Referenzhandbuch - Deutsch

FUJITSU

WebTransactions V7.5

Template-Sprache

Ausgabe August 2010

Kritik… Anregungen… Korrekturen…

Die Redaktion ist interessiert an Ihren Kommentaren zu diesem Handbuch. Ihre Rückmeldungen helfen uns, die Dokumentation zu optimieren und auf Ihre Wünsche und Bedürfnisse abzustimmen.

Sie können uns Ihre Kommentare per E-Mail an manuals@ts.fujitsu.com senden.

Zertifizierte Dokumentation nach DIN EN ISO 9001:2008

Um eine gleichbleibend hohe Qualität und Anwenderfreundlichkeit zu gewährleisten, wurde diese Dokumentation nach den Vorgaben eines Qualitätsmanagementsystems erstellt, welches die Forderungen der DIN EN ISO 9001:2008 erfüllt.

cognitas. Gesellschaft für Technik-Dokumentation mbH
www.cognitas.de

Copyright und Handelsmarken

Inhalt

1 Einleitung

Bei den meisten IT-Anwendern ist über die Jahre hinweg eine heterogene System- und Anwendungslandschaft entstanden: Mainframes stehen neben Unix- und Windows-Systemen, PCs neben Terminals. Unterschiedliche Hardware, Betriebssysteme, Netze, Datenbanken und Anwendungen werden parallel betrieben. Auf den Mainframe-Systemen und auch auf Unix- oder Windows-Servern existieren oft komplexe und funktional mächtige Anwendungen. Sie sind meist mit erheblichen Investitionen entwickelt worden und stellen in der Regel zentrale Geschäftsprozesse dar, die nicht ohne weiteres durch neue Software ersetzt werden können.

Die Integration vorhandener heterogener Anwendungen in ein einheitliches und transparentes IT-Konzept ist die zentrale Herausforderung der modernen Informationstechnik. Flexibilität, Investitionsschutz und Offenheit für neue Technologien sind dabei von entscheidender Bedeutung.

1.1 Charakterisierung des Produkts

Mit dem Produkt WebTransactions bietet Fujitsu Technology Solutions einen best-of-breed Web-Integration-Server, mit dem eine breite Palette geschäftsrelevanter Anwendungen in kürzester Zeit Browser- und Portal-fähig gemacht werden können. WebTransactions ermöglicht einen schnellen und kostengünstigen Zugang über Standard-PCs und mobile Endgeräte wie Tablett PCs, PDAs (Personal Digital Assistant) und Mobile Phones.

WebTransactions deckt alle Facetten ab, die typischerweise in einem Web-Integrationsprojekt auftreten: von der automatischen Bereitstellung der ursprünglichen „Legacy Oberfläche“ über die grafische Aufbereitung und die Anpassung der Arbeitsabläufe bis hin zu einer umfassenden Frontend-Integration mehrerer Anwendungen. WebTransactions bietet eine hoch-skalierbare Laufzeitumgebung und eine komfortable grafische Entwicklungsumgebung.

Sie können in einer ersten Integrationsstufe folgende Anwendungen und Inhalte über WebTransactions in einer direkten Umsetzung an das WWW anbinden und so Ihren Nutzern intern und extern einfacher zur Verfügung stellen:

– Dialoganwendungen im BS2000/OSD
– MVS- bzw. z/OS-Anwendungen
– systemübergreifende Transaktionsanwendungen auf Basis von openUTM
– dynamische Web-Inhalte

Der Benutzer greift im Internet oder Intranet mit einem Web-Browser seiner Wahl auf die Host-Anwendung zu.

Durch Nutzung modernster Technologie bietet WebTransactions als zweite Integrationsstufe an, die - oftmals noch alphanumerische - Oberfläche der bestehenden Host-Anwendung durch eine attraktive grafische Oberfläche zu ersetzen oder zu ergänzen. Außerdem kann die Host-Anwendung mit WebTransactions auch funktional erweitert werden, ohne dass Eingriffe auf der Host-Seite erforderlich wären (Dialog-Reengineering).

In einer dritten Integrationsstufe können Sie unter der einheitlichen Oberfläche des Browsers unterschiedliche Host-Anwendungen miteinander verknüpfen. Dabei ist es möglich, beliebige vormals heterogene Host-Anwendungen, beispielsweise MVS- oder OSD-Anwendungen miteinander zu verknüpfen oder mit beliebigen dynamischen Web-Inhalten zu kombinieren. Welche Datenquelle ursprünglich die Daten liefert, ist für den Endnutzer nicht mehr sichtbar.

Zusätzlich können Sie den Leistungsumfang und die Funktionalität von WebTransactions-Anwendungen durch eigene Clients beliebig erweitern. Dazu stellt Ihnen WebTransactions ein offenes Protokoll und Schnittstellen (APIs) bereit.

Parallel zum Zugriff über WebTransactions kann weiterhin auch über „herkömmliche“ Terminals oder Clients auf die Host-Anwendungen oder dynamische Web-Inhalte zugegriffen werden. So können Sie eine Host-Anwendung schrittweise ans Web anschließen und die Wünsche und Bedürfnisse unterschiedlicher Nutzergruppen berücksichtigen.

1.2 Dokumentation zu WebTransactions

Zusätzlich zum vorliegenden Handbuch enthält die Dokumentation zu WebTransactions folgende Einheiten:

- Ein einführendes Handbuch, das für alle Liefereinheiten gilt:

  Konzepte und Funktionen

  Das Handbuch beschreibt alle zentralen Konzepte von WebTransactions:

  – die unterschiedlichen Einsatzmöglichkeiten von WebTransactions.

  – das Konzept von WebTransactions und die Bedeutung der Objekte in WebTransactions, ihre wesentlichen Eigenschaften und Methoden, ihr Zusammenspiel und ihre Lebensdauer.

  – den dynamischen Ablauf einer WebTransactions-Anwendung.

  – die Administration von WebTransactions.

  – die Entwicklungsumgebung WebLab.

- Jeweils ein Benutzerhandbuch für jeden Host-Adapter mit speziellen Informationen, zugeschnitten auf den Typ der Partneranwendung:

  Anschluss an openUTM-Anwendungen über UPIC

  Anschluss an OSD-Anwendungen

  Anschluss an MVS-Anwendungen

  Alle Handbücher zu den Host-Adaptern enthalten eine ausführliche Beispielsitzung. Sie beschreiben

  – die Installation von WebTransactions mit dem jeweiligen Host-Adapter.

  – das Einrichten und Starten einer WebTransactions-Anwendung.

  – die Umsetzungs-Templates für die dynamische Umsetzung der Formate auf die Oberfläche eines Web-Browsers.

  – die Bearbeitung von Templates.

  – die Steuerung der Kommunikation zwischen WebTransactions und den Host-Anwendungen über verschiedene Attribute des Systemobjekts.

  – die Behandlung asynchroner Nachrichten und die Druckfunktionen von WebTransactions.

- Ein Benutzerhandbuch, das für alle Liefereinheiten gilt und die Möglichkeiten des HTTP-Host-Adapters beschreibt:

  Zugriff auf dynamische Web-Inhalte

  Das Handbuch beschreibt

  – wie Sie mit WebTransactions auf HTTP-Server zugreifen und deren Ressourcen nutzen.

  – die Einbettung des SOAP-Protokolls (Simple Object Access Protocol) in WebTransactions und den Anschluss von Web-Services über SOAP.

- Ein Benutzerhandbuch, das für alle Liefereinheiten gilt und das offene Protokoll und die Schnittstellen für die Client-Entwicklung für WebTransactions beschreibt:

  Client-APIs für WebTransactions

  Das Handbuch beschreibt

  – das Konzept der Client-Server-Schnittstelle von WebTransactions.

  – die Klasse `WT_RPC` und die Schnittstelle `WT_REMOTE`. Ein Objekt der Klasse `WT_RPC` repräsentiert eine Verbindung zu einer fernen WebTransactions-Anwendung, die auf der Server-Seite über die Schnittstelle `WT_REMOTE` abgewickelt wird.

  – Das Java-Package `com.siemens.webta`, das für die Kommunikation mit WebTransactions ausgeliefert wird.

