`-Elements dann:
`Text <b>mit fettem Text </b><i>und mit kursivem Text</i>`.
Die Methode `insertBefore()` erwartet also zwei Parameter:
1. der neu einzufügende Kindknoten,
2. ein Kindknoten, vor dem der neue Kindknoten eingefügt werden soll.

Wenn Sie einen Knoten *nach* einem bestimmten Kindknoten einfügen wollen, können Sie dazu ebenfalls `insertBefore()` verwenden. Geben Sie dazu als zweiten Parameter den Kindknoten an, der auf den Knoten folgt, nach dem der neue Knoten eingefügt werden soll. Dieser muss keine eigene ID haben, sondern Sie können ihn über nextSibling ausgehend vom vorigen Kindknoten ansprechen.

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title></head>
<body>
<p id="derText">Text <i id="derKursiveText">mit kursivem</i> Text</p>
<script type="text/javascript">
var neuB = document.createElement("b");
var neuBText = document.createTextNode(" und fettem");
neuB.appendChild(neuBText);
document.getElementById("derText").insertBefore(neuB, document.getElementById("derKursiveText").next
</script>
</body></html>
```

Erläuterung:

Das Beispiel funktioniert weitestgehend wie das erste, der Text "und fettem" wird jedoch nach dem `i`-Element eingefügt, sodass das `p`-Element am Ende `Text <i id="derKursiveText">mit kursivem</i><b> und fettem</b> Text` enthält. Als zweiter Parameter für `insertBefore()` wird `document.getElementById("derKursiveText").nextSibling` übergeben. Dies ist der nächstfolgende Knoten nach dem `i`-Element, also der Textknoten mit dem Inhalt " Text". Vor diesem Knoten wird das neu erzeugte `b`-Element eingefügt. Somit wird es nach dem `i`-Element und vor dem darauffolgenden Knoten eingefügt.

Wenn auf einen Knoten keine weiteren Kindknoten folgen, hat die Eigenschaft `nextSibling` den Wert `null`. Dies passt in diesem Fall, denn der zweite Parameter von `insertBefore()` kann auch `null` sein. In diesem Fall funktioniert `insertBefore()` genauso wie ↑ appendChild(): Der neue Knoten wird nach allen anderen Kindknoten, also ans Ende der Liste der Kindknoten eingefügt.

↑↓

insertData()

Fügt Zeichendaten in einem Textknoten ab einer bestimmten Zeichenposition ein.

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title>
<script type="text/javascript">
function italiano () {
  document.getElementsByTagName("h1")[0].firstChild.insertData(7, "duemilauno ");
}
</script></head>
<body>
<h1>2001 =  oder zweitausendeins</h1>
<a href="javascript:italiano()">sag es in italienisch!</a>
</body></html>
```

Erläuterung:

Das Beispiel enthält eine Überschrift erster Ordnung, in der noch irgendetwas fehlt an Text. Unterhalb davon ist ein Verweis notiert, bei dessen Anklicken die Funktion `italiano()` aufgerufen wird, die im Dateikopf notiert ist. Diese Funktion greift mit `document.getElementsByTagName("h1")[0].firstChild` auf den Textknoten der Überschrift zu und fügt mit `insertData()` ab dem 8. Zeichen (Zeichenposition 7, zu zählen begonnen wird bei 0) die Zeichenkette `duemilauno` ein. Der dahinterstehende Text `oder zweitausendeins` wird - wie beim Einfügemodus in der Textverarbeitung - einfach nach hinten geschoben.

Beachten Sie:

Der Internet Explorer 5.x interpretiert diese Methode noch nicht. Im Internet Explorer 6.0 und im Internet Explorer 5.0 Macintosh Edition wird die Methode dagegen unterstützt.

↑↓

removeAttribute()

Löscht aus einem Element ein Attribut.

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title>
<script type="text/javascript">
function wegMitAusrichtung () {
  document.getElementById("rechts").removeAttribute("align");
}
</script></head>
<body>
<p id="rechts" align="right"><a href="javascript:wegMitAusrichtung()">Weg mit der Ausrichtung!</a></
</body></html>
```

Erläuterung:

Das Beispiel enthält einen mit `align="right"` rechtsbündig ausgerichteten Textabsatz, der einen Verweis enthält. Beim Anklicken des Verweises wird die Funktion `wegMitAusrichtung()` aufgerufen, die im Dateikopf notiert ist. Diese Funktion greift mit `document.getElementById("rechts")` auf das `p`-Element zu und löscht mit `removeAttribute("align")` das `align`-Attribut. Der Absatz wird dadurch dynamisch neu ausgerichtet (per Voreinstellung linksbündig).

Beachten Sie:

Der Internet Explorer 5.0 Macintosh Edition interpretiert das Beispiel nicht. Konqueror 3.1 kennt zwar die Methode `removeAttribute()`, der rechtsbündige Absatz wird aber nach Entfernen des Attributs nicht wieder linksbündig.

removeAttributeNode()

Löscht aus einem Element einen Attributknoten.

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title></head>
<script type="text/javascript">
function wegMitAusrichtung () {
  document.getElementById("rechts").removeAttributeNode(document.getElementById("rechts").getAttribu
}
</script>
<body>
<p id="rechts" align="right"><a href="javascript:wegMitAusrichtung()">Weg mit der Ausrichtung!</a></
</body></html>
```

