window.document Objekt des Internet Explorer

Achtung: Das Objekt document.body ist nicht mehr verfügbar und wurde durch das Objekt document.documentElement ersetzt !

document.documentElement ist der Zeiger auf den Body und nicht mehr document.body !

Beispiel zur Abfrage auf CSS1-Konformität und damit zur Verwendung von document.body bzw. document.documentElement

```
function DocTypeAbfrage()
{
        // Annahme: CSS1-Standard, also !DOCTYPE wurde kodiert
         var ZeigerAufDolument=document.documentElement;

         if (document.compatMode=="BackCompat")
         {
                // nicht CSS1-Standard, also wurde kein !DOCTYPE kodiert
                ZeigerAufDolument =document.body;
         }

         return ZeigerAufDolument;
}
```

Hinweise: document.compatMode ist nur lesbar, also !DOCTYPE nicht setzbar
Wert "CSS1Compat" ab IE 6
geliefert wenn !DOCTYPE kodiert wurde
Rendern mit CSS1 Standard (Standard-Compilant-Modus)
Wert "BackCompat" ab IE 3
Geliefert wenn kein !DOCTYPE kodiert wurde
Rendern mit Nicht-kein Standad-Compilant-Modus

Ansatz:

Das Objekt ist das HTML-Dokument, das in Browserfenster geladen und angezeigt wird.
ist der Container für die HTML-Elemente der Webseite z.B. BODY oder DIV
umfasst sämtlichen Code **zwischen** <HTML...> und </HTML>:
Der Code hinter </HTML> (z.B. Script) wird nicht geparst, aber eventuell als Plain-Text angezeigt.

Scriptsteuerung erst ab IE 4.x

Hinweis: Es gibt noch das html Objekt, welches den **HTML-Tag** beschreibt.

Das HTML-Dokument im HTML-DOM:

Das Objekt wird als Hierarchie von Knoten dargestellt, die nicht der Folge der Elemente im HTML-Code entspricht. Ein HTML-Element ist ein Knoten als Kind des Dokumentes.

Das HTML-Element im HTML-DOM:

Das HTML-Element wird in seine Komponenten zerlegt, die Knoten und Kinder des HTML-Elementes sind.
Bsp.: SPAN-Element mit Text: Text ist ein Knoten von SPAN

Zur Referenzierung von HTML-Elementen des Dokumentes sollte im Regelfall **document.all** kodiert werden, damit die Browserperformance steigt (siehe Collection document.all).

Das HTML-Element beim IE und NS:
als HTML-Tag: Der IE und NS unterstützen z.T. verschiedene Tags.
als Objekt: Es gibt Objekte, die im IE **und** NS implementiert sind aber z.T. auf verschiedene Weise
z.B. Objekt document

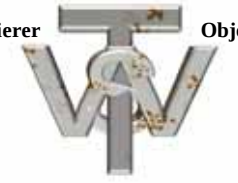
DOM und Collectionen zum Objekt document:

Zur Verwaltung des Dokumentes existieren Collectionen als Felder, die zugleich als Schnittstelle zum Programmierer dienen. Collectionen des Objektes document werden immer ohne .all kodiert, da sie keine HTML-Elemente darstellen. Der Netscape besitzt teilweise Collectionen, die auch der Internet Explorer bedient.

Objekt document und Browserfenster:

Das HTML-Dokument wird getrennt vom Fensterobjekt verwaltet: Ein Dokument muss zwar in ein Fenster geladen werden, aber das Dokument wird nur dann visualisiert (gerendert), wenn es sichtbare Elemente besitzt. Die Referenzierung des Dokumentes **im** Fenster erfolgt anhand des logischen Fensternames, der geliefert wird
per Methode .open() zum window Objekt und ein frei festgelegter oder vordefinierter Name sein kann
als Wert des ID-Attributes bei FRAMESET mit FRAMES bzw. IFRAMES (siehe dort).

HTML-Dokument und Druck-Layout im Browserfenster:



Im Internet Explorer ab 5.x lässt sich das Druck-Layout der Seite, also des Dokumentes, bezüglich Kopf- und Fusszeile beeinflussen:

Menüpunkt Datei-Seite einrichten und dort die Angaben editieren.

Die Angaben haben folgende vordefinierte Komponenten:

&w	Platzhalter für den Titel des Fensters, in dem das Dokument angezeigt wird (siehe auch Objekt window)
&u	Platzhalter für die URL des Dokumentes (siehe auch Objekt location)
&d	Platzhalter für das Datum im Kurzformat laut Uhr des PC des Users
&D	Platzhalter für das Datum im Langformat laut Uhr des PC des Users
&t	Platzhalter für die Uhrzeit im 12-Stunden-Format laut Uhr des PC des Users
&T	Platzhalter für die Uhrzeit im 24-Stunden-Format laut Uhr des PC des Users
&p	Platzhalter für die Seitennnummer
&P	Platzhalter für die Gesamtanzahl der Seiten im Dokument
&b	Text zentrieren, der hinter &b kodiert wird
&b&b	Text rechtsbündig setzen, der hinter &b&b kodiert wird
&&	das Zeichen "&"

Das Layout ist in der Seitenansicht sichtbar, wobei die Platzhalter durch konkrete Werte ersetzt sind (als wenn gedruckt werden würde).

Hinweis zur Pars-Reihenfolge des Internet Explorer innerhalb der HTML-Kodierung bezüglich dem Tag-Name und den Element-Attributen CLASS, ID, STYLE:

Folgende Reihenfolge wird beim Parsen eingehalten:

1. Element-Bezeichner
2. CLASS-Attribut mit Bezeichner aus Klassendeklaration im HEAD
3. ID-Attribut
4. STYLE-Attribut mit Style-Werten (nicht mit Bezeichner aus Style-Deklaration im HEAD)

Es gilt: Wenn gleiche Bezeichner verwendet, so nur Werte des **zuletzt** geparsten Bezeichners verwendet !
Die CLASS-Deklaration aus dem HEAD des Dokumentes wird wertmäßig durch die Style-Deklaration per Attribut des HTML-Elementes überschrieben, wenn gleiche Style-Eigenschaften betroffen sind (ansonsten hinzufügen).

Hinweis zu den Beschreibungen der Objekte und Collectionen aus der Hierarchie des Dokumentes:

Die Beschreibungen beziehen sich in der Regel aus Gründen der Vereinfachung auf das aktuelle Fenster und dessen (aktuelles) Dokument. sind analog anwendbar für ein per logischem Windownamen adressiertes Dokument.

Erzeugung:
durch den Browser oder per Methode .createDocumentFragment()

Beispiel für Ausdruck des aktuellen Dokumentes:

```
<BODY>
    <INPUT  TYPE="button" VALUE="Drucken"onclick="javascript:self.print()">
</BODY>
```

Beispiel für Webseite mit eigenem Icon ab IE 5.x:

```
<LINK   REL="SHORTCUT ICON" HREF="name.ico">
```

name.ico: name ist beliebig, auch mit Pfad

ICO-Datei: 32x32 Pixel mit16 Farben
oder 16x16 Pixel mit 256 Farben
ist BMP-Datei, die zu ICO-Datei konvertiert werden muss
(kann nicht jedes Grafik-Programm, aber Microsoft bietet dafür IconPro an)

Beispiel für Ermittlung der Verweilzeit des Users auf der Webseite:

```
<HTML>
<HEAD>
<SCRIPT>
<!--
    var start_zeit;         // muss global sein

     function start()
    {start_zeit = new Date();}

    function ende()
    {
        var ende_zeit = new Date();
        var wert=ende_zeit.getTime() - start_zeit.getTime();
```

```
                        wert = wert /1000;      // in Sekunden umrechnen
                        alert("Verweilzeit in Sekunden: " + Math.floor(wert).toString());
                }
//-->
</SCRIPT>
</HEAD>

<BODY ... onLoad="start();" onUnload="ende();">
</BODY>
<HTML>
```

Beispiel für Seitenelemente vom Druck ausschließen (ab IE 4.x):

Seitenteile, die nicht gedruckt werden sollen, in <DIV CLASS="keindruck"> und </DIV>
oder in <SPAN CLASS="keindruck"> und </SPAN> einschliessen.

