```
var StringLiteral     ="Text der SUP wird"
var HTML_Kette        = StringLiteral.sup();
HTML_Kette            = "Text der SUP wird".sup();

        entspricht <SUP>Text der SUB wird</SUP>

eval(HTML_Kette);             erzeugt Fehler, da HTML-Code von eval nicht ausgeführt werden kann
document.write(HTML_Kette);
```

.toLowerCase() String bzw. Stringliteral nach neuen String kopieren und dort nach Kleinbuchstaben umwandeln

Beispiel:

```
var StringLiteral ="StringLiteral";
StringLiteral.toLowerCase()          liefert "stringliteral"
"StringLiteral".toLowerCase()        liefert "stringliteral"
```

.toUpperCase() String bzw. Stringliteral nach neuen String kopieren und dort nach Grossbuchstaben umwandeln

Beispiel:

```
var StringLiteral ="StringLiteral";
StringLiteral.toUpperCase()          liefert "STRINGLITERAL"
"StringLiteral".toUpperCase()        liefert "STRINGLITERAL"
```

zusätzliche Methoden vom IE:

.localeCompare() Vergleich zweier Strings oder Stringliterale bezüglich ihrer Sortierfolge laut aktuelle Vorgaben zur Sortierungsfolge auf dem PC des Users
nur IE ab 5.5

.toLocaleLowerCase String bzw. Stringliteral nach neuen String kopieren und dort nach Kleinbuchstaben umwandeln laut aktuelle Sprach-Einstellungen der Umgebung auf PC des Users
nur IE ab 5.5
siehe Script-Objekt String

.toLocaleUpperCase() String bzw. Stringliteral nach neuen String kopieren und dort nach Grossbuchstaben umwandeln laut aktuelle Sprach-Einstellungen der Umgebung auf PC des Users
nur IE ab 5.5
siehe Script-Objekt String

4.3. vordefinierte Objekte zum Browser (Auswahl)

4.3.1. Ansatz

4.3.1.1. vordefinierte Objekte in Javascript /JScript

Es können alle Objekte aus Javascript/JScript mit deren Eigenschaften und Methoden in sinnvoller Kombination mit den zum Browser vordefinierten Objekten verwendet werden. Die objekt-übergreifenden, also objekt-unabhängigen Methoden, sind alle nutzbar aber z.T. in den zum Browser vordefinierten Objekten abgewandelt implementiert.

4.3.1.2. Browserfenster und HTML-Dokument (Objekt window und Objekt document)

Ein HTML-Dokument muss im Browserfenster nicht angezeigt werden, wenn das HTML-Element keine sichtbaren HTML-Elemente besitzt. So gesehen kann also ein HTML-Dokument nichts mit dem Browserfenster zu tun haben.

Der Browser kann allerdings nur dann ein HTML-Dokument verwalten, wenn es einem Browserfenster zugeordnet ist. So gesehen muss also ein HTML-Dokument immer in einem Fenster liegen, egal ob das Dokument sichtbar ist oder nicht.

Ein Browserfenster
- verwaltet das HTML-Dokument als Ganzheit (Container)
- verwaltet **nicht** die HTML-Elemente des Dokumentes
- visualisiert das HTML-Dokument
- besitzt weitere Objekte für die Realisierung eines Fensters

Das HTML-Dokument
- verwaltet seine Elemente anhand des HTML-DOM (siehe weiter unten)
- besitzt weitere Objekte und andere Elemente für die Verwaltung

Aus Sicht der gesamten Browserverwaltung wird folgende Hierarchie gebildet:

```
            Fenster
            /     \
      Dokument      andere Objekten, die der Browser benötigt
      /     \
HTML-Elemente   andere Objekte, die das Dokument benötigt
```

Zur Umsetzung der Hierarchie werden Zeiger verwendet:

Zeiger als Eigenschaften des Fensters (Objekt window)

Zeiger auf das HTML-Dokument

Zeiger auf andere benötigte Objekte

Zeiger als Eigenschaften des HTML-Dokumentes (Objekt window.document)

Zeiger auf die HTML-Elemente
Zeiger auf andere benötigte Objekte

Damit gilt: Jedes Fenster hat **sein** HTML-Dokument wie umgekehrt.

Solange der Scriptcode für das aktuelle Fenster gilt, muss der Zeiger auf das aktuelle Fenster (Zeiger ist das Schlüsselwort window) nicht kodiert werden.
Beispiel: window.document. ist synonym zu document.

In allen anderen Fällen muss die Punktnotation in der Form zeiger_auf_das_fenster. verwendet werden.
Beispiel: zeiger_auf_fenster.document.

wobei zeiger_auf_fenster
laut Methode .open() zum Fenster (Objekt window)

oder laut Eigenschaft .opener des Objektes window
(Zeiger auf Elternfenster, das **open() enthält** und das Kind-Fenster öffnet, welches in dieser .opener-Eigenschaft den Zeiger auf das Elternfenster enthält)

oder laut Eigenschaft .parent des Objektes window
(Zeiger auf das **in der Fensterhierarchie übergeordnete** Fenster, das aber nicht das .open() zum Kindfenster enthalten muss; z.B. Zeiger auf das Fenster, das die FRAMESET-Deklaration enthält, oder das diesem Fenster übergeordnet ist)

Achtung: Das **erste** Browserfenster hat keinen Zeiger, denn es stellt die Browserinstanz dar. Zur Referenzierung dieses Fensters muss in nachfolgenden Fenstern der Fensterhierarchie eine der Eigeschaften .opener bzw. .parent oder das Ziel _top verwendet werden.
Im **ersten** Fenster können in der Fensterhierarchie darunterliegende Fenster referenziert werden. Dazu müssen andere Eigenschaften des Objektes window benutzt werden (siehe dort).

Diese Zeigerzuordnungen spigelt die Vererbung nur z.T. wider:

Beispiele: Ein Fenster kann nicht an HTML-Elemente vererben
Das HTML-Dokument vererbt z.T. an seine Elemente

Aus Sicht der Vererbung ist die o.g. Hierarchie nicht sehr sinnvoll.

Nachfolgende Beschreibungen
halten sich an diese Hierarchie
zeigen Vererbungen
besitzen in Syntaxbeschreibungen immer Referenzen auf das aktuelle Fenster, so dass window im Regelfall dort nicht kodiert wird.
Beispiel: Beschreibung des HTML-Dokumentes document und nicht window.document, da im aktuellen Fenster window.document und document synonym sind.

4.3.1.3. HTML-Dokument (Objekt document) und seine HTML-Elemente

4.3.1.3.1. HTML-Dokument und die Hierarchie der HTML-Elemente

HTML-Elemente-Hierarchie im HTML-Quellcode des HTML-Dokumentes

Die Kodierung der HTML-Elemente im HTML-Code der Webseite legt die Folge der HTML-Elemente und damit das Layout der Webseite fest. Der Folge entspricht einer HTML-Elemente-Hierarchie laut Syntaxvorschriften der HTML-Konvention.

HTML-Elemente-Hierarchie der Objekte des HTML-Dokumentes

Mittels Umsetzung des HTML-Dokumentes z.B. zum Zweck der Visualisierung durch die Scriptmaschine wird aus jedem HTML-Element ein Objekt erzeugt, das innerhalb der Objekthierarchie des HTML-Dokumentes automatisch platziert wird. Diese Hierarchie wird als **HTML-Dokument-Objekt-Modell (DOM)** bezeichnet und hängt vom Layout der Webseite sowie von Vorgaben der Scriptmaschine ab.

Effektives Instrument der Abbildung einer Hierarchie ist die Baumdarstellung. Das liegt nahe, denn das HTML-Dokument ist der Container aller anderen HTML-Elemente.

DOM basiert fast ausschließlich auf der Baumstruktur. Die Abzweigungen analog vom Stamm zum Ast nennt man Knoten (Nodes). Jede Abzweigung kann weitere haben. Um diese zu unterscheiden, werden die Begriffe Eltern (Parent) und Kind (Child) verwendet. Ohne die Eltern kann ein Kind nicht existieren. Die Eltern können das Kind prägen. Das Kind kann aber auch von den Eltern differierende Auszeichnungen besitzen.

4.3.1.3.2. HTML-Dokument und HTML-DOM

Das HTML-DOM ist ein **symbolisches** Objekt der gesamten HTML-Seite **mit ihrem Kontext** im Browser. Das symbolische Objekt muss man sich als **Baumhierarchie der HTML-Elemente** im HTML-Dokument vorstellen. Diese Hierarchie ermöglicht es, für alle HTML-Elemente in Form von Objekten **jene gemeinsamen** Eigenschaften und Methoden festzulegen, die für die **Baumverwaltung**, also für die Kommunikation der HTML-Elemente in den Baumhierarchie-Ebenen zuständig sind. Dabei ist zu beachten, dass **nicht alle** Objekte die Baumstruktur verwalten können.

Zugleich ermöglicht DOM, element-spezifische Merkmale (Eigenschaften und Methoden) zu implementieren **und** deren Vererbung an Elemente der tiefer gelegenen Hierarchie-Ebene zu definieren.

Damit stellt das HTML-DOM die Grundsatzlogik für die Programmierung einer HTML-Seite per Javascript/JScript dar. Diese Scriptsprachen sind somit komplett objektorientiert und strukturiert (Punktnotation).

DOM bietet zugleich eine Möglichkeit, die Zeigerverwaltung von Eigenschaften und Methoden für den Programmierer sowie für die Scriptmaschine einfacherer und bezüglich Browserperformance optimaler ablaufen zu lassen. Diese Vereinfachung wurde als **Feldform** für diejenigen Objekte im DOM implementiert, die einen gemeinsamen Kontext, z.B. aufgrund gemeinsamer Eltern im Rahmen der Vererbung, besitzen: Vordefinierte **Collectionen** (Sammlungen) von Zeigern, die sich per **Index** verwalten lassen, vereinfachen den Programmierungsaufwand ungemein. Eine Collection ist also objektübergreifend und damit selbst kein Objekt. Das Prinzip der Collectionsbildung wird übrigens auch für einzelne Objekte realisiert.

