HTML-DOM beim Internet Explorer

HTML-DOM BEIM INTERNET EXPLORER 1

ausgewählte Objekte und deren Eigenschaften und Methoden des HTML-DOM

Die Objekte window, window.document, window.document.body und window.event sowie die Collection window.document.all werden wegen Umfang hier nicht beschrieben. Diese Objekte und Collectionen sind allerdings primär für die HTML-DOM-Verwaltung. Ansonsten sind DOM-Eigenschaften und Methoden (Kinder von document und document.body) im jeweiligen Objekt implementiert und anhand dessen Beschreibung zu ermitteln (inklusive Events). Die Eventsteuerung erfolgt im IE zentralisiert über das Objekt window, wobei je nach HTML-Objekt eine Eventdurchreichung zum window-Objekt erlaubt ist oder nicht (siehe Beschreibung Objekt event).

Attribut Objekt

Eigenschaft	.specified	prüfen ob ein Attribut im Element einen Wert hat, egal ob Element mit oder ohne HTML-Tag-Anweisung erzeugt wurde
Eigenschaft	.nodeValue	Knotenwert (Wert des Kindes, Node, Elementes) nur für Text- und Attribut-Elemente nicht für Element-Knoten (Knotentyp 1)
Eigenschaft	.value	Wert eines Attributes eines Objektes
Methode	.clearAttributes()	alle HTML-Attribute eines Objektes entfernen, außer ID, STYLE und per Script definierte Attribute
Methode	.createAttribute()	ein Attribut im Dokument erzeugen und Referenz liefern Achtung: Der Browser unterscheidet zwischen HTML-erzeugte oder mit dieser Methode erzeugten Attribute !
Methode	.getAttribute()	Wert eines Attributes, das durch HTML-Anweisungen erzeugt wurde, liefern
Methode	.getAttributeNode()	Referenz auf Eigenschaft des Attribute-Objektes liefern, also Zeiger auf attribute.name property. Eigenschaft kann mit HTML-Anweisung erzeugt worden sein, muss aber nicht Eigenschaft ist selbst ein Knoten in der Attribute-Objekt-Hierarchie
Methode	.mergeAttributes()	alle Attribute eines Elementes in ein anderes Element kopieren und im Ziel mit den vorhandenen Attributen mischen Attribute sind: HTML Events Styles ab IE 5.01 auch ID und NAME auch ab NS 6.x
Methode	.removeAttribute()	entfernen von Attribut, das durch HTML-Anweisungen erzeugt wurde
Methode	.removeAttributeNode()	entfernen von Attribut, egal ob es mit oder ohne HTML-Anweisung erzeugt wurde, und Referenz auf das entfernte Attribut liefern
Methode	.setAttribute()	Wert von vorhandenem Attribut neu definieren wenn Attribut nicht vorhanden ist, so es erzeugen und mit Wert belegen
Methode	.setAttributeNode()	Attribut einem Knoten zuweisen und Referenz liefern

Comment Objekt

Methode	.createComment()	Comment-Objekt (Kommentar-Objekt) im Dokument erzeugen und Referenz liefern

Document Objekt

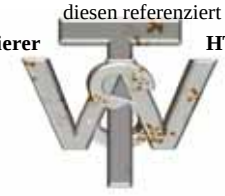
Eigenschaft	.documentElement	Referenz auf Wurzelknoten (root node) des Dokumentes liefern
Eigenschaft	.ownerDocument	Referenz auf das document-Objekt zu dem der Koten gehört, also in dem der Knoten erzeugt wurde
Eigenschaft	.XSLDocument	Referenz auf den obersten Knoten des XSL-Dokumentes (Style-Sheet-Dokument)
Methode	.createAttribute()	ein Attribut im Dokument erzeugen und Referenz liefern Achtung: Der Browser unterscheidet zwischen HTML-erzeugte oder mit dieser Methode erzeugten Attribute !
Methode	.createComment()	Comment-Objekt (Kommentar-Objekt) im Dokument erzeugen und Referenz liefern
Methode	.createElement()	HTML-Objekt (Tags) im Dokument erzeugen und Referenz liefern Achtung: Erzeugtes Objekt muss in DOM noch per Methode .insertBefore() bzw..appenChild() eingereiht werden. Hinweis: Attribute mit der Methode .createAttribute() erzeugen auch ab NS 6.x
Methode	.createStyleSheet()	Style-Sheet-Objekt im Dokument erzeugen und Referenz liefern
Methode	.createTextNode()	Textelement im Dokument erzeugen und Referenz liefern nur Plain-Text, also keine HTML-Tags auch ab NS 6.x
Methode	.getElementById()	Referenz auf das im Dokument zuerst gefundene Objekt laut ID liefern wenn mehrere Elemente mit ein und demselben ID, so das ERSTE Element von diesen referenzieren auch ab NS 6.x
Methode	.getElementsByName()	Referenz auf ein Feld (Collection) aller im Dokument befindlichen Objekte mit gemeinsamen NAME liefern auch ab NS 6.x

DOM als symbolisches Objekt

Methode	.normalize()	Normalisierung des DOM zur Erreichung einer konsistenten Struktur Achtung: CDATA-Sections dürfen nicht enthalten sein, da diese immer Inkonsistenz erzeugen

Element (Objekt / Knoten / Node) im DOM

Eigenschaft	.documentElement	Referenz auf Wurzelknoten (root node) des Dokumentes liefern
Eigenschaft	.nodeType	Knotentyp laut attributes Collection
Eigenschaft	.nodeValue	Knotenwert (Wert des Kindes, Node, Elementes) nur für Text- und Attribut-Elemente nicht für Element-Knoten (Knotentyp 1)
Eigenschaft	.ownerDocument	Referenz auf das document-Objekt zu dem der Koten gehört, also in dem der Knoten erzeugt wurde
Eigenschaft	.readyState	aktueller Status des Objektes beim Füllen des Objektes mit Daten
Eigenschaft	.scopeName	Namensraum laut XMLNS-Attribut
Eigenschaft	.tagUrn	Uniform Ressource Name (URN) laut Namensraum laut XMLNS-Attribut
Eigenschaft	.uniqueID	durch den Browser automatisch generiertes ID des Objektes Browser generiert zu verschiedenen Zeitpunkten auch verschiedene ID, wenn Objekt mehrmals geladen wurde
Eigenschaft	.XMLDocument	Referenz auf XML-Dokument (XML-DOM)
Methode	.addBehavior()	DHTML-Verhaltenseigenschaft einem Element hinzufügen ab IE 5.x bis unter IE 5.5
Methode	.cloneNode()	Objekt klonen und Referenz des Klone liefern
Methode	.getElementById()	Referenz auf das im Dokument zuerst gefundene Objekt laut ID liefern wenn mehrere Elemente mit ein und demselben ID, so das ERSTE Element von diesen referenziert



		auch ab NS 6.x
Methode	.getElementsByName()	Referenz auf ein Feld (Collection) aller im Dokument befindlichen Objekte mit gemeinsamen NAME liefern auch ab NS 6.x
Methode	.getElementsByTagName()	Referenz auf ein Feld (Collection) aller im Objekt befindlichen Kinder-Objekte mit gemeinsamen Tagnamen liefern, inklusive aller Kinder und Unterkinder etc. auch ab NS 6.x
Methode	**.implementation**.hasFeature()	Objektzugehörigkeit zum DOM (Document Object Model-Standard) (HTML-DOM bzw. XML-DOM)
Methode	.insertAdjacentElement()	Element in eine Element einfügen und Referenz liefern wenn Element bereits eingefügt vorhanden, so wird dieses nur verschoben nur nach dem kompletten Laden des Dokumentes
Methode	.removeBehavior()	per Methode .addBehavior() einem Element hinzugefügte Verhaltenseigenschaft entfernen (stets VOR dem Entfernen des Elementes mit der zugeordneten Eigenschaft aus der Dokument-Hierarchie)
Methode	.removeNode()	Knoten entfernen aus DOM und Referenz auf den entfernten Knoten liefern Sichtbarkeit erst wenn Ende-Tag geparst wurde
Methode	.replaceNode()	Objekt durch anderes Objekt komplett ersetzen und Referenz auf das komplett ersetzte Objekt liefern sichtbar erst mit parsen des Endetags
Methode	.swapNode()	Positionen von 2 Knoten in der Dokumenthierarchie tauschen nur sichtbar wenn Endetag geparst

Eltern in der Vererbung (Parent)

Eigenschaft	.canHaveChildren	prüfen ob Kind möglich ist, also ob Objekt Parent sein kann
Eigenschaft	.documentElement	Referenz auf Wurzelknoten (root node) des Dokumentes liefern
Eigenschaft	.parentNode	Referenz auf Elternknoten innerhalb der DOM-Hierarchie
Eigenschaft	.parentElement	Referenz auf das Elternobjekt, also nicht Elternknoten innerhalb DOM-Hierarchie
Eigenschaft	.parentTextEdit	Textbereich des Elternobjektes referenzieren
Methode	.applyElement()	Elementeigenschaft Kind oder Eltern festlegen und Referenz liefern Element kann selbst Kinder haben Element erst sichtbar, wenn Endetag des Elementes geparst wurde Achtung: Wenn Element per Methode .createElement() erzeugt wurde, aber nicht im Dokumentenbaum eingebunden ist, so wird die Eigenschaft .innerHTML gelöscht
Methode	.hasChildNodes()	prüfen auf Existenz von Kinder(n) von HTML-Elemente oder Textknoten

HTML-Element als Knoten (Node)

Eigenschaft	.canHaveHTML	prüfen ob Objekt HTML-Tags enthalten darf
Eigenschaft	.documentElement	Referenz auf Wurzelknoten (root node) des Dokumentes liefern
Eigenschaft	.tagName	Tag-Bezeichner des Objektes
Methode	.clearAttributes()	alle HTML-Attribute eines Objektes entfernen, außer ID, STYLE und per Script definierte Attribute
Methode	.createElement()	HTML-Objekt (Tags) im Dokument erzeugen und Referenz liefern Achtung: Erzeugtes Objekt muss in DOM noch per Methode .insertBefore() bzw..appenChild() eingereiht werden. Hinweis: Attribute mit der Methode .createAttribute() erzeugen auch ab NS 6.x
Methode	.expression()	Wert einer Style-Eigenschaft per STYLE-Attribut-Wert in HTML als Ausdruck definieren für Berechnung per Methode .getExpression()

		Ausdruck nur als Script kodierbar
Methode	.getAttribute()	Wert eines Attributes, das durch HTML-Anweisungen erzeugt wurde, liefern
Methode	.hasChildNodes()	prüfen auf Existenz von Kinder(n) von HTML-Elemente oder Textknoten
Methode	.insertAdjacentHTML()	HTML-Code und/oder Script-Code in ein Element einfügen nur nach dem kompletten Laden des Dokumentes HTML- und Script-Code müssen syntaktisch korrekt sein wenn nicht, so Einfügen nicht ausgeführt eingefügter Code wird sofort geparst und ausgeführt bei Script-Code: <SCRIPT DEFER .....> muss kodiert werden
Methode	.mergeAttributes()	alle Attribute eines Elementes in ein anderes Element kopieren und im Ziel mit den vorhandenen Attributen mischen Attribute sind: HTML Events Styles ab IE 5.01 auch ID und NAME auch ab NS 6.x
Methode	.removeAttribute()	entfernen von Attribut, das durch HTML-Anweisungen erzeugt wurde

HTML und Script in einem Element

Methode	.insertAdjacentHTML()	HTML-Code und/oder Script-Code in ein Element einfügen nur nach dem kompletten Laden des Dokumentes HTML- und Script-Code müssen syntaktisch korrekt sein wenn nicht, so Einfügen nicht ausgeführt eingefügter Code wird sofort geparst und ausgeführt bei Script-Code: <SCRIPT DEFER .....> muss kodiert werden

Kind in der Vererbung (Child)

Eigenschaft	.canHaveChildren	prüfen ob Kind möglich ist, also ob Objekt Parent sein kann
Eigenschaft	.documentElement	Referenz auf Wurzelknoten (root node) des Dokumentes liefern
Eigenschaft	.firstChild	Zeiger auf das ERSTE Kind laut childNodes-Collection eines Objektes
Eigenschaft	.lastChild	Zeiger auf das LETZTE Kind laut childNodes-Collection eines Objektes
Eigenschaft	.nextSibling()	Zeiger auf das NACHFOLGENDE Kind laut childNodes-Collection eines Objektes
Eigenschaft	.nodeName	String als Name des Kindes liefern, also TAG-Bezeichner, Attribut-Name, #text für Anker
Eigenschaft	.previousSibling	Zeiger auf das VORHERGEHENDE Kind
Methode	.appendChild()	Knoten als Kind an das DOM anhängen und Zeiger liefern Zeiger wird am Ende der Collection childNodes angehangen
Methode	.applyElement()	Elementeigenschaft Kind oder Eltern festlegen und Referenz liefern Element kann selbst Kinder haben Element erst sichtbar, wenn Endetag des Elementes geparst wurde Achtung: Wenn Element per Methode .createElement() erzeugt wurde, aber nicht im Dokumentenbaum eingebunden ist, so wird die Eigenschaft .innerHTML gelöscht
Methode	.contains()	prüfen ob Element innerhalb eines Elementes liegt, also ob das Element Eltern hat und somit ein Kind ist
Methode	.insertBefore()	Objekt als Kindknoten VOR dem einem anderen Kind-Objekt einfügen und Zeiger liefern einzufügendes Objekt muss mit Methode createElement() erzeugt worden sein Achtung: NICHT anwenden für einfügen VON bzw. VOR obersten Kindknoten Sichtbarkeit erst wenn Ende-Tag geparst wurde
Methode	.removeChild()	Kind entfernen aus DOM und Referenz auf das entfernte Kind liefern Sichtbarkeit erst wenn Ende-Tag geparst wurde

Methode	.replaceChild()	Kind ersetzen durch Objekt ersetzende Objekt muss per Methode .createElement() erzeugt worden sein Sichtbarkeit erst wenn Ende-Tag geparst wurde

Style Objekt

Methode	.createStyleSheet()	Style-Sheet-Objekt im Dokument erzeugen und Referenz liefern
Methode	.expression()	Wert einer Style-Eigenschaft per STYLE-Attribut-Wert in HTML als Ausdruck definieren für Berechnung per Methode .getExpression() Ausdruck nur als Script kodierbar
Methode	.getExpression()	Wert einer Style-Eigenschaft anhand des Ausdruckes berechnen und liefern Style-Eigenschaft ist per Methoden expression() oder setExpression() zu definieren
Methode	.mergeAttributes()	alle Attribute eines Elementes in ein anderes Element kopieren und im Ziel mit den vorhandenen Attributen mischen Attribute sind: HTML Events Styles ab IE 5.01 auch ID und NAME auch ab NS 6.x
Methode	.removeExpression()	Ausdruck für die Berechnung des Wert einer Style-Eigenschaft als Objektreferenz der Form objekt.style.eigenschaft..... entfernen
Methode	.setExpression()	Wert einer Style-Eigenschaft als Objektreferenz der Form objekt.style.eigenschaft..... anhand eines Ausdruck definieren für Berechnung per Methode getExpression() Ausdruck nur als Script kodierbar

Text Objekt

Eigenschaft	.nodeValue	Knotenwert (Wert des Kindes, Node, Elementes) nur für Text- und Attribut-Elemente nicht für Element-Knoten (Knotentyp 1)
Methode	.createTextNode()	Textelement im Dokument erzeugen und Referenz liefern nur Plain-Text, also keine HTML-Tags auch ab NS 6.x
Methode	.getAdjacentText()	Text eines Objektes liefern, wobei Textlage im Objekt definiert werden kann
Methode	.hasChildNodes()	prüfen auf Existenz von Kinder(n) von HTML-Elemente oder Textknoten
Methode	.insertAdjacentText()	Plain-Text (ohne HTML und Script) in ein Element einfügen nur nach dem kompletten Laden des Dokumentes
Methode	.replaceAdjacentText()	Plain-Text (ohne HTML und Script) eines Elementes durch anderen Text ersetzen und Referenz auf den zu ersetzenden Text liefern

TextNode

Dieses Objekt ist ein symbolisches Objekt und
verwaltet Plain-Textdaten im DOM (keine HTML-Daten), also **Textknoten**
kann von Objekten instanziert werden, die über nachfolgend beschriebene Methoden verfügen z.B. über .createTextNode()
ist Analogon zur Stringverabeitung
ist Basisobjekt für TextRange Objekt (siehe window.document.TextRange)

ab IE 6.x

Erzeugung:

durch Browser

Syntax:

object.methode()

object laut ID-Attribut

Textdaten erzeugen:

.createTextNode()	Plain-Textelement im Dokument erzeugen und Referenz liefern Plaintext kann keine keine HTML-Tags enthalten

DOM wird geändert, da Dokument erweitert wird
Syntax:
[var Zeiger =] document.createTextNode([Text])

Text	String	Text als Inhalt des Elementes
Zeiger	Referenz auf Objekt	

Beispiel 1:

```
<SCRIPT>
    function TextElementAendern()
    {
        var ZeigerAufTextElement = document.createTextNode("Neuer Text");
        var ZeigerAufSpanInhalt = ID_Span.childNodes(0);
        ZeigerAufSpanInhalt.replaceNode(ZeigerAufTextElement);
    }
</SCRIPT>
<SPAN ID = "ID_Span" onclick=" TextElementAendern()">
    Original Text
</SPAN>
```

Beispiel 2: var TextKnoten = document.createTextNode("Test 1");

Textdaten ersetzen:

.replaceData() Teilkette in einem Objekt ersetzen
Syntax:
object.replaceData(Offset, Anzahl, Kette)

Offset	Integer	Startposition ab der ersetzt werden soll ab 0
Anzahl	Integer	Anzahl der ersetzenden Zeichen ab 1 wenn Anzahl > object.length, so am Ende abgeschnitten
Kette	Zeichenkette, die ersetzt	

liefert nichts

Beispiel:

```
var TextKnoten = document.createTextNode("Test 1");
TextKnoten.replaceData(5, 1, "2 ");          // ergibt "Test 2"
```

Textdaten löschen:

.deleteData() Teilkette aus einem Objekt entfernen
Syntax:
object.deleteData(Offset, Anzahl)

Offset	Integer	Startposition der zu löschenden Teilkette ab 0
Anzahl	Integer	Anzahl der zu löschenden Zeichen ab 1 wenn Anzahl > object.length, so kein Fehler

liefert nichts

Beispiel:

```
var TextKnoten = document.createTextNode("Test 1");
TextKnoten.deleteData(4, 2);          // ergibt "Test"
```

Textdaten einfügen:

.insertData() Teilkette in ein Objekt einfügen
Syntax:
object.insertData(Offset, Kette)

Offset	Integer	Startposition ab der eingefügt werden soll ab 0
Kette	Zeichenkette	

liefert nichts

Beispiel:

```
var TextKnoten = document.createTextNode("Test 1");
TextKnoten.insertData(4, "reihe");          // ergibt "Testreihe 1"
```

Textdaten anhängen:

.appendData() String an das Ende des Objektes anhängen

Syntax:

object.appendData(Kette)

Kette Zeichenkette

liefert nichts

Beispiel:

```
var TextKnoten = document.createTextNode("Test 1");
TextKnoten.appendData("0");                    // ergibt "Test 10"
```

Eigenschaften zur Verwaltung des HTML-DOM im Internet Explorer

Nachfolgende Eigenschaften sind nicht in allen Objekten implementiert (siehe einzelne Objekte).