Erläuterung:

Das Beispiel tut das Gleiche wie das Beispiel bei der Beschreibung zu removeAttribute(). Der Unterschied ist nur, dass `removeAttributeNode()` als Parameter das Objekt eines Attributknotens erwartet, während `removeAttribute()` als Parameter den Namen des zu entfernenden Attributs nimmt. Im Beispiel wird `document.getElementById("rechts").getAttributeNode("align")` übergeben, welches das Objekt des Attributknotens des `align`-Attributs des `p`-Elements ist.

Beachten Sie:

Internet Explorer und Konqueror 3.1 interpretieren diese Methode nicht. Opera unterstützt diese Methode bereits in Version 7, hat aber offenbar Probleme, den Absatz sofort linksbündig darzustellen.

removeChild()

Löscht aus einem Element einen Kindknoten.

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title>
<script type="text/javascript">
function wegMitEintrag () {
  var Knoten = document.getElementsByTagName("ol")[0].firstChild;
  verschwunden = document.getElementsByTagName("ol")[0].removeChild(Knoten);
  alert(verschwunden.firstChild.nodeValue)
}
</script></head>
<body>
<ol start="1" type="I"><li>Element 1</li><li>Element 2</li><li>Element 3</li></ol>
<p><a href="javascript:wegMitEintrag()">Lösche das erste Element</a></p>
</body></html>
```

Erläuterung:

Das Beispiel enthält eine nummerierte Liste. Unterhalb davon ist ein Verweis notiert. Beim Anklicken des Verweises wird die **Funktion** `wegMitEintrag()` aufgerufen, die im Dateikopf notiert ist. In dieser Funktion wird auf den ersten Kindknoten mit `document.getElementsByTagName("ol")[0].firstChild` zugegriffen. Der Wert wird in der Variablen `Knoten` gespeichert. Mit `removeChild(Knoten)` wird der Listenpunkt gelöscht. Dabei wird der Methode als Parameter der zu löschende Knoten übergeben. Die Methode `removeChild()` gibt als Rückgabewert das gelöschte Element zurück. Dieser Wert ist im Beispiel in der globalen Variablen `verschwunden` gespeichert. Über diese Variable können Sie weiterhin auf die Eigenschaften des Knotens zugreifen. So wird im Beispiel noch einmal der Inhalt des ersten Kindknotens des gelöschten Elements ausgegeben.

Berücksichtigen Sie beim Nachvollziehen dieses Beispieles die ↑ **Besonderheit** des Internet Explorers unter Windows im Umgang mit Kindknoten.

↑↓

replaceChild()

Ersetzt aus Sicht eines Knotens einen seiner Kindknoten durch einen anderen.

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title>
<script type="text/javascript">
function andereAntwort () {
  var Textknoten = document.createTextNode("Du darfst!");
  document.getElementById("z2").replaceChild(Textknoten, document.getElementById("z2").firstChild);
}
</script></head>
<body>
<table border="1" bgcolor="#FFFFC0" width="30%">
<tr><td id="z1">Darf ich?</td><td id="z2">keine Ahnung!</td></tr>
</table>
<p><a href="javascript:andereAntwort()">andere Antwort!</a></p>
</body></html>
```

Erläuterung:

Das Beispiel enthält eine Tabelle mit zwei Zellen. Unterhalb der Tabelle ist ein Verweis notiert. Beim Anklicken des Verweises wird die **Funktion** `andereAntwort()` aufgerufen, die im Dateikopf notiert ist. Diese Funktion erzeugt zunächst mit **document.createTextNode()** einen neuen Textknoten für die Antwort. Die Antwort wird in Form eines Knotenobjekts benötigt, weil die Methode `replaceChild()` sowohl den neuen Knoten als auch den zu ersetzenden in Form eines Knotenobjekts erwartet. Im Beispiel wird mit `document.getElementById("z2")` auf die zweite Tabellenzelle zugegriffen, um deren Inhalt, also den Textknoten, der ihren Kindknoten darstellt, durch den neu erzeugten Textknoten zu ersetzen. Dazu wird `replaceChild()` angewendet. Als erster Parameter wird der neue Knoten übergeben, als zweiter der zu ersetzende. Der neue Knoten ist im Beispiel in der Variablen `Textknoten` gespeichert. Auf den zu ersetzenden Knoten wird mit `document.getElementById("z2").firstChild` zugegriffen.