Erzeugung von HTML-Elementen per HTML oder Script

HTML-Elemente können per HTML-Tag **oder** per Methode .createElement() erzeugt werden. Die Behandlung von per HTML-erzeugten Elementen kann von der per DOM-Methoden erzeugten **abweichen**, da DOM die Art der Erzeugungen **differenziert**. Für HTML-erzeugte Objekte gibt es **immer noch** eigene Methoden (Spezialfälle), obwohl das nicht nötig wäre (vermutlich wegen der schrittweisen Erweiterung des DOM auf alle Elemente im Dokument). Empfehlung: Man verwende - wenn möglich - **nur** Methoden, die HTML- **und** Script-erzeugte Elemente bearbeiten und erreicht dadurch die Vereinheitlichung im Quellcode.

Abbildung von Attributen eines Objektes im DOM

Attribute eines Objektes (egal ob per Script oder HTML erzeugt) sind selbst **Knoten und zugleich Kinder** des Objektes, das die Attribute besitzt. Werte von Attributen sind keine Knoten.

Wirksamkeit des HTML-Elementes

"Sichtbarkeit erst wenn Ende-Tag geparst wurde" bedeutet: Falls das Objekt ein Ende-Tag hat, kann es sich frühestens nach dem Parsen des Ende-Tags im Layout des Dokumentes auswirken. Eventuell ist ein expliziter Refresh des Dokumentes per explizitem document.location.reload() nötig.
Bei Tabellen gibt es ebenfalls die Methode .refresh().

Sichtbarkeit des HTML-Elementes

HTML-Elemente werden **in der Regel** sofort geparst (vom Browser interpretiert). Dabei ist zu beachten: Die Methoden beeinflussen nicht nur die DOM-Hierarchie sondern auch die Sichtbarkeit im Dokument. Bei Änderung des DOM wird die Sichtbarkeit erst beeinflusst, wenn das Ende-Tag geparst wurde. Das erfolgt garantiert mit dem Neuladen den Dokumentes per explizitem document.location.reload(), da die DOM-Methoden **leider nicht zwingend** das Layout des Dokumentes refreshen.

Änderung des DOM und Quellcode des Dokumentes

Änderungen an der DOM-Hierarchie sind nicht immer kongruent zu der Lage der Objekte im Quellcode des Dokumentes. Man gehe davon aus, dass die Lagen im DOM und im Dokument sich definitiv unterscheiden. Daher ist die Verwendung der Attribute ID bzw. NAME (falls diese zugelassen sind) zwingend nötig, um Eindeutigkeit der Bezüge der Objekte im Quellcode auf resultierende DOM-Objekte zur Laufzeit des Dokumentes herzustellen, wenn mit DOM-Eigenschaften und -Methoden per Script gearbeitet werden soll. Falls eine Collection existiert, kann auch diese verwendet werden.

Konsistenz des DOM und Zeiger

Die DOM-Hierarchie muss zu jedem Zeitpunkt konsistent sein (siehe Methode .normalize()). Zeiger dürfen **keinesfalls** ins Leere laufen.

DOM-Elemente entfernen

Das Entfernen eines Objektes aus der DOM-Hierarchie muss **nicht** zwingend mit
der physischen Löschung des Elementes
dem Entfernen von Kindern des Elementes
der geänderten Visualisierung des entfernten Elementes und seiner Kinder
verbunden sein.

Prototyping

Objekte können mit neuen Eigenschaften und Methoden erweitert werden (Prototyping per Eigenschaft .prototype). Um diese aber für den Browser verwaltbar zu machen, müssen diese Eigenschaften/Methoden **die vom Browser** unterstützten Eigenschaften und Methoden **letztendlich** aktivieren. Nur mit letzteren lässt der Browser eine Verarbeitung zu, weil er nur **SEINE** implementierten Eigenschaften und Methoden für das Rendern (Darstellen im Layout) von Objekten kennt.

Für private Objekte, die per new-Anweisung mit privatem Konstruktor erzeugt wurden, wird leider **nicht** die Eigenschaft .prototype erzeugt ! Prototyping kann also nur innerhalb des Konstruktors erfolgen und nicht nachträglich per Eigenschaft .prototype .
Nur für Objekte, die aus einem vordefiniertem Objekt abgeleitet sind, existiert die Eigenschaft .prototype .

DOM und Collectionen

DOM verwaltet eine Reihe von Feldern (Collectionen, Sammlungen) für Zeiger. Diese Felder müssen z.T. für die Parameterversorgung der Methoden - speziell für die Adressierung von Objekten – verwendet werden. Mit diesen Feldern wird DOM flexibler handhabbar. Diese Felder sind keine Objekte, enthalten aber gekapselte Daten des DOM mit genau **definierten** Zugriffsmöglichkeiten und Verwendungszwecken.

Normierung von Bezeichner von Methoden des DOM

Methoden mit dem Wort "Node" im Bezeichner behandeln per HTML und **in der Regel** per Script erzeugte Objekte.

Methoden mit dem Wort "Child" im Bezeichner behandeln Kind-Objekte, die genau 1 Eltern haben müssen.
(Den Begriff "Eltern" gibt es im Deutschen nur in der Mehrzahlform (Pluralform)).

Methoden mit dem Wort "Element" im Bezeichner beziehen sich auf Elemente im DOM, also Objekte,
wobei es Eltern, Kinder, per HTML und **in der Regel** per Script erzeugte Objekte sein können.

Methoden mit dem Wort "Attribut" im Bezeichner verwenden immer das attribute Objekt zum betroffenen Objekt.

Wird im Methodenbezeichner **nicht** zugleich das Wort "Node" verwendet, so werden **in der Regel** HTML-erzeugt Attribute behandelt, also
keine Attribute, die per Methode .createAttribute() erzeugt wurden.

erfolgtes komplettes Laden Dokumentes mit Parsen von </BODY>

"Nur nach dem kompletten Laden des Dokumentes möglich" bedeutet: Der Browser muss das Ende-Tag des BODY geparst haben, also </BODY>.

fehlendes BODY-Tag beim Internet Explorer

Wurde BODY nicht kodiert und befinden sich im HEAD-Teil des Dokumentes NICHT-HTML-DOM-konforme Anweisungen, die das Rendern von Objekten (Darstellung im Layout des Dokumentes) bewirken (z.B. document.write() mit Erzeugung eines HTML-Elementes), dann erzeugt der Internet Explorer genau dann **automatisch** ein BODY-Objekt, sobald das **erste** HTML-Element erzeugt wird.

Versionen von Browser, Scriptmaschine, HTML und CSS

Man beachte dabei zwingend die Unterschiede zwischen Browsern verschiedener Hersteller **und** Versionen von Browsern ein und desselben Herstellers, sowie für beide Fälle die Browser-Implementierung (Scriptmaschinen-Implementierungen) im jeweiligen Betriebssystem. Das Beachten ist Sache des Programmierers, welches noch durch die Verschiedenheit der Implementierung von Standards zu HTML-Versionen und bezüglich DHMTL zu CSS- und Script-Versionen (inklusive deren Umfang und browserhersteller-spezifischen Erweiterungen) erschwert wird. Man gehe davon aus, das Web-Projekte laufend zu pflegen sind, vor allem wenn DHTML zum Einsatz kommen soll (Kombination von HTML, Style-Sheets (CSS) und Scripten), oder man schränke sich auf weitverbreitete Browsertypen und -versionen ein (Verlust von Web-Seiten-Besuchern, die nicht-typische Browser benutzen) bzw. programmiere pures HTML mit Elementen, die von allen Browsern unterstützt werden.

Verbreitete Web-Editoren unterstützen in der Regel nicht das komplette Spektrum der Browserfähigkeiten, sondern decken typische und bedarfsgerechte Fähigkeiten ab. Das komplette Browser-Spektrum lässt sich z.Z. nur durch Programmierung von Hand unter Beachtung des DOM nutzen.

Abwärtskompatibilität des DOM und deprecated Elemente in DOM und HTML

Die **nicht** abwärtskompatible Änderung des DOM durch den Browserhersteller bzw. die zwischenzeitliche Aufgabe von HTML-Tags etc. (deprecated Elemente) ist ebenfalls zu beachten. Diese veralteten Elemente werden zwar z.T. noch unterstützt, sollten aber dringendst nicht mehr verwendet werden. Es ist Sache des Programmierers, diesbezüglich auf dem Laufenden zu sein.

Microsoft JScript.NET als Nachfolger von Microsoft JScript

Bezüglich Script favorisiert Microsoft inzwischen JScript.NET, welches abwärtskompatibel zu JScript ist, aber mehr in die Richtung der Sprachen Java oder C geht (z.B. lassen sich jetzt eine Objektklasse als Prototyp definieren und ein Datentyp bei der Variablendeklaration vergeben). Damit ist auch klar, dass sich DOM geändert hat.

Im Rahmen der NET-Produktpalette für Windows ab NT 5.x (Windows XP) wurde auch der Sprachumfang von Visual Basic verändert, allerdings in massiverem Umfang als zu JScript, so dass für VB-Programmierer ein Umdenken erfolgen **muss**.

XHTML als Nachfolger von HTML4.x

XHTML als "Nachfolger" ist eigentlich eine Erweiterung zu HTML 4.x per XML. XML wird in dieser Dokumentation nicht behandelt.

Microsoft hat XML und HTML ab IE 5.x verbunden bezüglich HTML-DOM und XML-DOM. Z.B. lassen sich XML-Dateninseln inmitten eines HTML-Dokumentes integrieren: Eine im Layout fertige Tabelle wird mit Daten gefüllt, ohne dass diese in das Tabellenlayout eingetragen werden müssen (nur die XML-Platzhalter für Daten). Vorteilhaft wird das dann erst richtig, wenn eine Datenbank die Datensätze in XML-Form erzeugen kann. Anstelle der XML-Dateninseln wird im HTML-Dokument und dessen Tabelle ein Bezug auf die XML-Dateien mit den Datensätzen erzeugt und die Tabelle kann dann diese Datensätze anzeigen. (Selbstverständlich müssen anzuzeigende Daten alle dem jeweiligen Datentyp der im Tabellenlayout kodierten XML-Platzhalter entsprechen). Das Tabellenlayout kann weiterhin mit CSS und Javascript, z.B. JScript, so dynamisch gehalten werden, dass beliebig viele Datensätze in das Tabellenlayout reinpassen (dynamische Tabellenverwaltung ist mit JScript über das DOM der Tabelle (TOM) programmierbar). Unter welchem Betriebssystem die Datenbank läuft, ist egal, hauptsache die Datenbank kann XML-konforme Datensätze als Ausgabe erzeugen. Damit wird HTML zur Anzeigeschnittstelle für beliebige Datenbanken. Der Pferdefuss: XML wird objektorientiert realisiert. Damit muss Javascript, z.B. JScript, verwendet werden, mit dem die Zeiger verwaltet werden können (zumal das Tabellenlayout zugleich auch dynamisch gehalten werden muss).