zwischen <HEAD> und </HEAD> einfügen: <LINK REL="stylesheet" MEDIA="print" HREF="print.css">

print.css enthält nur .keindruck {display:none;}

Beispiel für Seitenelemente nur beim Druck anzeigen (ab IE 4.x):

Seitenteile, die nur auf Ausdrucken sichtbar sein solllen, in <DIV CLASS="nurdruck"> und </DIV>
oder in <SPAN CLASS ="nurdruck"> und </SPAN> einschliessen

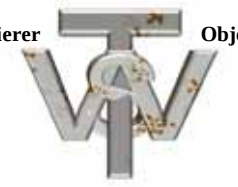
zwischen <HEAD> und </HEAD> einfügen: <LINK REL ="stylesheet" MEDIA="screen" HREF="screen.css">

screen.css enthält nur .nurdruck {display:none;}

Beispiel für Dokumenten-Hintergrund ein- bzw. ausblenden:

```
<HTML>
<HEAD>
<SCRIPT TYPE="text/javascript" LANGUAGE="JavaScript1.2">
<!--
//                      Dokument-Hintergrundfarbe ein- und ausblenden
//                              einblenden beim Dokument laden
//                              ausblenden beim Wechsel zu anderem Dokument, also entladen des aktuellen Dokumentes
//                                      mit Farbveränderung rückwärts gegenüber Laden

//*********************************************************************************************
//
//                      Nachfolgende Variablen muss der Programmierer anpassen
//
//*********************************************************************************************
//      Farbe wird aus Rot- und Gruen- und Blauanteil gebildet
//      Alle Farbanteilwerte gelten von 0 bis 255, alles ganzahlig, wird nicht geprüft
//      Startwerte für Startfarbe des Ein- bzw. Ausblenden, müssen <= Endwerte  sein, wird nicht geprüft
//      Endewerte  für Endefarbe nach Ein- bzw. Ausblenden, müssen >= Startwerte sein, wird nicht geprüft
//      Farbschritte = Stufen für Ein- bzw. Ausblenden
//                                      je mehr um so langsamer das Ein- bzw. Ausblenden
//                                      nur ganzahlig und maximal die kleinste paarige Differenz der obigen Werte
//                                              (keine Prüfung)

var RotAnteil_Start             = 0;
var GruenAnteil_Start                   = 0;
var BlauAnteil_Start            = 0;
var RotAnteil_Ende              = 255;
var GruenAnteil_Ende            = 255;
var BlauAnteil_Ende             = 255;
var FarbSchritte                = 64;

// *********************************************************************************************
//
//                      Nachfolgenden Code nicht verändern
//
//*********************************************************************************************

function DezimalZuHexaString(DezimalerWert)
// liefert String von 2 Ziffern bzw. Grossbuchstaben
// X_DezimalerWert  Gleitkomma
//                                      wenn nicht ganzahlig, so vor der Konvertierung gannzahlig gemacht
//                                      wenn <= 0   so '00' geliefert
```

```
//                                          wenn >= 255 so 'FF' geliefert
{
        var HexaZeichenFeld = "0123456789ABCDEF";
        var HexaKette="";

        // Dezimalwert zu ganzahlig
        X_DezimalerWert=Math.floor(X_DezimalerWert);

        if (DezimalerWert < 0)
        {HexaKette="00";}
        else
        {
                if (DezimalerWert > 255)
                {HexaKette="FF";}
                else
                {
                        //      Indexe im HexaZeichenFeld ermitteln
                        var HexaKette_Zeichen1_PositionImHexaZeichenFeld = Math.floor(DezimalerWert / 16);
                        var HexaKette_Zeichen2_PositionImHexaZeichenFeld =
                                DezimalerWert  - (HexaKette_Zeichen1_PositionImHexaZeichenFeld * 16);

                        //      HexaKette bilden
                        HexaKette =
                          HexaZeichenFeld.charAt(HexaKette_Zeichen1_PositionImHexaZeichenFeld)
                        + HexaZeichenFeld.charAt(HexaKette_Zeichen2_PositionImHexaZeichenFeld);
                }
        }

        return HexaKette;
}

function BackgroundEinAusBlenden(RotAnteil_Start,GruenAnteil_Start,BlauAnteil_Start,
                                 RotAnteil_Ende,GruenAnteil_Ende,BlauAnteil_Ende,
                                 FarbSchritte
                                )
{
//      Farbe wird aus Rot- und Gruen- und Blauanteil gebildet
//      Alle Farbanteilwerte gelten von 0 bis 255, alles ganzahlig, wird nicht geprüft
//      Startwerte für Startfarbe des Ein- bzw. Ausblenden, müssen <= Endwerte   sein, wird nicht geprüft
//      Endewerte  für Endefarbe nach Ein- bzw. Ausblenden, müssen >= Startwerte sein, wird nicht geprüft
//      Farbschritte = Stufen für Ein- bzw. Ausblenden
//                                          je mehr um so langsamer das Ein- bzw. Ausblenden
//                                          nur ganzahlig und maximal die kleinste paarige Differenz der obigen Werte
//                                                  (keine Prüfung)

        for(var i = 0; i <= FarbSchritte; ++i)
        {
                var RotAnteil   = Math.floor(    (RotAnteil_Start   * ((FarbSchritte - i) / FarbSchritte))
                                               + (RotAnteil_Ende    * (i / FarbSchritte))
                                             );

                var GruenAnteil = Math.floor(    (GruenAnteil_Start * ((FarbSchritte - i) / FarbSchritte))
                                               + (GruenAnteil_Ende  * (i / FarbSchritte))
                                             );

                var BlauAnteil  = Math.floor(    (BlauAnteil_Start  * ((FarbSchritte - i) / FarbSchritte))
                                               + (BlauAnteil_Ende   * (i / FarbSchritte))
                                             );

                document.bgColor =      "#"
                                      + DezimalZuHexaString(RotAnteil)
                                      + DezimalZuHexaString(GruenAnteil)
                                      + DezimalZuHexaString(BlauAnteil);
        }
}

//      Nachfolgende Funktionen exisitieren NUR, um die globalen Variablen für BODY verfügbar zu machen
//              da in der HTML-Body-Anweisung keine Javascript-Variablen akzeptiert werden
function EinAusBlenden_DokumentLaden()
{
        BackgroundEinAusBlenden(RotAnteil_Start,GruenAnteil_Start,BlauAnteil_Start,
                                RotAnteil_Ende,GruenAnteil_Ende,BlauAnteil_Ende,
                                FarbSchritte
```

```
                                        );
}

function EinAusBlenden_DokumentEntladen()
{
        //      ist das Laden rückwärts
        BackgroundEinAusBlenden(RotAnteil_Ende,GruenAnteil_Ende,BlauAnteil_Ende,
                                RotAnteil_Start,GruenAnteil_Start,BlauAnteil_Start,
                                FarbSchritte
                                );
}
// -->
</SCRIPT>
</HEAD>
<BODY BGCOLOR=#ffffff   onload="EinAusBlenden_DokumentLaden();"
                        onunload="EinAusBlenden_DokumentEntladen();"
>
</BODY>
</HTML>
```

Beispiel für Dokumenten-Hintergrund auswählen:

```
<HTML>
<HEAD>
<SCRIPT TYPE="text/javascript" LANGUAGE="JavaScript1.2">
<!--
//      Neues Fenster öffnen mit selektierten Hintergrund
//
function HintergrundSetzen(bgname)
{
        var Fenster=window.open("","","width=380,height=380");
        Fenster.focus();
        Fenster.document.open();
        Fenster.document.write(   "<HTML><HEAD></HEAD>"
                                + "<BODY BACKGROUND="
                                        + ""
                                        + bgname
                                        + ""
                                + ">"
                                + "</BODY>"
                        + "</HTML>");
        Fenster.document.close();
}
//-->
</SCRIPT>
</HEAD>
<BODY>
        <A HREF="getbackg.htm" onclick="HintergrundSetzen('bg1.jpg');return false">
                <IMG SRC="bg1.jpg" BORDER="0" WIDTH="96" HEIGHT="96">
        </A>
        <A HREF="getbackg.htm" onclick="HintergrundSetzen('bg2.jpg');return false">
                <IMG SRC="bg2.jpg" BORDER="0" WIDTH="96" HEIGHT="96">
        </A>
</BODY>
</HTML>
```