↑↓

replaceData()

Ersetzt Zeichendaten im Textinhalt eines Elements (Textknoten).

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title>
<script type="text/javascript">
function andereAntwort () {
  var Text = "Du darfst!";
  var rd_Start = 0;
  var rd_Laenge = document.getElementById("z2").firstChild.nodeValue.length;
  document.getElementById("z2").firstChild.replaceData(rd_Start, rd_Laenge, Text);
}
</script></head>
<body>
<table border="1" bgcolor="#FFFFC0" width="30%">
<tr><td id="z1">Darf ich?</td><td id="z2">keine Ahnung!</td></tr>
</table>
<p><a href="javascript:andereAntwort()">andere Antwort!</a></p>
</body></html>
```

Erläuterung:

Das Beispiel tut das Gleiche wie das Beispiel bei der Beschreibung zu ↑ replaceChild(). Der Unterschied ist nur, dass mit `replaceData()` auf Textebene gearbeitet wird, und dass diese Methode auch gezieltes Ersetzen ab einer bestimmten Zeichenposition und für eine bestimmte Länge erlaubt. Als Parameter erwartet `replaceData()` zuerst die Startposition in der Zeichenkette, ab der ersetzt werden soll. Im Beispiel wird dazu die Variable `rd_Start` übergeben, der zuvor `0` zugewiesen wurde. Damit wird ab dem ersten Zeichen ersetzt. Als zweiter Parameter wird übergeben, wie viele Zeichen ersetzt werden sollen. Im Beispiel wird die Variable `rd_Laenge` übergeben. Dieser wurde zuvor mit `document.getElementById("z2").firstChild.nodeValue.length` die Zeichenanzahl des gesamten Textknotens der zweiten Tabellenzelle zugewiesen. Im Beispiel wird auf diese Weise der komplette Inhalt der zweiten Tabellenzelle ersetzt. Mit einer Angabe wie `0,1` würden Sie beispielsweise nur das erste Zeichen ersetzen.
Als dritten Parameter erwartet `replaceData()` eine Zeichenkette, mit der die zuvor im Textknoten markierte Zeichenkette ersetzt werden soll. Im Beispiel wird die Variable `Text` übergeben, der zuvor ein Wert zugewiesen wurde.

Beachten Sie:

Der Internet Explorer 5.x interpretiert diese Methode noch nicht. Im Internet Explorer 6.0 und im Internet Explorer 5.0 Macintosh Edition wird die Methode dagegen unterstützt.

↑↓

setAttribute()

Setzt in einem Element einen Attributwert neu. Ist das Attribut bereits vorhanden, wird sein alter Wert durch den neuen ersetzt. Ist es noch nicht vorhanden, wird es neu angelegt und mit dem neuen Wert belegt.

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title>
<script type="text/javascript">
function setzen () {
  document.body.setAttribute("bgColor", document.Formular.bgcolor.value);
  document.body.setAttribute("text", document.Formular.text.value);
}
</script></head>
<body bgcolor="#FFFFCC" text="#E00000">
<form name="Formular" action=""><pre>
Hintergrundfarbe:      <input type="text" name="bgcolor">
Textfarbe:             <input type="text" name="text">
Werte:                 <input type="button" value="setzen" onclick="setzen()">
</pre></form>
</body></html>
```

Erläuterung:

Das Beispiel enthält ein Formular mit zwei Eingabefeldern, in denen der Anwender neue Werte für die beiden Attribute `bgcolor` und `text` des `body`-Elements eingeben kann. Beim Anklicken des Buttons unterhalb davon wird die Funktion `setzen()` aufgerufen, die im Dateikopf notiert ist. Diese Funktion greift mit `document.body` auf das `body`-Element zu und weist ihm mit `setAttribute()` die neuen Attribute zu. Als erster Parameter wird der Name des zu erzeugenden oder zu ersetzenden Attributs übergeben, als zweiter Parameter der gewünschte Wert. Im Beispiel wird als zweiter Parameter jeweils der Wert aus den Formulareingabefeldern übergeben.

Beachten Sie:

Im Internet Explorer 5.0 Macintosh Edition ist das Beispiel zwar nachvollziehbar, jedoch mit sehr seltsamen Ergebnissen.