XML dient vorrangig als Datenbeschaffung z.B. für Webseiten mit dynamischen Inhalt im vorgefertigtem dynamischen Layout. Unter der NET-Software von Windows XP von Microsoft wird XML netzwerkfähig und hat zusätzliche Layoutkomponenten (erweitertes XML-DOM), die aber mithin starr bleiben, so dass weiterhin JScript.NET und CSS verwendet werden sollte.

XML wird weiterhin als Schnittstelle für die Datenübergabe zwischen diversen Datenbanken verwendet, die ihre Daten nur im XML-Format erzeugen bzw. einlesen müssen. XML ist es völlig egal, wie die Daten in der Datenbank selbst verwaltet werden. So gesehen wird XML andere Schnittstellen wie IDE ersetzen und z.B. eine HTML-orientierte Datenanzeige per Webserver im Intranet eines Unternehmens oder im Internet (e-commerce) begünstigen. Wer XML-Software entwickelt, kann auch auf einfachere Weise HTML-bezogene Datenbankenausgaben erzeugen, als es z.B. das Unternehmen SAP mit ABAP4-eigenen HMTL-Erstellungsroutinen realisiert, die nicht XML nutzen und die daher auch nur von SAP-Software bzw. für SAP programmierte Software beherrscht werden. Zugleich sind andere softwarebedingte Ausgabeformate wie RTF ersetzbar, da Layoutkomponenten in XML genormt sind, ausschliesslich in Script-Form vorliegen und maschinell sowie vom Menschen lesbar sind. Mit anderen Worten: Ob sich Datenbankhersteller und Nutzer mit herstellerspezifischer Software für Datenanzeige herumschlagen oder ob beide sich auf XML einigen werden: Es ist eine Frage der Kosten, Konsistenz, Transparenz und vereinfachten Normung der netzwerkbezogenen Datenbank-Datenbereitstellung an den Endverbraucher, zu dessen Zweck die Daten ja verwaltet werden und damit dem Verbraucher Kosten verursachen.

Wer als privater Programmierer Datenbanken nutzt und Daten dynamisch, aber in ihrer Form und im Anzeige-Layout genormt darstellen will, sollte sich mit XML und JScript beschäftigen. Wird z.B. unter Linux die Datenbank gehalten, aber für Windows das Webdesign erstellt, müssen eine Datendateischnittstelle zwischen Linux und Windows geschaffen und die Datenbank zur XML-konformen Datensatzausgabe überredet werden. Anschliessend werden die XML-Dateien unter Windows im HTML-Dokument **eingelesen** und angezeigt. Scriptsprachen, die HTML- **und** XML-Code nicht verwalten bzw. erzeugen können, bleiben allerdings außen vor.

Plain-Text, HTML-Tags und Scriptcode

Für Text-Objekte sind Plain-Text (ohne HTML-Tags und Script) und Text mit HTML-Tags und/oder Scriptcode zu unterscheiden. Besonders spielt das eine Rolle bei der Programmierung des DIV-Objektes.

<u>wichtige Hinweise:</u>

Hinweis zur Verwendung logischer Objektnamen für HTML-Elemente und browserinterne Objekte:

Durch Referenzierung eines Objektes (implizit per HTML-Tag-Kodierung oder automatisch durch den Browser) wird ein interner Zeiger gebildet, über den das Objekt erreichbar ist. Zusätzlich ist im HTML-Tag eventuell die Verwendung des Attributes ID möglich, um einen logischen Namen zum Objekt zu vergeben. Dieser logische Name wird vom Browser als Referenz auf das interne Objekt erkannt. ID ist also die öffentliche Schnittstelle zum Programmierer, der ansonsten nur schwer an interne Zeiger gelangen kann (spezielle Eigenschaft beim Internet Explorer vorhanden). ID ist also ein Zeiger, der mit seinem Bezeichner zur Referenzierung in Script direkt per Punktnotation kodiert werden kann. So gesehen ist der Zeiger für den Programmierer eine **logische** Größe. Daher wird nachfolgend auch oft beim Wert des ID-Attributes oder NAME-Attributes vom **logischen** Namen des HTML-Elementes gesprochen.

Die Verwendung von NAME bzw. ID ist nicht für jedes HTML-Objekt möglich. Es ist also die HTML-Beschreibung des Objektes zu beachten. Wie immer ist zu prüfen, ob der Browserhersteller überhaupt das Tag unterstützt und wenn ja, in welcher Browserversion mit welchen Attributen, also z.B. mit oder ohne NAME bzw. ID, und ob diese Attribute auch in JavaScript unterstützt werden. Warum eine

einheitliche Schnittstelle über NAME bzw. ID zu allen HTML-Objekten nicht vorhanden ist, bleibt unklar wie der Fakt, dass HTML trotz existierendem HTML-Standard verschiedenartig in den Browsern implementiert ist.

Ist ein HTML-Objekt im Browser implementiert, so existiert auch eine interne objektorientierte Zeigerverwaltung, aber nicht zwingenderweise **die** Schnittstelle zum Programmierer. Daher ist es verständlich, wenn Java-Script-Programmierer für eine dynamische HTML-Programmierung (DHTML) denjenigen Browser vorziehen, der ein Maximum an Schnittstellen bietet. Diese Bevorzugung steht im Widerspruch zu den Gepflogenheiten der Browsernutzer, abgesehen von den Marktpositionen, der Schnelligkeit und eventuellen Sicherheitsproblemen der Browser.

Der logische Name kann für Zugriffe auf das Objekt verwendet werden. Die Art des Zugriffes ist den nachfolgenden Objekt-Beschreibungen zu entnehmen. Aber gerade bei Zugriffen über eine Objekteigenschaft als Feld ist der logische Name sehr sinnvoll: Er erspart die Indexermittlung.

```
Bsp:
<DIV ID="Logischer_DIV_name" .....> ....>/DIV>

document.all.Logischer_DIV_name.style.left=20;      // Internet Explorer
document.layers[Logischer_DIV_name].left=20;        // Netscape unter 6.x
```

In nachfolgenden Objektbeschreibungen wird bei Feldern immer der Zugriff per Index angegeben. Daher ist für diese Fälle nicht zu vergessen, dass ein vorhandener logischer Name anstelle des Indexes verwendet werden kann. Der Zugriff über den Index funktioniert immer. Der Zugriff über den logischen Namen kann nur funktionieren, wenn das HTML-Objekt diesen als kodierbar zulässt und der Browser diese Schnittstelle unterstützt.

Hinweise zur Beschreibung von Browserobjekten:

Betriebssystem-nahe Objekte werden z.T. **nicht** beschrieben.
Es wird das Betriebssystem MS Windows 32-Bit unterstellt – ohne Differenzierung nach den Varianten.
Schwerpunkt der Beschreibungen bildet der Microsoft Internet Explorer ab 6.x unter Windows 32-Bit.

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

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 zur dynamischen Eigenschaftenveränderung zur Laufzeit:

Die zuletzt während der Laufzeit getätigte Definition ersetzt wertmäßig den aktuellen Attributwert, wenn gleiche Attributnamen bzw. Eigenschaften betroffen sind (sonst hinzufügen).

Es ist zu beachten, dass der Internet Explorer diverse Collectionen hat, die Zeiger von Objekten zur Laufzeit mit den **aktuellen Werten** von Objekt-Eigenschaften verwalten. Diese Collectionen
werden **leider nur z.T.** automatisch aktualisiert
sind anzusprechen, wenn es um laufzeitaktuelle Eigenschaften geht
dienen dem internen Ablauf der Scriptmaschine.

Z.T. werden Objekte zur Laufzeit zugleich **verschieden** verwaltet, auch wenn es sich um die gleiche Eigenschaft handelt. Damit taucht die Objekt-Eigenschaft wertmäßig mehrmals auf - eine gewollte Redundanz des Herstellers der Scriptmaschine, auch um Kompatibilität der Versionen zu erhalten. (Programmierter Code sollte bezüglich Instanzen von Objekten und deren gekapselten Daten sowie bezüglich der Variablenverwaltung im Quellcode möglichst redundanzfrei sein.)

Windows 9x ist kein Echtzeitsystem, so dass sichtbare Eigenschaftenveränderungen **nicht** selbstverständlich **synchron** laufen müssen, auch wenn sie als synchron programmiert wurden. Für Synchronisierung spielt neben der CPU-Geschwindigkeit immer noch das **Multitasking in Echtzeit** ein Rolle. Der Programmierer sollte das testen und sich nicht auf die synchrone Programmierung verlassen (Synchron-Probleme sind für bestimmte Objekte an entsprechender Stelle in dieser Dokumentation kommentiert).

Das Laden von Daten zu einem HTML-Element, z.B. Laden von mehreren (und danach auf 1 Schlag anzuzeigenden) Bildern, erfolgt parallel. Außerdem unterliegt der Internetzugang diversen Schwankungen (auch bei ADSL etc.). Damit sind Synchron-Probleme objektiv möglich. Auch wenn JScript nur z.T. objektspezifische Hilfsmittel bietet (Events), die signalisieren, wann ein Laden beendet ist, werden diese Events ebenfalls parallel erzeugt. Programmtechnisch hieße das für das Laden der Bildfolge: Es muss das "Einsammeln" aller Events in einer Routine programmiert werden, die solange aktiv ist, bis das **letzte** Event gemeldet wurde, was aber leider **nicht** bedeutet, dass die Bildfolge auch mit Eintreten des letzten Events komplett angezeigt ist und schon garnicht alle Bilder auf 1 Schlag. Gewisse Abhilfe schafft nur das Vorladen von Daten, also im Falle der Bilder: In einer Schleife das

Instanzieren eines unsichtbaren IMG-Objektes per new-Anweisung (new Image()) **mit Urlzuweisung** pro Bild **und** das Abfragen der Events auf Ladezustand pro Bild. Ende der Schleife ist das Melden des letzten Events. Erst danach sind **alle** Bilder sichtbar zu machen (style.visibility). Aber auch letzteres muss nicht Synchronisierung der Anzeige erzeugen, kann es aber zumindest besser, als das nicht steuerbare Ladeverhalten von Daten ohne "Einsammel"-Routine. (Wichtig: Bei Bildern immer in der new Image()-Anweisung die korrekte Bilddimension angeben, denn die wird zur Anzeigeberechnung verwendet. Die Bilddimension sollte zur Dimension des Umfeldes (Kontextes) passend sein, z.B. zur Dimension des DIV's, der das Bild umgibt, wenn das Bild später anhand des DIV's z.B. bewegt werden soll).