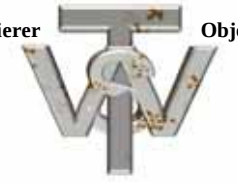
Beispiel für Änderung der Hintergrundfarben:

```
<HTML>
<BODY   bgColor="0000ff"
        onBlur="document.bgColor='ff0000'"
        onFocus="document.bgColor='0000ff'"
>
Dieses Fenster &auml;ndert die Hintergrundfarbe, wenn es nicht mehr aktiv ist.
</BODY>
</HTML>
```

Zugriff:

window.document.eigenschaft	oder	document.eigenschaft	oder	eigenschaft
window.document.methode()	oder	document.methode()	oder	methode()

```
logischer_window_name.document.eigenschaft
logischer_window_name.document.methode()
```

logischer_window_name laut open()

Der **logische** Fenstername muss nur dann kodiert werden, wenn der Bezug nicht auf das Dokument im aktuellen Fenster gehen soll.
Der logische Fenstername stammt aus
der Methode .open() zum window Objekt und kann ein frei festgelegter oder vordefinierter Name sein
aus dem Wert des ID-Attributes bei FRAMESET mit FRAMES.
Die Kodierung des Zeigers von document kann dann entfallen, wenn das aktuelle Dokument referenziert wird.
Allerdings ist von der Kodierung ohne window bzw. ohne window.document abzuraten, denn
es könnten gleichnamige Eigenschaften und Methoden geben, die in anderen Objekten auch implementiert sind
es wird damit die Browserperformance gesenkt.

Beispiel:

```
function OnClickHandler()
{
    alert(FensterZeiger.ID_TextArea.value;
    FensterZeiger.close();
}

....

// Fenster öffnen
var FensterZeiger=window.open("", null, "height=250,width=300,status=no,toolbar=no,menubar=no,location=no");
var FensterDokumentZeiger = FensterZeiger.document;

var Kette=    '<HTML>'
    + '<HEAD></HEAD>'
    + '<BODY >'
    + <TEXTAREA ID="ID_TextArea" ROWS="5" COLS="20"></TEXTAREA>'
    + '<BR>'
    + '<INPUT TYPE="button" VALUE="Text an Aufrufer übergeben"'
    +   'onclick="opener.OnClickHandler();"'
    +  '>'
    + '</BODY>'
    + '</HTML>';

FensterDokumentZeiger.open("text/html");
FensterDokumentZeiger.write(Kette);
FensterDokumentZeiger.close();
```

Eigenschaften:

Hinweis zu dynamischen Eigenschaftenveränderung zur Laufzeit:

Die zuletzt während der Laufzeit getätigte Defintion ersetzt wertmäßig den aktuellen Attributwert, wenn gleiche Attributnamen/Eigenschaften betroffen sind.

.activeElement — Referenz auf dasjenige Objekt im Dokument , das Focus hat
IE unter 5.x Zeiger auf BODY-Objekt
ab IE 5.x null

Beispiel:

```
var Zeiger =  document.activeElement
```

.alinkColor — Farbe aller aktiven Links im Dokument
document.body.alinkColor
"#rrggbb" oder vordefiniertef Farbname

.bgColor — deprecated und durch STYLE-Attribut zu ersetzen
Hintergrundfarbe des Objektes bzw. Dokumentes

.charset — Zeichensatz zum Encoden eines Objektes im Dokument

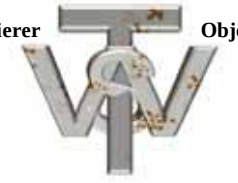
.compatMode — Kompatibilität des IE 6.x zu CSS1 bzw. seinen Browservorgängern
siehe style Objekt und Kodierung von !DOCTYPE
ab IE 6.x

Achtung: Das Objekt document.body ist nicht mehr verfügbar und wurde durch das Objekt document.documentElement ersetzt !

document.documentElement ist der Zeiger auf den Body und nicht mehr document.body !

Beispiel zur Abfrage auf CSS1-Konformität und damit zur Verwendung von document.body bzw. document.documentElement

```
function DocTypeAbfrage()
{
    // Annahme: CSS1-Standard, also !DOCTYPE wurde kodiert
```

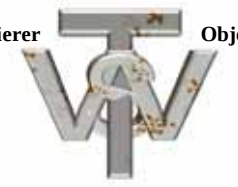
```
                var ZeigerAufDolument=document.documentElement;

                if (document.compatMode=="BackCompat")
                {
                        // nicht CSS1-Standard, also wurde kein !DOCTYPE kodiert
                        ZeigerAufDolument =document.body;
                }

                return ZeigerAufDolument;
        }
```

Hinweise: document.compatMode ist nur lesbar, also !DOCTYPE nicht setzbar
Wert "CSS1Compat" ab IE 6
geliefert wenn !DOCTYPE kodiert wurde
Rendern mit CSS1 Standard (Standard-Compilant-Modus)
Wert "BackCompat" ab IE 3
Geliefert wenn kein !DOCTYPE kodiert wurde
Rendern mit Nicht-kein Standad-Compilant-Modus

.cookie Cookie des Dokumentes als Stringwert

Beispiel:

```
<SCRIPT>
        function ErzeugeCookie(Name, Value)       // Name und Wert sind Strings
        {
                var date = new Date();
                var document.cookie =      Name
                                        + "="
                                        + escape(Value)
                                        + "; expires=" + date.toGMTString(); // aktuelles Datum
        }

        function LoescheCookie (Name, Value)      // Name und Wert sind Strings
        {
                var document.cookie =     Name
                                        + "="
                                        + escape(Value)
                                        + "; expires=Fri, 31 Dec 1999 23:59:59GMT;"; // altes Datum

        function LeseCookieWert(Name)
        // Name des zu lesenden Cookie ist String
        // liefert Cookiewert aus unescape() als String                                                    {
                var CookieWert="";   // Annahme: Cookie nicht gefunden oder mit Leerkette als Wert

                //  Feld der Cookies referenzieren
                //          erzeugt durch Separation anhand Semikolon
                var CookieFeld = document.cookie.split("; ");
                var AnzahlCookies = CookieFeld.length;

                // Feldelement umfasst Name und Wert des Cookie
                //          Aufbau     Cookie_Name = Cookie_Wert
                //                                  Separator ist Gleichheitszeichen
                //          Index 0 für Cookien_Name
                //          Index 1 für Cookie_Wert
                var CookieNameUndWertAlsFeld;

                // CookieFeld auslesen und auf Cookie laut Name prüfen
                var Gefunden=false;
                var Index = 0;
                do
                {
                        // aktuelles Cookie lesen
                        CookieNameUndWertAlsFeld = CookieFeld [Index].split("=");

                        // und auf Name prüfen
                        if (Name == CookieNameUndWertAlsFeld [0])
                        {
                                CookieWert = unescape(CookieNameUndWertAlsFeld [1]);
                                Gefunden=true;
                        }
                        else
                        {Index++;}
                }
```

```
                        while (         ( !Gefunden )
                                && ( Index < AnzahlCookies)
                            );

                        return CookieWert;
                }
        </SCRIPT>
```

.defaultCharset regionaler Standardzeichensatz (Charakter-Set) zum Encoden eines Objektes im Dokument

.designMode Editierbarkeit des Dokumentes (Manipulierbarkeit)
z.B. per Ausführung von Javscript
Syntax:
document.designMode [= Kette]
[var Kette =] document.designMode

Kette	String		
		"On"	Dokument kann editiert werden es wird kein Script ausgeführt !!
		"Off" oder "Inherit"	Standard Dokument kann nicht editiert werden Scripte werden ausgeführt "Inherit" bedeutet leider **nicht**: von Eltern geerbt, da Dokument keine Eltern haben muss (z.B. hat Frameset-Dokument Eltern)

lesen und schreiben

.dir Umflussrichtung
.doctype Referenz auf Dokumenttyp
.documentElement Referenz auf Wurzelknoten (Root Node) des Dokumentes liefern

Beispiel:

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

Achtung: Das Objekt document.body ist nicht mehr verfügbar und wurde durch das Objekt document.documentElement ersetzt !

document.documentElement ist der Zeiger auf den Body und nicht mehr document.body !