.canHaveChildren

prüfen ob Kind möglich ist, also ob Objekt Parent sein kann

Beispiel:

```
<SCRIPT>
	function KindHinzufuegen()
	{
		var ZeigerAufSPANObjekt = document.createElement("SPAN");

		var ZeigerAufTextObjekt = document.createTextNode("Test");

		ZeigerAufSPANObjekt.appendChild(ZeigerAufTextObjekt);

		for(var Index=0; Index <document.all.length; Index ++)
		{
			if(document.all[Index].canHaveChildren==true)
			{
			  document.all[Index].appendChild(ZeigerAufSPANObjekt);
			  break;
			}
		}
	}
</SCRIPT>
<INPUT TYPE=button VALUE="Kind hinzufügen " onclick="KindHinzufuegen()">
<DIV>
	Test<BR>
</DIV>
```

.canHaveHML

prüfen ob Objekt HTML-Tags enthalten darf

Beispiel:

```
<HTML>
<HEAD>
<SCRIPT>
	function Antwort(BooleanWert)
	{ BooleanWert ? alert("Ja") : alert("Nein");}
</SCRIPT>
</HEAD>
<BODY>
	<P>
		INPUT:
		<INPUT type="text" ID="ID_Input" VALUE="Test" >
	</P>

	<P>
		SPAN:
		<SPAN ID="ID_Span">Test</SPAN>
	</P>

	<BUTTON onclick="Antwort(ID_Input.canHaveHTML);">
		Kann INPUT-Element  HTML besitzen ?
	</BUTTON>

	<BUTTON onclick="Antwort(ID_Span.canHaveHTML);">
		Kann SPAN-Element HTML besitzen ?
	</BUTTON>
</BODY>
</HTML>
```

.documentElement

Referenz auf Wurzelknoten (root node) des Dokumentes liefern

Beispiel:

```
<SCRIPT>
        function fnGetHTML()
        {ID_Textarea.value= document.documentElement.innerHTML;}
</SCRIPT>
<TEXTAREA ID="ID_Textarea" COLS = 50 ROWS = 10>
</TEXTAREA>
```

.firstChild

Zeiger auf das ERSTE Kind laut childNodes-Collection eines Objektes

Beispiel:

```
<SCRIPT>
        var ZeigerAufErstesKind = Liste.firstChild; // liefert Referenz auf  Listenelement 1
</SCRIPT>
<BODY>
        <UL ID = "Liste">
                <LI> Listenelement 1
                <LI> Listenelement 2
                <LI> Listenelement 3
        </UL>
<BODY>
```

.lastChild

Zeiger auf das LETZTE Kind laut childNodes Collection eines Objektes

Beispiel:

```
<SCRIPT>
        var ZeigerAufLetztesKind = Liste.lastChild; // liefert Referenz auf  Listenelement 3
</SCRIPT>
<BODY>
        <UL ID = "Liste">
                <LI> Listenelement 1
                <LI> Listenelement 2
                <LI> Listenelement 3
        </UL>
<BODY>
```

.nextSibling

Zeiger auf das NACHFOLGENDE Kind laut childNodes Collection eines Objektes

Beispiel:

```
<SCRIPT>
        var ErstesKind_Index = 0;       // Index ab 0
        var ZeigerAufNachfolgenKind = Liste. childNodes(ErstesKind_Index).nextSibling;
                                                // liefert Referenz auf  Listenelement 2
</SCRIPT>
<BODY>
        <UL ID = "Liste">
                <LI> Listenelement 1
                <LI> Listenelement 2
                <LI> Listenelement 3
        </UL>
<BODY>
```

.nodeName

String als Name des Kindes (Knoten, Node, Element) also TAG-Bezeichner, Attribut-Name; #text für Anker

Beispiel:

```
<SCRIPT>
        var ErstesKind_Index = 0;       // Index ab 0
        var ErstesKind_Name = Liste.childNodes(ErstesKind_Index).nodeName;
        alert(ErstesKind_Name);         // liefert den Tagnamen 'LI' von Listenelement 1
                                        //  Hinweis: Listenelement 1 ist der Wert des Kindes
</SCRIPT>
```


```
<BODY>
    <UL ID = "Liste">
        <LI> Listenelement 1
        <LI> Listenelement 2
        <LI> Listenelement 3
    </UL>
<BODY>
```

.nodeType

Knotentyp laut attributes Collection

1	für Element-Knoten
2	für Attribut-Knoten
3	für Textknoten
4	für CDATA-Abschnitt
5	für Entity-Referenz
6	für Entity-Knoten
7	für Processing Instruction
8	für Kommentar-Knoten
9	für das Dokument
10	für den Dokumenttyp
11	für ein Dokument-Fragment
12	für Notation
null	wenn Knoten nicht vorhanden (null-Zeiger)

Beispiel 1:

```
var KnotenTypVonBODY = document.body.nodeType;
```

Beispiel 2:

```
var ZeigerAuf_B_Tag = document.createElement("B");          // B-Tag erzeugen
document.body.insertBefore(ZeigerAuf_B_Tag);
                                    // B-Tag in den BODY-Teil des Dokument einfügen,
                                    //          also in die Baumstruktur
var KnotenTypDes_B_Tag = ZeigerAuf_B_Tag.nodeType;
```

.nodeValue

Knotenwert (Wert des Kindes, Node, Elementes)

nur für Text- und Attribut-Elemente
nicht für Element-Knoten (Knotentyp 1)

Beispiel:

```
<SCRIPT>
    function KnotenWertAendern(  ZeigerAufListe,
                                 IndexVonListenElement,  // immer ab 0
                                 Zeichenkette            // Listenlement muss Text sein
                               )
    {
        var ReturnWert=false;          // Annahme: Änderung schlägt fehl

        // prüfen auf UL-Tag
        if (ZeigerAufListe.nodeName == "UL")
        {
            // Anzahl der Listenelemente holen
            var AnzahlListenelemente= ZeigerAufListe.childNodes.length;
                                                                    // immer ab 1
            //  und Anzahl und Index prüfen
            if (    (AnzahlListenelemente > 0)          // immer ab 1
                 && (IndexVonListenElement >= 0)        // immer ab 0
                 && (IndexVonListenElement < AnzahlListenelemente)
                                                        // Index ist zulässig zur Anzahl
               )
            {
                // Zeiger auf das Listenelement laut Index holen
                var ZeigerAufListenElement =
                        ZeigerAufListe.childNodes(IndexVonListenElement);

                // existiert das Listenelement ?
                if (ZeigerAufListenElement)
                {
                    // ZeigerAufListenElement ist nicht null

                    // Listenelement ist Textelement ?
```


```
                                                    if (ZeigerAufListenElement.nodeType == 3)
                                                    {
                                                            ZeigerAufListenElement.nodeValue =
                                                                                    Zeichenkette;
                                                            ReturnWert =true;
                                                    }
                                            }
                                    }
                            }

                            return ReturnWert;
                    }
            </SCRIPT>
            <UL ID="Liste" onclick=" KnotenWertAendern(this, 0, 'Listenelement Neu')">
                    <LI>Listenelement alt
            </UL>
```

.ownerDocument

Referenz auf das document Objekt zu dem der Koten gehört, also in dem der Knoten erzeugt wurde

Beispiel:

```
<SCRIPT>
        var ElternDokument = SpanTag.ownerDocument;
</SCRIPT>
<BODY>
        <SPAN ID="SpanTag">Hallo !</SPAN>
</BODY>
```

.parentElement

Referenz auf das Elternobjekt, also nicht Elternknoten innerhalb DOM

```
var Zeiger =  document.body.parentElement ;
```

.parentNode

Referenz auf Elternknoten innerhalb der DOM-Hierarchie

Beispiel 1:

```
<SCRIPT>
        var ElternZeiger = SpanTag.parentNode;
</SCRIPT>
<BODY>
        <SPAN ID="SpanTag">Hallo !</SPAN>
</BODY>
```

Beispiel 2:

```
document.body. parentNode  wird null liefern, da BODY das oberste Objekt
```

Beispiel 3:

```
var ZeigerAuf_B_Tag = document.createElement("B");                  // erzeugen
document.body.insertBefore(ZeigerAuf_B_Tag);                        // einfügen
var ElternZeiger = ZeigerAuf_B_Tag.parentNode;                      // Zeiger auf BODY
```

.parentTextEdit

Textbereich des Elternobjektes referenzieren

Beispiel:

```
<SCRIPT>
        function Selektion()
        {
                var ZeigerAufSelektionsQuelle = window.event.srcElement ;

                if (!ZeigerAufSelektionsQuelle.isTextEdit)
                { ZeigerAufSelektionsQuelle = ZeigerAufSelektionsQuelle.parentTextEdit;}

                if (ZeigerAufSelektionsQuelle != null)
                {
```


```
                                        var ZeigerAufTextBereich =
                                                ZeigerAufSelektionsQuelle.createTextRange();

                                        ZeigerAufTextBereich.moveToElementText(window.event.srcElement);
                                        ZeigerAufTextBereich.collapse();
                                        ZeigerAufTextBereich.expand("SelektionsText");
                                        ZeigerAufTextBereich.select();
                                }
                        }
                </SCRIPT>
```

.previousSibling

Referenz auf das Vorgängerkind

Beispiel:

```
<SCRIPT>
        var AktuellesKind_Index = 1;    // Index ab 0
        var VorgaengerKind_Zeiger = Liste.childNodes(AktuellesKind_Index).previousSibling;
                                        //  Listenelement 1 wird referenziert
</SCRIPT>
<BODY>
        <UL ID = "Liste">
                <LI> Listenelement 1
                <LI> Listenelement 2
                <LI> Listenelement 3
        </UL>
<BODY>
```

.readyState

aktueller Status des Objektes beim Füllen des Objektes mit Daten

"uninitialized"	Objekt ist nicht initialisiert
"loading"	Objekt ist nicht initialisiert aber lädt gerade Daten zur Initialisierung
"loaded"	Objekt hat alle Daten komplett geladen und ist komplett initialisiert
"interactive"	Objekt kann vom User interaktiv verwendet werden zum Füllen mit Daten
"complete"	Object hat alle Daten geladen und ist mit diesen komplett initialisiert

Beispiel:

```
alert(document.body.readyState);
```

.scopeName

Namensraum laut XMLNS-Attribut

Beispiel:

```
<HTML XMLNS:InetSDK='http://www.test.de'>
<STYLE>
        @media all {InetSDK\:HalloDu { behavior:url (simple.htc) }}
</STYLE>
<SCRIPT>
        function window.onload()        // überschreibt Standard window.onload-Routien !!!!
        {
                // Statuszeile als Ausgabebereich nutzen
                window.status =     'scopeName = '  +  Hallo.scopeName +  ';'
                                + ' tagUrn = '      + Hallo.tagUrn;
        }
</SCRIPT>
<BODY>
        <InetSDK:HelloWorld ID='Hallo'></InetSDK:HalloDu>
</BODY>
</HTML>
```

.specified

prüfen ob Objekt Attribute hat

true, so Attribute ist vorhanden
false, so keine Attribute vorhanden

Beispiel:

```
<SCRIPT>
```


```
function Anzeigen()
{
	var FeldAllerAttribute = ID_List.attributes;
	alert(FeldAllerAttribute(0).nodeName);	// Knotenname

	for( var i=0; i< FeldAllerAttribute.length; i++)
	{
		// neues LI-Element als Knoten der Liste anhängen
		var NeuesListenElement = document.createElement("LI");
		ID_List.appendChild(NeuesListenElement);

		// Wert des neuen LI-Elementes ist Text, also Textknoten
		var WertNeuesListenElement = document.createTextNode(
			i
			+ " "
			+ FeldAllerAttribute (i).nodeName
			+ " = "
			+ FeldAllerAttribute (i).nodeValue
			);
		NeuesListenElement.appendChild(WertNeuesListenElement);

		// auf specified prüfen
		if(FeldAllerAttribute (i).nodeValue != null )
		{
			alert(	FeldAllerAttribute(i).nodeName
				+ " specified: "
				+ FeldAllerAttribute(i).specified   // boolean
			);
		}
	}
}
</SCRIPT>
<UL ID="ID_List" onclick = " Anzeigen()">
	<LI>Klick mich
</UL>
```

.tagName

Tag-Bezeichner des Objektes

Beispiel:

```
<SCRIPT>
	var ID_Wert = window.prompt("Bitte ID eines Tags eingeben:");
	if (ID_Wert != null)
	{alert(document.all[ID_Wert].tagName) }
</SCRIPT>
```

.tagUrn

Uniform Ressource Name (URN) laut Namensraum laut XMLNS-Attribut

Beispiel:

```
<HTML XMLNS:InetSDK='http://www.test.de'>
<STYLE>
	@media all {InetSDK\:HalloDu { behavior:url (simple.htc) }}
</STYLE>
<SCRIPT>
	function window.onload()		// überschreibt Standard window.onload-Routien !!!!
	{
		// Statuszeile als Ausgabebereich nutzen
		window.status =	'scopeName = ' +  Hallo.scopeName +  ';'
				+ ' tagUrn = '	+ Hallo.tagUrn;
	}
</SCRIPT>
<BODY>
	<InetSDK:HelloWorld ID='Hallo'></InetSDK:HalloDu>
</BODY>
</HTML>
```

.uniqueID

durch den Browser automatisch generiertes ID des Objektes

Browser generiert zu verschiedenen Zeitpunkten auch verschiede ID, wenn Objekt mehrmals geladen wurde
kann anstelle eines privat vergebenen ID als ID-Attributwert weiterverwendet werden

Beispiel:

```
<PUBLIC:ATTACH EVENT="onload" FOR="window" ONEVENT="init()"/>
<SCRIPT LANGUAGE="JScript">
    function init()
    {
        var newTextAreaID = element.document.uniqueID;
        element.document.body.insertAdjacentHTML (  "beforeEnd",
                                                    "<P><TEXTAREA STYLE='height: 200 ;"
                                                  + "width: 350' ID=" + newTextAreaID
                                                  + "></TEXTAREA></P>"
                                                  );
    }
</SCRIPT>
```

.value

Wert eines Objekt-Attributes

Hinweis: Wert eines Elementes ist z.T. über das VALUE-Attribut definierbar

Beispiel für Input-Objekt und seine Varianten:

für Input-Elemente (input Objekt) gelten folgende Standardwerte:

INPUT type=checkbox	on
INPUT type=reset	Reset
INPUT type=submit	Submit Query

alle anderen Input-Elemente haben keinen Standardwert

für Input-Elemente (input Objekt) sind folgenden Werte sendbar:

INPUT type=checkbox	selektierter Wert
INPUT type=file	Dateiname laut Eingabe
INPUT type=hidden	selektierter Wert einer Box-Control (selektierte Option)
INPUT type=password	Eingabewert
INPUT type=radio	selektierter Wert
INPUT type=reset	das Label des Elementes (falls Label existent ist)
INPUT type=submit	das Label des Elementes (falls Label existent ist)
INPUT type=text	Eingabewerte

Werte sind les- und schreibbar
nur lesen bei INPUT type=file

Beispiel für option objekt eines select objektes:

```
<SCRIPT>
    function Anzeigen()
    {
        var Kette = "Auswahl = " + ID_select.options(ID_select.selectedIndex).value;
        alert(Kette);
    }
</SCRIPT>
<SELECT ID="ID_select" onchange = "Anzeigen()">
    <OPTION VALUE="1">Auswahl 1 </OPTION>
    <OPTION VALUE="2">Auswahl 2 </OPTION>
    <OPTION VALUE="3">Auswahl 3 </OPTION>
</SELECT>
```

Beispiel für Zeichenkette auf Wertebereich der Zeichen prüfen:

```
<HTML>
<HEAD>
 <SCRIPT  LANGUAGE="JavaScript">
<!--
    function pruefe_zeichenkette(zeichenkette, wertebereich)
    {   //  wertebereich ist Zeichenkette z.B. "0123456789 -+/.,()"

        var rueckgabewert = true;       // Annahme: Zeichenkette hat NUR Zeichen
                                        //                  aus dem Wertebereich
        var zeichen_aus _zeichenkette;
```


```
                        //          Zeichenkette zeichenweise analysieren: Jedes Zeichen mit Wertebereich vergleichen
                        for (var i = 0; i < zeichenkette.length; i++)
                         {
                                zeichen_aus_zeichenkette = zeichenkette.charAt(i);

                                if (wertebereich.indexOf(zeichen_aus_zeichenkette) == -1)
                                {
                                        rueckgabewert = false;
                                        break;      // ungültiges Zeichen gefunden, also abbrechen
                                 }
                        }

                        return rueckgabewert;
                }

                function pruefe_eingabe(zu_pruefende_zeichenkette)
                {
                        if (pruefe_zeichenkette(zu_pruefender_zeichenkette, "0123456789 -+/.,()") )
                         {alert("Eingabe ist korrekt !");}
                         else
                         {alert("Eingabe ist nicht korrekt !"); }
                }
//-->
</SCRIPT>
</HEAD>
<BODY>
 <FORM>
                Telefon:
                <INPUT  TYPE="text"
                        NAME="Telefon"
                        VALUE=""
                >
                <INPUT  TYPE="button"
                        VALUE="Ueberpruefen"
                        onclick="pruefe_eingabe(this.form.Telefon.value)">
 </FORM>
</BODY>
</HTML>
```

.XMLDocument

Referenz auf XML-Dokument (XML-DOM)

Beispiel:

```
<HTML>
<HEAD>
<SCRIPT>
        var ZeigerAufXMLDocument = ID_Div.XMLDocument;
</SCRIPT>
</HEAD>
<BODY>
        <DIV ID="ID_Div">
        </DIV>
</BODY>
</HTML>
```

.XSLDocument

Referenz auf den obersten Knoten des XSL-Dokumentes (Style-Sheet-Dokument)

```
var Zeiger =  document.XSLDocument;
```


Methoden zur Verwaltung des HTML-DOM im Internet Explorer

Nachfolgende Methoden sind nicht in allen Objekten implementiert (siehe einzelne Objekte).