Hinweise zu Syntaxbeschreibungen:

String	bedeutet Zeichenkette oder Zeichenketten-Literal oder numerisches Literal	
Integer	bedeutet ganzzahlig (32-Bit),	positiv, negativ, inklusive 0
Floating point	bedeutet Kommazahl mit Punkt als Komma,	positiv, negativ, inklusive 0

Die Symbole [ ] schließen optionale Größen ein, die also nicht kodiert werden müssen, aber können. wobei z.T. Verschachtelung möglich ist

Bsp.: [parameter1 [,parameter2 [,parameter3]]]

Parameter 1 bis 3 sind optional
Parameter 2 bis 3 sind optional
Parameter 3 ist optional
mindestens Parameter 1 kodierbar
für Parameter3 muss zuvor Parameter2 kodiert werden
wenn Parameter 2 und / oder Parameter 3 kodiert, so mit Komma

Achtung: Bei Feldern und Collectionen bedeuten **[ ]** die Indexbegrenzer und nicht "optional".

Der Index einer Collection sollte in **[ ]** kodiert werden, kann aber auch in **( ) kodiert sein**, wenn es der Browser zulässt. In Syntaxbeschreibungen von Collectionen sind also die Klammern **[ ] zum Index nicht** die Zeichen für **optional**. Der Index ist in der Regel Integer ab 0, kann aber eventuell der **logische Objektname** laut Wert des ID-Attributes oder des NAME-Attributes sein. Wenn der logische Objektname zulässig ist, so ist es immer ein String, der in " " bzw. ' ' kodiert sein **kann**, aber nicht muss. Letzteres gilt auch für die Kodierung des Wertes des ID- bzw. NAME-Attributes in einem HTML-Tag.

Instanz bedeutet ein zur Ausführungszeit (RunTime) des Dokumentes existierendes Objekt im Hauptspeicher.

Referenz bedeutet Zeiger, also ein Bezug auf eine Instanz .

Ein Bezug ist die Adresse der Instanz, wobei die Adresse in einem Zeiger gespeichert ist. Für den Programmierer sind also Zeiger und Adresse synonym.

Eine Adresse ist die Position der Instanz im Hauptspeicher. Die Adresse liegt in einer Variablen als Zeiger auf diese Adresse. Hinweis: Die Variable selbst muss ebenfalls eine Adresse haben, worum sich der Programmierer nicht zu kümmern braucht.

Ein Zeiger ist eine Variable, die eine Adresse enthält UND zugleich mit dem Typ der Variablen bekannt gibt, wie der Browser den Wert der Variablen zu verarbeiten hat.

Bsp: var Text = "Otto";

Hinter Text steckt die Adresse der Zeichenkette "Otto" im Hauptspeicher.
Zeichenkette "Otto" ist der Wert der Variablen Text und liegt an der Position im Hauptspeicher, auf die die Variable zeigt.
Anhand von "" weiß der Browser, dass es sich um eine Zeichenkette handelt, mit der z.B. nicht addiert werden kann.
Die Variable Text hat ebenfalls eine Adresse, um die sich der Programmierer nicht zu kümmern braucht.

Objekt oder Element oder HTML-Tag oder HTML-Element sind synonym, wenn es um die objektorientierte Darstellung geht. Ein HTML-Tag ist ein Element eines Dokumentes und zugleich als Objekt realisiert, genau wie das Dokument selbst.

Dokument ist synonym zu HTML-Dokument, das z.B. die Elemente HEAD und BODY hat. Alle 3 sind Objekte !

Die Objektstrukturen im Browser basieren nur zum Teil auf der Kodierungsreihenfolge im Quellkode des Dokumentes. Der Quellcode wird vom Browser interpretiert und ausgeführt, wobei dazu das DOM-Modell des jeweiligen Browsers herangezogen wird, das die Kodierungs-Reihenfolge der Elemente im Quellkode beim Ausführen des Kodes verändern kann. Maßgebend ist die Implementierung des DOM in den Browser als Verhaltensregeln für den Browser zur Verarbeitung des Quellkodes und dessen objektorientierte Umsetzung. Es gibt im Prinzip keine Elemente, die nicht Objekt bzw. Teil eines Objektes sind.

Eine Collection ist eine Sammlung von (nicht unbedingt typengleichen) Elementen in Form eines Feldes (array), z.B. eine Sammlung von Zeigern. Eine DOM-Collection dient zur Erweiterung der Baumstruktur anhand von Zeigern als Möglichkeit der direkten Referenzierung, ohne die Baumstruktur ablaufen zu müssen (Zwischenspeicher von Zeigern). Zugriff auf Elemente der Collection erfolgt per Index oder z.T. auch per logischen Name z.B. laut ID-Attribut.

null ist der NULL-Zeiger, der einen Dummy-Wert hat (nicht numerisch Null !!!), also auf nichts weist und nichts referenziert. Eine instanzierte Variable, die den Wert null zugewiesen bekommt, zeigt dann auf nichts, und das Objekt, auf den die Variable **vor** der Null-

Zuweisung gezeigt hat, wird mit der Null-Zuweisung aus dem Hauptspeicher entfernt. Hinweis: Die Adresse der Variablen selbst wird davon nicht betroffen.

Microsoft ändert fortlaufend die Active-X-Eigenschaften von Windows und somit auch des Internet Explorers

Diese fortlaufenden Änderungen muss der Programmierer in Erfahrung bringen.
Der Programmierer kann sich definitiv nicht auf Verfügbarkeit von Active-X-Controls verlassen und muss damit rechnen, dass seine Webseiten schlagartig nicht mehr komplett laufen weil u.a. Programmcode noch nicht angepasst ist. Ebenfalls muss der Programmierer Varianten von Windows und Patchzustände beachten, die prinzipiell Kostenprobleme verursachen können.
Mit anderen Worten: Wer Microsoft-Komponenten nutzt, muss wissen, was ihm blüht ... siehe nachfolgende Beispiel für Risiken.

Prinzipielle Lizenzprobleme für den Programmierer

Microsoft verlangt Linzensierung von Windows. Bezüglich Windows-Versionen gibt es die Updatestufen z.B. per Servicepacks
Ein Windows mit Servicepack fällt unter die Lizenz des geupdateten Windows.
Ein Windows mit Vorversion zum Servicepack bedarf einer anderen Lizenz.
Will man z.B. den Internet Explorer 7 und 6 parallel testen, benötigt man 2 Windowslizenzen, da beide Versionen nicht parallel installierbar. Dazu kommt, dass es den IE 6 in 2 Versionen gibt: Win SP1 und SP2 (IE 7 nur ab Win SP2).
Für 3 Browserversionen benötigt man 3 Windowslizenzen, will man parallel testen.
Ein Blick auf Browser-Konkurrenzprodukte klärt die Sachlage unschlagbar: Opera ist z.B. parallel installierbar.

Hinweis: Man suche doch mal im Internet nach einem kostenlosen HTTP-Server vom Microsoft, um IE-Seite testen zu können, die JScript nutzen (inklusive Debugger). Denn sollte kein kostenloses Angebot findbar sein, kommen die Kosten von Entwicklungssoftware zum IE hinzu. Ein Blick auf Konkurrenz-HTTP-Server klärt die Sachlage: Apache-HTTP-Server ist kostenlos, allerdings nicht einfach einzurichten (Hinweis: Der HTTP-Server sollte virtuelle Hosts einrichten können und korrekt mit der Firewall des Users zusammenarbeiten können).

Abänderungen wegen Sicherheitspatches der jeweiligen Windows-Versionen

Abschaltungen von Active-X-Controls erfolgen auch im Rahmen der Sicherheitspatches zu Windows-Versionen.
Es ist auch möglich, dass wegen Sicherheitslücken abgeschaltet wird und somit Komponenten einer Webseite je nach Windowsversion nicht mehr laufen.

Im Rahmen der Sicherheitspatches ist es Microsoft sogar gelungen, Webseiten, die den MS-Encoder zur Komprimierung von HTML- und JScript-Code nutzen, schlagartig unnutzbar zu machen: Ein Bug in einem Patch zu Windows XP - Q918899
Das Patch verusacht IE-Browser-Absturz bei per MS ScriptEncoder gepacktem JScript unter SP1 und 2 wenn HTTP 1.1 mit Kompression genutzt wird z.B. bei
onlick-Handler auf IMG
klick ins Fenster per aktivem Popup

Der Absturz ist "read" -Fehler von immer ein und derselben Speicherstelle.
User, die dieses Patch installiert haben, können ab sofort keine IE-Seiten mit codiertem Script mehr ansehen.
Microsoft stellt Abhilfe nach geraumer Zeit zur Verfügung, jedoch spezifisch nach Windows XP-Version:
Patch Q918899 für
Windows XP SP1Download für jedermann bereitgestellt
SP2 nur auf kostenpflichte telefonische Anfrage des Users per Downloadlink bereitgestellt, da Microsoft explizit die User registriert haben will, bei denen das Patchproblem auftritt (User muss sich Telefonnummer besorgen)
Solange also das Patch zum fehlerhatem Patch vom User nicht installiert wird, z.B. weil der User keine Ahnung hat, dass und wo er sich die Telefonnummer von Microsoft besorgen muss bzw. zu besorgen hat, wird der User IE-Seiten mit komprimierten Code dauerhaft nicht nutzen können. (Microsoft-Support ist z.T. nur in Englisch).