Beispiel zur Abfrage auf CSS1-Konformität und damit zur Verwendung von document.body bzw. document.documentElement

```
function DocTypeAbfrage()
{
        // Annahme: CSS1-Standard, also !DOCTYPE wurde kodiert
         var ZeigerAufDolument=document.documentElement;

         if (document.compatMode=="BackCompat")
        {
                // nicht CSS1-Standard, also wurde kein !DOCTYPE kodiert
                ZeigerAufDolument =document.body;
        }

        return ZeigerAufDolument;
}
```

Hinweise: document.compatMode ist nur lesbar, also !DOCTYPE nicht setzbar
Wert "CSS1Compat" ab IE 6
geliefert wenn !DOCTYPE kodiert wurde
Rendern mit CSS1 Standard (Standard-Compilant-Modus)
Wert "BackCompat" ab IE 3
Geliefert wenn kein !DOCTYPE kodiert wurde
Rendern mit Nicht-kein Standad-Compilant-Modus

.domain Domain-Suffix des Dokumentes
Kommunikation von Dokumenten verschiedener Urls mit gemeinsamen Domainsuffix

Beispiel:
home.*test.com* und **www.*test.com*** können kommunizieren,
wenn Kette auf "***test.com***" gesetzt wurde in den Dokumenten beider Urls

.expando Wirksamkeit von Attributen im Dokument ein/aus
true ein, Default
false aus

Beispiel:

```
<HTML>
<HEAD>
<SCRIPT>
    document.expando = false; // für gesamtes Dokument abschalten
</SCRIPT>
</HEAD>
<BODY>
<DIV>
    <SPAN ID="ID_Span" UNSELECTABLE="on">
        Dieser Text ist <B>selektierbar !!<B>
    </SPAN>
</DIV>
</BODY>
</HTML>
```

.fgColor Vordergrundfarbe (Textfarbe) im Dokument
#rrggbb Standard ist #000000
vordefinierter Farbname (browserspezifisch)

.fileCreatedDate Datum der Dokumenterstellung

Beispiel 1:

"Monday, December 08, 2000"

Beispiel 2:

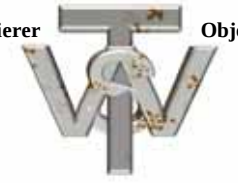
```
<SCRIPT>
    window.onload=fnInit;

    function fnInit()
    {
        var AnzahlMillisekundenProTag =86400000;

        var DatumDokumentErzeugung =new Date(document.fileCreatedDate);

        var DatumHeute =new Date();

        var TagesDifferenz =    (   DatumHeute.getTime()
                                    - DatumDokumentErzeugung.getTime()
                                )
                                / AnzahlMillisekundenProTag;

        alert(    "Dokument erzeugt am " + DatumDokumentErzeugung
                + "\n..... also vor " +parseInt(TagesDifferenz) + " Tagen"
            );
    }
</SCRIPT>
```

.fileModifiedDate Datum der letzten Dokumentveränderung
.fileSize Größe des Dokumentes
.implementation Referenz auf Objekt implementation zum Dokument
.lastModified Datum der letzten Dokumentveränderung
.linkColor Farbe von noch nicht benutzten Links im Dokument
"#rrggbb" Standard ist "#0000FF"
vordefinierter Farbname (browserspezifisch)

.location Zeiger auf das gleichnamige Objekt

Beispiel 1:

```
<HTML>
<HEAD>
<SCRIPT>
    function LocationAendern()
    {
        for(i=0;i<document.all.length;i++)
        {
            if (document.all[i].tagName=="IFRAME")
            {document.all[i].contentWindow.location = "http://www.test.com";}
        }
    }
</SCRIPT>
</HEAD>
<BODY>
```

```
                <BUTTON onclick="LocationAendern();">Location aendern</BUTTON>
                <IFRAME SRC="http://www.test.de"></IFRAME>
        </BODY>
        </HTML>
```

Beispiel 2:

```
        <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>
```

.parentWindow — Referenz auf Elternfenster des Dokumentes

.protocol — Protokoll-Teil einer Url inklusive http und ftp liefern
nur lesen bei Objekten document
img
location

Beispiel:

```
location.protocol liefert z.B. http: und immer inklusive dem Doppelpunkt
```

.readyState — aktueller Status des Objektes beim Füllen des Objektes mit Daten

"uninitialized"	Objekt ist nicht initialisiert
"loading"	Objekt ist nicht intialisiert aber lädt gerade Daten zur Initialisierung
"loaded"	Objekt hat alle Daten komplett geladen und ist komplett intitialisiert
"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:

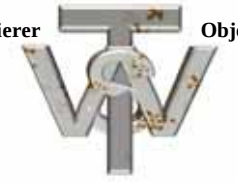
```
alert(document.body.readyState);
```

.referrer — Url der direkt vorhergehenden Seite der aktuellen Seite
aktuelle Seite muss durch Link angesprungen sein
Eingabe in Adresszeile oder per Menü Datei-Öffnen etc. nicht verwendet
ist Leerkette, wenn aktuelle Seite nicht durch Link von der vorherigen aktiviert wurde

.title — Fenstertitel des Dokumentes
siehe Objekt document

Beispiel für permanenten Fenstertitel-Wechsel:

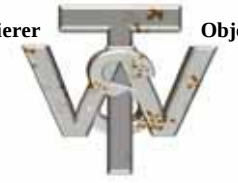
```
        <SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
        <!--
                var titel_texte_feld = [
                                        "Titel 1",
                                        "Titel 2",
                                        "Titel 3",
                                        "Titel 4"
                                        ];

                var wechsel_tempo = 2500;       // Zeit in ms zwischen zwei Schritten
                var zahler = 0;
```

```
		var id;

		function start()
		{
			titel_texte_feld[titel_texte_feld.length] = document.title;
			// beim ersten Aufruf wird der Original-Titel mit gespeichert
			// Länge ab 1, aber Index ab 0
			//    Länge == Index +1, an desssen Position der Original-Titel abgelegt wird

			window.setTimeout("wechseln()", wechsel_tempo);
		}

		function wechseln()
		{
			document.title = titel_texte_feld [zahler];
			zahler ++;

			if(zahler >= texte.length)
			{zahler = 0}

			id = window.setTimeout("wechseln()", wechsel_tempo);
		}
//-->
</SCRIPT>
```

Beispiel für Fenstertitelzeile mit scrollendem Text:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript1.2">
//
//		In der aktuellen Fenster-Titelzeile einen freien Text durch endloses Scrollen von lins nach rechts vorsetzen

//*************************************************************************************************
//
//			nachfolgende Variablen sind vom Programmierer zu setzen
//
//*************************************************************************************************
var VorsetzText_Wert		= "Freien Text vorsetzen durch scrollen und warten ";
var ScrollGeschwindigkeit	= "150";   // in Millisekunden, immer > 0, wird nicht geprüft
var DummyLeerzeichenAnzahl  = 10;      // immer > 0 , wird nicht geprüft
                                       // Die Leerzeichen werden nicht sichtbar angezeigt, aber gescrollt.
                                       //		So entsteht eine Wartezeit (pro Leerzeichen laut
                                       //		ScrollGeschwindigkeit)
                                       //		für die vollständige Anzeige von VorsetzText NACH dem
                                       //		vollständigem Vorsetzen per Scrollen von links nach
                                       //		rechts, ehe der Scroller wieder von vorn beginnt

//*************************************************************************************************
//
//			nachfolgender Code darf nicht verändert werden
//			wird anstelle von onLoad innerhalb BODY verwendet
//
//*************************************************************************************************
//		rechtsbündige Auffüllung mit DummyLeerzeichen
for (i=0; i <=DummyLeerzeichenAnzahl; i++)
{VorsetzText_Wert+=" ";}

//		Länge NACH Auffüllung ermitteln
var VorsetzText_Laenge=VorsetzText_Wert.length;

//		globale Variablen für EndlosScrollen(), die dort laufend manipuliert werden,
//			also hier initialisiert werden müssen
var Zahler="0";
var VorsetzText="";
//-->
</SCRIPT>
</HEAD>

<BODY>
<SCRIPT LANGUAGE="JavaScript1.2">

//*************************************************************************************************
```

```
		//
		//			nachfolgender Code darf nicht verändert werden
		//
		//*************************************************************************************************

		function EndlosScrollen()
		{
			// endloses Scrollen

			if (		(document.all)
				|| (document.getElementById)
			  )
			{
				// VorsetzText um nächsten Buchstaben aus dem VorsetzText_Wert erweitern
				// (einschliesslich DummyLeerzeichen)
				VorsetzText+=VorsetzText_Wert.charAt(Zahler);   // charAt ab Null