.addBehavior()

DHTML-Verhaltenseigenschaft einem Element hinzufügen

Empfehlung: Standard-IE-Eigenschaften nutzen, da diese mit "#default#behaviorName" komplett erfasst werden und bereits im Browser implementiert sind (keine HTC-Datei nötig).

ab IE 5.x bis unter IE 5.5

Beispiel:

```
<SCRIPT>
    var FeldDerEigenschaftenID      = new Array(); // für removeBehavior
    var FeldDerTagsLIimDokument     = new Array ();
    var FeldDerTagsLIimDokument_Laenge = 0;

    function EigenschaftHinzufuegen()
    {
        FeldDerTagsLIimDokument      = document.all.tags ("LI");
        FeldDerTagsLIimDokument_Laenge = FeldDerTagsLIimDokument.length;
        for ( var i=0; i < FeldDerTagsLIimDokument_Laenge; i++)
        {
            var EigenschaftenID    // immer neu anlegen wegen Zeigerprüfung
                = FeldDerTagsLIimDokument [i].addBehavior ("hilite.htc");

            if (iEigenschaftenID)
            {FeldDerEigenschaftenID[i] = EigenschaftenID;}
        }
    }

    function EigenschaftEntfernen()
    {
        for ( var i=0; i < FeldDerTagsLIimDokument_Laenge; i++)
        {FeldDerEigenschaftenID[i].removeBehavior (FeldDerEigenschaftenID[i]); }
    }
</SCRIPT>
<A HREF="javascript: EigenschaftHinzufuegen()">Eigenschaft hinzufuegen</A>
<A HREF="javascript: EigenschaftEntfernen()">Eigenschaft entfernen</A>
```

.appendChild()

Knoten als Kind an die DOM-Hierarchie anhängen und danach den Zeiger laut DOM liefern

DOM wird geändert

Zeiger wird zugleich immer am Ende der Collection childNodes angehangen

Anhängen wird danach nur sichtbar, wenn zusätzlich dem BODY-Objekt angehangen wird UND Ende-Tag (falls vorhanden) des Knoten geparst wurde

kann erst nach createElement() erfolgen

.readyState wird erst belegt mit .appendChild()

Beispiel 1:

```
var ZeigerAufDiv =document.createElement("DIV");
document.body.appendChild(ZeigerAufDiv);
```

Beispiel 2:

```
<SCRIPT>
    function Anhaengen()
    {
        var ZeigerAufNeuesLI = document.createElement("LI");
        Liste.appendChild(ZeigerAufNeuesLI);          // dem BODY anhängen
                                                      //  also sichtbar werdend
        ZeigerAufNeuesLI.innerText="Listenelement 5";
    }
    </SCRIPT>
    <BODY>
        <UL ID = "Liste" >
            <LI>Listenelement 1
            <LI>Listenelement 2
            <LI>Listenelement 3
            <LI>Listenelement 4
        </UL>
```


```
                <INPUT TYPE = "button" VALUE = "Anhaengen " onclick = " Anhaengen()">
        </BODY>
```

.applyElement()

Elementeigenschaft "Kind sein" oder "Eltern sein" festlegen, also die Lage im DOM, und danach Referenz
laut DOM liefern
DOM wird geändert
Element kann selbst Kinder haben
Element erst sichtbar, wenn Endetag (falls vorhanden) des Elementes geparst wurde
Achtung: Wenn Element per Methode .createElement() erzeugt wurde, aber nicht im Dokumentenbaum eingebunden ist, so wird die Eigenschaft .innerHTML gelöscht !

Beispiel:

```
<SCRIPT>
        function Hinzufuegen()
        {
                var Zeiger = document.createElement("LI");
                Liste.applyElement(Zeiger);
        }
</SCRIPT>
<UL ID = Liste>
        <LI>Listenelement 1
        <LI>Listenelement 2
        <LI>Listenelement 3
        <LI>Listenelement 4
</UL>
<INPUT   TYPE="button"   VALUE=" Hinzufuegen"  onclick=" Hinzufuegen()">
```

.clearAttributes()

alle HTML-Attribute eines Objektes entfernen, außer ID, STYLE und per Script definierte Attribute
Script-erzeugte Attribute nicht entfernbar
DOM wird geändert

Beispiel:

```
<SCRIPT>
        function Loeschen()
        {
                ID_Span.children[0].clearAttributes();
        }
</SCRIPT>
<SPAN ID="ID_Span">
        <DIV    ID="ID_Div"
                ATTRIBUTE1="true"
                ATTRIBUTE2="true"
                onclick="alert('click');"
                onmouseover="this.style.color='#0000FF';"
                onmouseout="this.style.color='#000000';"
        >
                Test eines<B>Div</B>Elementes.
        </DIV>
</SPAN>
<INPUT  TYPE="button" VALUE=" Loeschen" onclick="Loeschen()">
```

.cloneNode()

Objekt klonen und Referenz des erzeugten Klone liefern
DOM wird nicht geändert, da Klone nicht in DOM eingebunden wird (reines Neu-Instanzieren eines DOM-Objektes im Hauptspeicher außerhalb des DOM)

Beispiel:

```
<SCRIPT>
        function Klonen()
        {
                var ZeigerAufKlone = Liste.cloneNode(true);  // DOM wird nicht geändert
                document.body.insertBefore(ZeigerAufKlone); // DOM wird geändert
        }
</SCRIPT>
<UL ID = "Liste" >
        <LI>Listenelement 1
        <LI>Listenelement 2
        <LI>Listenelement 3
```


```
		<LI>Listenelement 4
	</UL>
	<INPUT TYPE="button" VALUE=" Klonen" onclick=" Klonen()">
```

.contains()

prüfen ob Element innerhalb eines Elementes liegt, also ob das innere, eingeschlossene Element Eltern (Eltern-Objekt, Container) hat und somit ein Kind-Objekt ist

DOM nicht geändert
true Element liegt innerhalb eines Elementes
false Element liegt nicht innerhalb eines Elementes

Beispiel:

```
<SCRIPT>
	function TesteMaus(Zeiger)
	{
		if(  ! Zeiger.contains(event.fromElement)  )
		{alert("Maus über Button erkannt");}
	}
</SCRIPT>
<BUTTON onmouseover=" TesteMaus(this)">Ueberfahre mich mit der Maus</BUTTON>
```

.createAttribute()

ein Attribut im Dokument erzeugen und Referenz auf das erzeugte Attribut liefern

Achtung: Der Browser unterscheidet zwischen HTML-erzeugte oder mit dieser Methode erzeugte Attribute!
DOM nicht geändert

Beispiel:

```
<HTML>
<HEAD>
<SCRIPT>
	function Hinzufuegen()
	{
		// Attribut mit Wert erzeugen
		var ZeigerAufAttribut = document.createAttribute("title");
		ZeigerAufAttribut.value = "Tooltip-Text ";

		// Attribut als Feldelement anhängen
		var ZeigerAufFeld = ID_Div.attributes;
		ZeigerAufFeld.setNamedItem(ZeigerAufAttribut);
	}
</SCRIPT>
</HEAD>
<BODY onload=" Hinzufuegen();">
	<DIV ID="ID_Div">Es wird ein Tooltip-Text hinzugefuegt</DIV>
</BODY>
</HTML>
```

.createComment()

Comment-Objekt (Kommentar-Objekt) im Dokument erzeugen und Referenz liefern

verwendbar anstelle von direkt im HTML- bzw. Script-Quellcode kodiertem Kommentar
DOM wird geändert, da das Dokument erweitert wird

Beispiel:

```
var Kommentar = document.createComment("Das ist ein Kommentar");
```

.createElement()

HTML-Objekt (Tags) im Dokument erzeugen und Referenz liefern

Achtung: Erzeugtes Objekt muss in DOM noch per Methode .insertBefore() bzw..appenChild() eingereiht werden.
Hinweis: Attribute mit der Methode .createAttribute() erzeugen
DOM wird geändert

nach createElement() muss irgendwann z.B. appendChild() folgen
.readyState wird erst belegt mit .appendChild()
Tags sind auch in der Form '< ...>' mit Attributen möglich
Ende-Tag kann muss aber nicht kodiert werden (falls HTML-Syntax einen Endetag vorschreibt)
Bsp: '<DIV STYLE="....">' oder '<DIV STYLE="...."></DIV>'
niemals Elemente zwischen den Tags angeben sondern diese NACH der Erzeugung

per .innerText bzw. .innerHTML erzeugen
Achtung: beide Eigenschaften schalten aktives BGSOUND auf stumm !
also z.B. nicht ' <DIV STYLE="...."><B>TestText</B></DIV>'
'<A STYLE="....">TestText</DIV>'
es werden z.B. HTML-Code-Fehler ignoriert, aber nicht alle
wenn ignoriert, so Objekt erzeugt aber eben nicht mit vollständigem HTML-Code

Beispiel 1:

```
<SCRIPT>
    function Erzeugen()
    {
        ID_Span.innerHTML="";
        var FeldDerZeigerAufOption = ID_Select.options[ID_Select.selectedIndex];

        if(FeldDerZeigerAufOption.text.length>0)
        {
            var ZeigerAufElement=
                document.createElement(FeldDerZeigerAufOption.text);
            eval(  " ZeigerAufElement."
                + FeldDerZeigerAufOption.value
                + "='"
                + ID_Input.value
                + "'"
              );

            if(FeldDerZeigerAufOption.text=="A")
            { ZeigerAufElement.href="javascript:alert('A link.')"; }
        }
        ID_Span.appendChild(ZeigerAufElement);
    }
</SCRIPT>
<SELECT ID="ID_Select" onchange="Erzeugen()">
    <OPTION VALUE="innerText">A
    <OPTION VALUE="value">&lt;INPUT TYPE="button"&gt;
</SELECT>
<INPUT TYPE="text" ID="ID_Input" VALUE="Beispiel Text">
<SPAN ID="ID_Span" ></SPAN>
```

Beispiel 2:

```
<HTML>
<HEAD>
<SCRIPT>
    function Erzeuge()
    {
        var Zeiger = document.createElement(
                "<INPUT TYPE='RADIO' NAME='RADIOTEST' VALUE='Eins'>"
                );
        document.body.insertBefore(Zeiger);

        Zeiger = document.createElement(
                "<INPUT TYPE='RADIO' NAME='RADIOTEST' VALUE='Zwei'>"
                );
        document.body.insertBefore(Zeiger);
    }
</SCRIPT>
</HEAD>
<BODY>
    <INPUT  TYPE="BUTTON"
        ONCLICK=" Erzeuge()"
        VALUE="Zwei Radio Buttons erzeugen">
    <BR>
    <INPUT  TYPE="BUTTON"
        ONCLICK="alert(document.body.outerHTML)"
        VALUE="Click um HTML zu sehen">
    <BODY>
</HTML>
```

.createStyleSheet()

Style-Sheet-Objekt im Dokument erzeugen und Referenz liefern
DOM wird geändert
Beispiel:

```
document.createStyleSheet('styles.css');
```

.createTextNode()

Plain-Textelement im Dokument erzeugen und Referenz liefern

Plaintext kann keine HTML-Tags enthalten
DOM wird geändert, da Dokument erweitert wird

Beispiel:

```
<SCRIPT>
    function TextElementAendern()
    {
        var ZeigerAufTextElement = document.createTextNode("Neuer Text");
        var ZeigerAufSpanInhalt = ID_Span.childNodes(0);
        ZeigerAufSpanInhalt.replaceNode(ZeigerAufTextElement);
    }
</SCRIPT>
<SPAN ID = "ID_Span" onclick=" TextElementAendern()">
    Original Text
</SPAN>
```

.expression()

Wert einer Style-Eigenschaft per STYLE-Attribut-Wert in HTML als Ausdruck definieren für
spätere Berechnung per Methode.getExpression()

Ausdruck nur als Script kodierbar
DOM wird geändert
siehe auch Methode .setExpression()

Beispiel 1:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE ="JScript">
    function width_init()
    {
        ID_Span.style.setExpression(    "width",
                                        "    trueBlueSpan.style.pixelWidth
                                          + oldYellowSpan.style.pixelWidth
                                        ",
                                         "jscript"
                                    );
    }

    function Berechne()
    {alert(ID_Span.style.getExpression("width");}
</SCRIPT>
</HEAD>
<BODY onload= width_init();>
    <SPAN   ID="ID_Span"
        STYLE="background-color:lightgreen;
            width:expression(    trueBlueSpan.style.pixelWidth
                                + oldYellowSpan.style.pixelWidth
                            )
                "
    >
    </SPAN>
    <BUTTON onclick= Berechne();>
</BODY>
</HTML>
```

Beispiel 2:

```
ID_Span.style.setExpression("height","document.style.fontSize + 13");
ID_Span.style.setExpression("width","document.body.style.fontSize");
<SAPN   ID="ID_Span"
    STYLE="background-color:lightgreen;
        width:expression(       trueBlueSpan.style.pixelWidth
                            + oldYellowSpan.style.pixelWidth
                        )
            "
>
</SPAN>
```


.getAdjacentText()

Text eines Objektes liefern, wobei Textlage im Objekt definiert werden kann
Text kann HTML-Tags enthalten, muss aber nicht
DOM nicht geändert

Beispiel:

```
<SCRIPT>
    function Anzeigen()
    {
        var Woher_Kette = ID_Selektion.options[ID_Selektion.selectedIndex].text;
        alert(ID_P.getAdjacentText(Woher_Kette));
    }
</SCRIPT>
Text vor dem Beginn des P-Tag (woher ist beforeBegin)
<P ID="ID_P">
    Text nach dem Beginn des P-Tag (woher ist afterBegin)
    <B>Irgend ein Text</B>
    Text vor dem Ende des P-Tag (woher ist beforeEnd)
</P>
Text nach dem Ende des P-Tag (woher ist afterEnd)

<SELECT ID="ID_Selektion">
    <OPTION SELECTED>woher ist beforeBegin
    <OPTION>woher ist afterBegin
    <OPTION>woher ist beforeEnd
    <OPTION>woher ist afterEnd
</SELECT>
<INPUT TYPE="button" VALUE="Anzeigen" onclick=" Anzeigen()">
```

.getAttribute()

Wert eines per **HTML** erzeugten Attributes liefern
DOM nicht geändert

Beispiel:

```
<HTML>
<HEAD>
<STYLE>
    .user_data_speicher_klasse {behavior:url(#default#userData);}
</STYLE>
<SCRIPT>
    // Cachname festlegen
    //      es können diverse Cachenamen definiert und somit Versionen von Cache
    //          verwaltet werden
    var FreierCacheName = "InputCache";

    // Cache-Attribut festlegen
    //      es können diverse Attribute definiert und somit Versionen von Input-Daten
    //          verwaltet werden
    var FreiesCacheAttribut = "InputCacheAttribut";

    // zu cachende Daten referenzieren
    var InputDatenObjekt = ID_Formular.ID_Input;

    function InputSichern()
    {
        // +++++++++ Zeitstempel ++++++++++++++++++++++++
        var ZeitpunktJetzt = new Date();
        var ZeitpunktJetztInMinuten = ZeitpunktJetzt.getMinutes();

        // Zeitstempel festlegen: Ab jetzt + 20 Minuten
        var Zeitstempel = ZeitpunktJetztInMinuten + 20;

        // und als UTC-Format erzeugen
        var ZeitstempelUTC = Zeitstempel.toUTCString();

        // und Zeitstempel dem Input-Objekt verpassen
        ID_Input.expires = ZeitstempelUTC;

        // ++++++++ Daten chachen +++++++++++++++++++++
        // aktuelle Daten holen laut Input-Objekt
        var InputDaten = InputDatenObjekt.value;

        // Attrbibut instanzieren und mit Daten füllen
```


```
                    InputDatenObjekt.setAttribute(FreiesCacheAttribut, InputDaten);

                    // und Cache saven
                    InputDatenObjekt.save(FreierCacheName);
                }

                function InputLaden()
                {
                    // Cache laden
                    InputDatenObjekt.load(FreierCacheName);

                    // und Daten zum Attribut laut Sicherung lesen
                    var InputDaten = InputDatenObjekt.getAttribute(FreiesCacheAttribut);
                }