Abänderungen wegen Browser-Inkompatibilität

Popupblocker-Fehler

Die Microsoft Browser-Version IE 7 ist nicht abwärtskompatibel bezüglich Popup per window.createPopup()
Popup per window-Objekt ist ein Markenzeichen des IE, das im IE 7 nicht mehr fehlerfrei nutzbar ist.
Der Fehler liegt in der Popup-Blockerverwaltung des IE und wurde mit dem IE 7 implementiert.
Der Fehler tritt nicht auf, wenn ein Fenster per window.open() erzeugt wurde.
Bedingung:
Scripfehleranzeige ist erlaubt im IE 7
Popupblocker ist im IE abgeschaltet
ein aktives Fenster (Register) mit Dokument, dass fortlaufend (rekursiv) genau 1 window.popup per .show()erzeugt.
ein weiteres Fenster (Register) z.B. leere Seite (about:blank)
beide (Register) liegen in einer gemeinsamen IE-Instanz
Ablauf: Wird Focus auf Register der leeren Seite gehalten und wird parallel das Popup per .show() erzeugt, bricht der Browser das Dokument mit .show() ab (Scriptfehler).

Der Popupblocker für die leere Seite verursacht den Programmfehler im Dokument mit .show(). Es wird folgende Meldung angezeigt (in der Informationsleiste):

'Ein Popup wurde geblockt. Klicken Sie hier, um das Popup bzw. weitere Optionen anzuzeigen.'

Die Bedeutung der Meldung laut Microsoft-Hilfe im IE 7:

Der Popupblocker hat ein Popupfenster geblockt. Sie können den Popupblocker deaktivieren oder Popups temporär zulassen, indem Sie auf die Informationsleiste klicken.

Die Realität zur obigen Meldung ist völlig anders:

Linke oder rechte Maus auf die Meldung liefert z.B. Einstellungen darunter

Popupblocker einschalten
weitere Informationen

jedoch keine Möglichkeit wie laut Bedeutung

Damit gilt: Der abgeschaltete Popupblocker ist in Wirklichkeit aktiv.

Pikant: Ein Popup erscheint normalerweise auch über fremde Fenster, die nicht das Popup erzeugt haben (z.B. Fenster einer Windowsanwendung z.B. einer anderen IE-Instanz)

Der Popupblocker des IE bemeckert aber NUR Webseite, die das Popup erzeugt.

Durch das Abwürgen von Popup wird das Popup natürlich auf und für anderen Seiten nicht relevant; im Falle einer anderen IE-Instanz also auch für diese nicht relevant, obwohl diese Instanz per Popupblocker verwaltet wird.

Der Popupblocker beschneidet die Popup-Reichweite an der Wurzel, ist aber nicht objektorientiert zu den anderen Webseiten (die nicht das Popup erzeugt haben).

Der Popupblocker ist nicht als Filter aufgesetzt sondern reingestrickt worden.

Der Popupblockerfehler verändert die Eventverwaltung:

Es werden u.a. ignoriert
onfocus
onblur
onfocusin
onfocusout

und viele andere, so dass trotz Events z.B. des Body der Popupblockerfehler entsteht.

```
// nachfolgender Code setzt focus nicht neu: Fenstereintrag in Taskleiste blinkt eventuell
window.focus();
window.document.focus();
if(document.body!=null)
{if(document.body.style!='hidden')          // wenn  hidden so focus() nicht möglich (Scriptfehler erzeugt)
 {document.body.focus();}
}
// wenn paralleles Fenster offen (on oder offline), so Scriptfehler erzeugt
popupzeiger.show(...);
```

Hinweis: Der Popupfehler ist so elementar, dass die vielen Beta-Testphasen des IE mehr als fragwürdig erscheinen, wie die Angabe von Microsoft, dass Code neu programmiert wurde, um den IE sicherer zu machen.

focus-Methode beim IE 7

windows.focus() document.focus() und body.focus() funktionieren NICHT

zwischen Register in einem IE-Fenster
zwischen Fensters z.B. in Taskleiste

Hinweis:

.focus() setzt Element aktiv, gibt dem Element den Focus und feuert dann onfocus
.setActive() ist Teilmenge von .focus(): nur das aktiv setzen
funktioniert nicht mit allen Elementen, mit denen .focus() funktioniert

animierte Gif (mit Timer)

Animierte Gifs (mit Timer), die unter IE 6 korrekt laufen, müssen unter IE 7 im Timer nicht mehr laufen:

z.B. garnicht mehr sichtbar, oder Timer nicht verwendet.

Dann müssen animierte Gif-Bilder nach IE-Version bereitgestellt werden.

Abänderungen wegen Rechtstreitigkeiten von Microsoft mit Fremdanbietern

Ein sehr bekanntes Beispiel ist die nachträglich eingeführte Einschränkung von Active-X-Controls wegen Patentwahrung durch Microsoft, wobei für den JScript-Programmierer massive Änderungen eintreten.

Wegen Patentwahrung hat Microsoft ein zunächst freiwilliges Patch herausgegeben, dass bei ActiveX-Control per APPLET, EMBED oder OBJECT, die auf dem Bildschirm rendern (mit oder ohne Userschnittstelle), dafür sorgt, dass bei mouseover über das Control eine Sprechblase erscheint, die darauf hinweist, dass das Objekt als ActiveX-Control klickbar ist.
Diese Sprechblase erscheint auch, wenn das Control keine Userschnittstelle hat, also diese gar nicht klickbar ist.

Es wurde das Eventmodell gleichzeitig geändert:

Es werden alle Events solange unterdrückt, bis der User die Sprechblase geklickt hat.

Das Klicken muss auf das Objekt im Sprechblasenrahmen erfolgen, der so groß ist, wie die Dimension, in der gerendert wurde.
Es muss also ERST per Mausklick das Control aktiviert werden, ehe das Control klickbar und damit die Eventsteuerung aktiviert ist.
Ein Control, dass programmtechnisch zwar was rendert, aber ansonsten ohne sichtbare programmtechnisch startet, muss ebenfalls geklickt werden, obwohl es bereits läuft und es nichts zu klicken gäbe (wenn keine Eventsteuerung eingebaut wurde).
Wegen blockierter Eventsteuerung ist also die Sprechblase z.B. nicht automatisch klickbar.
Die Eventauslösung per nicht-objekteigenen Eventhandler, der für das Objekt per fireEvent() ein Event auslöst, ist solange blockiert, bis der User die Sprechblase geklickt hat.

style.visibility='hidden' wird ignoriert

Die Sprechblase erscheint auch dann, wenn das Control mit style.visibility='hidden' belegt ist, also sich unsichtbar rendert:
Der Sprechblasenrahmen hat genau die Dimension wie die des unsichtbaren Controls. Der Sprechblasenrahmen erscheint also Zusammenhangslos, und der User weiß nicht, warum er klicken soll, wenn er nichts sieht. Vor allem weiß er nicht, WAS er klickt ... ideale Basis für Schadsoftware per Script.

Diese Sprechblase erscheint nur DANN NICHT, wenn die Userschnittstelle mit Breite == Höhe == 0 gerendert wird. Sollte die Userschnittstelle in einem Container liegen, z.B. DIV, dann wird der Container, wenn er in der Dimension kleiner ist, also die Userschnittstelle, angepasst. Daher muss der Container ebenfalls mit Breite == Höhe == 0 gerendert werden. Wegen Dimensionierung auf 0 sollte style.visivility="hidden" sein. Im Falle eines Containers reicht es, den style des Containers zu ändern, da visibility normalerweise vererbt wird an Kinder, also auch an das Control.

Abänderung wegen Abschaltungen

DirectX ist wegen Abschaltung von Active-X--Controls nicht mehr abwärtskompatibel:
Z.B. wurde bei Win XP SP2 Direct Animation aus DirectX schlagartig durch Abschaltung von Bibliotheken dezimiert, die es bei Win XP SP1 aber noch gibt.

Hier ein Beispiel aus dem Jahr 2004: Abschaltungen von Active-X-Controls

ActiveX-Controls und Unterstützung/Verbot 20041215

erlaubt sind noch

Tabular Data-Steuerelement{333C7BC4-460F-11D0-BC04-0080C7055A83} Das TDC (Tablular Data-Steuerelement) ermöglicht die Weiterverarbeitung von Daten, die nur im Textformatvorliegen, beispielsweise durch Darstellung in einer Tabelle oder Sortierung. Weitere Informationen:• http://msdn.microsoft.com/workshop/database/tdc/tabular_data_control_node_entry.asp(http://msdn.microsoft.com/workshop/database/tdc/tabular_data_control_node_entry.asp)

Microsoft Agent Control - Version 2.0 {D45FD31B-5C6E-11D1-9EC1-00C04FD7081F} Microsoft Agent repräsentiert die neue Generation des ursprünglichen Office-Assistenten. Anstatt den Assistenten jedoch innerhalb eines Rahmens darzustellen wird hier lediglich der Charakter bzw. Agent selbst dargestellt und kannauch in Webseiten verwendet werden. Weitere Informationen:• http://msdn.microsoft.com/library/partbook/egvb6/introducingmicrosoftagent.htm(http://msdn.microsoft.com/library/partbook/egvb6/introducingmicrosoftagent.htm)

Microsoft MSChat-Steuerelement-Objekt 2.0 - 2.5 {D6526FE0-E651-11CF-99CB-00C04FD64497} Dieses Steuerelement wird von Webautoren verwendet, um text- und graphisch basierte Chatgemeinden für Echtzeitkonversationen im Web zu erstellen.

Microsoft ActiveX Upload-Steuerelement, Version 1.5 {886e7bf0-c867-11cf-b1ae-00aa00a3f2c3} Dieses Steuerelement kann auf vielerlei Art genutzt werden, um auf einfache Weise Webinhalte via Drag and Drop zuveröffentlichen. Weitere Informationen:• 230298 (http://support.microsoft.com/kb/230298/DE/) - Posting Acceptor Release Notes
• http://msdn.microsoft.com/workshop/management/tools/reference/file_upload_control.asp (http://msdn.microsoft.com/workshop/management/tools/reference/file_upload_control.asp)

verboten sind

Datenbindung RDS {BD96C556-65A3-11D0-983A-00C04FC29E36} {BD96C556-65A3-11D0-983A-00C04FC29E33} Die RDS (Remote Data Service) Steuerelemente ermöglichen dem Browser, client-basierte SQL Abfragen an einen Webserver zu stellen. Inzwischen wurde RDS jedoch durch neuere Standards wie SOAP abgelöst, von einer weiteren Verwendung von RDS wird daher abgeraten. Weitere Informationen:• 184375 (http://support.microsoft.com/kb/184375/DE/) - Sicherheitsaspekte bei RDS 1.5, IIS 3.0 oder 4.0 und ODBC

http://msdn.microsoft.com/library/en-us/iissdk/iis/remotedatabindingwithremotedataservice.asp (http://msdn.microsoft.com/library/en-us/iissdk/iis/remotedatabindingwithremotedataservice.asp)

http://msdn.microsoft.com/library/en-us/dnmdac/html/data_mdacroadmap.asp (http://msdn.microsoft.com/library/enus/dnmdac/html/data_mdacroadmap.asp)

XMLDSO, XMLDocument, DOMDocument, und XMLIslandPeer {550dda30-0541-11d2-9ca9-0060b0ec3d39} {CFC399AF-D876-11d0-9C10-00C04FC99C8E} {e54941b2-7756-11d1-bc2a-00c04fb925f3}{7108ECB4-AFDC-11D1-ADC1-00805FC752D8}XMLDSO, XMLDocument, DOMDocument, und XMLIslandPeer ermöglichen die Verarbeitung von XML Daten,etwa die Bindung von HTML Elementen an einen XML Datensatz, oder das Einlesen, Manipulieren, und Zurückschreiben von XML Daten.