				// Fenster-Titelzeile neu belegen
				document.title=VorsetzText;

				Zahler++;
				if (Zahler==VorsetzText_Laenge)
				{
					Zahler="0";
					VorsetzText="";
				}

				setTimeout("EndlosScrollen()",ScrollGeschwindigkeit);
			}
		}

		window.onload=EndlosScrollen; // ohne () kodieren, damit nicht sofort sondern beim Laden ausgeführt wird
		//-->
		</SCRIPT>
		</BODY>
		</HTML>
```

.uniqueID — durch den Browser automatisch-genriertes 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>
```

.URL — Url des Dokumentes
Achtung: Gross-und kleinschreibung wird unterschieden !!!
ist identisch mit location.href

.URLUnencoded — Url des Dokumentes ohne Zeichenencoding

.vlinkColor — Farbe bereits geklickter Links

.XMLDocument — Referenz auf XML-Dokument (XML-DOM)

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

Methoden:

.attachEvent() — Einschalten des Registrieren eines Events durch Eventhandler
Hinweis: Abschalten mit Methode .detachEvent()
Achtung: Wenn mehrere Eventhandler zum Event, so Aufruf der Handler leider
NICHT verkettet sondern in **Zufallsfolge**, es sei denn
die Handler prüfen ihre Aufruffolge (muss programmiert werden)

Beispiel:

```
<PUBLIC:ATTACH EVENT="ondetach" ONEVENT=" EventEntfernen()"/>

<SCRIPT LANGUAGE="JScript">
	function CursorNeu()
	{
```

```
                    if (event.srcElement == element)
                    {
                        normalColor          = style.color;
                        runtimeStyle.color  = "red";
                        runtimeStyle.cursor = "hand";
                    }
                }

                function CursorNormal()
                {
                    if (event.srcElement == element)
                    {
                        runtimeStyle.color   = normalColor;
                        runtimeStyle.cursor = "";
                    }
                }

                function EventEntfernen()
                {
                    detachEvent ('onmouseover', CursorNeu);                    // nicht () kodieren !!!
                    detachEvent ('onmouseout', CursorNormal);// nicht () kodieren !!
                }

                attachEvent ('onmouseover', CursorNeu);
                attachEvent ('onmouseout', CursorNormal);
            </SCRIPT>
```

.clear() Dokument löschen

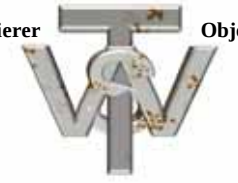
.close() Ausgabe-Datenstream schliessen und Anzeige des Dokumentes beenden
schliessen einer mit .open() erzeugten Dokument-Instanz

Beispiel:

```
<HTML>
<HEAD>
<SCRIPT>
    function ErzeugeZurLaufZeit()
    {
        // Dokumentinstanz erzeugen
        var ID_Dokument = document.open("text/html", "replace");

        // Dokumentinhalt festlegen und anzeigen
        ID_Dokument.write("<HTML><BODY>Hallo</BODY></HTML>");

        // Dokumentinstanz schliessen
        ID_Dokument.close();
    }
</SCRIPT>
</HEAD>
<BODY>
<INPUT TYPE="button" onclick=" ErzeugeZurLaufZeit();">
</BODY>
</HTML>
```

Beispiel:

```
function OnClickHandler()
{
    alert(FensterZeiger.ID_TextArea.value;
    FensterZeiger.close();
}

....

// Fenster öffnen
var FensterZeiger=window.open("", null, "height=250,width=300,status=no,toolbar=no,menubar=no,location=no");
var FensterDokumentZeiger = FensterZeiger.document;

var Kette=    '<HTML>'
    + '<HEAD></HEAD>'
    + '<BODY >'
    + <TEXTAREA ID="ID_TextArea" ROWS="5" COLS="20"></TEXTAREA>'
    + '<BR>'
    + '<INPUT TYPE="button" VALUE="Text an Aufrufer übergeben"'
    +   'onclick="opener.OnClickHandler();"'
```

```
                    + '>'
                    + '</BODY>'
                    + '</HTML>';

                FensterDokumentZeiger.open("text/html");
                FensterDokumentZeiger.write(Kette);
                FensterDokumentZeiger.close();
```

.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:

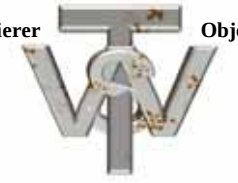
```
var Kommentar = document.createComment("Das ist ein Kommentar");
```

.createDocumentFragment() erzeugt neues Dokument

.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

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>
```

.createEventObject() Event-Objekt im Dokument erzeugen nur für Methode .fireEvent()

Beispiel:

```
var EventObjekt = document.createEventObject();
document.fireEvent("onclick", EventObjekt);
```

.createStyleSheet() styleSheet Objekt im Dokument erzeugen

Beispiel:

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

.createTextNode() Text-Objekt im Dokument erzeugen
nur Plain-Text, also keine HTML-Tags

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>
```

.detachEvent() Abschalten des Registrieren eines Events durch Eventhandler
wobei Registrierung mit Methode .attachEvent() aktiviert wurde
Abschalten = ordnet dem Window-Objekt das Event laut Parameter event_bezeichner
zu, das **nicht** behandelt werden soll, falls es für das Window-Objekt auftritt
(also Standardbehandlung aktiv)

Beispiel:

```
<PUBLIC:ATTACH EVENT="ondetach" ONEVENT=" EventEntfernen()"/>

<SCRIPT LANGUAGE="JScript">
	function CursorNeu()
	{
		if (event.srcElement == element)
		{
			normalColor	= style.color;
			runtimeStyle.color  = "red";
			runtimeStyle.cursor = "hand";
		}
```

```
                }

                function CursorNormal()
                {
                        if (event.srcElement == element)
                        {
                                runtimeStyle.color   = normalColor;
                                runtimeStyle.cursor = "";
                        }
                }

                function EventEntfernen()
                {
                        detachEvent ('onmouseover', CursorNeu);          // nicht () kodieren !!!
                        detachEvent ('onmouseout', CursorNormal);// nicht () kodieren !!
                }

                attachEvent ('onmouseover', CursorNeu);
                attachEvent ('onmouseout', CursorNormal);
        </SCRIPT>
```

.elementFromPoint() liefert Referenz auf Objekt an Pixelpos relativ zum Fenster
Fenster linke obere Ecke (0,0)
Objekt muss Mausevents unterstützen

.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

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>
```

.focus()

Focus setzen und Focus-Event auslösen
nur nach dem kompletten Laden des Dokumentes
vor IE 5.x: Objekt muss TABINDEX-Attribut besitzen

Beispiel:

```
<BODY onload="document.body.focus();>
```

.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");
```

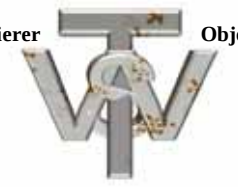
Hinweis: entspricht var DIV_KinderZeigerFeld = document.body.all.tags("DIV");

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>
```

.hasFocus() Objekt im Focus-Zustand
true Objekt hat den Focus
false Objekt hat keinen Focus

.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()">
```

.open() Dokument instanzieren und mit/ohne Fenster öffnen und Referenz liefern auf Dokument
Ausgabe-Datenstream öffnen und Anzeige des Dokumentes
Instanzieren ohne .open(): siehe .write() und .writeln()
Hinweis: neues Fenster erzeugen als unterstes der aktuellen Fensterhierarchie
und dann Fenster öffnen (anzeigen, rendern)
Syntax:
für Öffnen mit Fenster:
[var ZeigerAufWindow =] document.open(
[URL] [, Name] [, Features] [, Replace])

für Öffnen im aktuellen Fenster
[var ZeigerAufDokument =] document..open([mime_typ] [,Replace])

für Öffnen im beliebigen Fenster
[var ZeigerAufDokument =] window_instanz.document..open([mime_typ] [,Replace])

URL String mit Multipurpose Internet Mail Extensions (MIME) Typ
Standard: MIME-Typ "text/html"
nur "text/html" zulässig