        </SCRIPT>
        </HEAD>
        <BODY>
            <FORM ID="ID_Formular">
                <INPUT   ID="ID_Input"
                         CLASS="user_data_speicher_klasse"
                         TYPE="text"
                >
                <INPUT TYPE="button" VALUE="sichern der Input-Daten" onclick="InputSichern()">
                <INPUT TYPE="button" VALUE="laden der Input-Daten"    onclick="InputLaden()">
            </FORM>
        </BODY>
        </HTML>
```

.getAttributeNode()

Referenz auf Eigenschaft des attribute-Objektes liefern, also Zeiger auf attribute.name Eigenschaft.
Eigenschaft kann mit HTML-Anweisung erzeugt worden sein, muss aber nicht
Eigenschaft ist selbst ein Knoten in der Attribute-Objekt-Hierarchie zum Objekt
Wert des Attributes wird somit über die Referenz laut DOM erreichbar
DOM nicht geändert

Beispiel:

```
<HTML>
<HEAD>
<SCRIPT>
    function ToolTipKnotenErmitteln()
    {
        return (ID_Div.getAttributeNode("TITLE"));
    }
</SCRIPT>
</HEAD>
<BODY onload="ToolTipKnotenErmitteln();">
    <DIV ID="ID_Div" TITLE="Tooltip-Text">Es ist ein Tooltip-Text vorhanden</DIV>
</BODY>
</HTML>
```

.getElementById()

Referenz auf das im Dokument ZUERST gefundene Objekt laut ID (analog zum ID-Attribut) liefern
Achtung: Objekte, die kein ID besitzen, werden nicht erfasst !
Für Verwaltung per NAME (analog zum NAME-Attribut):
siehe Methode getElementsByName()
Für Verwaltung per Tag-Name
siehe Methode .getElementsByTagName()
wenn mehrere Elemente mit ein und demselben ID, so das ERSTE Element von diesen referenziert
Eine Referenzierung einer Collection der Objekte mit gemeinsamen ID ist leider nicht möglich. Deswegen der strenge Hinweis:
In der Regel werden ID vom Programmierer objektweise getrennt vergeben, es sei denn, man will bewusst eine Gruppe von Objekten (z.B. RadioBox) verwaltbar machen und kennt diese Objekte. Die maschinelle Analyse eines fremden Dokumentes mit der Methode .getElementById() ist nicht möglich.
DOM nicht geändert

Beispiel:

```
<SCRIPT>
    function Referenziere()
    {
```


```
                    var Zeiger = document.getElementById("ID_Div");
                }
            </SCRIPT>
            <DIV ID="ID_Div">Test</DIV>
            <INPUT TYPE="button" VALUE=" Referenziere" onclick=" Referenziere()">
```

.getElementsByName()

Referenz auf ein Feld (Collection) aller im Dokument befindlichen Objekte mit gemeinsamen NAME
(analog zum Attribut NAME) liefern
Achtung: Objekte, die kein NAME besitzen, werden nicht erfasst !
Für Verwaltung per ID (analog zum ID-Attribut):
siehe Methode getElementsById()
Für Verwaltung per Tag-Name
siehe Methode .getElementsByTagName()
DOM nicht geändert

Beispiel:

```
<SCRIPT>
    function Referenziere()
    {
        var FeldDerInput = document.getElementsByName("GemeinsamerName");
    }
</SCRIPT>
<INPUT TYPE="text" NAME="GemeinsamerName">
<INPUT TYPE="text" NAME="GemeinsamerName">
<INPUT TYPE="button" NAME="Button" VALUE=" Referenziere" onclick=" Referenziere()">
```

.getElementsByTagName()

Referenz auf ein Feld (Collection) aller im Objekt befindlichen Kinder-Objekte mit gemeinsamen
Tagnamen liefern, inklusive aller Kinder und Unterkinder etc.
Hinweis: Natürlich kann auch das document-Objekt so verarbeitet werden
(beachte dabei document.all-Collection)
Achtung: Kinder-Objekte, die keinen Tag-Name besitzen, werden nicht erfasst !
Für Verwaltung per ID (analog zum ID-Attribut):
siehe Methode getElementsById()
Für Verwaltung per NAME (analog zum NAME-Attribut):
siehe Methode .getElementsByName()
DOM nicht geändert

Beispiel 1:

```
var DIV_KinderZeigerFeld = document.body.getElementsByTagName("DIV");

Hinweis: entspricht  var DIV_KinderZeigerFeld = document.body.all.tags("DIV");
```

Beispiel 2:

```
<SCRIPT>
    var Feld_Span = ID_DivEltern.getElementsByTagName("SPAN");
                                        // alle SPAN-Kinder referenzieren
</SCRIPT>
<DIV ID="ID_DivEltern">
    <SPAN>
        Span-Kind von ID_DivEltern
        <DIV>
            Div-Kind vonDivEltern-Span
            <SPAN>
            Span-Kind vonDivEltern-Span-Div
            </SPAN>
        </DIV>
    </SPAN>
</DIV>
```

Beispiel 3:

```
<SCRIPT>
    function Anzeige()
    {
        var ZeigerAufOnClickEventQuelle=event.srcElement;
        var Feld =
            ZeigerAufOnClickEventQuelle.parentElement.getElementsByTagName("LI");
        alert(      "Anzahl LI : "
                + Feld.length
                + "\nErster Eintrag: "
```


```
                                        + Feld [0].childNodes[0].nodeValue
                                    );
                }
</SCRIPT>
<UL onclick="Anzeige()">
        <LI>Menuepunkt 1
        <UL>
                <LI> Menuepunkt 1.1
                        <OL>
                                <LI> Menuepunkt 1 1.1
                                <LI> Menuepunkt 1 1.2
                        </OL>
                <LI> Menuepunkt 1.2
                <LI> Menuepunkt 1.3
        </UL>
        <LI> Menuepunkt 2
        <UL>
                <LI> Menuepunkt 2.1
                <LI> Menuepunkt 2.3
        </UL>
        <LI> Menuepunkt 3
</UL>
```

.getExpression()

Wert einer Style-Eigenschaft anhand des Ausdruckes berechnen und liefern

Style-Eigenschaft ist per Methoden
expression() oder setExpression()
zu definieren

DOM wird nicht verändert (nur Werteveränderung), aber das Dokument-Layout
(nach dem eventuellen expliziten Dokument-Refresh)

In nachfolgenden Beispielen wurde aus Platzgründen die Zeichenkette von STYLE umgebrochen,
was eigentlich nicht zulässig ist, und es sind nicht alle SPAN kodiert.

Beispiel 1:

```
<SPAN ID="ID_Span"
        STYLE=  "background-color:lightgreen;
                 width:expression(      trueBlueSpan.style.pixelWidth
                                        + oldYellowSpan.style.pixelWidth
                                )
                "
>
</SPAN>
<BR>
<BUTTON onclick=alert(ID_Span.style.getExpression("width"));>
```

Beispiel 2:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE ="JScript">
        function width_init()
        {
                ID_Span.style.setExpression("width",
                                                "
                                                        trueBlueSpan.style.pixelWidth
                                                  + oldYellowSpan.style.pixelWidth
                                                ",
                                                "jscript"
                                               );
        }

        function Berechne()
        {alert(ID_Span.style.getExpression("width");}
</SCRIPT>
</HEAD>
<BODY onload= width_init();>
        <SPAN   ID="ID_Span"
                STYLE="background-color:lightgreen;
                        width:expression(   trueBlueSpan.style.pixelWidth
                                          + oldYellowSpan.style.pixelWidth
```


```
                                                                    )
                                                        "
                        >
                        </SPAN>
                        <BUTTON onclick= Berechne();>
        </BODY>
        </HTML>
```

.hasChildNodes()

prüfen auf Existenz von Kinder(n) als HTML-Elemente oder Textknoten
(Textelemente) in einem Objekt
DOM nicht geändert

Beispiel:

```
<HTML>
<BODY onload="alert(ID_Div.hasChildNodes();">
        <DIV ID="ID_Div" TITLE="Tooltip-Text">Es ist ein Tooltip-Text vorhanden</DIV>
</BODY>
</HTML>
```

.insertAdjacentElement()

Objekt in eine Objekt einfügen und Referenz liefern, wobei die Lage definiert werden kann
wenn Element bereits eingefügt vorhanden, so wird dieses nur verschoben laut Lage des Objektes im DOM
nur nach dem kompletten Laden des Dokumentes möglich
DOM wird geändert

Beispiel:

```
<SCRIPT>
        function Hinzufuegen()
        {
                var ZeigerAufLI = document.createElement("LI");
                ID_Liste.children(0).insertAdjacentElement("beforeBegin", ZeigerAufLI);
                ZeigerAufLI.innerText = "Listeneintrag 0";
        }
</SCRIPT>
<BODY>
        <OL ID = "ID_Liste ">
                <LI>Listeneintrag 1</LI>
                <LI>Listeneintrag 2</LI>
                <LI>Listeneintrag 3</LI>
        </OL>
        <INPUT  TYPE = "button" VALUE = " Hinzufuegen" onclick=" Hinzufuegen()">
</BODY>
```

.insertAdjacentHTML()

HTML-Code und/oder Script-Code in ein Element einfügen, wobei die Lage definiert sein kann
nur nach dem kompletten Laden des Dokumentes möglich
HTML- und Script-Code müssen syntaktisch korrekt sein
wenn nicht, so wird das Einfügen **nicht** ausgeführt
eingefügter Code wird **nur** dann sofort geparst und ausgeführt, wenn syntaktisch korrekt ist
bei Script-Code: <SCRIPT DEFER .....> muss kodiert werden
DOM wird geändert

Beispiel:

```
var HTML_Code =         "<INPUT TYPE =button"
                                +           "  VALUE='Bitte klicken'"
                        +           " onclick='Anzeige()'"
                        +   ">"
                        + "<BR>";

var JavaScript_Code =     "<SCRIPT DEFER>"                      // DEFER muss kodiert sein
                        +  "function Anzeige(){alert('Anzeige aktiv') }"
                        +  "</SCRIPT>";

ID_Div.insertAdjacentHTML("afterBegin", HTML_Code + JavaScript_Code);

<DIV ID="ID_Div">
        Test
<DIV>
```


.insertAdjacentText()

Plain-Text (ohne HTML und Script) in ein Element einfügen, wobei die Lage definiert werden kann
nur nach dem kompletten Laden des Dokumentes
DOM wird geändert

Beispiel:

```
var PlainText = " Hinzugefuegter Text";
ID_Div.insertAdjacentText("afterBegin", PlainText);

<DIV ID="ID_Div">
    Test
<DIV>
```

.insertBefore()

Objekt als Kindknoten VOR dem einem anderen Kind-Objekt einfügen und Zeiger liefern
einzufügendes Objekt muss mit Methode createElement() erzeugt worden sein
Achtung: NICHT anwenden für einfügen VON bzw. VOR obersten Kindknoten
Sichtbarkeit erst wenn Ende-Tag geparst wurde
DOM wird geändert

Beispiel:

```
<SCRIPT>
    function Einfuegen()
    {
        var ZeigerAufNeuesLI = document.createElement("LI");
        ID_UL.insertBefore(ZeigerAufNeuesLI, ID_LI);
        ZeigerAufNeuesLI.innerText="2";
    }
</SCRIPT>
</HEAD>
<BODY>
        <SPAN onclick= Einfuegen()>Klick</SPAN>
    <UL ID="ID_UL">
        <LI >1</LI>
        <LI ID="ID_LI">3</LI>
        <LI >4</LI>
    </UL>
</BODY>
```

.mergeAttributes()

alle Attribute eines Elementes in ein anderes Element kopieren und eventuell die Attribute im Ziel mischen
Attribute sind: HTML
Events
Styles
ab IE 5.01 auch ID, NAME
Achtung: Diese Methode ist mir Vorsicht zu genießen !!
DOM wird geändert

Beispiel:

```
<SCRIPT>
    function Mischen()
    { ID_SPAN.children[1].mergeAttributes(ID_SPAN.children[0]);}
</SCRIPT>
<SPAN ID=ID_SPAN>
    <DIV    ID="ID_Div_Quelle"
            ATTRIBUTE1="true"
            ATTRIBUTE2="true"
            onclick="alert('click');"
            onmouseover="this.style.color='#0000FF';"
            onmouseout="this.style.color='#000000';"
    >
            Quell<B>Div</B>
    </DIV>
    <DIV    ID="ID_Div_Ziel">
            Ziel-Div
    <DIV>
</SPAN>

<INPUT  TYPE="button" VALUE=" Mischen" onclick="Mischen()">
```


.normalize()

Normalisierung des DOM zur Erreichung einer konsistenten Struktur
Achtung: CDATA-Sections dürfen nicht enthalten sein, da diese immer Inkonsistenz erzeugen

Beispiel:

```
<HTML>
<BODY onload="ID_Div.normalize();">
    <DIV ID="ID_Div" TITLE="Tooltip-Text ">Es ist ein Tooltip-Text vorhanden</DIV>
</BODY>
</HTML>
```

.removeAttribute()

entfernen eines per HTML erzeugten Attributes
Achtung: Der Browser unterscheidet zwischen HTML-erzeugte oder mit dieser Methode erzeugte Attribute!
per Methode .createAttribute() erzeugte Attribute werden nicht erfasst
DOM wird geändert

Beispiel:

```
<HTML>
<HEAD>
<SCRIPT>
    function Entfernen()
    {
        ID_Div.removeAttribute("TITLE");
    }
</SCRIPT>
</HEAD>
<BODY onload="Entfernen();">
    <DIV ID="ID_Div" TITLE="Tooltip-Text">Es ist ein Tooltip-Text vorhanden</DIV>
</BODY>
</HTML>
```

.removeAttributeNode()

entfernen von Attribut, egal ob es mit oder ohne HTML-Anweisung erzeugt wurde, und Referenz auf das
entfernte Attribut liefern
DOM wird geändert

Beispiel:

```
<HTML>
<HEAD>
<SCRIPT>
    function ToolTipKnotenEnfernen()
    {
        var Knoten = ID_Div.getAttributeNode("TITLE");
        ID_Div removeAttributeNode(Knoten);
    }
</SCRIPT>
</HEAD>
<BODY onload="ToolTipKnotenEntfernen();">
    <DIV ID="ID_Div" TITLE="Tooltip-Text">Es ist ein Tooltip-Text vorhanden</DIV>
</BODY>
</HTML>
```

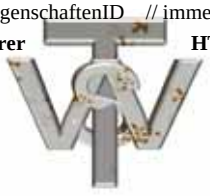
.removeBehavior()

per Methode .addBehavior() einem Element hinzugefügte Verhaltenseigenschaft entfernen (stets VOR dem
Entfernen des Elementes mit der zugeordneten Eigenschaft aus der Dokument-Hierarchie)
DOM wird geändert

Beispiel:

```
<SCRIPT>
    var FeldDerEigenschaftenID          = new Array(); // für removeBehavior
    var FeldDerTagsLIimDokument         = new Array ();
    var FeldDerTagsLIimDokument_Laenge = 0;

    function EigenschaftHinzufuegen()
    {
        FeldDerTagsLIimDokument         = document.all.tags ("LI");
        FeldDerTagsLIimDokument_Laenge = FeldDerTagsLIimDokument.length;
        for (i=0; i < FeldDerTagsLIimDokument_Laenge; i++)
        {
            var EigenschaftenID   // immer neu anlegen wegen Zeigerprüfung
```

```
                                        = FeldDerTagsLIimDokument [i].addBehavior ("hilite.htc");

                                    if (iEigenschaftenID)
                                    {FeldDerEigenschaftenID[i] = EigenschaftenID;}
                            }
                    }

                    function EigenschaftEntfernen()
                    {
                            for (i=0; i < FeldDerTagsLIimDokument_Laenge; i++)
                            {FeldDerEigenschaftenID[i].removeBehavior(FeldDerEigenschaftenID[i]); }
                    }
            </SCRIPT>
            <A HREF="javascript: EigenschaftHinzufuegen()">Eigenschaft hinzufuegen</A>
            <A HREF="javascript: EigenschaftEntfernen()">Eigenschaft entfernen</A>
```

.removeChild()

Kind-Objekt aus einem Objekt entfernen aus DOM und Referenz auf das entfernte Kind liefern
Sichtbarkeit erst, wenn Ende-Tag geparst wurde, also das Dokument neu geladen wurde
DOM wird geändert

Beispiel:

```
<HEAD>
<SCRIPT>
        function Entfernen()
        {
                // versuche den Text zu entfernen
                try
                {
                        var KindZeigerAufTextImDiv = ID_Div.children(0);
                        ID_Div.removeChild(KindZeigerAufTextImDiv);
                        // Achtung: Der Text ist noch sichtbar !!!!
                }
                // oder fange das Ereignis des bereits entfernten Textes ein
                //                      und behandle das Ereignis
                catch(x)
                {
                        alert(          "Text wurde entfernt !\n"
                                + "Das Dokument muss neu geladen werden !
                            );
                        document.location.reload();
                }
        }
</SCRIPT>
</HEAD>
<BODY>
        <DIV ID="ID_Div" onclick=" Entfernen()">
                Klick, um diesen Text zu entfernen !
        </DIV>
</BODY>
```

.removeExpression()

Ausdruck entfernen, der für die Berechnung des Wertes einer Style-Eigenschaft als Objektreferenz der
Form objekt.style.eigenschaft. dient.
Ausdruck muss mit der Methode .setExpression() gesetzt worden sein
DOM wird nicht geändert

Beispiel: aus Platzgründen die Zeichenkette von STYLE umgebrochen,
was eigentlich nicht zulässig ist, und es sind nicht alle SPAN kodiert.

```
ID_Span.style.setExpression("width","document.body.style.fontSize");
<SAPN   ID="ID_Span"
        STYLE="background-color:lightgreen;
                width:expression(       trueBlueSpan.style.pixelWidth
                                + oldYellowSpan.style.pixelWidth
                        )
                "
>
</SPAN>
ID_Span.style.removeExpression("width");
```


.removeNode()

Knoten entfernen aus DOM und Referenz auf den entfernten Knoten liefern
Sichtbarkeit erst wenn Ende-Tag geparst wurde
DOM wird geändert