Die Steuerelemente DOMDocument und XMLIslandPeer bzw. die dazugehörigen ClassIDs sind nicht mehr aktuell, so dass von einer generellen Freigabe dieser Steuerelementgruppe abgeraten wird. Weitere Informationen:• http://msdn.microsoft.com/library/en-us/xmlsdk/htm/xml_concepts2_7ook.asp(http://msdn.microsoft.com/library/en-us/xmlsdk/htm/xml_concepts2_7ook.asp)

Internet Explorer

Active Setup / IE Active Setup-Steuerelement {F72A7B0E-0DD8-11D1-BD6E-00AA00B92AF1}Dieses Steuerelement enthält die in Microsoft Security Bulletin MS99-037 beschriebene Sicherheitsanfälligkeit. Umeine weitere Ausführung zu verhindern wurde im Rahmen dieses Security Bulletins ein Kill-Bit gesetzt, so dass selbst bei einer Freigabe dieses Controls eine Ausführung blockiert wird. Weitere Informationen:• http://www.microsoft.com/technet/security/bulletin/ms99-037.mspx(http://www.microsoft.com/technet/security/bulletin/ms99-037.mspx)

http://www.microsoft.com/technet/security/bulletin/fq99-037.mspx (http://www.microsoft.com/technet/security/bulletin/fq99-037.mspx)

240797 (http://support.microsoft.com/kb/240797/DE/) - So verhindern Sie die Ausführung von ActiveX-Steuerelementen in Internet Explorer

Media Player / Active Movie Runtime {A4001DE0-7075-11d0-89AB-00A0C9054129} Die Funktionalität dieses Steuerelements wird wird nun durch das Windows Media Player ActiveX Steuerelement abgedeckt. Das Active Movie Runtime Steuerelement wird daher nicht mehr unterstützt, von einer Freigabe wirdabgeraten.

Media Player / ActiveMovie-Steuerelement{05589FA1-C356-11CE-BF01-00AA0055595A} Die Funktionalität dieses Steuerelements wird wird nun durch das Windows Media Player ActiveX Steuerelement abgedeckt. Das Active Movie Steuerelement wird daher nicht mehr unterstützt, von einer Freigabe wird abgeraten.

Media Player / Microsoft NetShow Player {2179C5D3-EBFF-11CF-B6FD-00AA00B4E220} Die Funktionalität dieses Steuerelements wird wird nun durch das Windows Media Player ActiveX Steuerelement abgedeckt. Das NetShow Player Steuerelement wird daher nicht mehr unterstützt, von einer Freigabe wird abgeraten.

Media Player / Windows Media Player {22D6F312-B0F6-11D0-94AB-0080C74C7E95} Dies ist das Steuerelement für Windows Media Player version 6.4 und war Installationsbestandteil bis einschließlich

Windows Media Player Version 8. Ab Windows Media Player 9 wurde diese ClassID durch die neue ClassID {6BF52A52-394A-11D3-B153-00C04F79FAA6} abgelöst, deren Verwendung stattdessen empfohlen wird. Ab Windows Media Player Version 9 wirdferner die alte ClassID anhand eines Wrappers automatisch auf die neue ClassID umgeleitet. Die ClassID für Windows Media Player Version 9 ist jedoch nicht in der Liste der vom Administrator genehmigten Steuerelemente enthalten, undmuss bei Bedarf manuell hinzugefügt werden.

Animierte Schaltflächen{0482B100-739C-11CF-A3A9-00A0C9034920} Dieses Steuerelement erlaubte in frühen Versionen des Internet Explorer die Verwendung animierter Schaltfächen auf Webseiten. Das Steuerelement wird nicht mehr unterstützt und dürfte nur noch vereinzelt im Einsatz sein. Von derFreigabe des Steuerelements wird daher abgeraten.

IE Label-Steuerelement
{99B42120-6EC7-11CF-A6C7-00AA00A47DD2} Dieses Steuerelement ist nicht mehr aktuell und seit Internet Explorer Version 5 auch kein Bestandteil der Installation mehr. Das Steuerelement wird nicht mehr unterstützt und dürfte nur noch vereinzelt im Einsatz sein. Von einer Freigabe des Steuerelements wird daher abgeraten. Weitere Informationen:• 190045 (http://support.microsoft.com/kb/190045/DE/) - INFO: ActiveX Controls That Are Removed from Internet Explorer 5

IE Menu-Steuerelement {74701400-9DD9-11CF-A662-00AA00C066D2} Dieses Steuerelement ermöglicht die Handhabung von Menüstrukturen in Webseiten, wird jedoch nicht mehr unterstützt und dürfte nur noch selten Verwendung finden. Von einer Freigabe des Steuerelements wird daher abgeraten.

IE Preloader-Steuerelement {16E349E0-702C-11CF-A3A9-00A0C9034920} Dieses Steuerelement ermöglichte das Vorladen von Webseiten, ist jedoch inzwischen nicht mehr aktuell, wird nicht mehr unterstützt und dürfte nicht mehr im Einsatz sein. Aufgrund einer potentiellen Sicherheitsanfälligkeit in diesem Steuerelement wird von einer Freigabe abgeraten. Weitere Informationen:• 231452 (http://support.microsoft.com/kb/231452/DE/) - Update Available for "Legacy ActiveX Control" Issue

IE Timer-Steuerelement {59CCB4A0-727D-11CF-AC36-00AA00A47DD2} Dieses Steuerelement ist nicht mehr aktuell und seit Internet Explorer Version 5 kein Bestandteil der Installation mehr. Das Steuerelement wird nicht mehr unterstützt und dürfte nur noch vereinzelt im Einsatz sein. Von einer Freigabe des Steuerelements wird daher abgeraten. Weitere Informationen:• 190045 (http://support.microsoft.com/kb/190045/DE/) - INFO: ActiveX Controls That Are Removed from Internet Explorer 5

MCSiMenü {275E2FE0-7486-11D0-89D6-00A0C90C9B67} Dieses Steuerelement dient der Anpassung von Popupmenüs, ist jedoch nicht mehr aktuell und wurde nach Windows 98 nicht mehr ausgeliefert. Das Steuerelement wird nicht mehr unterstützt und dürfte nur noch vereinzelt im Einsatz sein. Von einer Freigabe des Steuerelements wird daher abgeraten.

Popupmenüobjekt {7823A620-9DD9-11CF-A662-00AA00C066D2} Dieses Steuerelement ist nicht mehr aktuell und seit Internet Explorer Version 5 kein Bestandteil der Installation mehr. Das Steuerelement wird nicht mehr unterstützt und dürfte nur noch vereinzelt im Einsatz sein. Von einer Freigabe des Steuerelements wird daher abgeraten. Weitere Informationen:• 190045 (http://support.microsoft.com/kb/190045/DE/) - INFO: ActiveX Controls That Are Removed from Internet Explorer 5

Microsoft Agent Control - Version 1.5 {F5BE8BD2-7DE6-11D0-91FE-00C04FD701A5} Microsoft Agent repräsentiert die neue Generation des ursprünglichen Office-Assistenten. Anstatt den Assistenten jedoch innerhalb eines Rahmens darzustellen wird hier lediglich der Charakter bzw. Agent selbst dargestellt und kann auch in Webseiten verwendet werden. Diese Version des Steuerelements ist jedoch nicht mehr aktuell und wird nicht mehr unterstützt. Von einer Freigabe des Steuerelements wird daher abgeraten. Weitere Informationen:• http://msdn.microsoft.com/library/partbook/egvb6/introducingmicrosoftagent.htm (http://msdn.microsoft.com/library/partbook/egvb6/introducingmicrosoftagent.htm)

4.3.1.3.3. HTML-DOM

DOM wurde browserspezifisch und zugleich abhängig von der Browserversion bei gleichem Browserhersteller implementiert.

Aus Platzgründen sind z.T. die Beispiele in den Anhang verlegt worden. Viele Objekte haben gemeinsame Methoden und Eigenschaften, so dass diese Art der Platzierung für Übersichtlichkeit sorgt.

4.3.1.3.3.1. HTML-DOM beim Netscape 6.x (Übersicht)

HTML-DOM wird ab NS 6.x um Eigenschaften, Methoden sowie um das Objekt document.documentElement erweitert.

4.3.1.3.3.1.1. Methoden vom Objekt document im HTML-DOM des Netscape

Hinweis: Bezeichner von Tags immer in Grossbuchstaben

4.3.1.3.3.1.1.1. document.getElementById() des Netscape

Zeiger auf HTML-Element ermitteln: Element **muss** mit **ID-Attribut** versehen sein (optional zusätzlich Attribut NAME).

Bsp: <P ID="**P_ID**" .....> ... </P>

var P_Tag_Zeiger = document.getElementById("**P_ID**");

P_Tag_Zeiger dient zur Referenzierung von eigenschaften und Methoden des P-Elementes

Hinweise:

Der Internet Explorer unterstützt z.T. ebenfalls diese Methode, so dass dann diese Methode nicht zur Identifizierung des NS 6.x verwendet werden kann.

4.3.1.3.3.1.1.2. document.getElementsByTagName() des Netscape

Zeiger auf Feld aller Objekte im Dokument ermitteln, die einen gemeinsamen Tag besitzen.

Feldeigenschaft	.length	Anzahl der Objekte
Feldmethode	.item()	Zeiger auf Feldelement anhand Feldindex liefern
Feldindex		Integer, ab 0

Element muss **nicht** das ID-Attribut kodiert haben (kann aber).