- Ein Benutzerhandbuch, das für alle Liefereinheiten gilt und das Web-Frontend von WebTransactions beschreibt, das den Zugriff auf allgemeine Web-Services ermöglicht:

  Web-Frontend für Web-Services

  Das Handbuch beschreibt

  – das Konzept des Web-Frontends für objektorientierte Backend-Systeme.

  – die Generierung von Templates für den Anschluss von allgemeinen Web-Services an WebTransactions.

  – den Test und die Weiterentwicklung des Web-Frontends für allgemeine Web-Services.

1.3 Konzept und Zielgruppe dieses Handbuchs

Dieses Handbuch wendet sich an alle, die aktiv WebTransactions-Anwendungen gestalten wollen. Es enthält eine Beschreibung aller Sprachmittel der Template-Sprache WTML. Diese erlaubt es Ihnen, bei der Web-Integration von Host-Anwendungen die Ergebnisse der automatischen Umsetzung individuell anzupassen.

Das Handbuch ist als Referenzhandbuch konzipiert, um schnelles Nachschlagen zu ermöglichen. Es enthält jedoch auch zahlreiche Beispiele, die die beschriebenen Sprachmittel illustrieren.

Für die Lektüre ist es von Vorteil, wenn Sie bereits mit dem Grundprinzip der Auszeichnungssprache HTML vertraut sind. Auch Kenntnisse einer höheren Programmiersprache sind empfehlenswert, werden aber keineswegs vorausgesetzt.

Die Kapitel des Handbuchs lassen sich in vier Blöcke gliedern:

- Einführung
  Das Kapitel 2 stellt die Sprachkonzepte der Template-Sprache zunächst kurz vor.
- Referenz
  Die Kapitel 3 bis Kapitel 12 bieten eine komplette Beschreibung aller Sprachmittel. Kapitel 13 beschreibt die grundlegenden Konzepte der XML-Konvertierung.
- Beispiele
  Das Kapitel 14 enthält Beispiele, die das Zusammenspiel der Sprachmittel illustrieren.
- Überblick und Nachschlagehilfen
  Die Kurzreferenzen in Kapitel 15 sollen Ihnen einen Überblick bieten und schnelles und gezieltes Nachschlagen ermöglichen.

1.4 Neue Funktionen

In diesem Abschnitt werden nur die Neuerungen bezüglich der Template-Sprache genannt. Einen allgemeinen Überblick über die Neuerungen finden Sie im WebTransactions-Handbuch „Konzepte und Funktionen“.

Globale Funktionen: – Neue Funktion `moveFile()` – Neue Funktion `copyFile()` – Neue Funktion `isRequestWaiting()` – Neuer Parameter `all` in `listFolder()`	 Seite 114 Seite 91 Seite 110 Seite 112
Eingebaute Klassen und Methoden: – Neue Methode `String.fromCharCode` – Neue Methode `string.charCodeAt` – Neue Methode `WT_Filter.dataObjectToFormattedXML` – Änderung bei der Ausgabe der Methode `toString()` an Objekten und Arrays: bessere Serialisierung/Deserialisierung	 Seite 208 Seite 206 Seite 234
Exceptions: Neue Attribute `strLine`, `strColumn` und `strText` am Exception-Objekt	Seite 323
C/C++-Userexits: Neues Argument `SendMail`	Seite 387

1.5 Darstellungsmittel

Diese Dokumentation verwendet die folgenden Darstellungsmittel:

`dicktengleiche Schrift`	feste Teile, die genau in dieser Form ein- oder ausgegeben werden, wie z.B. Schlüsselwörter, URLs, Dateinamen
kursive Schrift	variable Teile, für die Sie konkrete Angaben einsetzen müssen
fette Schrift	Zitate, die genauso am Bildschirm oder in der grafischen Oberfläche angezeigt werden, sowie Menübefehle
[ ]	optionale Angaben. Die eckigen Klammern selbst dürfen Sie nicht angeben.
{*alternative1* | *alternative2* }	alternative Angaben. Einen der Ausdrücke innerhalb der geschweiften Klammern müssen Sie auswählen. Die einzelnen Ausdrücke sind durch senkrechte Striche voneinander getrennt. Die geschweiften Klammern und senkrechten Striche selbst dürfen Sie nicht angeben
...	optionale ein oder mehrmalige Wiederholung des vorhergehenden Elements
	wichtige Hinweise und weiterführende Informationen

2 Überblick über die Template-Sprache WTML

Die Template-Sprache WTML (WebTransactions **M**arkup **L**anguage) ermöglicht es Ihnen, WebTransactions-Anwendungen individuell zu gestalten. Dies umfasst nicht nur die optische Gestaltung der Oberfläche, sondern auch die Verarbeitungslogik: Sie können mit WTML den Dialog mit der Host-Anwendung aktiv steuern und auch mehrere Host-Anwendungen unter einer Web-Oberfläche integrieren.

WebTransactions ist also mehr als ein bloßer Kopplungsbaustein, der die Nachrichten zwischen Browser und Host-Anwendung überträgt. Die mächtigen Sprachmittel der Template-Sprache ermöglichen echte Anwendungsintegration: Mit WebTransactions können Sie Ihre IT-Infrastruktur einem Redesign unterziehen, ohne hierfür die Host-Anwendungen ändern zu müssen.

Um Missverständnissen vorzubeugen:
WebTransactions zwingt Sie nicht zum Programmieren. Mit den automatischen Umsetzungs-Tools von WebTransactions können Sie Ihre Host-Anwendungen auch ans Web anschließen, ohne hierfür eine einzige Zeile zu programmieren. Die Umsetzungs-Tools erzeugen nach Standard-Vorgaben aus jeder Einheit der Benutzeroberfläche (Seite/Blatt) ein Template. Nähere Informationen hierzu finden Sie in den Benutzerhandbüchern der verschiedenen Produktvarianten. Diese Standard-Templates können Sie unverändert einsetzen oder als Grundlage für individuelle Anpassungen verwenden.

2.1 WebLab - WebTransactions-Entwicklungsumgebung

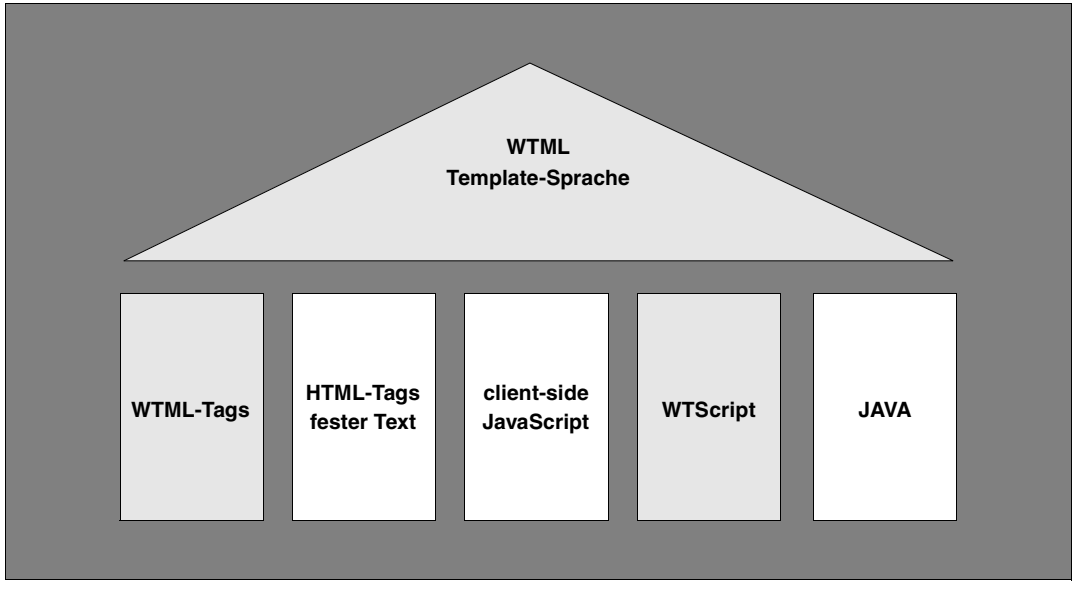
Für die Bearbeitung der Templates können Sie einen beliebigen Text-Editor nutzen. Besonders komfortabel gestaltet sich die Bearbeitung mit der WebTransactions-Entwicklungsumgebung WebLab. Mit WebLab können Sie z.B. WTML-Sprachmittel per Mausklick in ein Template einfügen und Standard-Operationen über Menüs und Dialogboxen definieren. Mit WebLab lassen sich die Templates auch „on the fly“ bei laufender WebTransactions-Anwendung bearbeiten. Die aktuellen Objekte werden grafisch dargestellt und sind direkt modifizierbar.