`setAttribute()` ist eine Methode des allgemeinen DOM (Kern-DOM). Wenn Sie auf Attribute von HTML-Elementen zugreifen möchten, brauchen Sie `setAttribute()` strenggenommen nicht, denn das HTML-spezifische DOM definiert alle Attribute als Eigenschaften von Elementobjekten, auf die Sie direkt zugreifen können.
Das Setzen von Attributen über `setAttribute()` ist meist unnötig umständlich. Anstatt `document.body.setAttribute("bgColor", document.Formular.bgcolor.value);` können Sie unter Verwendung des HTML-spezifischen DOM ebenso `document.body.bgColor = document.Formular.bgcolor.value;` notieren. In der Objektreferenz zu den HTML-Elementobjekten werden alle Attribute aufgelistet, die Sie auf diese Weise ansprechen können.

Davon abgesehen hat `setAttribute()` eine noch viel größere Tücke: Im Internet Explorer hat die Methode einen Fehler. Sie müssen bei einigen Attributen eine spezielle Schreibweise des Attributnamens beachten, damit der Internet Explorer das Attribut tatsächlich setzt. Im Beispiel wurde absichtlich `setAttribute("bgColor", ...)` und nicht etwa **`bgcolor`** notiert. Denn der MSIE verlangt die Schreibweise des Attributnamens, die im HTML-spezifischen DOM für den Direktzugriff definiert wurde - diese lautet in Beispielfall `bgColor`. Es kommt also auf die korrekte Groß- und Kleinschreibung an.

Im HTML-spezifischen DOM werden Attribute in der Regel klein geschrieben. Es gibt aber neben `bgColor` eine Reihe von weiterer Attribute, beim denen der Direktzugriff über eine besondere Schreibweise erfolgen muss. Der Unterschied ist zumeist ein Mittelinitial und der Wegfall des Bindestrichs, den einige Attribute enthalten. Es handelt sich um folgende Attribute:

`acceptCharset` (von `accept-charset`), `accessKey`, `aLink`, `bgColor`, `cellPadding`, `cellSpacing`, `chOff` (von `charoff`), `className` (von `class`), `codeBase`, `codeType`, `colSpan`, `dateTime`, `frameBorder`, `htmlFor` (von `for`), `httpEquiv` (von `http-equiv`), `isMap`, `longDesc`, `marginHeight`, `marginWidth`, `noHref`, `noResize`, `noWrap`, `readOnly`, `rowSpan`, `tabIndex`, `useMap`, `vAlign` und `vLink`.

Wenn Sie diese Attribute mit `setAttribute()` setzen möchten, müssen Sie die angegebene Schreibweise verwenden, damit der Internet Explorer korrekt arbeitet. Allerdings akzeptieren andere Browser diese speziellen Schreibweisen nicht. Deshalb ist es einfacher, den Internet-Explorer-Fehler zu umgehen, indem Sie auf `setAttribute()` möglichst verzichten und den Direktzugriff nutzen.

setAttributeNode()

Fügt in ein Element einen neuen Attributknoten ein. Ist der Attributknoten bereits vorhanden, wird der alte Knoten durch den neuen ersetzt. Ist er noch nicht vorhanden, wird er neu angelegt.

Beispiel:

Anzeigebeispiel: So sieht's aus

```
<html><head><title>Test</title></head>
<body>
<h1>Element ohne Eigenschaften?</h1>
<script type="text/javascript">
var Ausrichtung = document.createAttribute("align");
Ausrichtung.nodeValue = "center";
document.getElementsByTagName("h1")[0].setAttributeNode(Ausrichtung);
</script>
</body></html>
```

Erläuterung:

Das Beispiel enthält eine Überschrift erster Ordnung - ohne Attribute. Unterhalb davon ist ein JavaScript notiert. Dort wird zunächst mit document.createAttribute() ein neuer Attriutknoten für ein Attribut namens `align` erzeugt. Der Knoten wird in der Variablen `Ausrichtung` gespeichert. Durch `Ausrichtung.nodeValue` lässt sich dem erzeugten Knoten dann ein Wert zuweisen. Mit `document.getElementsByTagName("h1")[0]` greift das Script schließlich auf das Überschriftenelement zu und weist ihm mit `setAttributeNode(Ausrichtung)` den zuvor erzeugten und in `Ausrichtung` gespeicherten Attributknoten zu.

Beachten Sie:

Der Internet Explorer 5.x interpretiert diese Methode noch nicht. Im Internet Explorer 6.0 und im Internet Explorer 5.0 Macintosh Edition wird die Methode dagegen unterstützt.

all
HTML-Elementobjekte