Name String Dokumentname für Verlauf (History)
wenn kodiert, so immer das aktuelle Dokument durch das Neue ersetzt im Verlauf
wenn nicht kodiert, so neues Dokument angehängt an History

physischer Window-Name: identisch mit Wert laut Attribut TARGET z.B. im Link
dient der Referenz
auch null kodierbar für keine Referenz
Vordefinierte Namen:

"_blank"	Standard : ohne Name
"_media"	Dokument laut Url wird in den HTML-Content-Bereich der Media Bar geladen ab IE 6.x
"_parent"	Dokument laut Url wird in den Elternframe geladen wird. wenn kein Frame so wirkunglos
"_search"	Dokument laut Url wird in das Browser-Search-Fenster geladen ab IE 5.x
"_self"	Dokument laut Url wird in das neue Fenster geladen
"_top"	Dokument laut Url wird in das oberste Fenster geladen identisch mit _self, da neues Fenster das oberste wird

mime_typ Datentyp des Dokumentes
meistens **"text/html"** HTML-Dokument
oder **"text/plain"** Normaltext-Dokument

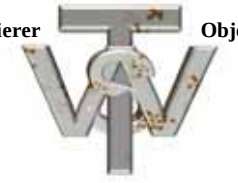
Features String als Parameterliste mit Kommatrennung
(Liste der Fensterkomponenten)
gesamte Liste in " " bzw. ' ' setzen
z.B. "fullscreen=yes, toolbar=yes"
gesamte Liste ist 1 Zeichenkette,
die in 1 Quelltextzeile passen muss,
oder in Teilketten zerlegt mit dem + Operator zusammengesetz wird
Blanks in Liste **nicht** zulässig
Listenelement: option=wert
wenn mindestens 1 Element kodiert, so alle anderen nicht kodierten Elemente automatisch deaktiviert, also immer alle gewünschten Optionen kodieren !!!
Standardwert eines Merkmales, wenn
Liste kodiert wurde: no oder 0
Liste nicht kodiert wurde: yes oder 1
keine Standardwerte vorhanden für Pixelangaben
da diese vom Browser automatisch belegbar sind

alwaysLowered = { yes | no }
yes: Fenster öffnen und permanent als unterstes in der Fensterhierarchie belassen
nur per signiertem Script
beachte .setZOptions()
nur NS

alwaysRaised = { yes | no}
yes: Fenster öffnen und permanent als oberstes in der Fensterhierarchie belassen
nur per signiertem Script
beachte .setZOptions()
nur NS

channelmode = { yes | no | 1 | 0 }
default ist no bzw. 0
yes oder 1 für Channelleiste anzeigen
(Fenster im Channel-Mode anzeigen)
nur IE

dependent = { yes | no }
yes: Fenster an den .opener bezüglich



Fensterschliessen koppeln: auch
schliessen, wenn .opener geschlossen wird
sowie geöffnetes Fenster nicht in
der Taskleiste von Windows
anzeigen
nur NS

directories = { yes | no | 1 | 0 }
default ist yes bzw. 1
yes oder 1 Directory Buttons anzeigen

fullscreen = { yes | no | 1 | 0 }
default ist no bzw. 0
yes bzw. 1 Vollbild anzeigen also ohne
Leisten etc., wobei damit fast alle
Steuerungselemente unsichtbar werden
nur IE
Hinweis: ALT+F4 schliesst das Fenster

height = Pixelwert, Fensterhöhe
bei IE : mindestens 100 (wenn < 100, so auf
100 automatisch gesetzt)
bei NS: nur mit signiertem Script < 100
ohne signiertem Script: wenn < 100,
so auf 100 automatisch gesetzt

hotkeys = { yes | no}
yes: alle sicherheitsrelevanten
Tastenkombinationen
verwendbar machen
no: nur noch ALT+F4 funktioniert
(schliessen des Fensters)
nur NS

innerHeight= anzeigebereich_höhe_in_pixel
ohne signiertes Script: >= 100
Anzeigebereich = Fenster abzüglich
Leisten, Scrollbars etc.
nur NS

innerWidth= anzeigebereich_breite_in_pixel
ohne signiertes Script: >= 100
Anzeigebereich = Fenster abzüglich
Leisten, Scrollbars etc.
nur NS

left = X-Koordinate in Pixels bezüglich linker
oberer Screen-Ecke (0,0)
nur IE

location = { yes | no | 1 | 0 }
default ist yes bzw. 1
yes bzw 1 für URL-Eingabezeile anzeigen
(Adresszeile anzeigen)

menubar = { yes | no | 1 | 0 }
default ist yes bzw. 1
yes bzw. 1 für Menübar anzeigen
(Leiste der Pull-Down-Menüs anzeigen)

outerHeight= fenster_höhe_in_pixel
ohne signiertes Script: >= 100
Fenster = Anzeigebereich sowie Leisten,
Scrollbars etc.
nur NS

outerWidth= fenster_breite_in_pixel
ohne signiertes Script: >= 100
Fenster = Anzeigebereich sowie Leisten,
Scrollbars etc.
nur NS

personalbar = { yes | no }

yes: persönliche Toolbar anzeigen
nur NS

resizable = { yes | no | 1 | 0 }
default ist yes bzw. 1
yes bzw. 1 für Größenveränderung des
Fensters erlaubt per Mausziehen
bei NS beachte .setResizable()

screenX= horizontale_pixel_pos des Fensters
bezüglich dem Bildschirm,
dessen Ursprung (0,0) in der
linken oberen Ecke liegt
nur NS

screenY= vertikale_pixel_pos des Fensters
bezüglich dem Bildschirm,
dessen Ursprung (0,0) in der
linken oberen Ecke liegt
nur NS

scrollbars = { yes | no | 1 | 0 }
default ist yes bzw. 1
yes bzw. 1 für Scrollbalken anzeigen
sobald Fensterinhalt größer als
Anzeigebereich des Fensters

status = { yes | no | 1 | 0 }
default ist yes bzw. 1
yes bzw. 1 für Statuszeile anzeigen

titlebar = { yes | no | 1 | 0 }
default ist yes bzw. 1
yes bzw. 1 für Titelzeile anzeigen
Titel werden immer angezeigt bei Dialog-Box
bei NS: nur mit signiertem Script setzbar

toolbar = { yes | no | 1 | 0 }
default ist yes bzw. 1
yes bzw. 1 für Toolbar (Navigationsleiste)
anzeigen (Button Zurück etc.)

top = Y-Koordinate in Pixels bezüglich linker oberer
Screen-Ecke (0,0)
nur IE

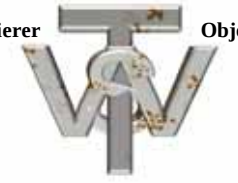
width = Pixelwert, Fensterbreite,
bei IE : mindestens 100 (wenn < 100, so auf
100 automatisch gesetzt)
bei NS: nur mit signiertem Script < = 100
ohne signiertem Script: wenn < 100,
so auf 100 automatisch gesetzt

z-lock = { yes | no}
yes: Fenster kann kein anderes
überlagern
nur per signiertem Script
beachte .setZOptions()
nur NS

Replace true, so History-Eintrag des Dokumentes, in dem das neue
Fenster erzeugt wird, ersetzen durch den Eintrag
des neuen Fensters, also Zurück zum Dokument,
dass das Fenster öffnet, nicht möglich
false, so neuen History-Eintrag zum neuen Fenster
erzeugen, also zurück zum alten Fenster möglich

ZeigerAufWindow Referenz (ID) auf das neue Fenster mit Dokument
z.B. für Methode window.close()

ZeigerAufDokument
Referenz (ID) auf das neue Dokument
z.B. für Methode ZeigerAufDokument.close()



Beispiel:

```
<HTML>
<HEAD>
<SCRIPT>
    function ErzeugeZurLaufZeit()
    {
        // Dokumentinstanz erzeugen
        var ID_Dokument = document.open("text/html", "replace");

        // Dokumentinhalt festlegen und anzeigen
        ID_Dokument.write("<HTML><BODY>Hallo</BODY></HTML>");

        // Dokumentinstanz schliessen
        ID_Dokument.close();
    }
</SCRIPT>
</HEAD>
<BODY>
<INPUT TYPE="button" onclick=" ErzeugeZurLaufZeit();">
</BODY>
</HTML>
```

Beispiel:

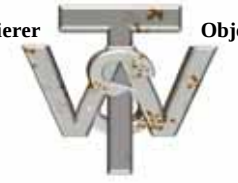
```
function OnClickHandler()
{
    alert(FensterZeiger.ID_TextArea.value;
    FensterZeiger.close();
}

....

// Fenster öffnen
var FensterZeiger=window.open("", null, "height=250,width=300,status=no,toolbar=no,menubar=no,location=no");
var FensterDokumentZeiger = FensterZeiger.document;

var Kette=    '<HTML>'
    + '<HEAD></HEAD>'
    + '<BODY >'
    + <TEXTAREA ID="ID_TextArea" ROWS="5" COLS="20"></TEXTAREA>'
    + '<BR>'
    + '<INPUT TYPE="button" VALUE="Text an Aufrufer übergeben"'
    +    'onclick="opener.OnClickHandler();"'
    +  '>'
    + '</BODY>'
    + '</HTML>';

FensterDokumentZeiger.open("text/html");
FensterDokumentZeiger.write(Kette);
FensterDokumentZeiger.close();
```

.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

.recalc() dynamischen Eigenschaften des Dokumentes neu berechnen
Hinweis: andere Objekte, die Eigenschaften des Dokumentes nutzen, werden auch neu berechnet, wenn Eigenschaften nicht in einem Berechungsausdruck vorliegen

Beispiel 1:

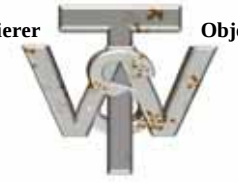
```
<HTML>
<HEAD>
<STYLE>
    BUTTON {font-size:14;width:150}
</STYLE>
<SCRIPT>
    var timerID = null;
    var seconds = 0;

    function init()
    {
        ID_Div1.style.setExpression("width","seconds*10");
        ID_Div2.setExpression("innerText","seconds.toString()");
```

```
		}

		function timer()
		{
			seconds++;
			document.recalc();
		}

		function starttimer()
		{
			if (timerID == null)
			{
				timerID = setInterval("timer()", 1000);
				ID_Button1.disabled = true;
				ID_Button2.disabled = false;
			}
		}

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

		function resettimer()
		{seconds = 0;}
</SCRIPT>
</HEAD>
<BODY onload="init()">
		<DIV ID="ID_Div1" STYLE ="background-color:lightblue" ></DIV>
		<DIV ID="ID_Div2"  STYLE="color:hotpink;font-weight:bold"></DIV>
		<BR>
		<BUTTON ID="ID_Button1"                    onclick="starttimer()">Start Timer</BUTTON></BR>
		<BUTTON ID="ID_Button2" DISABLED="true"  onclick="stoptimer()">Stop  Timer</BUTTON><BR>
		<BUTTON                                    onclick="resettimer()">Reset Timer</BUTTON><BR>
</BODY>
</HTML>
```

Beispiel 2 für Sekundenbalken:

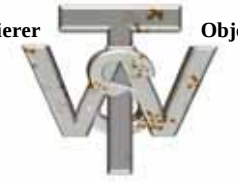
```
<HTML>
<HEAD>
<STYLE>
	BUTTON {font-size:14;width:150}
</STYLE>
<SCRIPT>
	var timerID = null;
	var Zahler = 0;

	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>
```

Beispiel 3 für Sound mit Sekundenanzeige:

```
	<HTML>
	<HEAD>
	<SCRIPT>
		// ++++++++++++ globale Variablen, die verändert werden können
		var SoundUrl			= "56sec.mid";
		var SoundDauerInSekunden	= 56; // Dauer muss exakt stimmen !

		var PixelBreiteProBalkenErweiterung = 10;

		// ++++++++++++ Browser-Typ ermitteln
		// Dieser Quellcode muss VOR allen anderen Routinen codiert sein, damit zuerst abgearbeitet
```

```
var ns = document.layers ? true : false;
var ie = document.all ? true : false;

// ++++++++++ Routinen der Sekundenzählung
var SekundenZahler              = 0;
var SekundenZahlerTimoutID    = null;

function SekundenZaehlen()      // wird durch Rekursion alle Sekunde neu gestartet
{
        // Zähler erhöhen
        SekundenZahler++;

        // visuelle Anzeige im Dokument auffrischen, also alle Audrucke der Style-Werte
        //        neu berechnen und damit alle DIV's neu visualisieren
        document.recalc();
}

function SekundenZaehlen_Start()
{
        // prüfen ob Sekundenzählen nicht bereits läuft
        if (SekundenZahlerTimoutID == null)
        {
                // nicht aktiv

                // Rekursion starten: SekundenZaehlen() wird permanent alle Sekunde aktiviert
                SekundenZahlerTimoutID = setInterval("SekundenZaehlen()", 1000);
        }
}

function SekundenZaehlen_Stop()
{
        // prüfen ob Sekundenzählen aktiv ist
        if (SekundenZahlerTimoutID != null)
        {
                // aktiv, also stoppen
                clearInterval(SekundenZahlerTimoutID);
                SekundenZahlerTimoutID = null;
        }
}

// ++++++++++ Routine zur Erzeugung Sound-Objekt
function SoundObjektErzeugen(SoundFileUrl,SoundFileDauerInSekunden)
{
        this.SoundFileUrl                               = SoundFileUrl;
        this.SoundFileDauerInMillisekundeSekunden = SoundFileDauerInSekunden  * 1000;
                                                                // Timerzeit für Rekursion
        this.SoundFileBeendet                           = true;    // kein Sound aktiv
}

// ++++++++++ Routinen zur Wiedergabe Sound-Objekt
var SoundTimeoutID=0;

function SoundAbspielen()
{
        // prüfen ob Sound nicht bereits aktiv ist
        if (SoundObjekt.SoundFileBeendet)
        {
                // Anzeige intialisieren
                SekundenAnzeigeInit();

                // Sekundenzähler starten, wobei das Zählen eigenständig und parallel erfolgt
                SekundenZaehlen_Start();

                // Sound erzeugen und sofort starten durch  Url-Zuweisung
                ID_BGSound.src=SoundObjekt.SoundFileUrl ;
                SoundObjekt.SoundFileBeendet=false;

                // Millisekunden warten und danach die Funktion SoundAbspielen() neu aufrufen
                SoundTimeoutID = setTimeout(
                                        "SoundAbspielen()",
                                        SoundObjekt.SoundFileDauerInMillisekundeSekunden
                                                );
        }
```

```
		else
		{
			// Dieser Zweig wird erst mit dem 2. Aufruf der Funktion abgearbeitet

			// Sound zu Ende
			SoundObjekt.SoundFileBeendet=true;

			// Sekundenzähler stoppen
			SekundenZaehlen_Stop();

			// und Meldung
			var TimerUngenauigkeit1 = SoundDauerInSekunden - SekundenZahler;
			var TimerUngenauigkeit2 = TimerUngenauigkeit1 / SoundDauerInSekunden;
			alert(		"Wiedergabe beendet\nUngenauigkeit des Timers = "
				+ TimerUngenauigkeit1.toString()+ " Sekunden\n"
				+ "also " + TimerUngenauigkeit2.toString() + " Ticks pro Sekunde"
			    );
		}
}

// ++++++++++ Sekunden-Anzeige initialiseren
function SekundenAnzeigeInit()
{
		// ----- Variablen init
		SekundenZahler			= 0;
		SekundenZahlerTimoutID		= null;

		// ----- visuelle Anzeige erzeugen
		// - - - Sekundenbalken und Sekundenzähler dynamisch visualisieren
		//		Es wird jedem DIV als Style-Wert ein Ausdruck hinterlegt, also kein Wert.
		//			Der Ausdruck liefert den Wert , welcher sofort das Layout der
		//			DIV's beeinflusst.
		//		Jeder Ausdruck besitzt den SekundenZahler als Komponente.
		//			Damit ändert sich der Wert des Ausdruckes.
		//		Für die Neuberechnung des Ausdruckes ist der Aufruf von
		//				document.recal()
		//			nötig.
		//			Dieser Aufruf erfolgt in SekundenZaehlen(), also permanent pro Sekunde.
		//			Damit wird der Style-Wert permanent neu berechnet.
		//			Damit visualisieren sich die DIV's permanent neu.

		// Sekundenbalken in der Style-Eigenschaft width (Breite) mit Ausdruck belegen,
		//		also dynamisch anzeigen
		ID_DIV_Balken.style.setExpression(  "width",
						    "SekundenZahler * PixelBreiteProBalkenErweiterung"
						  );

		// Sekundenzähler  in der  Eigenschaft .innerText mit Ausdruck belegen,
		//		also dynamisch anzeigen
		ID_DIV_SekundenZahler.setExpression("innerText","SekundenZahler.toString()");

		// - - - Messlatte statisch anzeigen
		ID_DIV_MessLatte.style.width  = SoundDauerInSekunden * PixelBreiteProBalkenErweiterung;
		ID_DIV_MessLatte.innerText    =    "Der Sound dauert "
						   + SoundDauerInSekunden.toString()
						   + " Sekunden";
}

// ################ Dieser Teil wird mit dem Laden des Dokumentes abgearbeitet ################
if (ie)
{
		document.write('<BODY></BODY>');

		document.write(		'<BGSOUND ID= "ID_BGSound" LOOP="0"">'

		document.write(		 '<DIV	ID="ID_DIV_Balken"'
				+		 'STYLE="background-color:lightblue"'
				+ '>'
				+ '</DIV>'
				+ '<BR>'
			);

		document.write(		 '<DIV	ID="ID_DIV_SekundenZahler"'
```

```
                                        +           'STYLE="color:hotpink;font-weight:bold"'
                                        + '>'
                                        + '</DIV>'
                                        + '<BR>'
                                    );

                                document.write(     '<DIV   ID="ID_DIV_MessLatte"'
                                        +           'STYLE="color:white;background-color:gray"'
                                        + '>'
                                        + '</DIV>'
                                    );

                                // ++++++++++ Sound initialisieren und starten mit Laden des Dokumentes

                                //  ----- Sound-Objekt erzeugen anhand globaler Variablen
                                SoundObjekt = new SoundObjektErzeugen(SoundUrl, SoundDauerInSekunden);

                                // ----- Sound-Objekt wiedergeben
                                SoundAbspielen();  // meldet wenn Wiedergabe beendet ist
                        }
                </SCRIPT>
                </HEAD>
                <BODY>
                        <! -- BODY-Teil muss leer bleiben //-->
                </BODY>
                </HTML>
```