Bsp.: var Feld_aller_P_Tag = document.getElementsByTagName("P");

Bsp.: var ZeigerAufBODY = document.getElementsByTagName("BODY").item(0); // es gibt nur 1 BODY im Dokument

Bsp.:
```
var myTable = document.createElement("TABLE");
// ... diverse TR erzeugen
var TR_Feld = tableBody.getElementByTagName("TR")
```

Anzahl der TR laut TR_Feld.length ab 1

Feldelement laut TR_Feld.item(index_ab_0)

z.B. lastRow = TR_Feld.item(allRows.length-1);
firtRow TR_Feld.item(0);

4.3.1.3.3.1.1.3. document.createElement() des Netscape

HTML-Element leer erzeugen und Zeiger liefern, aber leider ohne Einbindung des Objektes in das Dokument.

Beispiel für nicht-komplexes Element wie <P>:

```
var P_Tag_Zeiger=document.createElement("P");
```

Beispiel für komplexes Element wie <TABLE>:

```
// Zeiger auf BODY holen
var ZeigerAufBODY = document.getElementsByTagName("body").item(0);

// Tabelle leer erzeugen mit ID
var ZeigerAufTabelle = document.createElement("TABLE");
ZeigerAufTabelle.id = "ID_Table";

// Tabellenkörper leer erzeugen
```


```
var ZeigerAufTBODY = document.createElement("TBODY");

// Tabellenkörper zeilenweise füllen (3 Zeilen)
for (var i = 0; i < 3; i++)
{
        // Tabellenzeile leer erzeugen
        var ZeigerAufTR = document.createElement("TR");

        // Tabellenzeile zellenweise füllen (3 Zellen)
         for (var j = 0; j < 3; j++)
        {
                // Tabellenzelle leer erzeugen
                var  ZeigerAufTD = document.createElement("TD");

                // Tabellenzelle dimensionieren
                ZeigerAufTD.setAttribute("WIDTH","50");
                ZeigerAufTD.setAttribute("HEIGHT","50");

                // Tabellenzelle mit Text füllen
                var textNode = document.createTextNode("Test");
                ZeigerAufTD.appendChild(textNode);

                // Tabellenzelle an die Tabellenzeile anhängen als Kind der Zeile
                ZeigerAufTR.appendChild(ZeigerAufTD);
        }

        // gefüllte Tabellenzeile an den Tabellenkörper anhängen als Kind des Körpers
         ZeigerAufTBODY.appendChild(ZeigerAufTR);
}

// Tabellenkörper an die Tabelle anhängen als Kind der Tabelle
ZeigerAufTabelle.appendChild(ZeigerAufTBODY);

// Tabelle an BODY des Dokumentes anhängen als Kind des Dokumentes
ZeigerAufBODY.appendChild(ZeigerAufTabelle);
```

4.3.1.3.3.1.1.4. document.createAttribute() des Netscape

Attribut eines HTML-Elementes ohne Wert erzeugen.

```
var Attribut_Zeiger = document.createAttribute("attribut_bezeichner");
```

```
Bsp:    var myTable = document.createElement("TABLE");
        myTable.id ="TableOne";
        myTable.border = 1;
```

4.3.1.3.3.1.1.5. document.setAttribute() des Netscape

Attribut eines HTML-Elementes mit Wert erzeugen.

```
P_Tag_Zeiger.setAttribute("attribut_bezeichner", wert);

        wert ja nach Attribut z.B. als String
```

```
Beispiel:  var myImg = document.createElement("IMG");
        myImg.setAttribute("id","ID_IMG");
        myImg.setAttribute("src","test.gif");
```

4.3.1.3.3.1.1.6. document.createTextNode() des Netscape

Textknoten erzeugen (kein HTML-Knoten)

```
Bsp: var Text_Zeiger = document.createTextNode("freier_text");
```

Bsp: Füllen einer Tabelle:

```
// Zeiger auf  BODY holen
var ZeigerAufBODY = document.getElementsByTagName("body").item(0);

// Tabelle leer erzeugen mit ID
var ZeigerAufTabelle = document.createElement("TABLE");
ZeigerAufTabelle.id = "ID_Table";

// Tabellenkörper leer erzeugen
var ZeigerAufTBODY = document.createElement("TBODY");
```


```
// Tabellenkörper zeilenweise füllen (3 Zeilen)
for (var i = 0; i < 3; i++)
{
        // Tabellenzeile leer erzeugen
        var ZeigerAufTR = document.createElement("TR");

        // Tabellenzeile zellenweise füllen (3 Zellen)
         for (var j = 0; j < 3; j++)
        {
                // Tabellenzelle leer erzeugen
                var  ZeigerAufTD = document.createElement("TD");

                // Tabellenzelle dimensionieren
                ZeigerAufTD.setAttribute("WIDTH","50");
                ZeigerAufTD.setAttribute("HEIGHT","50");

                // Tabellenzelle mit Text füllen
                var textNode = document.createTextNode("Test");
                ZeigerAufTD.appendChild(textNode);

                // Tabellenzelle an die Tabellenzeile anhängen als Kind der Zeile
                ZeigerAufTR.appendChild(ZeigerAufTD);
        }

        // gefüllte Tabellenzeile an den Tabellenkörper anhängen als Kind des Körpers
         ZeigerAufTBODY.appendChild(ZeigerAufTR);
}

// Tabellenkörper an die Tabelle anhängen als Kind der Tabelle
ZeigerAufTabelle.appendChild(ZeigerAufTBODY);

// Tabelle an BODY des Dokumentes anhängen als Kind des Dokumentes
ZeigerAufBODY.appendChild(ZeigerAufTabelle);
```

4.3.1.3.3.1.1.7. document.createTextRange() des Netscape

Textbereich erzeugen

Bsp.: var TextRange=document.body.createTextRange();

4.3.1.3.3.1.2. Objekt document.documentElement des Netscape

Das Objekt ist ein symbolisches Objekt und repräsentiert ein HTML-Element.
Die Referenzierung des HTML-Elementes erfolgt über document.getElementById() oder document.getElementsByTagName().

Hinweis: Bezeichner von Attributen oder Tags immer in Kleinbuchstaben

4.3.1.3.3.1.2.1. Eigenschaften von document.documentElement des Netscape

.attributes	Zeiger auf Feld der Attrivute Bsp: var Feld = zeiger_auf_element.attributes;
.attributes.length	Anzahl der Attribute , ab 1
.childNodes	Zeiger auf Feld der Kinderknoten Bsp: var Feld = zeiger_auf_element.childNodes;
.childNodes.length	Anzahl der Kinder, ab 1 Bsp: var Anzahl_kinder = zeiger_auf_element.childNodes.length;
.childNodes.nodeValue	Zeiger auf Wert eines Kindes Bsp: var Wert_des_kindes = zeiger_auf_element.childNodes.item(index_ab_0). nodeValue;
.firstChild	Zeiger auf das erste Kind für das HTML-Dokument wird immer HEAD geliefert Bsp: var zeiger_auf_erstes_kind = zeiger_auf_element.firstChild; Hinweis: document.documentElement.firstChild.ownerDocument.documentElement.nodeName liefert HTML
.lastChild	Zeiger auf das letzte Kind Bsp: var zeiger_auf_letztes_kind = zeiger_auf_element.lastChild; Hinweis: document.documentElement.lastChild.ownerDocument.documentElement.nodeName

liefert BODY

.nodeName Zeiger auf Eigenschaften eines Elementes, egal ob Eltern oder Kind
Bsp.: var Element_Tag = zeiger_auf_element.nodeName;

.nodeValue Zeiger auf Wert eines Knotens
Bsp.: var Element_Wert = zeiger_auf_element.nodeValue;

.nodeType Zeiger auf Knotentyp
Bsp.: var Element_Typ = zeiger_auf_element.nodeType;

z.B. 1 = Tag wie HTML, BODY HEAD etc.
2 = Attribut wie ID, NAME etc.
3 = reiner Text der Tags wie z.B. in. <A> .... </A>

.ownerDocument Zeiger auf das HTML-Dokument
Bsp: var zeiger_auf_html_dokument = zeiger_auf_element.ownerDocument;

.parentNode Zeiger auf Eltern-Element
Bsp: var zeiger_auf_eltern = zeiger_auf_element.parentNode;

.text den Text des HTML-Elementes liefern, das die Eigenschaft .text unterstützt
z.B. bei OPTION
Bsp: var Text = zeiger_auf_element.text;

.value den Wert des HTML-Elementes liefern, das die Eigenschaft .value unterstützt
z.B. SELECT
Bsp: var Wert = zeiger_auf_element.value;

4.3.1.3.3.1.2.2. Methode von document.documentElement des Netscape

.appendChild() HTML-Element einbinden als Kind
Bsp: zeiger_auf_eltern.appendChild(zeiger_auf_zu_neues_kind);

.attributes.item() Zeiger auf Attribut eines HTML-Elementes ermitteln

Bsp: var Attribut_Zeiger=P_Tag_Zeiger.attributes.item(index_des_attributes);

alle Attribute werden ab 0 in der Kodierungsrichtung von links nach rechts indiziert
0 ist also das linkeste Attribut, also das erste

.childNodes.item() Zeiger auf ein Kind
Bsp: var zeiger_auf_kind = zeiger_auf_element.childNodes.item(index_ab_0);

getAttribute() Wert eines Attribute ermitteln

Bsp: var wert=P_Tag_Zeiger.getAttribute("attribut_bezeichner");

.hasChildNodes() prüfen ob Kinder vorhanden sind
Bsp: var kinder_vorhanden = zeiger_auf_element.hasChildNodes();

vorhanden so true geliefert

.insertBefore() HTML-Element einfügen

zeiger_auf_eltern.insertBefore(zeiger_auf_einzufügendes_kind, zeiger_auf_kind_vor_dem_eingefügt_wird);