Grundlegende Informationen zu WebLab finden Sie im WebTransactions-Handbuch „Konzepte und Funktionen“. Eine detaillierte Beschreibung ist in der ausführlichen Online-Hilfe von WebLab enthalten.

2.2 Sprachmittel im Überblick

Folgendes Bild zeigt die Sprachmittel, die Ihnen für die aktive Gestaltung der Templates zur Verfügung stehen. Die unterschiedlichen Farben sollen verdeutlichen, dass HTML und client-seitiges JavaScript kein Bestandteil von WTML sind: Welche HTML- oder JavaScript-Mittel Sie hier verwenden können, hängt nicht von WebTransactions sondern von den eingesetzten Web-Browsern ab. Sie können somit immer die neuesten HTML- und JavaScript-Features in Ihren Templates verwenden. Z.B. können Sie im Microsoft Internet Explorer auch VBScript bzw. JScript verwenden. Alle Mittel zur Gestaltung von Web-Seiten stehen auch für Templates zur Verfügung.

HTML-Tags und client-seitiges JavaScript werden im Weiteren zwar kurz erwähnt, im Handbuch aber nicht im Einzelnen beschrieben.



Standard-HTML-Tags und Text

In Templates können Sie alle HTML-Tags sowie konstanten Text verwenden, die der beim Benutzer eingesetzte Browser interpretieren kann. Diese HTML-Bereiche des Templates (HTML-Tag und Text) werden unverändert an den Browser geschickt.

JavaScript (client-seitig)

Sie können in Templates **alle** HTML-Tags verwenden, also auch `<SCRIPT>`-Tags. Damit stehen Ihnen alle JavaScript-Sprachmittel zur Verfügung, die die eingesetzten Browser interpretieren können. Da solche Scripts wie die Standard-HTML-Tags nicht von WebTransactions sondern vom Browser interpretiert werden, spricht man auch von client-seitigem JavaScript.

Diese client-seitigen JavaScripts werden wie HTML-Tags und fester Text unverändert an den Browser geschickt. Sie gehören aus Sicht von WebTransactions zum HTML-Bereich.

WTML-Tags

Mit WTML-Tags können Sie HTML-Seiten dynamisch generieren bzw. verändern sowie die WebTransactions-Anwendung steuern. Durch sie werden z.B. Werte berechnet oder Informationen von der Host-Anwendung übermittelt.

WTML-Tags sind WebTransactions-spezifische Funktionen im „HTML-Kleid“. D.h. alle Funktionen, die WebTransactions für die Bearbeitung des Templates und der Daten und für die Steuerung der WebTransactions-Anwendung bereitstellt, sind in WTML-Tags eingeschalt. Sie haben die Form: `<wt...>`. Damit orientiert sich die Syntax des Templates am HTML-Standard. Das Template können Sie deshalb nicht nur mit dem WebTransactions-Editor WebLab sondern darüber hinaus mit HTML-Editoren bearbeiten - vorausgesetzt, diese HTML-Editoren versuchen nicht, unbekannte Tags automatisch zu korrigieren. Sie können es auch offline direkt am Browser anzeigen lassen; die WTML-Tags werden in diesem Fall als „unbekannte“ HTML-Tags einfach ignoriert.

Bereiche mit WTML-Tags werden nicht an den Browser geschickt. Sie gehören also nicht zu einem HTML-Bereich.

Die WTML-Tags sind ausführlich im Kapitel „WTML-Tags“ auf Seite 277ff beschrieben. Eine Kurzreferenz finden Sie im Abschnitt „WTML-Tags“ auf Seite 401ff.

WTScripts

WTScripts stehen ähnlich wie client-seitige JavaScripts in Bereichen, die mit speziellen Tags eingeleitet und beendet werden. Statt HTML-`SCRIPT`-Tags verwenden Sie hierfür jedoch WTML-Tags: `WTOnCreateScript` und `WTOnReceiveScript`. Damit zeigen Sie an, dass diese Scripts von WebTransactions und nicht vom Browser ausgeführt werden sollen und signalisieren zusätzlich den gewünschten Ausführungszeitpunkt. OnCreate-Scripts werden ausgeführt, bevor die Seite an den Browser geschickt wird. OnReceive-Scripts werden erst ausgeführt, nachdem die Antwort vom Browser empfangen wurde.

In WTScripts stehen Ihnen eine große Zahl von JavaScript-Sprachmitteln zur Verfügung (die Anweisungen finden Sie im Kapitel „WTScript-Anweisungen (innerhalb von OnCreate-Script/OnReceiveScript)“ auf Seite 293) und zusätzlich spezielle WebTransactions-Klassen und Funktionen - z.B. für den Nachrichtenaustausch mit den Host-Anwendungen (siehe Abschnitte „Host-Datenobjekt-Klasse“ auf Seite 166 und „WT_Communication-Klasse“ auf Seite 226).

Wie WTML-Tag-Bereiche werden WTScript-Bereiche nicht an den Browser geschickt und gehören damit nicht zum HTML-Bereich.

Auswertungsoperatoren

Durch die Auswertungsoperatoren können Sie beliebige Ausdrücke auswerten, z.B. die Werte von Objekten und Attributen. Dabei werden die Auswertungsoperatoren durch aktuelle Werte ersetzt, d.h. sie werden wie eine Variable verwendet.

Auswertungsoperatoren können Sie im Text, in HTML-Tags und in bestimmten WTML-Tags verwenden. Sie haben die Form: `## ... #`. In einem Auswertungsoperator ist jeder beliebige WTScript-Ausdruck möglich: Sie können also den Wert, der ausgegeben werden soll, direkt im Auswertungsoperator in einer beliebig komplexen Operation ermitteln lassen und - wenn Sie wollen - ganz nebenbei gleich einer Variablen zuweisen, die den ermittelten Wert dann im Template global verfügbar macht.

Der Auswertungsoperator ist im Abschnitt „Auswertungsoperator ##...#“ auf Seite 87 und im Kapitel „Klassen-Templates (*.clt)“ auf Seite 329 beschrieben.

2.3 Beispiel: Aufbau eines Templates

Das folgende Beispiel soll Ihnen den Aufbau eines Templates an einem einfachen Beispiel verdeutlichen:

```
<html>                                                              (1)
<head>
<title>TRAVEL Main Menue</title>
</head>
<body>

<wtinclude Name="header">                                           (2)
<form WebTransactions name="myForm">                                (3)
<wtoncreatescript>
WT_HOST.OSD_0.receive();                                            (4)
<</wtoncreatescript>

Datum: ##WT_Host.COMM1.DATE.Value# <br>                             (5)
<input type="button" name="SELBUTTON" value="Book" >
<input type="button" name="SELBUTTON" value="Inquire" >
<input type="button" name="SELBUTTON" value="Cancel" >
<input type="button" name="SELBUTTON" value="Connect" >
...

<wtonreceivescript>                                                 (6)

host=WT_HOST[WT_SYSTEM.HANDLE];                                     (7)
switch (WT_POSTED.SELBUTTON)
{
   case "Book":
      host.SELBUTTON.Value = 1;
      break;
   case "Inquire":
      host.SELBUTTON.Value = 2;
      break;
   case = "Cancel":
      host.SELBUTTON.Value = 3;
      break;
   case "Connect":
      host.SELBUTTON.Value = 4;
}
host.send();

</wtonreceivescript>                                                (8)

...                                                                 (9)

</form>                                                             (10)
</body>
</html>
```

(1) Das Beispiel beginnt mit einem HTML-Bereich, der ungeändert an den Browser weitergereicht wird.

(2) Das erste WTML-Tag ist ein `wtinclude`, das den Inhalt der Datei `header.htm` an dieser Stelle inkludiert. Der Name der Datei ist - abgesehen vom Suffix `.htm` - frei wählbar. Diese Datei kann beliebigen Text und auch HTML- und WTML-Code enthalten. In einer solchen Include-Datei könnten Sie beispielsweise die Definition eines Formularkopfes hinterlegen, um ein einheitliches, später leicht modifizierbares „Look and Feel“ der erzeugten HTML-Seiten zu gewährleisten.