Beispiel:

```
<SCRIPT>
    function Entfernen()
    {Tabelle.removeNode(true);}
</SCRIPT>
<TABLE ID = "Tabelle" >
<TR>
    <TD>Zelle 1</TD>
    <TD>Zelle 2</TD>
</TR>
</TABLE>
<INPUT TYPE = button VALUE = " Entfernen" onclick = " Entfernen()">
```

.replaceAdjacentText()

Plain-Text (ohne HTML und Script) eines Elementes durch anderen Text ersetzen und Referenz auf den zu ersetzenden Text liefern
DOM wird nicht geändert

Beispiel:

```
var PlainText = "Neuer Text";
ID_Div.replaceAdjacentText("afterBegin", PlainText);

<DIV ID="ID_Div">
    Test
<DIV>
```

.replaceChild()

Kind-Objekt ersetzen durch ein Objekt
ersetzende Objekt muss per Methode .createElement() erzeugt worden sein
Sichtbarkeit erst wenn Ende-Tag geparst wurde
DOM wird geändert

Beispiel:

```
<HEAD>
<SCRIPT>
    function Ersetze()
    {
        var KindZeigerAufDivText = ID_Div.children(0);
        var RetteInnerHTML = KindZeigerAufDivText.innerHTML;

        // prüfen auf Tag im Div-Text
        if (KindZeigerAufDivText.tagName=="B")
        {
            // Bold-Tag <B>gefunden, also I-Tag erzeugen
            var ZeigerAufNeuenSchriftStilTag =document.createElement("I");

            // komplettes ersetzen von Div-Text,
            ID_Div.replaceChild(ZeigerAufNeuenSchriftStilTag, KindZeigerAufDivText);
            ZeigerAufNeuenSchriftStilTag.innerHTML= RetteInnerHTML;
        }
        else
        {
            // keinen Bold-Tag <B>gefunden
            var ZeigerAufNeuenSchriftStilTag =document.createElement("B");

            // komplettes ersetzen von Div-Text,
            ID_Div.replaceChild(ZeigerAufNeuenSchriftStilTag, KindZeigerAufDivText);
            ZeigerAufNeuenSchriftStilTag.innerHTML= RetteInnerHTML;
        }
    }
</SCRIPT>
</HEAD>
<BODY>
    <DIV ID="ID_Div" onclick=" Ersetze()">
```


```
                    Klicke für den Wechseln des <B>Schriftstils<B>
                </DIV>
        </BODY>
```

.replaceNode()

Objekt durch anderes Objekt komplett ersetzen und Referenz auf das komplett ersetzte Objekt liefern
sichtbar erst mit parsen des Endetags
DOM wird geändert

Beispiel:

```
<SCRIPT>
        function Ersetze()
        {
                var RetteInnerHTML = Liste.innerHTML;
                var ZeigerAufNeuenKnoten = document.createElement("OL");
                Liste.replaceNode(ZeigerAufNeuenKnoten);
                ZeigerAufNeuenKnoten.innerHTML = RetteInnerHTML;
        }
</SCRIPT>
<UL ID = "Liste" >
        <LI>Listeneintrag 1
        <LI>Listeneintrag 2
        <LI>Listeneintrag 3
        <LI>Listeneintrag 4
</UL>
<INPUT TYPE = button VALUE = "Ersetze" onclick = "Ersetze()">
```

.setAttribute()

Wert von vorhandenem Attribut setzen
wenn Attribut nicht vorhanden, so wird es automatisch erzeugt und mit dem Wert gefüllt
DOM wird nur bei Erzeugung geändert

Beispiel:

```
<HTML>
<HEAD>
<STYLE>
        .user_data_speicher_klasse {behavior:url(#default#userData);}
</STYLE>
<SCRIPT>
        // Cachname festlegen
        //      es können diverse Cachenamen definiert und somit Versionen von Cache
        //              verwaltet werden
        var FreierCacheName = "InputCache";

        // Cache-Attribut festlegen
        //      es können diverse Attribute definiert und somit Versionen von Input-Daten
        //              verwaltet werden
        var FreiesCacheAttribut = "InputCacheAttribut";

        // zu cachende Daten referenzieren
        var InputDatenObjekt = ID_Formular.ID_Input;

        function InputSichern()
        {
                // +++++++++ Zeitstempel ++++++++++++++++++++++++
                var ZeitpunktJetzt = new Date();
                var ZeitpunktJetztInMinuten = ZeitpunktJetzt.getMinutes();

                // Zeitstempel festlegen: Ab jetzt + 20 Minuten
                var Zeitstempel = ZeitpunktJetztInMinuten + 20;

                // und als UTC-Format erzeugen
                var ZeitstempelUTC = Zeitstempel.toUTCString();

                // und Zeitstempel dem Input-Objekt verpassen
                ID_Input.expires = ZeitstempelUTC;

                // ++++++++ Daten chachen +++++++++++++++++++++
                // aktuelle Daten holen laut Input-Objekt
                var InputDaten = InputDatenObjekt.value;
```


```
                        // Attrbibut instanzieren und mit Daten füllen
                        InputDatenObjekt.setAttribute(FreiesCacheAttribut, InputDaten);

                        // und Cache saven
                        InputDatenObjekt.save(FreierCacheName);
                }

                function InputLaden()
                {
                        // Cache laden
                        InputDatenObjekt.load(FreierCacheName);

                        // und Daten zum Attribut laut Sicherung lesen
                        var InputDaten = InputDatenObjekt.getAttribute(FreiesCacheAttribut);
                }

        </SCRIPT>
        </HEAD>
        <BODY>
                <FORM ID="ID_Formular">
                        <INPUT   ID="ID_Input"
                                CLASS="user_data_speicher_klasse"
                                TYPE="text"
                        >
                        <INPUT TYPE="button" VALUE="sichern der Input-Daten" onclick="InputSichern()">
                        <INPUT TYPE="button" VALUE="laden der Input-Daten"    onclick="InputLaden()">
                </FORM>
        </BODY>
        </HTML>
```

.setAttributeNode()

Attribut einem Knoten zuweisen und Referenz liefern
DOM wird geändert

Beispiel:

```
        <HTML>
        <HEAD>
        <SCRIPT>
                function Hinzufuegen()
                {
                        // Attribut mit Wert erzeugen
                        var ZeigerAufAttribut = document.createAttribute("title");
                        ZeigerAufAttribut.value = "Tooltip-Text";

                        // Attribut als Knoten erzeugen
                        var AttributKnoten = ID_Div.setAttributeNode(ZeigerAufAttribut);
                }
        </SCRIPT>
        </HEAD>
        <BODY onload="Hinzufuegen();">
                <DIV ID="ID_Div">Es wird ein Tooltip-Text hinzugefuegt</DIV>
        </BODY>
        </HTML>
```

.setExpression()

Wert definieren, der als Ausdruck für die Methode .getExpression() zur Berechnung einer Style-Eigenschaft als Objektreferenz der Form objekt.style.eigenschaft. dient
Ausdruck nur als Script kodierbar
DOM wird nicht geändert

In nachfolgenden Beispielen wurde aus Platzgründen die Zeichenkette von STYLE umgebrochen, was eigentlich nicht zulässig ist, und es sind nicht alle SPAN kodiert.

Beispiel 1:

```
        <HTML>
        <HEAD>
        <SCRIPT LANGUAGE ="JScript">
                function width_init()
```


```
		{
			ID_Span.style.setExpression("width",
							"    trueBlueSpan.style.pixelWidth
							   + oldYellowSpan.style.pixelWidth
							",
							"jscript"
						       );
		}

		function Berechne()
		{alert(ID_Span.style.getExpression("width");}
	</SCRIPT>
	</HEAD>
	<BODY onload= width_init();>
		<SPAN	ID="ID_Span"
			STYLE="background-color:lightgreen;
				width:expression(	trueBlueSpan.style.pixelWidth
							+ oldYellowSpan.style.pixelWidth
						)
			"
		>
		</SPAN>
		<BUTTON onclick= Berechne();>
	</BODY>
	</HTML>
```

Beispiel 2:

```
	ID_Span.style.setExpression("height","document.style.fontSize + 13");
	ID_Span.style.setExpression("width","document.body.style.fontSize");
	<SAPN ID="ID_Span"
		STYLE="background-color:lightgreen;
			width:expression(	trueBlueSpan.style.pixelWidth
						+ oldYellowSpan.style.pixelWidth
					)
		"
	>
	</SPAN>
```

Beispiel 3 für Sekundenbalken:

```
	<HTML>
	<HEAD>
	<STYLE>
		BUTTON {font-size:14;width:150}
	</STYLE>
	<SCRIPT>
		var timerID = null;
		var Zahler = 0;

		function Init()
		{
			// DIV-Eigenschaften festlegen

			// DIV-Breite je nach Sekundenanzahl, also dynamische DIV-Breite als Balken
			ID_Div1.style.setExpression("width","Zahler *10");

			// DIV-Inhalt als Sekundentext, der permanent aktualisiert wird
			ID_Div2.setExpression("innerText","Zahler.toString()");
		}

		function Uhr()
		{
			// Sekunden kumulieren
			Zahler ++;

			// und Anzeige neu berechnen
			document.recalc();
		}

		function Starten()
		{
			if (timerID == null)
			{
```


```
                                        // Start-Button nicht aktivierbar machen
                                        ID_Button1.disabled = true;

                                        // Stop-Button aktivierbar machen
                                        ID_Button2.disabled = false;

                                        // Uhr neu starten
                                        timerID = setInterval("Uhr()", 1000);
                                }
                        }

                        function Stoppen()
                        {
                                if (timerID != null)
                                {
                                        clearInterval(timerID);
                                        timerID = null;
                                        ID_Button1.disabled = false;
                                        ID_Button2.disabled = true;
                                }
                        }

                        function ZurueckSetzen()
                        { Zahler = 0; }
                </SCRIPT>
                </HEAD>
                <BODY onload="Init()">
                        <DIV ID="ID_Div1"  STYLE="background-color:lightblue" ></DIV>
                        <DIV ID="ID_Div1"  STYLE="color:hotpink;font-weight:bold"></DIV>
                        <BR>
                        <BUTTON         ID="ID_Button1"
                                        onclick="Starten()"
                        >
                                Start
                        </BUTTON>
                        </BR>
                        <BUTTON         ID="ID_Button2"
                                        DISABLED="true"
                                        onclick="Stoppen()"
                        >
                                Stop
                        </BUTTON>
                        <BR>
                        <BUTTON         ID="ID_Button3"
                                        onclick="ZurueckSetzen()"
                        >
                                Reset
                        </BUTTON>
                        <BR>
                        <P      STYLE="width:200;color:white;background-color:gray">
                                Sekundenbalken
                        </P>
                </BODY>
                </HTML>
```

.swapNode()

Positionen von 2 Knoten im DOM tauschen (Zeigertausch)

nur sichtbar wenn Endetag geparst

DOM wird geändert

Beispiel:

```
        <SCRIPT>
                function Tauschen()
                {Liste.children(0).swapNode(Liste.children(1)); }
        </SCRIPT>
        <UL ID = Liste>
                <LI>Listeneintrag 1
                <LI>Listeneintrag 2
                <LI>Listeneintrag 3
                <LI>Listeneintrag 4
        </UL>
        <INPUT TYPE = button VALUE = "Tauschen" onclick = "Tauschen()">
```


Collectionen zur Verwaltung des HTML-DOM im Internet Explorer

Nachfolgende Collectionen sind nicht in allen Objekten implementiert (siehe einzelne Objekte).

Die Collectionen werden in ihren Eigenschaften und Methoden beschrieben. Aus Platzgründen sind Beispiele z.T. in den Anhang verlegt worden. Das gilt auch für identische Eigenschaften und Methoden der verschiedenen Collectionen.

attributes Collection des HTML-DOM im Internet Explorer

Diese Collection sammelt die Zeiger aller Attribute eines HTML-Elementes.

Hinweis: Es gibt noch das Objekt attribute, also ohne den Buchstaben S. Dieses Objekt dient zur Verwaltung von Attributen eines Objektes.
Die Collection attribute**s**, also mit Buchstabe S, ist die Zeigersammlung von einzelnen attribute Objekten.

Syntax:

[var FeldZeiger =] object.attributes
[var FeldElementZeiger =] object.attributes**[**Index**]**

Index: Integer und ab 0
muss in **[ ]** kodiert sein

Beispiel:

```
<SCRIPT>
    function ShowAttribs(ZeigerAufObjekt)
    {
        var ZeigerAufFeld  = ZeigerAufObjekt.attributes;

        for (var Index = 0; Index < ZeigerAufFeld.length; Index ++)
        {
            var ZeigerAufFeldElement  = ZeigerAufFeld [Index];

            alert(      ZeigerAufFeldElement.nodeName
                    + '='
                    + ZeigerAufFeldElement.nodeValue
                    + ' ('
                    + ZeigerAufFeldElement.specified
                    + ') '
                );
        }
    }
</SCRIPT>
```

Eigenschaften:

.length — Anzahl der Feldelemente also Feldlänge

Methoden:

.getNamedItem() — Zeiger auf Attribut liefern anhand des Attributnamen (analog zu ID oder NAME-Attribut)
ab IE 6.x

Beispiel:

```
<HTML>
<HEAD>
<SCRIPT>
    function Init()
    {
        var ZeigerAufFeld = ID_P.attributes;
        var ZeigerAufFeldElement = ZeigerAufFeld.getNamedItem("align");
        alert("ALIGN Attribut Wert =  " + ZeigerAufFeldElement.value);
    }
</SCRIPT>
</HEAD>
<BODY ONLOAD="Init()">
    <P ID="ID_P" ALIGN="center">Test</P>
</BODY>
</HTML>
```

.item() — Referenz auf Feldelement anhand des Integer-Indexes oder des Attributnamen (analog zu ID oder NAME-Attribut) liefern
außer bei Formular mit <INPUT TYPE=image ...>
da dafür die children-Collection verwendet werden muss !!!

Beispiel 1:

```
<SCRIPT LANGUAGE="JScript">
    var ZeigerAufCollectionDocumentAll = document.all;

    if (ZeigerAufObjekt!=null)
    {
```


```
                    for (i=0; i< ZeigerAufCollectionDocumentAll.length; i++)
                    {alert(ZeigerAufCollectionDocumentAll.item(i).tagName);}
                }
            </SCRIPT>
```

Beispiel 2:

```
            <HTML>
            <HEAD>
            <SCRIPT>
                function Init()
                {
                    var ZeigerAufFeld = ID_P.attributes;
                    for (Index = 0; Index < ZeigerAufFeld.length; Index ++)
                    {
                        var ZeigerAufFeldElement  = ZeigerAufFeld.item(Index);
                                                                    // hier numerischer Index
                        var AttributWertSpezifiziert = ZeigerAufFeldElement.specified;
                                                                    // true oder false
                        var KnotenName = ZeigerAufFeldElement.nodeName;
                                                                    // String
                        var KnotenWert = ZeigerAufFeldElement.nodeValue;
                        alert(      "Knotenname = "
                                + KnotenName
                                + "     mit spezified = "
                                + AttributWertSpezifiziert
                                + "     und Wert = "
                                + KnotenWert
                            );
                    }
                }
            </SCRIPT>
            </HEAD>
            <BODY ONLOAD="Init()">
                <P ID="ID_P">Test</P>
            </BODY>
            </HTML>
```

.removeNamedItem() Attribut entfernen anhand Attributname (analog zu ID oder NAME-Attribut),
wobei danach der Standard-Attributwert automatisch weiterverwendet wird
(falls Standard vorhanden ist),
und Zeiger auf gelöschtes Attribut liefern
ab IE 6

Beispiel:

```
            <HTML>
            <HEAD>
            <SCRIPT>
                function Entfernen()
                {
                    var ZeigerAufFeld = ID_Div.attributes;
                    ZeigerAufFeld.removeNamedItem("TITLE");
                }
            </SCRIPT>
            </HEAD>
            <BODY>
                <DIV onclick="Entfernen();" ID="ID_Div" TITLE="Tooltip-Text ">
                    Klick um den Tooltip-Text zu entfernen
                </DIV>
            </BODY>
            </HTML>
```

.setNamedItem() Attribut hinzufügen anhand Zeiger auf Attribut
wenn noch nicht im Feld vorhanden, so Anhängen an das Feldende
wenn schon im Feld vorhanden, so überschreiben und Referenz auf das
überschriebene Attribut (vor dem Überschreiben) liefern
Bsp: Attribut erzeugen document.createAttribute("title");
Hinweis: in HTML können Attributnamen groß oder klein geschrieben werden
ab IE 6

Beispiel:

```
            <HTML>
            <HEAD>
            <SCRIPT>
                function Hinzufuegen()
                {
```

```
                        // Attribut mit Wert erzeugen
                        var ZeigerAufAttribut = document.createAttribute("title");
                        ZeigerAufAttribut.value = "Tooltip-Text ";

                        // Attribut als Feldelement anhängen
                        var ZeigerAufFeld = ID_Div.attributes;
                        ZeigerAufFeld.setNamedItem(ZeigerAufAttribut);
                }
        </SCRIPT>
        </HEAD>
        <BODY onload="Hinzufuegen();">
                <DIV ID="ID_Div">Es wird ein Tooltip-Text hinzugefuegt</DIV>
        </BODY>
        </HTML>
```

childNodes Collection des HTML-DOM im Internet Explorer

Feld der Zeiger aller Kinder-Knoten eines Objektes, also Feld der Zeiger aller HTML-Knoten-Kinder **und** Textknoten-**Kinder** des Objektes
Collection dient zur Ermittlung **nur von Knoteneigenschaften laut DOM**, falls diese im Element implementiert sind:

Elementefolge NICHT laut HTML-Coding sondern laut DOM.

Element (Knoten) kann per HTML oder Methode .createElement() erzeugt worden sein (falls Methode erlaubt ist).
Diese Collection sammelt die Zeiger aller Kinder**knoten** eines HTML-Elementes.