Beispiel:

```
// Zeiger auf  BODY holen
var ZeigerAufBODY = document.getElementsByTagName("body").item(0);

// Tabelle leer erzeugen mit ID
var ZeigerAufTabelle = document.createElement("TABLE");
ZeigerAufTabelle.id = "ID_Table";

// Tabellenkörper leer erzeugen
var ZeigerAufTBODY = document.createElement("TBODY");

// Tabellenkörper zeilenweise füllen (3 Zeilen)
for (var i = 0; i < 3; i++)
{
    // Tabellenzeile leer erzeugen
```


```
                        var ZeigerAufTR = document.createElement("TR");

                        // Tabellenzeile zellenweise füllen (3 Zellen)
                         for (var j = 0; j < 3; j++)
                        {
                                // Tabellenzelle leer erzeugen
                                var  ZeigerAufTD = document.createElement("TD");

                                // Tabellenzelle dimensionieren
                                ZeigerAufTD.setAttribute("WIDTH","50");
                                ZeigerAufTD.setAttribute("HEIGHT","50");

                                // Tabellenzelle mit Text füllen
                                var textNode = document.createTextNode("Test");
                                ZeigerAufTD.appendChild(textNode);

                                // Tabellenzelle an die Tabellenzeile anhängen als Kind der Zeile
                                ZeigerAufTR.appendChild(ZeigerAufTD);
                        }

                        // gefüllte Tabellenzeile an den Tabellenkörper anhängen als Kind des Körpers
                         ZeigerAufTBODY.appendChild(ZeigerAufTR);
                }

                // Tabellenkörper an die Tabelle anhängen als Kind der Tabelle
                ZeigerAufTabelle.appendChild(ZeigerAufTBODY);

                // Tabelle an BODY des Dokumentes anhängen als Kind des Dokumentes
                ZeigerAufBODY.appendChild(ZeigerAufTabelle);

                var ZeigerAufTRNeu = document.createElement("TR");
                var ZeigerAufTRAlt = document.getElementByTagName("TR").item(0);
                ZeigerAufBODY.insertBefore(ZeigerAufTRNeu, ZeigerAufTRAlt);
```

.removeAttribute() Attribut eines HTML-Elementes mit Wert löschen.

Bsp: var geloeschtes_Attribut_Zeiger = P_Tag_Zeiger.removeAttribute("attribut_bezeichner");

leider ist ein Stringparameter zu übergeben und nicht der Zeiger auf das Attribut.

.removeChild() Kind entfernen

Bsp: zeiger_auf_eltern.removeChild(zeiger_auf_kind);

Beispiel:

```
                // Zeiger auf  BODY holen
                var ZeigerAufBODY = document.getElementsByTagName("body").item(0);

                // Tabelle leer erzeugen mit ID
                var ZeigerAufTabelle = document.createElement("TABLE");
                ZeigerAufTabelle.id = "ID_Table";

                // Tabellenkörper leer erzeugen
                var ZeigerAufTBODY = document.createElement("TBODY");

                // Tabellenkörper zeilenweise füllen (3 Zeilen)
                for (var i = 0; i < 3; i++)
                {
                        // Tabellenzeile leer erzeugen
                        var ZeigerAufTR = document.createElement("TR");

                        // Tabellenzeile zellenweise füllen (3 Zellen)
                         for (var j = 0; j < 3; j++)
                        {
                                // Tabellenzelle leer erzeugen
                                var  ZeigerAufTD = document.createElement("TD");

                                // Tabellenzelle dimensionieren
                                ZeigerAufTD.setAttribute("WIDTH","50");
                                ZeigerAufTD.setAttribute("HEIGHT","50");
```


```
                                        // Tabellenzelle mit Text füllen
                                        var textNode = document.createTextNode("Test");
                                        ZeigerAufTD.appendChild(textNode);

                                        // Tabellenzelle an die Tabellenzeile anhängen als Kind der Zeile
                                        ZeigerAufTR.appendChild(ZeigerAufTD);
                                }

                                // gefüllte Tabellenzeile an den Tabellenkörper anhängen als Kind des Körpers
                                 ZeigerAufTBODY.appendChild(ZeigerAufTR);
                        }

                        // Tabellenkörper an die Tabelle anhängen als Kind der Tabelle
                        ZeigerAufTabelle.appendChild(ZeigerAufTBODY);

                        // Tabelle an BODY des Dokumentes anhängen als Kind des Dokumentes
                        ZeigerAufBODY.appendChild(ZeigerAufTabelle);

                        var ZeigerAufTR = document.getElementByTagName("TR").item(0);
                        ZeigerAufBODY.removeChild (ZeigerAufTRNeu, ZeigerAufTR);
```

.replaceChild() HTML-Element ersetzen

```
                        Bsp:    zeiger_auf_eltern.replaceChild(  zeiger_auf_zu_neues_kind,
                                                                 zeiger_auf_zu_ersetzendes_kind
                                                              );
```

Beispiel:

```
                        // Zeiger auf  BODY holen
                        var ZeigerAufBODY = document.getElementsByTagName("body").item(0);

                        // Tabelle leer erzeugen mit ID
                        var ZeigerAufTabelle = document.createElement("TABLE");
                        ZeigerAufTabelle.id = "ID_Table";

                        // Tabellenkörper leer erzeugen
                        var ZeigerAufTBODY = document.createElement("TBODY");

                        // Tabellenkörper zeilenweise füllen (3 Zeilen)
                        for (var i = 0; i < 3; i++)
                        {
                                // Tabellenzeile leer erzeugen
                                var ZeigerAufTR = document.createElement("TR");

                                // Tabellenzeile zellenweise füllen (3 Zellen)
                                 for (var j = 0; j < 3; j++)
                                 {
                                        // Tabellenzelle leer erzeugen
                                        var  ZeigerAufTD = document.createElement("TD");

                                        // Tabellenzelle dimensionieren
                                        ZeigerAufTD.setAttribute("WIDTH","50");
                                        ZeigerAufTD.setAttribute("HEIGHT","50");

                                        // Tabellenzelle mit Text füllen
                                        var textNode = document.createTextNode("Test");
                                        ZeigerAufTD.appendChild(textNode);

                                        // Tabellenzelle an die Tabellenzeile anhängen als Kind der Zeile
                                        ZeigerAufTR.appendChild(ZeigerAufTD);
                                }

                                // gefüllte Tabellenzeile an den Tabellenkörper anhängen als Kind des Körpers
                                 ZeigerAufTBODY.appendChild(ZeigerAufTR);
                        }

                        // Tabellenkörper an die Tabelle anhängen als Kind der Tabelle
                        ZeigerAufTabelle.appendChild(ZeigerAufTBODY);

                        // Tabelle an BODY des Dokumentes anhängen als Kind des Dokumentes
                        ZeigerAufBODY.appendChild(ZeigerAufTabelle);

                        var ZeigerAufTRNeu = document.createElement("TR");
```


```
var ZeigerAufTRAlt = document.getElementByTagName("TR").item(0);
ZeigerAufBODY.replaceChild(ZeigerAufTRNeu, ZeigerAufTRAlt);
```

4.3.1.3.3.2. HTML-DOM beim Internet Explorer

4.3.1.3.3.2.1. thematisierte Zuordung von Eigenschaften und Methoden des HTML-DOM zu ausgewählten Objekten und zu Anwendungsbereichen im Internet Explorer (Übersicht)

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) 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

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

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

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

4.3.1.3.3.2.2. 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;

4.3.1.3.3.2.3. 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()">
```

4.3.1.3.3.2.4. 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.

4.3.1.3.3.2.4.1. 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>
```

4.3.1.3.3.2.4.2. 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>
```

4.3.1.3.3.2.4.3. 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>
```

4.3.1.3.3.2.4.4. 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

4.3.1.3.3.2.5. 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

4.3.1.3.3.2.6. TextNode Objekt des HTML-DOM im Internet Explorer

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"
```

4.3.2. window Objekt

Objekt eines Browserfenster das im Browser erzeugt wird
das in browserspezifischen Versionen erzeugt werden kann

Das window Objekt hat diverse Zeiger auf andere Objekte:

z.B. beim IE und NS:

.document	Zeiger auf das Objekt document im Fenster
.event	Zeiger auf das Objekt event im Fenster
.history	Zeiger auf das Objekt history (Verlauf)
.location	Zeiger auf das Objekt location
.navigator	Zeiger auf das Objekt navigator

z.B. beim IE:

.screen	Zeiger auf das Objekt screen

Diese Zeiger dienen **nur** der Zuordnung der Objekte zum Fenster. Solange es sich um das aktuelle Fenster handelt, kann also in der Punktnotation die Kodierung von window entfallen.

Das Objekt window beschreibt obige Objekt nicht: Aus Sicht des HTML-DOM sind Objekte, die im DOM hinterlegt sind, keine Kinder des Fensters, denn das ist im DOM nur **durch** das HTML-Dokument präsent.

Hinweis: Wenn ein Fenster mit dem Unterstrichsymbol rechts oben in der Fensterecke minimiert wurde, dann ist es durch Windows in die Hintergrund-Prioritätenfolge eingeordnet worden und arbeitet im Hintergrund bzw. garnicht. Mit anschliessendem Maximieren (Symbol in der Fensterleiste rechts oben) wird das Fenster wieder angezeigt und das geladene Dokument **erneut** geöffnet. Konsequenz daraus ist, dass dieses Maximieren einem Neustart des Dokumentes entspricht, auch wenn der Programmierer diesen Zustand nicht erwünscht. Sound, der z.B. mit Start des Dokumentes erzeugt wird, erklingt erneut mit Maximierung nach einer Fensterminimierung. Analogon ist die Fenstereinengung im Browser durch z.B. Ein- und Ausblenden der Favoritenleiste. Dieses Verhalten des Fensters mit seinem Dokument ist aber **nicht identisch** mit dem Verhalten nach einer Fenstergrößenänderung z.B. durch Rahmenverschiebung (resize). Die Aktionen des Fensters und seines Dokumentes bezüglich Resize müssen programmiert werden, sind also nicht standardmäßig vorhanden - im Gegensatz zum Verhalten mit/nach Fensterminimierung/-maximierung per Symbole in der rechten oberen Ecke der Fensterleiste. Sollte ein Frameset minimiert werden, dann wird das für den gesamten Frameset getan (Framseset hat 1 Fenster für alle Frames gesamt), allerdings mit Maximierung wird auch das Frameset-Dokument neu geladen. Wichtig dabei sind für Zeiger-Bezüge der Frames auf den Frameset per parent-Zeiger, dass die Daten im Frameset-Dokument mit Maximierung initialisiert werden und somit ebenfalls Daten der Frames, die im Framset-Dokument abgelegt wurden, also z.B. Daten, die Verhaltensweisen der Frames vor einer Änderung der Frames-Inhalte gespeichert haben.

Erzeugung:

per Methode .open()