(3) Als Nächstes folgt das WTML-Tag `form WebTransactions`. Es wird intern durch ein HTML-Tag `form` und eine Reihe verdeckter Felder ersetzt. Dieses Tag markiert den Beginn eines Formularbereichs mit Daten, die vom Browser zur WebTransactions-Anwendung zurückgeschickt werden sollen.

(4) Es folgt ein `wtoncreatescript`-Skript, das WebTransactions sofort während der Erzeugung des HTML-Outputs ausführt. In diesem Beispiel wird die Funktion `receive` an einem Kommunikationsobjekt aufgerufen. WebTransactions soll hier also eine Nachricht von der Host-Anwendung einlesen. Die von der Host-Anwendung empfangenen Daten stehen unmittelbar anschießend in Form von Host-Datenobjekten zur Verfügung, die den einzelnen Feldern des umgesetzten Formats entsprechen.

(5) Im darauf folgenden HTML-Bereich wird auch gleich in einem Auswertungsoperator auf das Value-Attribut eines dieser Host-Datenobjekte zugegriffen. Der Auswertungsoperator sorgt dafür, dass der ermittelte Wert an Ort und Stelle in den HTML-Bereich integriert wird. Weiterhin werden in diesem HTML-Bereich eine Reihe von Buttons definiert.

(6) Das nächste Tag ist wieder ein WTML-Tag: `wtonreceivescript` zeigt an, dass ein WTScript-Bereich folgt. Es signalisiert außerdem, dass das folgende WTScript nicht augenblicklich ausgeführt werden soll, sondern erst „onReceive“, also nachdem die HTML-Seite generiert, an den Browser geschickt und die vom Browser gepostete Antwort empfangen wurde.

(7) Da sich WebTransactions die in diesem WTScript definierten Schritte „merkt“ und erst ausführt wenn die Antwort vom Browser vorliegt, können Sie hier bereits die Verarbeitung dieser Antwort definieren: die zurückgelieferten Werte für `SELBUTTON` werden ausgewertet und dem entsprechenden Host-Datenobjekt zugewiesen. Als letzte Aktion werden die Host-Datenobjekte an die Host-Anwendung geschickt.

(8) Das schließende WTML-Tag `/wtonreceivescript` zeigt das Ende des WTScripts an.

(9) Hier könnten beispielsweise noch HTML-Tags definiert werden, um weitere Buttons oder Eingabefelder zu erzeugen. Diese würden dann in der Darstellung am Browser unmittelbar auf die in (5) erzeugten Buttons folgen. Hier in (9) definierte Buttons könnten dann ebenfalls vom WTScript in (7) oder in einem weiteren `onReceive`-Script ausgewertet werden.

(10) Das schließende WTML-Tag `/form` beendet den Formularbereich.

Weitere Informationen zum dynamischen Ablauf der Kommunikation und zur Abarbeitungsreihenfolge der Tags finden Sie im WebTransactions-Handbuch „Konzepte und Funktionen“.

3 Lexikalische Elemente

In diesem Kapitel werden die lexikalischen Elemente der Template-Sprache beschrieben:

Es fehlen noch die Operatoren: Obwohl Operatoren eigentlich auch lexikalische Grundelemente sind, werden sie nicht hier, sondern im Kapitel „Ausdrücke und Operatoren“ auf Seite 69ff beschrieben.

3.1 Zeichensätze

In den WebTransactions-Templates sind alle Zeichen aus 8-Bit-Zeichensätzen zulässig (z.B. ISO-Zeichensätze wie ISO-8859-1 oder Windows-Zeichensätze wie Windows-1250).

Diese Zeichensätze enthalten 256 Zeichen. Davon sind die ersten 128 identisch mit dem klassischen 7-Bit-ASCII-Zeichensatz. Die zweiten 128 Zeichen beinhalten dagegen etliche Sonderzeichen und Buchstaben für bestimmte Sprachen, z.B. "ö", "ß" oder "á".

3.2 WhiteSpace-Zeichen

Unter dem Begriff WhiteSpace-Zeichen werden alle Zeichen zusammengefasst, die aus Sicht von WebTransactions keinerlei Bedeutung haben und bei der Abarbeitung des Templates nicht berücksichtigt werden. WhiteSpaces haben nur die Aufgabe, das Coding des Templates für den menschlichen Leser übersichtlicher zu gestalten. Die Zwischenraum-Zeichen Leerzeichen, Tabulator, Zeilenumbruch und Seitenvorschub fungieren häufig als WhiteSpaces.

Dabei hängt es vom Kontext ab, ob ein Zeichen von WebTransactions als WhiteSpace ignoriert wird: Leerzeichen beispielsweise, mit denen eine Anweisung in einem OnCreate- oder OnReceive-Script eingerückt ist, sind WhiteSpaces und werden ignoriert. Ein Leerzeichen dagegen, das innerhalb einer solchen Anweisung ein Schlüsselwort von einem Bezeichner trennt, ist kein WhiteSpace sondern fungiert als Trennzeichen.

Beispiel

```
function myfunction(param1,param2) {
   var x = 99999;
   vary  = 99999;
   ...
}
```

Erläuterung

In der ersten Zuweisung wird mit dem Schlüsselwort `var` die lokale Variable `x` definiert, in der zweiten Zuweisung dagegen eine globale Variable namens `vary`. Das Leerzeichen zwischen `var` und `x` hat also sehr wohl eine Bedeutung und ist somit kein WhiteSpace. Gleiches gilt für das Leerzeichen zwischen `function` und `myfunction`. Alle anderen Leerzeichen sind WhiteSpaces: Sie dienen der optischen Gliederung und werden von WebTransactions ignoriert.

Dies gilt jedoch nur unter der Voraussetzung, dass diese Funktion innerhalb eines OnCreate- oder OnReceive-Scripts definiert ist. Falls sie innerhalb eines client-seitigen Scripts (zwischen den HTML-Tags `<SCRIPT>` und `</SCRIPT>`) steht, so ist sie Teil eines HTML-Literals: Alle Zeichen innerhalb eines HTML-Literals werden unverändert an den Browser geschickt, es gibt also in diesem Fall keine WhiteSpaces.

3.3 Trennzeichen

Es gibt folgende Trennzeichen:

"bedeutungstragende" Zwischenraumzeichen
(Leerzeichen, Tabulator, Zeilenumbruch, Seitenvorschub)

`()` zur Klammerung von Teilausdrücken.

`{}` zur Bildung von Blöcken.

`,` zur Trennung von Parametern bei Funktionsdefinitionen und Funktionsaufrufen. Beachten Sie, dass das Komma-Zeichen in anderen Kontexten als Operator wirkt (siehe Abschnitt „Komma-Operator (,)“ auf Seite 81).

`;` zum Abschluss von Anweisungen.

3.4 Zeilenabschlusszeichen

Als Zeilenumbruch werden die Zeichen `CR` (Wagenrücklauf), `LF` (Zeilenvorschub) oder die Folge `CR LF` interpretiert. Dabei steht `CR LF` für **einen** Zeilenumbruch. Damit können Templates, die Sie auf Unix-Systemen oder unter Windows erstellt haben, gleichermaßen bearbeitet werden.

Zeilenabschlusszeichen innerhalb von OnCreate- oder OnReceive-Scripts sind WhiteSpaces oder Trennzeichen. Das Anweisungsende erkennt WebTransactions jeweils am Trennzeichen „;“, das in server-seitigen Scripts als Anweisungsabschluss obligatorisch ist (in client-seitigen Scripts kann der abschließende Strichpunkt fehlen, falls die Anweisung in einer eigenen Zeile steht).

3.5 Kommentare

Innerhalb von HTML-Bereichen werden Kommentare durch `wtRem`-Tags gekennzeichnet (siehe Abschnitt „Rem - Kommentare einfügen“ auf Seite 280).