Syntax:

```
[ var ZeigerAufFeld = ] object.childNodes
[ var ZeigerAufFeldElement = ] object.childNodes[Index]
```

object	Zeiger auf Elternobjekt
Index	Integer und ab 0 muss in **[ ]** kodiert sein
ZeigerAufFeldElement	ist null, wenn Feldelement nicht vorhanden

Zugriff auf Element:

Je nach Art des Kind-Elementes stehen dem Kind Eigenschaften und Methoden zur Verfügung (sieh Objektbeschreibungen):

```
object.childNodes[Index].eigenschaft_des_kindes
object.childNodes[Index].methode_des_kindes
```

object	Zeiger auf Elternobjekt
Index	Integer und ab 0 muss in **[ ]** kodiert sein

Beispiel 1:

```
        <SCRIPT>
                var ZeigerAufFeld = ID_Body.childNodes;
        </SCRIPT>
        <BODY ID="ID_Body">
                <SPAN >Test </SPAN>
        </BODY>
```

Beispiel 2:

```
        // DIV erzeugen
        var ZeigerAufDivKnoten = document.createElement("DIV");

        // B-Tag im DIV erzeugen
        var ZeigerAufBKnoten = document.createElement("B");
        ZeigerAufDivKnoten.insertBefore(ZeigerAufBKnoten);

        // erst jetzt das DIV in den BODY einfügen, also DIV sichtbarmachen
        document.body.insertBefore(ZeigerAufDivKnoten);

        // Collection referenzieren
        var ZeigerAufFeld = ZeigerAufDivKnoten.childNodes;
```

Beispiel 3:

```
        <SCRIPT LANGUAGE="JScript">
                var ZeigerAufFeldAllerPTag = document.all.tags("P");
                if (ZeigerAufFeldAllerPTag!=null)
                {
                        for (i=0; i< ZeigerAufFeldAllerPTag.length; i++)
                        { ZeigerAufFeldAllerPTag [i].style.textDecoration="underline";} // auch .item(i) kodierbar
                }
```


```
</SCRIPT>
```

Beispiel 4:

```
<SCRIPT>
    var ErstesKind_Index = 0;        // Index ab 0
    var ErstesKind_Name = Liste.childNodes(ErstesKind_Index).nodeName;
    alert(ErstesKind_Name);          // liefert den Tagnamen 'LI' von Listenelement 1
                                     //  Hinweis: Listenelement 1 ist der Wert des Kindes
</SCRIPT>
<BODY>
    <UL ID = "Liste">
        <LI> Listenelement 1
        <LI> Listenelement 2
        <LI> Listenelement 3
    </UL>
<BODY>
```

Beispiel 5:

```
<SCRIPT>
    function KnotenWertAendern(   ZeigerAufListe,
                                  IndexVonListenElement,        // immer ab 0
                                  Zeichenkette                  // Listenlement muss Text sein
                              )
    {
        var ReturnWert=false;           // Annahme: Änderung schlägt fehl

        // prüfen auf UL-Tag
        if (ZeigerAufListe.nodeName == "UL")
        {
            // Anzahl der Listenelemente holen
            var AnzahlListenelemente= ZeigerAufListe.childNodes.length;
                                                                  // immer ab 1
            //  und Anzahl und Index prüfen
            if (    (AnzahlListenelemente > 0)        // immer ab 1
                 && (IndexVonListenElement >= 0)              // immer ab 0
                 && (IndexVonListenElement < AnzahlListenelemente)
                                                  // Index ist zulässig zur Anzahl
               )
            {
                // Zeiger auf das Listenelement laut Index holen
                var ZeigerAufListenElement =
                    ZeigerAufListe.childNodes(IndexVonListenElement);

                // existiert das Listenelement ?
                if (ZeigerAufListenElement)
                {
                    // ZeigerAufListenElement ist nicht null

                    // Listenelement ist Textelement ?
                    if (ZeigerAufListenElement.nodeType == 3)
                    {
                        ZeigerAufListenElement.nodeValue =
                                                    Zeichenkette;
                        ReturnWert =true;
                    }
                }
            }
        }

        return ReturnWert;
    }
</SCRIPT>
<UL ID="Liste" onclick=" KnotenWertAendern(this, 0, 'Listenelement Neu')">
    <LI>Listenelement alt
</UL>
```

Beispiel 6:

```
<SCRIPT>
    function TextElementAendern()
    {
        var ZeigerAufTextElement = document.createTextNode("Neuer Text");
        var ZeigerAufSpanInhalt = ID_Span.childNodes(0);
        ZeigerAufSpanInhalt.replaceNode(ZeigerAufTextElement);
```

```
		}
	</SCRIPT>
	<SPAN ID = "ID_Span" onclick=" TextElementAendern()">
		Original Text
	</SPAN>
```

Beispiel 7:

```
	<SCRIPT>
		function Anzeige()
		{
			var ZeigerAufOnClickEventQuelle=event.srcElement;
			var Feld =
			    ZeigerAufOnClickEventQuelle.parentElement.getElementsByTagName("LI");
			alert(		"Anzahl LI : "
				+ Feld.length
				+ "\nErster Eintrag: "
				+ Feld [0].childNodes[0].nodeValue
			    );
		}
	</SCRIPT>
	<UL onclick="Anzeige()">
		<LI>Menuepunkt 1
		<UL>
			<LI> Menuepunkt 1.1
				<OL>
					<LI> Menuepunkt 1 1.1
					<LI> Menuepunkt 1 1.2
				</OL>
			<LI> Menuepunkt 1.2
			<LI> Menuepunkt 1.3
		</UL>
		<LI> Menuepunkt 2
		<UL>
			<LI> Menuepunkt 2.1
			<LI> Menuepunkt 2.3
		</UL>
		<LI> Menuepunkt 3
	</UL>
```

Eigenschaften:

.length Anzahl der Feldelemente also Feldlänge

Methoden:

.item() Referenz auf Feldelement anhand des Integer-Indexes oder des Attributnamen (analog zu ID oder NAME-Attribut) liefern außer bei Formular mit <INPUT TYPE=image ...> da dafür die children-Collection verwendet werden muss !!!

Beispiel 1:

```
	<SCRIPT LANGUAGE="JScript">
		var ZeigerAufCollectionDocumentAll = document.all;

		if (ZeigerAufObjekt!=null)
		{
			for (i=0; i< ZeigerAufCollectionDocumentAll.length; i++)
			{alert(ZeigerAufCollectionDocumentAll.item(i).tagName);}
		}
	</SCRIPT>
```

Beispiel 2:

```
	<HTML>
	<HEAD>
	<SCRIPT>
		function Init()
		{
			var ZeigerAufFeld = ID_P.attributes;
			for (Index = 0; Index < ZeigerAufFeld.length; Index ++)
			{
				var ZeigerAufFeldElement  = ZeigerAufFeld.item(Index);
				                                        // hier numerischer Index
				var AttributWertSpezifiziert = ZeigerAufFeldElement.specified;
				                                        // true oder false
				var KnotenName = ZeigerAufFeldElement.nodeName;
				                                        // String
				var KnotenWert = ZeigerAufFeldElement.nodeValue;
				alert(		"Knotenname = "
```


```
                                                + KnotenName
                                                + "     mit spezified = "
                                                + AttributWertSpezifiziert
                                                + "     und Wert = "
                                                + KnotenWert
                                        );
                                }
                        }
                </SCRIPT>
                </HEAD>
                <BODY ONLOAD="Init()">
                        <P ID="ID_P">Test</P>
                </BODY>
                </HTML>
```

.urns() Referenz auf Feld aller Elemente mit gemeinsamer URN liefern

Beispiel:

```
<SCRIPT LANGUAGE="JScript">
        var ZeigerAufFeldAllerURN1 coll  = document.all.urns("URN1");
        var Text = "";

        if (ZeigerAufFeldAllerURN1 != null)
        {
                for (i=0; i< ZeigerAufFeldAllerURN1.length; i++)
                {Text += ZeigerAufFeldAllerURN1.item(i).id + ', ';}
                alert (Text);
        }
</SCRIPT>
```

children Collection des HTML-DOM im Internet Explorer

Feld der Zeiger aller **HTML-Elemente-Kinder** eines Objektes
Collection dient **auch** zur Ermittlung von Knoteneigenschaften, wenn diese im Element implementiert sind.
Elementefolge laut HTML-Coding und nicht laut DOM
Element kann per HTML oder Methode .createElement() erzeugt worden sein (falls Methode erlaubt ist).
Für das Objekt form.input image **muss** die children Collection verwendet werden.

Syntax:

[var ZeigerAufFeld =] object.children
[var ZeigerAufFeldElement =] object.children**[**Index [, SubIndex] **]**

object		Zeiger auf Elternobjekt	
Index			Integer ab 0
	oder	String	Name oder ID des Elementes laut ID-Attribut bzw. NAME-Attribut muss in **[ ]** kodiert sein
SubIndex		optional Integer Unterindex also Unterelement eines Elementes nur kodieren wenn Index ein String ist	
ZeigerAufFeldElement		ist null, wenn Feldelement nicht vorhanden	

Beispiel 1:

```
<SCRIPT>
        var ZeigerAufFeld = ID_Body.children;
</SCRIPT>
<BODY ID="ID_Body">
        <SPAN >Test </SPAN>
</BODY>
```

Beispiel 2:

```
<SCRIPT>
        function Loeschen()
        {
                ID_Span.children[0].clearAttributes();
        }
</SCRIPT>
<SPAN ID="ID_Span">
        <DIV    ID="ID_Div"
                ATTRIBUTE1="true"
                ATTRIBUTE2="true"
                onclick="alert('click');"
                onmouseover="this.style.color='#0000FF';"
```

```
                    onmouseout="this.style.color='#000000';"
            >
                    Test eines<B>Div</B>Elementes.
            </DIV>
    </SPAN>
    <INPUT  TYPE="button" VALUE=" Loeschen" onclick="Loeschen()">
```

Beispiel 3:

```
    <SCRIPT>
            function Hinzufuegen()
            {
                    var ZeigerAufLI = document.createElement("LI");
                    ID_Liste.children(0).insertAdjacentElement("beforeBegin", ZeigerAufLI);
                    ZeigerAufLI.innerText = "Listeneintrag 0";
            }
    </SCRIPT>
    <BODY>
            <OL ID = "ID_Liste ">
                    <LI>Listeneintrag 1</LI>
                    <LI>Listeneintrag 2</LI>
                    <LI>Listeneintrag 3</LI>
            </OL>
            <INPUT  TYPE = "button" VALUE = " Hinzufuegen" onclick=" Hinzufuegen()">
    </BODY>
```

Beispiel 4:

```
    <SCRIPT>
            function Tauschen()
            {Liste.children(0).swapNode(Liste.children(1)); }
    </SCRIPT>
    <UL ID = Liste>
            <LI>Listeneintrag 1
            <LI>Listeneintrag 2
            <LI>Listeneintrag 3
            <LI>Listeneintrag 4
    </UL>
    <INPUT TYPE = button VALUE = "Tauschen" onclick = "Tauschen()">
```

Beispiel 5:

```
    <HEAD>
    <SCRIPT>
            function Entfernen()
            {
                    // versuche den Text zu entfernen
                    try
                    {
                            var KindZeigerAufTextImDiv = ID_Div.children(0);
                            ID_Div.removeChild(KindZeigerAufTextImDiv);
                            // Achtung: Der Text ist noch sichtbar !!!!
                    }
                    // oder fange das Ereignis des bereits entfernten Textes ein
                    //                      und behandle das Ereignis
                    catch(x)
                    {
                            alert(          "Text wurde entfernt !\n"
                                    + "Das Dokument muss neu geladen werden !
                                );
                            document.location.reload();
                    }
            }
    </SCRIPT>
    </HEAD>
    <BODY>
            <DIV ID="ID_Div" onclick=" Entfernen()">
                    Klick, um diesen Text zu entfernen !
            </DIV>
    </BODY>
```

Beispiel 6:

```
    <HEAD>
    <SCRIPT>
            function Ersetze()
            {
```


```
                                        var KindZeigerAufDivText = ID_Div.children(0);
                                        var RetteInnerHTML = KindZeigerAufDivText.innerHTML;

                                        // prüfen auf Tag im Div-Text
                                        if (KindZeigerAufDivText.tagName=="B")
                                        {
                                                // Bold-Tag <B>gefunden, also I-Tag erzeugen
                                                var ZeigerAufNeuenSchriftStilTag =document.createElement("I");

                                                // komplettes ersetzen von Div-Text,
                                                ID_Div.replaceChild(ZeigerAufNeuenSchriftStilTag, KindZeigerAufDivText);
                                                ZeigerAufNeuenSchriftStilTag.innerHTML= RetteInnerHTML;
                                        }
                                        else
                                        {
                                                // keinen Bold-Tag <B>gefunden
                                                var ZeigerAufNeuenSchriftStilTag =document.createElement("B");

                                                // komplettes ersetzen von Div-Text,
                                                ID_Div.replaceChild(ZeigerAufNeuenSchriftStilTag, KindZeigerAufDivText);
                                                ZeigerAufNeuenSchriftStilTag.innerHTML= RetteInnerHTML;
                                        }
                                }
                        </SCRIPT>
                        </HEAD>
                        <BODY>
                                <DIV ID="ID_Div" onclick="Ersetze()">
                                        Klicke für den Wechseln des <B>Schriftstils<B>
                                </DIV>
                        </BODY>
```

Eigenschaften:

.length — Anzahl der Feldelemente also Feldlänge

Methoden:

.item() — Referenz auf Feldelement anhand des Integer-Indexes oder des Attributnamen (analog zu ID oder NAME-Attribut) liefern
außer bei Formular mit <INPUT TYPE=image ...>
da dafür die children-Collection verwendet werden muss !!!

Beispiel 1:

```
<SCRIPT LANGUAGE="JScript">
        var ZeigerAufCollectionDocumentAll = document.all;

        if (ZeigerAufObjekt!=null)
        {
                for (i=0; i< ZeigerAufCollectionDocumentAll.length; i++)
                {alert(ZeigerAufCollectionDocumentAll.item(i).tagName);}
        }
</SCRIPT>
```

Beispiel 2:

```
<HTML>
<HEAD>
<SCRIPT>
        function Init()
        {
                var ZeigerAufFeld = ID_P.attributes;
                for (Index = 0; Index < ZeigerAufFeld.length; Index ++)
                {
                        var ZeigerAufFeldElement  = ZeigerAufFeld.item(Index);
                                                                // hier numerischer Index
                        var AttributWertSpezifiziert = ZeigerAufFeldElement.specified;
                                                                // true oder false
                        var KnotenName = ZeigerAufFeldElement.nodeName;
                                                                // String
                        var KnotenWert = ZeigerAufFeldElement.nodeValue;
                        alert(          "Knotenname = "
                                + KnotenName
                                + "     mit spezified = "
                                + AttributWertSpezifiziert
                                + "     und Wert = "
                                + KnotenWert
                            );
                }
        }
```


```
</SCRIPT>
</HEAD>
<BODY ONLOAD="Init()">
        <P ID="ID_P">Test</P>
</BODY>
</HTML>
```

.tags() Referenz auf Feld aller HTML-Elemente mit gemeinsamen Tag-Namen liefern
siehe tags Collection des DOM

Beispiel:

```
<SCRIPT LANGUAGE="JScript">
        var ZeigerAufFeldAllerPTag = document.all.tags("P");
        if (ZeigerAufFeldAllerPTag!=null)
        {
                for (i=0; i< ZeigerAufFeldAllerPTag.length; i++)
                { ZeigerAufFeldAllerPTag [i].style.textDecoration="underline";}
        }
</SCRIPT>
```

.urns() Referenz auf Feld aller Elemente mit gemeinsamer URN liefern

Beispiel:

```
<SCRIPT LANGUAGE="JScript">
        var ZeigerAufFeldAllerURN1 coll  = document.all.urns("URN1");
        var Text = "";

        if (ZeigerAufFeldAllerURN1 != null)
        {
                for (i=0; i< ZeigerAufFeldAllerURN1.length; i++)
                {Text += ZeigerAufFeldAllerURN1.item(i).id + ', ';}
                alert (Text);
        }
</SCRIPT>
```

tags Collection des HTML-DOM im Internet Explorer

Diese Collection sammelt die Zeiger aller HTML-Elemente, die gemeinsamen HTML-Tag besitzen.
wird **nur** von Collectionen und Objekten instanziert, die die Methode .tags() besitzen

z.B. childNodes Collection des DOM
children Collection des DOM
document.all Collection des DOM

Zugriff auf das Feld **nur** über die Methode .tags()

Syntax:

[var Zeiger =] zeiger_auf_collection_oder_objekt.tags(Kette)

Kette String mit HTML-Tag-Bezeichner

Zeiger weist auf ein leeres Feld, wenn keine HTML-Elemente mit dem Tag gefunden wurden

Beispiel:

```
<SCRIPT LANGUAGE="JScript">
        var ZeigerAufFeldAllerPTag = document.all.tags("P");
        if (ZeigerAufFeldAllerPTag!=null)
        {
                for (i=0; i< ZeigerAufFeldAllerPTag.length; i++)
                { ZeigerAufFeldAllerPTag [i].style.textDecoration="underline";}
        }
</SCRIPT>
```

Eigenschaften:

.length Anzahl der Feldelemente also Feldlänge

Methoden:

keine

readyState und onreadystatechange im HTML-DOM des Internet Explorer

ReadyState-Eigenschaft dient der dynamischen Datenverwaltung von (HTML-)-Objekten z.B. wie die des XMLHttpRequest-Objektes (Letzteres hat Microsoft nach elender Wartezeit endlich in HTML ohne Active-X-Control eingeführt).

Man kann mit dem Internet Explorer schon seit ewig das HTML-Element BGSOUND für Media-Daten nicht nur per HTML-Attribute designen, sondern per readyState in Echtzeit direkt steuern (per Script).