.releaseCapture() Maus-Überwachung ausschalten für ein Objekt
Maus-Events sind : onmousedown, onmouseup, onmousemove, onclick, ondblclick,
onmouseover und onmouseout.
Hinweis: einschalten per Methode .setCapture()

Beispiel:

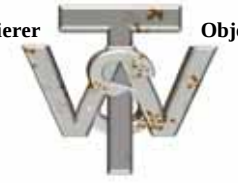
```
<BODY   onload="ID_Div.setCapture();"
        onclick="document.releaseCapture();"
>
        <DIV ID="ID_Div"
                onmousemove="ID_Textarea.value = event.clientX + event.clientY;"
                onlosecapture="alert(event.srcElement.id + ' hat keine Mausueberwachung mehr');"
        >
                <TEXTAREA ID="ID_Textarea" COLS=2>Test</TEXTAREA>
        </DIV>
</BODY>
```

.setActive() Objekt für die Eventdurchreichung aktivieren
aber ohne es zu fokussieren
und ohne es scrollbar zu machen

Beispiel:

```
<HTML>
<HEAD>
<SCRIPT>
        var ID_Fenster;

        function FensterErzeugen()
        {
                ID_Fenster = window.open( "/test /test.htm",
                                          " ID_Fenster",
                                          "top=10px,left=480px,height=375px,width=200px,resizable=1"
                                        );
                this.focus();
        }

        function ButtonAktivieren()
        {window.parent.ID_Fenster.ID_Button.setActive();}
</SCRIPT>
</HEAD>
<BODY onload="FensterErzeugen();">
        <BUTTON ID="ID_Button" onclick="ButtonAktivieren();">
                Button aktivieren
        </BUTTON>
</BODY>
</HTML>
```

.write() Text bzw. HTML-Ausdruck in das Dokument ausgeben und anzeigen bzw. sofort parsen



Hinweis:

Die **ERSTE** Erzeugung eines HTML-Tags bewirkt das **automatische Öffnen eines neuen** HTML-Dokumentes, wenn das aktuelle Dokument **bereits komplett geladen**, also das Ereignis onload **bereits** ausgelöst wurde. Letzteres ist immer dann der Fall, wenn der BODY-Teil des Dokumentes komplett geparst wurde. Grund: Ein komplett geparstes Dokument kann per write() bzw. writeln() nicht um HTML-Elemente verändert werden, da diese Methoden keine des HTML-DOM sind. Mit anderen Worten: Nur die Verwendung von Methoden des HTML-DOM lassen eine **nachträgliche** HTML-Elemente-Veränderung des Dokumentes zu.

Plain-Text
HTML-Text
Script

.writeln() Text bzw. HTML-Ausdruck in das Dokument ausgeben und anzeigen bzw. sofort parsen

Hinweis:

Die **ERSTE** Erzeugung eines HTML-Tags bewirkt das **automatische Öffnen eines neuen** HTML-Dokumentes, wenn das aktuelle Dokument **bereits komplett geladen**, also das Ereignis onload **bereits** ausgelöst wurde. Letzteres ist immer dann der Fall, wenn der BODY-Teil des Dokumentes komplett geparst wurde. Grund: Ein komplett geparstes Dokument kann per write() bzw. writeln() nicht um HTML-Elemente verändert werden, da diese Methoden keine des HTML-DOM sind. Mit anderen Worten: Nur die Verwendung von Methoden des HTML-DOM lassen eine **nachträgliche** HTML-Elemente-Veränderung des Dokumentes zu.

nach der Anzeige einen Zeilenvorschub anzeigen
Plain-Text
HTML-Text
Script

Events:

onactivate Fires when the object is set as the active element.
onbeforeactivate Fires immediately before the object is set as the active element.
onbeforecut Fires on the source object before the selection is deleted from the document.
onbeforedeactivate Fires immediately before the activeElement is changed from the current object to another object in the parent document.
onbeforeeditfocus Fires before an object contained in an editable element enters a UI-activated state or when an editable container object is control selected.
onbeforepaste Fires on the target object before the selection is pasted from the system clipboard to the document.
onclick Fires when the user clicks the left mouse button on the object.
oncontextmenu Fires when the user clicks the right mouse button in the client area, opening the context menu.
oncontrolselect Fires when the user is about to make a control selection of the object.
oncut Fires on the source element when the object or selection is removed from the document and added to the system clipboard.
ondblclick Fires when the user double-clicks the object.
ondeactivate Fires when the activeElement is changed from the current object to another object in the parent document.
ondrag Fires on the source object continuously during a drag operation.
ondragend Fires on the source object when the user releases the mouse at the close of a drag operation.
ondragenter Fires on the target element when the user drags the object to a valid drop target.
ondragleave Fires on the target object when the user moves the mouse out of a valid drop target during a drag operation.
ondragover Fires on the target element continuously while the user drags the object over a valid drop target.
ondragstart Fires on the source object when the user starts to drag a text selection or selected object.
ondrop Fires on the target object when the mouse button is released during a drag-and-drop operation.
onfocusin Fires for an element just prior to setting focus on that element.
onfocusout Fires for the current element with focus immediately after moving focus to another element.
onhelp Fires when the user presses the F1 key while the browser is the active window.
onkeydown Fires when the user presses a key.
onkeypress Fires when the user presses an alphanumeric key.
onkeyup Fires when the user releases a key.
onmousedown Fires when the user clicks the object with either mouse button.
onmousemove Fires when the user moves the mouse over the object.
onmouseout Fires when the user moves the mouse pointer outside the boundaries of the object.
onmouseover Fires when the user moves the mouse pointer into the object.
onmouseup Fires when the user releases a mouse button while the mouse is over the object.
onmousewheel Fires when the wheel button is rotated.
onmove Fires when the object moves.
onmoveend Fires when the object stops moving.
onmovestart Fires when the object starts to move.
onpaste Fires on the target object when the user pastes data, transferring the data from the system clipboard to the document.
onpropertychange Fires when a property changes on the object.
onreadystatechange Fires when the state of the object has changed.
onresizeend Fires when the user finishes changing the dimensions of the object in a control selection.
onresizestart Fires when the user begins to change the dimensions of the object in a control selection.
onselectionchange Fires when the selection state of a document changes.
onstop Fires when the user clicks the Stop button or leaves the Web page.