Auch innerhalb von WTML-Tags und in Script-Bereichen können Sie - wie in den HTML-Bereichen - für Kommentare `wtRem`-Tags verwenden. Zusätzlich sind hier mehrzeilige und einzeilige Kommentare im JavaScript-Format möglich. Kommentare dieses Formats können nicht geschachtelt werden:

`//` *Kommentar* (einzeiliger Kommentar)
`/*` *Kommentar*`*/` (ein- oder mehrzeiliger Kommentar)

3.6 Schlüsselwörter

Innerhalb der WTML-Tags gelten die folgenden Schlüsselwörter, die in beliebiger Groß-/Kleinschreibung angegeben werden können.

```
action
case
default
fromAttribute
fromObject
function
lower
name
onSubmit
scope
toAttribute
toObject
type
upper
value
wtArgument
wtDataForm
wtDo
wtDoWhile
wtElse
wtEndIf
wtExit
wtIf
wtInclude
wtOnCreate
wtOnCreateScript
wtOnReceive
wtOnReceiveScript
wtUntil
```

Innerhalb der OnCreate- und OnReceive-Scriptbereiche gelten folgende Schlüsselwörter. Hier wird Groß-/Kleinschreibung unterschieden.

```
break
catch
continue
delete
do
else
finally
for
function
if
in
instanceof
new
prototype
return
this
try
typeofvar
void
while
with
WT_THIS
```

Schlüsselwörter können Sie nicht als Bezeichner verwenden.

Manche andere Ausdrücke wie z.B. `true` und `false` wirken zwar auf den ersten Blick wie Schlüsselwörter, sind aber technisch gesehen bestimmte Werte. Als Bezeichner sind solche speziellen Werte jedoch ebenfalls unzulässig.

3.7 Literale

Literale dienen dazu, in Templates konstante Werte unmittelbar anzugeben („wörtliche“ Wertangabe).

Es gibt Literale für HTML-Bereiche, natürliche Zahlen, Gleitkommazahlen, Zeichenketten (Strings), logische Werte, das Nullobjekt und reguläre Ausdrücke.

3.7.1 Text-Literale

Ein HTML-Bereich besteht jeweils aus einem Text-Literal, das unverändert an den Browser weitergegeben wird. Text-Literale können beliebige HTML-Tags, auszugebenden festen Text und client-seitigen JavaScript-Code enthalten.

Die in HTML-Literalen enthaltenen Zeichen werden ungeändert an den Browser geschickt. Einzige Ausnahme ist ein Gegenschrägstrich am Zeilenende, der das Zeilenende entwertet: `\Zeilenumbruch` wird durch nichts ersetzt. Falls Sie in der HTML-Seite einen Gegenschrägstrich am Zeilenende ausgeben wollen oder Zeichen darstellen wollen, die normalerweise den HTML-Bereich beenden, verwenden Sie folgende HTML-Escape-Sequenzen bzw. oktale oder hexadezimale Eingaben:

Darzustellende Zeichen	**im HTML**	**im client-seitigen JavaScript (innerhalb von Strings)**
\ am Zeilenende	\Zeilenumbruch	`\134` oder `\x5C`
##	##	`\43\43` oder `\x23\x23`
<	< oder <	`\74` oder `\x3C`

Beispiel

```
<p> Der Auswertungsoperator hat folgende Form: &#35;&#35;...&#35; </p>
```

Ausgabe im Browser:
Der Auswertungsoperator hat folgende Form: ##...#

3.7.2 Natürliche Zahlen

Literale für natürliche Zahlen (Integer-Werte) können im dezimalen, oktalen oder hexadezimalen System eingegeben werden. Eine vorangestellte `0` signalisiert oktal, vorangestelltes `0x` oder `0X` signalisiert hexadezimal:

Art des Literals	Format	Maximalwert
Oktale Literale	`0` *octalDigit* ...	`037777777777`
Dezimale Literale	`0 | {` *digit1* `[` *digit* ... `] }`	`4294967295`
Hexadezimale Literale	`0 {x | X}` *hexDigit* ...	`0xffffffff`

octalDigit eines der Zeichen `0-7`.

digit1 eines der Zeichen `1-9`.

digit eines der Zeichen `0-9`.

hexDigit eines der Zeichen `0-9,A-F,a-f`.

3.7.3 Gleitkommazahlen

Literale für Gleitkommazahlen können als Ziffernfolge mit Stellen vor oder nach dem Dezimalpunkt angegeben werden. Eine Exponential-Schreibweise ist ebenfalls möglich.

Ein Literal für eine Gleitkommazahl muss von der folgenden Form sein:

```
{digit ... .[digit...] [{e|E}[+|-]digit ... ]} |
{[digit ...].digit ... [{e|E}[+|-]digit ... ]} |
{digit ... {e|E}[+|-]digit ... }
```

wobei `digit` eines der Zeichen `0-9` ist.

Der Zahlenbereich und die Genauigkeit bei der Umwandlung der Literale in die interne Darstellung ist maschinenabhängig.

Beispiele

```
3.1415
31415.e-4
.031415E2
31415E-4
```

3.7.4 Zeichenketten (String-Literale)

Literale für Zeichenketten werden in einfache ' oder doppelte " Hochkommas eingeschlossen. Dabei müssen sich öffnende und schließende Hochkommas jeweils entsprechen (entweder beide Male ' oder beide Male "). Soll das jeweils als Begrenzungszeichen verwendete Hochkomma innerhalb der Zeichenkette vorkommen, so ist es zu entwerten.

String-Literale haben folgendes Format:

```
{"[{char1 | \"}] ... "} | {'[{char2 | \'}] ... '}
```

wobei *char1* ein Zeichen ungleich " und *char2* ein Zeichen ungleich ' ist.

Escape-Sequenzen in Strings

Durch das Entwertungszeichen Gegenschrägstrich \ wird eine Escape-Sequenz eingeleitet, die als Ersatzdarstellung für ein bestimmtes Zeichen interpretiert wird. Die folgende Tabelle zählt die möglichen Escape-Sequenzen auf:

Escape-Sequenz	**Umsetzung**
\b	Backspace BS
\t	horizontaler Tabulator HT
\n	Zeilenumbruch LF
\f	Seitenvorschub FF
\r	Wagenrücklauf CR
\"	doppeltes Hochkomma "
\'	einfaches Hochkomma '
\Zeilenumbruch	nichts
\\	Gegenschrägstrich \
octalDigit...	Zeichen entsprechend Oktalwert
xhexDigit...	Zeichen entsprechend Hexadezimalwert
char	für jedes andere Zeichen: *char*

Beispiel

```
document.write("the words \n\"apple\"");
document.write(' and "pear"');
```

Ausgabe:

```
the words
"apple" and "pear"
```

3.7.5 Logische Werte

Die möglichen logischen (boolean) Literale sind `true` und `false`.

3.7.6 Literal für ein Array-Objekt

Arrays können bei Zuweisungen oder als Argumente von Funktionsaufrufen folgendermaßen als Literale angegeben werden:

```
[element1, element2, ...]
```

Beispiele

```
x=[1,2,3];          // x ist jetzt ein Array mit drei Elementen 1, 2 und 3
c = f([1,2]);       // f wird mit einem Array mit den Elementen 1 und 2
                    // aufgerufen
```

3.7.7 Literal für ein Object-Objekt

Objekte können bei Zuweisungen oder als Argumente von Funktionsaufrufen folgendermaßen als Literale angegeben werden:

```
{attribut1: wert1, attribut2: wert2, ... }
```

Beispiele

```
x={y:1, z:23};          // x ist jetzt ein Objekt mit den Attributen y und z,
                        // die wiederum die Werte 1 bzw. 23 haben
c = f({Name: "Inge Neumann", Abteilung: "Entwicklung"});
                        // f wird mit einem Argument aufgerufen, dem Objekt
                        // mit den Attributen Name und Abteilung
```

3.7.8 Literal für das null-Objekt

Das Literal `null` verweist auf das `null`-Objekt. Das `null`-Objekt ist ein spezielles Objekt, das „kein Objekt“ referenziert. Einige Methoden, z.B. die String-Methode `match`, liefern das `null`-Objekt, wenn das Suchmuster nicht gefunden wird. Andere Methoden liefern das `null`-Objekt, wenn Fehler aufgetreten sind, z.B. die Kommunikationsobjekt-Methoden `send` und `receive`.

Das Literal `null` können Sie beispielsweise in Abfragen verwenden, um zu prüfen, ob ein bestimmtes Objekt existiert oder nicht.