Auch schon seit ewig kann man im Internet Explorer Script-Objekt per readyState steuern: Man kann so Scripte, die inhaltlich abhängig sind, in gezielter Reihenfolge landen und sogar in Echtzeit des Internets verfolgen, wann welche Daten eintreffen. XMLHttpRequest, das wesentlich später kam, ist also nicht die Voraussetzung von dynamischer Programmierung gewesen, sondern eine moderne Variante z.B. auf XML-Basis. Tipp: Man prüfe doch mal, ob sich das Laden des Objektes Body, also des Unterobjektes vom document-Objekt, per readyState steuern lässt, um z.B. das HTML-Dokument erst anzuzeigen, wenn sämtliche Elemente komplett geladen (deren readyState auf "complete") und gerendert sind (Anzeige des Dokumentes ohne Ruckeln und Zuckeln etc.). ReadyState muss mit Timer-Rekursionen bei Anzapfung der Ressourcen des Betriebssystemes programmiert werden: Timer erhöhen CPU-Bedarf (nicht nur der XMLHttpRequest selbst kann träge sein).

Ajax ist ebenfalls ein moderner Trittbrettfahrer von readyState, das für den clientzugroff auf Server verwendet wird: ReadyState als Teil der XMLHttpRequest-Komponente.

ReadyState gibt es im Internet Explorer seit der Version 4.

Es ist anzunehmen, dass Objekte, die das Event onreadystatechange besitzen, auch readyState haben. Man muss das aber an der konrketen Objektbeschreibung prüfen.

Folgende Objekte des IE besitzen onreadystatechange:

a
acronym
address
applet
area
b
base
basefont
bdo
bgsound
big
blockquote
body
br
button
caption
center
cite
code
col
colgroup
comment
dd
del
dfn
dir
div
dl
dt
em
embed
fieldset
font
form
h1
h2
h3
h4
h5
h6
head
hr
html
i
iframe

img
input
ins
isindex
kbd
label
legend
li
link
map
marquee
nobr
noframes
noscript
object
ol
optgroup
option
p
pre
q
rt
ruby
s
samp
script
select
small
span
strike
strong
style
sub
sup
table
tbody
td
textarea
tfoot
th
thead
title
tr
tt
u
ul
var
xml
xmp

window.XDomainRequest Objekt im HTML- und XML-DOM des IE (ab IE 8)

Objekt für domainübergreifender Zugriff über HTTP per XML
Sicherer Datenservice für jede Seite bei anonymen Zugriff auf jeden Server zum Zweck des XML-Datenaustausches zwischen Domains.
Hinweis: XMLHttpRequest hat eine Sicherheitskomponente implementiert: Domainübergriff nicht möglich.

Attribute:

contentType	Contenttyp-Header setzen bzw. liefern
responseText	Body der Antwort des Servers
timeout	Timeoutwert setzen bzw. liefern

Methoden:

abort()	beendet das aktive Senden / Empfangen
open()	erzeugt eine Verbindung zum Server
send()	Datenstring an den Server senden, der die Daten verarbeiten soll

Event:

onerror	tritt ein, wenn Zugriff auf Objekt-Daten nicht komplett werden kann
onload	tritt ein wenn Objekt-Daten vom Server komplett eingetroffen sind
onprogress	tritt ein, wenn der Browser beginnt, Objekt-Daten vom Server zu empfangen
ontimeout	tritt ein, wenn Zugriff auf Objekt-Daten wegen Überschreitung der Zeit laut Attribut timeout nicht komplett werden kann

Beispiel:

```
if (window.XDomainRequest)
{
  var oReq = new XDomainRequest();
  if (oReq) {
    oReq.open("GET", "http://example.com/test.xml");
    oReq.send();
    alert(oReq.statusText);
  }
}
```

Attribute

Attribut contentType

content-type header setzen oder liefern

Syntax:

```
[ v = ] XDomainRequest.contentType [ = v ]
```

v text des content-type header

lesen und schreiben
kein Standardwert

Beispiel:

```
// 1. Create XDR object
xdr = new XDomainRequest();
// 2. Open connection with server using POST method
xdr.open("POST", "http://www.contoso.com/xdr.txt");
// 3. Set the content-type
xdr.contentType = "text/plain";
// 4. Send string data to server
xdr.send("data to be processed");
```

Attribut responseText

Body der Antwort vom Server liefern
Zugriff nur möglich, wenn onprogress oder onload eingetreten ist (sonst erzeugt der Zugriff einen Ausnahmefehler erzeugt)
onload tritt nach onprogress ein
Body nur komplett verfügbar, wenn onload eingetreten ist (bei onprogress nicht vollständig verfügbar)

Syntax:

```
[ p = ] XDomainRequest.responseText
```

p String

Attribut timeout

Timeoutwert setzen oder liefern

maximale Wartezeit auf Antwort des Servers

Syntax:

[v =] XDomainRequest.timeout [= v]

v Millisekunden

lesen und schreiben
kein Standardwert

Methoden

Methode abort()

beendet ein Senden / Empfangen
nur aufrufbar nach senden und bevor onload-Event eintritt
ansonsten immer Error erzeugt

Syntax:
XDomainRequest.abort():

liefert nichts

Beispiel:

```
// 1. Create XDR object
xdr = new XDomainRequest();
// 2. Open connection with server using POST method
xdr.open("POST", "http://www.contoso.com/xdr.txt");
// 3. Send string data to server
xdr.send("data to be processed");
// 4. abort
xdr.abort();
```

Methode open()

Verbindung zum Server erzeugen

Syntax:
XDomainRequest.open(methode, url)

methode String
'GET'
'POST'

url String
Url vom Server

liefert nichts

Beispiel:

```
// 1. Create XDR object
xdr = new XDomainRequest();
// 2. Open connection with server using POST method
xdr.open("POST", "http://www.contoso.com/xdr.txt");
// 3. Send string data to server
xdr.send("data to be processed");
```

Methode send()

sendet einen Datenstring zum Server, der den String verarbeiten soll

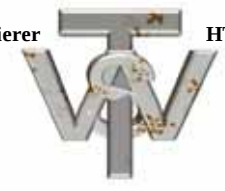
Syntax:
XDomainRequest.send([varBody])

varBody pptional
String
Standard ist ""

liefert nichts

Beispiel:

```
// 1. Create XDR object
xdr = new XDomainRequest();
```

```
// 2. Open connection with server using POST method
xdr.open("POST", "http://www.contoso.com/xdr.txt");
// 3. Send string data to server
xdr.send("data to be processed");
```

Events

Event onerror

tritt ein, wenn ein Fehler das Zugriff verhindert
welcher Fehler es ist, ist nicht ermittelbar
onerror wird nicht ausgelöst durch ontimeout

Syntax:

```
<SCRIPT FOR = XDomainRequest EVENT = onerror>
XDomainRequest.onerror=handler;
```

Event Information:

Bubbles No
Cancels No

Beispiel:

```
<script type="text/javascript">
function err()
{
   alert("XDR onerror");
}
...
xdr.onerror = err;
```

Event onload

tritt ein, wenn Objektdaten komplett vom Server empfangen wurden (nach einem send)
wenn eingetreten, dann ist responseText komplett gefüllt und zugreifbar

Syntax:

```
<SCRIPT FOR = XDomainRequest EVENT = onload>
XDomainRequest.onload=handler;
```

Event Information:

Bubbles No
Cancels No

Beispiel:

```
function loadd()
{
   alert("XDR onload");
   alert("Got: " + xdr.responseText);
}
...
xdr.onload = loadd;
```

Event onprogress

tritt ein, solange Browser Daten vom Server empfängt (nach send und vor Eintritt vom Event onload)
wenn eingetreten, dann ist responseText nicht komplett gefüllt

Syntax:

```
 <SCRIPT FOR = XDomainRequest EVENT = onprogress>
XDomainRequest.onprogress=handler;
```

Event Information:

Bubbles No
Cancels No

Beispiel:

```
function progres()
{
   alert("XDR onprogress");
   alert("Got: " + xdr.responseText);
}
```


```
...
xdr.onprogress = progres;
```

Event ontimeout

tritt ein, wenn Zugriff innerhalb Zeitspanne laut Attribut timeout nicht komplett erfolgt ist (nach send)
responseText ist nicht verfügbar, so dass ein Zugriff auf dieses Attribut einen Fehler erzeugt

Hinweis: Zugriff ist komplett erfolgt, wenn Event onload ausgelöst wurde

Syntax:

```
<SCRIPT FOR = object EVENT = ontimeout>
XDomainRequest.ontimeout=handler;
```

Event Information:

```
Bubbles No
Cancels No
```

Beispiel:

```
function timeo()
{
   alert("XDR ontimeout");
}
...
xdr.ontimeout = timeo;
```


window.XMLHttpRequest Objekt im HTML- und XML-DOM des IE (ab IE 7 bzw. 8)

Refrenz auf ein Objekt,
anhand dessen der XML-Datentransfer per HTTP möglich wird.
das erst erzeugt werden muss
ab IE 7, erweitert ab IE 8

Die XML-Datenbeschaffung über HTTP per Serverzugriff erfolgt, um vom Server empfangene Daten als XML von der Webseite verarbeiten zu lassen, wobei XML-Daten in den HTML-Kontext per XML-DOM konvertiert werden müssen (Daten als XML-DOM-Objekt lieferbar, die dann von der Webseite per Script anhand XML-DOM und HTML-DOM verarbeitbar sind). Man kann zur Erzeugung von HTML Extensible Stylesheet Language Transformations (XSLT) benutzen. Monderne Variante als Laufzeitsystem ist Ajax, das auch auf Server z.B. von Microsoft mit ASP-Zugang aufsetzen kann (je nach Ajax-Anbieter).

Will man das Objekt lokal testen, kann man das nur, wenn lokal ein Server läuft z.B. Apache, da der Serverdienst benötigt wird.

Hinweis: Mit window.XDomainRequest kann ein sicherer Datenservice für jede Seite bei anonymen Zugriff auf jeden Server zum Zweck des XML-Datenaustausches zwischen Domains implementiert werden.

Eigenschaften:

onreadystatechange	Eventhandler-Refenrenz für asynchronen HTTP-Zugriff setzen bzw. liefern
readyState	aktueller Status des Objektes z.B. des Zugriffes
responseBody	Körper der Antwortdaten liefern als Feld unsigned Bytes)
responseText	Körper der Antwortdaten liefern als String
responseXML	Körper der Antwortdaten liefern als eine Refrenz auf ein Objekt des XML-DOM
status	HTTP-Status-Code des Zugriffes liefern
statusText	HTTP-Status-Text des Zugriffes liefern
timeout	Timeout-Wert setzen bzw. liefern, erst ab IE 8

Methoden:

abort()	aktuellen Zugriff abbrechen
getAllResponseHeaders()	komplette Liste der Antwort-Kopf liefern
getResponseHeader()	einen bestimmten Anwort-Kopf liefern
open()	Zugriff creieren und öffnen, aber nicht starten Zuweisungmethode Ziel-Url Attribute des Zugriffes
send()	per geöffneten Zugriff Daten an den Server Senden und dann auf Antwort vom Server warten
setRequestHeader()	Benutzdefinierten HTTP-Kopf für einen Zugriff setzen

Event:

ontimeout	Event erzeugt, wenn Zugriffbeenddigung einen Timeout-Wert zeitlich überschritten hat erst ab IE 8

XMLHttpRequest bis zum IE 6 musste per Active-X-Control erfolgen:

```
var XMLHttpRequestObjekt = new ActiveXObject("MSXML2.XMLHTTP.3.0");
```

Beispiel für den IE7:

```
if (window.XMLHttpRequest)
{
  var XMLHttpRequestObjekt = new XMLHttpRequest();
  XMLHttpRequestObjekt.open("GET", "http://localhost/test.xml");
  XMLHttpRequestObjekt.send();
  alert(XMLHttpRequestObjekt.statusText);
}
```

Attribute

Attribut onreadystatechange

Eventhandler für asynchronen Zugriff zuweisen bzw. Referenz liefern

Syntax:

```
[ vHandler = ] XMLHttpRequestObjekt.onreadystatechange [ = vHandler ]

vHandler   Referenz auf Handler

lesen und schreiben
kein Standardwert
```


Beispiel:

```
function reportStatus()
{
   if (XMLHttpRequestObjekt.readyState == 4)
      alert('Transfer complete.');
}

var XMLHttpRequestObjekt = new XMLHttpRequest();
XMLHttpRequestObjekt.onreadystatechange = reportStatus;
XMLHttpRequestObjekt.open("GET", "http://localhost/test.xml", true);
XMLHttpRequestObjekt.send();
```

Attribut readyState

aktuellen Status des Zugriffes ermitteln
responseText und responseBody nur auswertbar, wenn Status 4 anliegt (ansonsten erzeugt ein Zugriff auf die beiden Eigenschaften eventuell einen Fehler)

Syntax:

[nState =] XMLHttpRequestObjekt.readyState

nState Integer
0 Objekt erzeugt und nicht initialisiert (open() bisher nicht aktiviert worden)
1 Objekt ist offen und bisher kein send() aktiviert worden
2 Objekt sendet (send() wurde aktiviert)
3 Objekt empfängt da Daten eingehen
4 Objekt hat empfangen beendet und alle Daten sind eingetroffen
responseText ist auswertbar
responseBody ist auswertbar

nur lesen
hat keinen Standardwert

Attribut responseBody

liefert den Antwort-Body als ein Feld aus unsigned Bytes

Syntax:

[vBody =] XMLHttpRequestObjekt.responseBody

vBody Feld aus unsigned Bytes

nur lesen
kein Standardwert

Attribut responseText

liefert den Antwort-Body als String

Syntax:

[sBody =] XMLHttpRequestObjekt.responseText

sBody String des Antwort-Body

nur lesen
kein Standardwert

Attribut responseXML

liefert den Anwortbody als XML-DOM-Objekt
Es können sämtliche Methoden zum Objekt laut XML-DOM aktiviert werden, da XML-DOM und HTML-DOM parallel nutzbar sind per Script.

Syntax:

[oXMLDocument =] XMLHttpRequestObjekt.responseXML

oXMLDocument Referenz aufXML-DOM-Objekt

nur lesen
kein Standardwert

Beispiel:

```
var XMLHttpRequestObjekt = new XMLHttpRequest();
```


```
XMLHttpRequestObjekt.open("GET", "http://localhost/books.xml", false);
XMLHttpRequestObjekt.send();
alert(XMLHttpRequestObjekt.responseXML.xml);
```

Attribut status

liefert den HTTP-Statuscode des Zugriffes

Syntax:

```
[ nStatus = ] XMLHttpRequestObjekt.status

nStatus     Integer
```

100	Continue
101	Switching protocols
200	OK
201	Created
202	Accepted
203	Non-Authoritative Information
204	No Content
205	Reset Content
206	Partial Content
300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
307	Temporary Redirect
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Request Entity Too Large
414	Request-URI Too Long
415	Unsupported Media Type
416	Requested Range Not Suitable
417	Expectation Failed
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported

```
nur lesen
kein Standardwert
```

Beispiel:

```
if (XMLHttpRequestObjekt.status == 401)
   alert('Access denied.');
else
   alert(XMLHttpRequestObjekt.responseText);
```

Attribut statusText

liefert HTTP-Status des Zugriffes als Text

Syntax:

```
[ sStatus = ] XMLHttpRequestObjekt.statusText

sStatus     String

nur lesen
kein Standardwert
```

Beispiel:

```
XMLHttpRequestObjekt.open("GET", "http://localhost/test.xml", false);
```


```
XMLHttpRequestObjekt.send()
if (XMLHttpRequestObjekt.statusText == "OK")
  alert(XMLHttpRequestObjekt.responseText);
else
  alert(XMLHttpRequestObjekt.statusText);
```

Attribut timeout erst ab Internet Explorer 8

setzen oder liefern des Timeout-Wertes als Wartezeit des Browsers auf Antwort des Servers.
Wird innerhalb der Wartetzeit keine Antwort an den Browser geliefert, dann wird responseText auf null-Zeiger gesetzt.
Wert nur setzbar nach open() und vor send()

Syntax:

[v =] XMLHttpRequestObjekt.timeout [= v]

v Integer
Miilisekunden als Wartezeit des Browser auf Antwort des Servers
Standard ist 0
Wert nur setzbar nach open() und vor send()

lesen und schreiben

Methoden

Methode abort()

aktuellen Zugriff abbrechen
aktuellen onreadystatechange-Event-Handler dekativieren und entfernen
readyState auf 0 setzen (Objekt ist uninitialisiert).

Syntax: XMLHttpRequestObjekt.abort();

liefert nichts

Methode getAllResponseHeaders()

liefert die komplette Liste der Antwort-Header
Liste ist Text
Jedes Paar aus Name und Wert ist begrenzt von CRLF (Return mit Linefeed)

Syntax:

sHeaders = XMLHttpRequestObjekt.getAllResponseHeaders();

liefert String

Methode getResponseHeader()

liefert einen bestimmten Antwort-Header

Syntax:

p = XMLHttpRequestObjekt.getResponseHeader(bstrHeader)

bstrHeader String
muss kodiert werden
Name des Antwort-Headers

liefert String

Methode open()

Request öffnen, ermöglicht danach send()
Methode des Zugriffes zuweisen
gemischte Protokolle sind nicht erlaubt
Protokollwechsel nicht erlaubt
Quelle und Ziel des Datentransportes müssen identisches Protokoll benutzen
Ziel-Url des Zugriffes zuweisen
Domainbereich verlassen ist nicht erlaubt
nur Dateien innerhalb derselben Domaine
Quelle und Ziel des Datentransportes müssen identische Domain benutzen
Eigenschaften des Zugriffes zuweisen

Daten werden im Cache (Temporary Internet Files) des Internet Explorers
gepuffert bei Operation "GET"

nicht gepuffert bei Operation "POST"

Userkennung und Password (beide optional) werden erst dann übertragen, wenn die Verbindung bereits aktiv ist
Erst nach open() ist Senden möglich, da die Verbindung dann steht.