Beispiel

```
<wtOnCreateScript>
<!--
  host = WT_HOST[WT_SYSTEM.HANDLE];
  if (host.WT_SYSTEM != null )
    host_system = host.WT_SYSTEM;
  else
    host_system = WT_SYSTEM;
//-->
</wtOnCreateScript>
```

Die if-Anweisung dieses Scripts fragt ab, ob es ein verbindungsspezifisches Systemobjekt gibt. Gleichbedeutend wäre die Abfrage `if(host.WT_SYSTEM)`. Durch die Verwendung des `null`-Literals ist die Bedeutung der if-Anweisung für den menschlichen Leser jedoch einfacher zu erkennen.

3.7.9 Literale für reguläre Ausdrücke

Literale für reguläre Ausdrücke werden in Schrägstriche `/` eingeschlossen. Innerhalb der Schrägstriche muss der Schrägstrich und der Gegenschrägstrich durch einen Gegenschrägstrich entwertet werden. Literale für reguläre Ausdrücke dürfen nicht mit einem Stern beginnen und nicht leer sein, da sie sonst als Beginn eines Kommentars interpretiert würden.

Zwischen den Schrägstrichen können Sie jeden reguläre Ausdruck wie in der folgenden Tabelle beschrieben angeben. Der abschließende Schrägstrich kann von einem `i` und/oder `g` gefolgt sein, was Nichtbeachtung der Groß-/Kleinschreibung bzw. globale Suche bedeutet.

Beispiel

/*pattern*/`ig`

Reguläre Ausdrücke werden wie in JavaScript bzw. Perl aufgebaut. Umfang und Bedeutung der Metazeichen für diese regulären Ausdrücke sind in der folgenden Tabelle zusammengefasst.

Wenn Sie reguläre Ausdrücke als String-Literale verarbeiten wollen, müssen Sie die folgenden Metazeichen jeweils einzeln mit einem Gegenschrägstrich entwerten:

Beispiel

```
str = "Amsterdam | Brüssel | Chemnitz | Dortmund";
document.write(str.split("\\\|"));
```

Ausgabe am Browser:

```
[Amsterdam,Brüssel,Chemnitz,Dortmund]
```

Metazeichen	Bedeutung
`\`	Entwertung des folgenden Metazeichens: Das Zeichen `*` beispielsweise ist ein Metazeichen und bedeutet: null oder mehr Wiederholungen des vorhergehenden Zeichens (z.B. bedeutet `/a*/` kein `a`, ein einzelnes `a` oder eine Folge von mehreren `a`'s). Um nach dem Zeichen `*` selbst zu suchen, muss ein Gegenstrich vorangestellt werden (z.B. passt `/b\*a/` zur Zeichenkette `b*a`) . Beachten Sie, dass bei manchen Metazeichen der Gegenschrägstrich bereits Bestandteil des Metazeichens ist, z.B.bei `\n` für Zeilenvorschub. Wenn Sie nicht nach einem Zeilenvorschub sondern nach der Zeichenfolge `\n` suchen wollen, müssen Sie auch hier einen Gegenschrägstrich voran stellen: `\\n`

Metazeichen	Bedeutung
^	Anfang der durchsuchten Zeichenkette oder Anfang der Zeile. Der reguläre Ausdruck /^a/ passt zu dem a in der Zeichenkette "another b", nicht jedoch zu dem a in der Zeichenkette "Another a".
$	Ende der durchsuchten Zeichenkette oder Ende der Zeile. Der reguläre Ausdruck /d$/ beispielsweise passt zum d in "head", nicht jedoch zum d in "header".
.	jedes Zeichen außer Zeilenumbruch. /.T/ beispielsweise passt nur zu CT in "WEB\nTRANSACTIONS", nicht jedoch zu \nT.
[*chars*]	eines der Zeichen aus *chars*. Sie können einen Bindestrich verwenden, um einen Zeichenbereich festzulegen. /[abcdefg]/ bedeutet das Gleiche wie /[a-g]/ und passt zum g in "spring".
[^*chars*]	keines der Zeichen aus *chars*. Sie können einen Bindestrich verwenden, um einen Zeichenbereich festzulegen. Ein solcher regulärer Ausdruck passt zu allen Zeichen, die **nicht** angegeben sind. /[^abcdefg]/ beispielsweise bedeutet das Gleiche wie /[^a-g]/ und passt in "spring" zu allen Zeichen außer g. /[^1-5]/ passt in "x246" zu x und 6.
|	Trennung mehrerer Alternativen. /Dampf|Diesel/ beispielsweise passt zu Dampf in "Dampfschiff" und zu Diesel in "Dieselschiff".
( )	Gruppierung von Suchmuster-Teilen und Abspeichern der entsprechenden Teile der Fundstelle: Auf die Teile der Fundstelle, die den geklammerten Suchmuster-Teilen entsprechen, können Sie über die Elemente des Ergebnis-Arrays oder über die Eigenschaften $1,...$9 des vordefinierten Objekts RegExp zugreifen. /(Dampf)((schiff)(fahrt))/ beispielsweise passt im String "Dampfschifffahrtsgesellschaft" zu Dampfschifffahrt. Die Teile der Fundstellen, die den vier geklammerten Teilausdrücken entsprechen, sind danach in den Attributen des RegExp-Objekts abgespeichert: $1: Dampf $2: schifffahrt $3: schiff $4: fahrt

Metazeichen	Bedeutung
`\`*n*	Ist eine Referenz auf das *n*-te geklammerte Suchmusterteil, wobei *n* eine positive ganze Zahl ist. In `/(Dampf)((schiff)(fahrt)).*\3/` beispielsweise steht `\3` für den dritten geklammerten Suchmuster-Teil (`schiff`). Das Suchmuster passt somit im String `"Dampfschifffahrt oder Dieselschiffahrt"` zu `"Dampfschifffahrt oder Dieselschiff"`.
`*`	davorstehender Ausdruck beliebig oft (null mal, einmal oder mehrmals). `/ba*/` beispielsweise passt zum `ba` in `"bang!"`, zu `baaaa` in `"baaaang!"` und auch zu `b` in `"bong!"`.
`+`	davorstehender Ausdruck mindestens einmal (einmal oder mehrmals). `/ba+/` beispielsweise passt zu `ba` in `"bang!"` und zu `baaaa` in `"baaaang!"` aber **nicht** zum `b` in `"bong!"`.
`?`	davorstehender Ausdruck null oder einmal. `/ba?/` beispielsweise passt zu `ba` in `"bang!"`, zu `ba` in `"baaaag!"` und zu `b` in `"bong!"`.
`{`*n*`}`	davorstehender Ausdruck *n* mal (wobei *n* eine positive ganze Zahl ist). `/ba{3}/` beispielsweise passt zu `baaa` in `"baaaaag!"` hat aber weder eine Entsprechung in `"bang!"`, noch in `"bong!"`.
`{`*n*`,}`	davorstehender Ausdruck mindestens *n* mal (wobei *n* eine positive ganze Zahl ist). `/ba{3,}/` beispielsweise passt zu `baaaaa` in `"baaaaag!"` hat aber keine Entsprechung in `"baang!"`.
`{`*n*`,`*m*`}`	davorstehender Ausdruck *n* bis *m* mal (wobei *n* und *m* positive ganze Zahlen sind). `/ba{2,3}/` beispielsweise passt zu `baaa` in `"baaaaag!"` und zu `baa` in `"baang"`, hat aber keine Entsprechung in `"bang!"`.
`\t`	Tabulator.
`\n`	Zeilenumbruch (newline).
`\r`	Wagenrücklauf (carriage return).
`\f`	Seitenvorschub (form feed).
`\v`	vertikaler Tabulator.
`\o`*octalDigit...*	Oktalzahl (wobei *octalDigit* eine Ziffer zwischen 0 und 7 ist). Diese Angabe ermöglicht es Ihnen, oktale Escape-Sequenzen für ASCII-Zeichen in reguläre Literale einzubetten.

Metazeichen	Bedeutung
\x*hexDigit*...	Hexadezimalzahl (wobei *hexDigit* eines der Zeichen 0-9,A-F oder a-f ist). Diese Angabe ermöglicht es Ihnen, hexadezimale Escape-Sequenzen für ASCII-Zeichen in reguläre Literale einzubetten.
\c*A*	wobei *A* ein Controlzeichen angibt, nach dem gesucht werden soll. \cJ\ passt beispielsweise zu einem Zeilenvorschub (Linefeed), \cM\ passt zu einem Carriage Return.
\b	Wortgrenze. /.ns\b/ beispielsweise passt im String "WebTransactions" zu ons, aber nicht zu ans.
\B	keine Wortgrenze. /.ns\B/ beispielsweise passt im String "WebTransactions" zu ans, aber nicht zu ons.
\w	eines der Zeichen a-z A-Z 0-9 oder Unterstrich (_). \w bedeutet somit das Gleiche wie [A-Za-z0-9_]. /\w+/ beispielsweise passt in "!x my_Array 77" zu x, zu my_Array und zu 77.
\W	keines der Zeichen a-z A-Z 0-9 oder Unterstrich (_). \W bedeutet somit das Gleiche wie [^A-Za-z0-9_]. /\W+/ beispielsweise passt in "!x my_Array 77" zu ! und zu den beiden Leerzeichen (vor myArray und vor 77).
\s	einzelnes WhiteSpace-Zeichen (Leerzeichen, horizontaler/vertikaler Tabulator, Wagenrücklauf, Seiten- oder Zeilenvorschub). \s bedeutet somit das Gleiche wie [\t\v\r\n\f]. /\s\w+/ beispielsweise passt in "tool bar" zu " bar".
\S	beliebiges einzelnes Zeichen außer WhiteSpace. \S bedeutet somit das Gleiche wie [^ \t\v\r\n\f]. /\S\w+/ beispielsweise passt in "tool bar" zu "tool" und zu "bar".
\d	eine Ziffer (eines der Zeichen 0-9). \d bedeutet das Gleiche wie [0-9]. /\d/ passt beispielsweise in "57a" zu 5 und 7.
\D	keine Ziffer (keines der Zeichen 0-9). \D bedeutet das Gleiche wie [^0-9]. /\D/ passt beispielsweise in "57a" zu a.

3.8 Bezeichner

Bezeichner sind die Grundelemente, aus denen sich Namen zusammensetzen.

Bei einfachen Namen entspricht ein Bezeichner einem Namen (z.B. „x“ in x=9). Bei qualifizierten Namen besteht der Name aus mehreren Bezeichnern, wobei entweder der Punkt-Operator (z.B. `myarray.length`) oder der Indexoperator verwendet wird (z.B. `myarray[3]`, `myarray["length"]`). Da solche qualifizierten Namen keine lexikalischen Grundelemente sind, sondern bereits Kombinationen solcher Elemente, werden sie nicht in diesem Kapitel behandelt sondern in Abschnitt „Aufbau von Namen“ auf Seite 59.

Aufbau von Bezeichnern

In Bezeichnern sind folgende Zeichen zulässig:

`A-Z,a-z (`Groß- und Kleinbuchstaben)

`0-9` (Ziffern)

`_` (Unterstrich)

`$` (Dollarzeichen)

Das erste Zeichen eines Bezeichners darf keine Ziffer sein und der Bezeichner darf keinem Schlüsselwort entsprechen. Bezeichner können beliebig lang sein.

Groß-/Kleinschreibung

Groß- und Kleinschreibung wird bei Bezeichnern grundsätzlich unterschieden. Eine Ausnahme bilden die Bezeichner für die vordefinierten Objekte `WT_SYSTEM`, `WT_HOST` und `WT_POSTED`, die Sie beliebig in Groß-und Kleinschreibung angeben können.

`ResultArray` und `resultarray` bezeichnen also unterschiedliche Variablen, während `Wt_System`, `wt_system` und `WT_SYSTEM` ein und dasselbe Objekt bezeichnen.

Beispiele

zulässige Bezeichner: `ResultArray, my_array, $input1, _hits`

4 Datentypen, Variablen und Namen

Variablen sind benannte Speicherbereiche, in denen Sie Daten, die Sie im Template benötigen, speichern können. Der Inhalt, der in einer Variablen gespeichert ist, wird als „Wert“ bezeichnet.

Neben ihrem Wert hat jede Variable auch einen bestimmten Typ. Das Typ-Konzept der Template-Sprache entspricht dem von JavaScript (siehe Abschnitt „Datentypen“ auf Seite 46).

Genau wie in JavaScript haben Sie in der Template-Sprache beim Umgang mit Variablen große Freiheiten, die viele andere Programmiersprachen nicht bieten:

- Sie müssen Variablen nicht eigens deklarieren.
- Die Template-Sprache ist „loose typing“: Auch wenn Sie eine Variable explizit mit einer var-Anweisung deklarieren, legen Sie dabei keinen Typ fest. Der Typ der Variable ergibt sich dynamisch durch den Typ des zugewiesenen Werts.

  Sie können also nicht nur den Wert sondern auch den Typ einer Variablen jederzeit ändern:

  ```
  a=10;   // typeof a is number
  a="10"; // typeof a is string
  ```

- Für natürliche Zahlen (Integer) und Gleitkommazahlen benötigen Sie keine unterschiedlichen Variablen-Typen. Der Typ `number` ist für beide Arten geeignet.
- Datentypen werden automatisch konvertiert, wenn die Ausführung dies erfordert (siehe Abschnitt „Typkonvertierung“ auf Seite 49). Der Inhalt von numerischen Variablen kann z.B. ohne explizite Konvertierung in auszugebende Strings integriert werden (siehe auch Beispiele im Abschnitt „String-Verknüpfungsoperator (+)“ auf Seite 79):

  ```
  a=4+2;
  document.write("Das Ergebnis lautet: " + a);
  ```

4.1 Datentypen

Jede Variable hat - wie auch jedes Literal und jeder konstante Wert - einen bestimmten Datentyp. Das Typ-Konzept der Template-Sprache entspricht dem von JavaScript.

Es gibt folgende Datentypen: `undefined`, `number`, `boolean`, `string`, `object` und `function`. Diese werden in den Abschnitten „number“ auf Seite 47 bis „function“ auf Seite 48 vorgestellt. Der Abschnitt „Stringähnliche Datentypen“ auf Seite 49 erklärt den Begriff „stringähnlich“ und im Abschnitt „Typkonvertierung“ auf Seite 49 werden die Regeln genannt, die WebTransactions bei der automatischen Typ-Konvertierung verwendet.

Einfache Datentypen und Referenz-Datentypen

Die Datentypen `number`, `boolean` und `undefined` werden häufig als einfache Datentypen bezeichnet, während die Datentypen `string`, `object` und `function` unter dem Begriff Referenz-Datentypen zusammengefasst werden. Dies hat folgenden Hintergrund:

Bei einer Variablen eines einfachen Typs ist der Wert der Variable „einfach“ an der durch den Variablennamen symbolisch bezeichneten Speicherstelle abgelegt: Wenn Sie z.B. fünf Variablen mit dem Wert 1000 definieren, dann ist dieser Wert an fünf verschiedenen Stellen gespeichert.

Bei einer Variablen eines Referenz-Datentyps dagegen ist der interne Wert die Adresse des „eigentlichen“ Wertes, oder - anders ausgedrückt - eine Referenz: Wenn Sie z.B. für eine Funktion fünf verschiedene Namen definieren, so ist diese Funktion trotzdem nur einmal im Speicher. Sie können diese Funktion dann mit jedem der fünf Namen ansprechen (referenzieren).

Beispiel: die unterschiedlichen Datentypen

```
document.writeln (typeof a); // Ausgabe: undefined (noch keine Zuweisung)
a=4.118;
document.writeln (typeof a); // Ausgabe: number

a="Peter";
document.writeln (typeof a); // Ausgabe: string

a=false;
document.writeln (typeof a); // Ausgabe: boolean

a=new Array();
document.writeln (typeof a); // Ausgabe: object

function myfunction() {      // Definition einer Funktion
   return 10 }
a=myfunction;
document.writeln (typeof a);  // Ausgabe: function
```

```
document.writeln (a());       // Ausgabe: 10 (Aufruf der Funktion über den
                              //              neuen Namen a)
```

4.1.1 number

Der Datentyp `number` umfasst natürliche Zahlen und Gleitkommazahlen. Die Arithmetik für Gleitkommazahlen folgt wie bei JavaScript dem Standard IEEE 754-1985 (IEEE, New York).