Syntax:

XMLHttpRequestObjekt.open(sMethod, sUrl [, bAsync] [, sUser] [, sPassword])

sMethod muss kodiert werden
HTTP-Methode zum Öffnern der HTTP-Connection
Gros-kleinschreibung egal
Quelle und Ziel des Datentransportes müssen identisches Protokoll benutzen

"GET"	Request URI
"POST"	Send data to server
"HEAD"	Request URI without body
"PUT"	Store data for URI
"DELETE"	Delete data for URI
"MOVE"	Move URI to to new location
"PROPFIND"	Request URI Properties
"PROPPATCH"	Update or Delete URI Properties
"MKCOL"	Create collection at URI
"COPY"	Create copy of URI
"LOCK"	Create Lock
"UNLOCK"	Remove Lock
"OPTIONS HTTP"	Request URI Options

sUrl muss kodiert werden
absolute oder relative URL der XML daten
des XML Web Services auf dem Server
Quelle und Ziel des Datentransportes müssen identische Domain benutzen

bAsync optional
boolean
true asynchroner Zugriff
Es muss eine Handler per onreadystatechange eingebunden sein, der feststellt, wann der Zugriff komplett ist (readyState benutzen)
Standardwert
false synchroner Zugriff
Browser wartet direkt auf Ende des Zugriffes, so dass der Browser weder Eingaben akzeptiert noch Ausgaben produziert.
Die HTTP-Verbindung sollte also zuverlässig und schnell sein.

sUser optional
String
Name des Users, der
den Zugriff auslöst
sich einloggen muss
wenn Name kodiert so einloggen mit Zugriff automatisch
wenn keine Name oder Leerkette "" kodiert so Login-Fenster angezeigt.

sPassword
optional
String
Password für Autentifizierung
wenn nicht kodiert oder Leerkette "" so kein Password z.B. per Loginfenster verlangt

liefert nichts

Methode send()

Einen HTTP-Zugriff an den Server senden und auf Antwort des Servers warten.
Der Zugriff erfolgt je nach open() synchron oder ansynchron (Art des Wartens auf Anwort, siehe open())

Syntax:

XMLHttpRequestObjekt.send([varBody])

varBody optional
Körper der Meldung, die mit dem Zugriff gesendet wird
String oder Feld aus unsigned Bytes oder Zeiger auf XML-DOM-Objekt

liefert nichts

Methode setRequestHeader()

einen benutzerdefinierten Header dem Zugriff hinzufügen (für nächstes send() bereitstellen)

Syntax:

zeiger_auf_objekt.setRequestHeader(sName, sValue)

sName muss kodiert werden, 1. Teil des Paares
String
Name des Headers

sValue muss kodiert werden, 2. Teil des Paares
String
Wert des Headers

liefert nichts

Beispiel: HTTP Content-Type Header auf 'text/xml' setzen, bevor der Sendezugriff erfolgt (request body)

```
var XMLHttpRequestObjekt = new XMLHttpRequest();
XMLHttpRequestObjekt.open("POST", sURL, false);
XMLHttpRequestObjekt.setRequestHeader("Content-Type", "text/xml");
XMLHttpRequestObjekt.send(sRequestBody);
```

Events

Event ontimeout

Event ausgelöst, wenn
ein Fehler im Zugriffverlauf erfolgt ist
timeout-Periode endete bevor ein onload-Event ausgelöst werden konnte
Wenn Event ausgeöst, so ist responseText nicht verfügbar (Zugriff auf responseText erzeugt Fehler).

Syntax:

<SCRIPT FOR = object EVENT = ontimeout>

Event Information

Bubbles: No
Cancels: No

Beispiel:

```
<script type="text/javascript">
function timeo()
{
   alert("XDR ontimeout");
}
...
xdr.ontimeout = timeo;
```

command Objekt zum HTML-DOM des Internet Explorer

Das Objekt ist ein symbolisches Objekt zum Verwalten von externen vordefinierten Kommandos des IE. Diese Kommandos sind eine andere Variante von Objekterzeugungen (z.B. von HTML-Elementen) als Analogon zu Makros. Das Objekt besitzt nur Methoden, die an andere Objekte vererbt werden. Die meisten Methoden sind **erst** nach dem kompletten Laden des Dokumentes anwendbar. Falls die Methoden Werte liefern, so sind die Werttypen kommandospezifisch.

Erzeugung:

durch Browser

Syntax:

object.methode()

object Zeiger laut ID-Attribut

Beispiel 1:

```
<HTML>
<BODY>
    <H1 UNSELECTABLE="on">Demo</H1>
    <SCRIPT>
        function AddLink()
        {
            var SelektierterText = document.selection.createRange();

            if (!SelektierterText =="")
            {
                // Link erzeugen
                document.execCommand("CreateLink");

                if (SelektierterText.parentElement().tagName == "A")
                {

                    // markierten Text mit Eltern-Url ersetzen
                    SelektierterText.parentElement().innerText=
                        SelektierterText.parentElement().href;

                    // Vordergrundfarbe setzen im Dokument
                    document.execCommand(
                        "ForeColor","false","#FF0033");
                }
            }
            else
            {alert("Bitte im blauen Text selektieren !");
            }
    </SCRIPT>
    <P UNSELECTABLE="on">
        Selektiere (markiere) im nachfolgenden blauen Text die Stelle mit
        dem Text  MARKIERE_MICH.<BR>
        Danach auf das Button klicken.<BR>
        Anstelle von MARKIERE_MICH  im blauen Text erscheint dort
        nun eine Url.
    </P>
    <P STYLE="color=#3366CC">
        Meine beliebeste Webseite MARKIERE_MICH bitte besuchen !
    </P>
    <BUTTON onclick="AddLink()" UNSELECTABLE="on">Klick</BUTTON>
</BODY>
</HTML>
```

Beispiel 2:

```
<HTML>
<HEAD>
<SCRIPT>
    function HandlerFuerOnMoveStart()
    {
        // anstelle des ID vom DIV falls mehrere bewegbare Objekte vorhanden sind
        var ZeigerAufObjektMitEvent = event.srcElement;

        ZeigerAufObjektMitEvent.style.backgroundColor = "green";
        ZeigerAufObjektMitEvent.innerText = "DIV wird bewegt ";
    }

    function HandlerFuerOnMove ()
    {
```


```
                    ID_Span1.innerHTML = event.srcElement.offsetLeft;
                    ID_Span2.innerHTML = event.srcElement.offsetTop;
                }

                function HandlerFuerOnMoveEnd()
                {
                    // anstelle des ID vom DIV falls mehrere bewegbare Objekte vorhanden sind
                    var ZeigerAufObjektMitEvent = event.srcElement;

                    ZeigerAufObjektMitEvent.style.backgroundColor = "red";
                    ZeigerAufObjektMitEvent.innerText = "DIV wird nicht mehr bewegt";
                }

                // 2-D Positionierung einschalten
                document.execCommand("2D-position",false,true);
            </SCRIPT>
            </HEAD>
                <BODY   onmovestart="HandlerFuerOnMoveStart();"
                        onmove="HandlerFuerOnMove();"
                        onmoveend="HandlerFuerOnMoveEnd();"
                >
                    offsetLeft = <SPAN ID="ID_Span1"></SPAN>
                    <BR>
                    offserTop = <SPAN ID="ID_Span2"></SPAN>
                    <BR>
                    <DIV CONTENTEDITABLE="true">
                        <DIV STYLE=
                            "position:absolute;width:300px;height:100px; background-color:red;"
                        >
                            bewegbarer DIV
                        </DIV>
                    </DIV>
                </BODY>
            </HTML>
```

Eigenschaften:
keine
Methoden:

.execCommand() Kommando ausführen z.B. im aktuellen Dokument
in aktueller Selektion
im aktuellen Bereich
erst nach dem kompletten Laden des Dokumentes zulässig
Hinweis: Selektion = Markierung z.B. von Textbereich (Block)
Control = Element zur Steuerung analog zum HTML-Element (Tag)
Input-Control = Element mit Eingabeeigenschaft
Syntax:

```
var Wert = object.execCommand(Command [, UserInterface] [, Value])
```

Command String als Kommando, wobei Gross-Kleinschreibung egal ist

"2D-Position"	absolute Positionierung von Elementen durch Bewegen im Dragging erlauben
"AbsolutePosition"	Element-Eigenschaft der Position auf "absolute" setzen nur für selektierbare Elemente nicht für STYLE-Deklarationen im Dokument
"BackColor"	lesen oder setzen der Hintergrundfarbe der aktuellen Selektion
"Bold"	Wechsel zwischen bold und nonbold in der aktuellen Selektion
"Copy"	Aktuelle Selektion in das Clipboard kopieren
"CreateBookmark"	Bookmark setzen oder lesen für aktuelle Selektion bzw. Einfügepunkt
"CreateLink"	Hyperlink in der aktuellen Selektion setzen oder einfügen bei Einfügen erscheint Dialog-Box
"Cut"	Aktuelle Selektion in das Clipboard verschieben
"Delete"	Aktuelle Selektion löschen Hinweis: Dokument nicht löschbar
"FontName"	Font für aktuelle Selektion setzen oder holen
"FontSize"	Fontsize für aktuelle Selektion setzen oder holen
"ForeColor"	Vordergrundfarbe (Textfarbe) für aktuelle Selektion setzen oder holen
"FormatBlock"	Blockformat setzen
"Indent"	Ident des selektierten Textes erhöhen
"InsertButton"	in Textselektion das Button-Control einfügen, wenn eines bereits vorhanden so überschreiben
"InsertFieldset"	in Textselektion die Box einfügen,

wenn eine bereits vorhanden so überschreiben

"InsertHorizontalRule"
in Textselektion die horizontale Linie einfügen
wenn eine bereits vorhanden so überschreiben

"InsertIFrame" in Textselektion den inline-frame (IFRAME) einfügen
wenn einer bereits vorhanden so überschreiben

"InsertImage" in Textselektion das Image einfügen
wenn eines bereits vorhanden so überschreiben

"InsertInputButton" in Textselektion das Input-Button-Control einfügen
wenn eines bereits vorhanden so überschreiben

"InsertInputCheckbox"
in Textselektion das Input-Check-Box-Control einfügen
wenn eines bereits vorhanden so überschreiben

"InsertInputFileUpload"
in Textselektion das Input-File-Upload-Control einfügen
wenn eines bereits vorhanden so überschreiben

"InsertInputHidden" in Textselektion das Input-Hidden-Control einfügen
wenn eines bereits vorhanden so überschreiben

"InsertInputImage" in Textselektion das Input-Image-Control einfügen
wenn eines bereits vorhanden so überschreiben

"InsertInputPassword"
in Textselektion das Input-Password-Control einfügen
wenn eines bereits vorhanden so überschreiben

"InsertInputRadio" in Textselektion das Input-Radio-Control einfügen
wenn eines bereits vorhanden so überschreiben

"InsertInputReset" in Textselektion das Input-Reset-Control einfügen
wenn eines bereits vorhanden so überschreiben

"InsertInputSubmit" in Textselektion das Input-Submit-Control einfügen
wenn eines bereits vorhanden so überschreiben

"InsertInputText" in Textselektion das Input-Text-Control einfügen
wenn eines bereits vorhanden so überschreiben

"InsertMarquee" in Textselektion das Marquee einfügen
wenn eines bereits vorhanden so überschreiben

"InsertOrderedList" Wechsel zwischen ordered list und normalen Block
für aktuelle Textselektion

"InsertParagraph" in Textselektion Zeilenumbruch einfügen
wenn eines bereits vorhanden so überschreiben

"InsertSelectDropdown"
in Textselektion das Drop-Down-Selections-Control einfügen
wenn eines bereits vorhanden so überschreiben

"InsertSelectListbox" in Textselektion die List-Box-Selektion einfügen
wenn eines bereits vorhanden so überschreiben

"InsertTextArea" in Textselektion das Input-Textarea-Control einfügen
wenn eines bereits vorhanden so überschreiben

"InsertUnorderedList"
Wechsel zwischen unordered list und normalen Block
für aktuelle Textselektion

"Italic" Wechsel zwischen italic und non-italic
für aktuelle Textselektion

"JustifyCenter" zentrieren für aktuelle Textselektion

"JustifyLeft" linksbündig für aktuelle Textselektion

"JustifyRight" rechtssbündig für aktuelle Textselektion

"MultipleSelection" Mehrfachselektion per CTRL+ bzw. Shift + erlauben

"Outdent" Outdent des selektierten Textes erniedrigen

"OverWrite" Wechsel zwischen überschreiben und nicht überschreiben

"Paste" Aktuelle Selektion aus Clipboard überschreiben

"Print" Druck-Dialogbox öffen

"Refresh" aktuelles Dokument refreshen

"RemoveFormat" formatierende Tags der aktuellen Selektion entfernen

"SaveAs" Aktuelles Dokument speichern als Datei

"SelectAll" alles markieren (selektieren)

"UnBookmark" Alle Bookmark der aktuellen Selektion entfernen

"Underline" Wechsel zwischen underline und nicht-underline
für aktuelle Textselektion

"Unlink" Alle Hyperlink der aktuellen Selektion entfernen

"Unselect" Alles demarkieren (de-selektieren)

UserInterface false Default
user interface soll nicht angezeigt werden (Dialogbox)

true user interface soll angezeigt werden
(falls Kommando das unterstützt)

Value z.B. String, number
immer passend zu Command, kann optional sein
Kodierung null (nicht numerisch Null !!) als Wert
entspricht Weglassen des optionalen Wertes

Kommando	IE ab	Dialogbox	Value
2D-Position	5.5	nein	true für on false für off
AbsolutePosition	5.5	nein	true für "absolut" false für nicht"absolut"
BackColor	4.x	nein	rrggbb OHNE führendes # vordefinierter Farbname (browserspezifisch)
Bold	4.x	nein	null oder omit oder weglassen
Copy	4.x	nein	null oder omit oder weglassen
CreateBookmark	4.x	nein	String mit Ankername keine Leerkette
CreateLink	4.x	ja	String mit Url keine Leerkette
Cut	4.x	nein	null oder omit oder weglassen
Delete	4.x	nein	null oder omit oder weglassen
FontName	4.x	nein	String mit Fontname oder Fontliste Fontliste: Folge von Fonteigenschaften
FontSize	4.x	nein	String mit Fontsize von einschliesslich 1 bis einschliesslich 7
ForeColor	4.x	nein	rrggbb OHNE führendes # vordefinierter Farbname (browserspezifisch)
FormatBlock	4.x	nein	String mit Block-Tag
Indent	4.x	nein	null oder omit oder weglassen
InsertButton	4.x	nein	String mit Attributen des Button-Control
InsertFieldset	4.x	nein	String mit Attributen der Box
InsertHorizontalRule	4.x	nein	String mit Attributen der Linie
InsertIFrame	4.x	nein	String mit Attributen des IFRAME
InsertImage	5.x	ja	String mit Pfad und Dateiname
InsertInputButton	4.x	nein	String mit Attributen des Input- Button-Control
InsertInputCheckbox	4.x	nein	String mit Attributen des Input- Checkbox-Control
InsertInputFileUpload	4.x	**nein !!!**	String mit Attributen des Input- Fileupload-Control
InsertInputHidden	4.x	nein	String mit Attributen des Input- Hidden-Control
InsertInputImage	4.x	nein	String mit Attributen des Input- Image-Control
InsertInputPassword	4.x	nein	String mit Attributen des Input- Password-Control
InsertInputRadio	4.x	nein	String mit Attributen des Input- Radio-Control

Kommando	IE ab	Dialogbox	Value
InsertInputReset	4.x	nein	String mit Attributen des Input- Reset-Control
InsertInputSubmit	4.x	nein	String mit Attributen des Input- Submit-Control
InsertInputText	4.x	nein	String mit Attributen des Input- Text-Control
InsertMarquee	4.x	nein	String mit Attributen des Marquee-Control
InsertOrderedList	4.x	nein	String mit Attributen des Ordered-List-Control
InsertParagraph	4.x	nein	String mit Attributen des Paragraph-Control
InsertSelectDropdown	4.x	nein	String mit Attributen des Drop- Down-Control
InsertSelectListbox	4.x	nein	String mit Attributen des List- Box-Control

InsertTextArea	4.x	nein	String mit Attributen des Text-Area-Control
InsertUnorderedList	4.x	nein	String mit Attributen des Unordered-List-Control
Italic	4.x	nein	null oder omit oder weglassen
JustifyCenter	4.x	nein	null oder omit oder weglassen
JustifyLeft	4.x	nein	null oder omit oder weglassen
JustifyRight	4.x	nein	null oder omit oder weglassen
MultipleSelection	5.5	nein	true für on false für off
Outdent	4.x	nein	null oder omit oder weglassen
OverWrite	4.x	nein	true für Überschreiben false für Nicht-Überschreiben
Paste	4.x	nein	null oder omit oder weglassen
Print	5.5	ja	null oder omit oder weglassen **kein String !!!**
Refresh	4.x	nein	null oder omit oder weglassen
RemoveFormat	4.x	nein	null oder omit oder weglassen
SaveAs	4.x	ja	wenn Dialogbox anzeigen so kann Wert null sein wenn Dialogbox nicht anzeigen so muss Wert die Paramter enthalten
SelectAll	4.x	nein	null oder omit oder weglassen
UnBookmark	4.x	nein	null oder omit oder weglassen
Underline	4.x	nein	null oder omit oder weglassen
Unlink	4.x	nein	null oder omit oder weglassen
Unselect	4.x	nein	null oder omit oder weglassen

Wert true wenn Kommando ausgeführt wurde
false wenn Kommando nicht ausgeführt wurde

.queryCommandEnabled()	prüfen ob Kommando ausführbar ist
.queryCommandIndeterm()	prüfen ob Kommando-Status bestimmbar ist oder nicht
.queryCommandState()	Status des aktuellen Kommando ermitteln: ob ausgeführt wurde oder nicht
.queryCommandSupported()	prüfen ob Kommando im aktuellen Bereich unterstützt wird
.queryCommandValue()	Wert eines Kommandos liefern