Die darstellbaren Maschinenzahlen liegen betragsmäßig zwischen 4.94065645841247e-324 und 1.79769313486232e+308.

Daneben gibt es einige spezielle numerische Werte:

`NaN` (not a number)
: Dieser Wert wird bei nicht definierten arithmetischen Operationen als Ergebnis geliefert, z.B. wenn Sie versuchen zwei Strings wie „Peter“ und „Maria“ zu multiplizieren.
`NaN` steht außerhalb der linearen Ordnung: Die Vergleichsoperatoren `==`, `<`, `<=`, `>` und `>=` liefern `false`, wenn einer oder beide der Operanden `NaN` ist, `!=` liefert `true`, wenn einer oder beide Operanden `NaN` sind.

`-Infinity` (minus unendlich)
: Dieser Wert wird geliefert, wenn eine Zahl kleiner als die kleinste darstellbare negative Zahl ist.

`Infinity` (plus unendlich)
: Dieser Wert wird geliefert, wenn eine Zahl größer als die größte darstellbare Zahl ist.

4.1.2 boolean

Der Datentyp boolean hat die möglichen logischen Werte `true` und `false`.
Auf Operanden vom Typ `boolean` können Sie die boolschen Operatoren anwenden (siehe Abschnitt „Boolsche Operatoren (&&, ||, !)“ auf Seite 76)

4.1.3 undefined

Jede Variable, der noch kein Wert zugewiesen wurde, hat den Typ `undefined` und den Wert `undefined`.

4.1.4 string

Ein String ist eine Folge von ASCII-Zeichen. Jeder String hat ein vordefiniertes Längenattribut (*string*`.length`), das die Anzahl der Zeichen angibt. Für Objekte der Klasse `String` gibt es eine Reihe vordefinierter Methoden (siehe Abschnitt „String-Klasse“ auf Seite 204), die Dank Typkonvertierung auch für den Typ `string` gelten.

Operanden vom Typ `string` können Sie mit dem Operator + verknüpfen (siehe Abschnitt „String-Verknüpfungsoperator (+)“ auf Seite 79).

4.1.5 object

Eine Variable vom Typ `object` ist ein Behälter für benannte Attribute, es handelt sich somit um ein assoziatives Array. Der Wert eines solchen Objekts ist die Referenz auf einen solchen Behälter oder die `null` Referenz. Wenn Sie beispielsweise einer Variablen ein Objekt zuweisen, haben Sie anschließend einen neuen Namen für dieses Objekt und nicht zwei Objekte.

Die Attribute können beliebigen Typs sein. Ihre Namen können beliebige Bezeichner oder ganze Zahlen sein; im letzteren Fall heißt das Attribut auch Index (siehe Abschnitt „Aufbau von Namen“ auf Seite 59).

Es gibt ein spezielles Objekt, das `null`-Objekt:
Das `null`-Objekt referenziert „kein Objekt“. Einige Methoden, z.B. die String-Methode `match`, liefern das `null`-Objekt, wenn das Suchmuster nicht gefunden wird. Andere Methoden liefern das `null`-Objekt, wenn Fehler aufgetreten sind, z.B. die Kommunikationsobjekt-Methoden `send` und `receive`.

Zum Erzeugen von Objekten steht Ihnen der `new`-Operator zur Verfügung, zum Löschen von Objekten der `delete`-Operator (siehe Abschnitt „new-Operator“ auf Seite 82).

4.1.6 function

Eine Funktion wird durch eine `function`-Anweisung oder als Objekt der `Function` Klasse definiert (siehe Abschnitt „function-Anweisung“ auf Seite 317). Diese Definition legt ein Funktionsobjekt mit dem Namen der definierten Funktion an.

4.1.7 Stringähnliche Datentypen

Bei einigen Operationen, z.B. bei Vergleichsoperationen, ist es von Bedeutung, wie „ähnlich“ ein Datentyp einem String ist. Als stringähnlich gelten die Datentypen `string` und `function`, sowie alle Objekte von Klassen, bei denen es eine Methode valueOf gibt, die einen String liefert (im Augenblick sind das Objekte der Klasse `String` und es können Objekte selbstdefinierter Klassen sein).

Zum Begriff „Stringähnlichkeit“ siehe auch: „Beispiel: arithmetische Addition vs. Stringverknüpfung“ auf Seite 79.

4.1.8 Typkonvertierung

Wie bereits am Anfang dieses Kapitels erwähnt haben Sie in der Template-Sprache beim Umgang mit Datentypen einige Freiheiten. Wenn der Datentyp eines Ausdrucks nicht in den Verwendungskontext passt, konvertiert WebTransactions den Ausdruck - wo immer dies möglich ist - auf den benötigten Datentyp. Wenn Sie z.B. versuchen, zwei Strings zu multiplizieren, dann werden diese Strings auf den Typ `number` konvertiert. Falls es sich um Zahlen-Strings handelt, wird wie gewohnt multipliziert, wenn es sich nicht um Zahlenstrings handelt, liefert die Multiplikation das Ergebnis `NaN` (Not a Number), was auf eine ungültige numerische Operation hinweist.

Folgende Tabelle zeigt die Regeln, die WebTransactions für die Typkonvertierung verwendet:

Ausgangs-Datentyp	**Ziel-Datentyp**				
	function	**object**	**number**	**boolean**	**string**
undefined	Fehler	null	"NaN"	false	"undefined"
function	-	Fehler	Fehler	true	Header[1]
object					
(not null)	Fehler	-	valueOf/ "NaN"	valueOf/ true	toString/ valueOf
(null)	Fehler	-	0	false	"null"
number					
(zero)	Fehler	Number	-	false	"0"
(nonzero)	Fehler	Number	-	true	Ziffernstring
(NaN)	Fehler	Number	-	false	"NaN"
(Infinity)	Fehler	Number	-	true	"+Infinity"
(-Infinity)	Fehler	Number	-	true	"-Infinity"
boolean					
(false)	Fehler	Boolean	0	-	"false"
(true)	Fehler	Boolean	1	-	"true"
string					
(empty)	Fehler	String	0	false	-
(non-empty)	Fehler	String	numerischer Wert / "NaN"	true	-
selbst definiert	Bei selbstdefinierten Klassen bestimmt die Basisklasse den Typ. Es gilt deshalb die entsprechende Zeile weiter oben.				

[1] Es wird nur der Header der Funktion umgewandelt und nicht - wie bei JavaScript - der gesamte Programmtext.

Erläuterung zur Tabelle

In den einzelnen Kästchen der Tabelle ist jeweils das Ergebnis angegeben, das WebTransactions beim Versuch liefert, den in der linken Spalte angegebenen Ausgangstyp in den oben angegebenen Zieltyp umzuwandeln. Wenn in einem Kästchen zwei Möglichkeiten - getrennt durch einen Schrägstrich - genannt sind, so bedeutet dies, dass WebTransactions zunächst die erste Möglichkeit versucht und - falls diese scheitert - die zweite.

Im Einzelnen bedeuten:

`undefined, function, object, number boolean, string`
: Variable oder Werte des entsprechenden Datentyps

`Number, Boolean, String`
: Variable oder Werte vom Typ object der entsprechenden Klasse

`toString`
: Ergebnis der `toString`-Methode.

`valueOf`
: Ergebnis der `valueOf`-Methode, falls diese ein Ergebnis vom Ziel-Datentyp liefert.

4.2 Lokale und globale Variablen

Alle Variablen, die Sie außerhalb einer Funktion deklarieren, sind global - gleichgültig ob Sie das Schlüsselwort `var` verwenden oder nicht. Eine Variable, die Sie innerhalb einer Funktion deklarieren, ist nur dann lokal, wenn Sie hierfür das Schlüsselwort `var` verwenden, ansonsten ist sie automatisch global.

Eine globale Variable gilt überall im WTScript-Code.

Eine lokale Variable gilt innerhalb der gesamten Funktion, in der sie deklariert ist, unabhängig davon, wo in der Funktion sie definiert wurde.

Wenn Sie innerhalb einer Funktion eine Variable mit `var` deklarieren und bereits eine globale Variable (oder bei geschachtelten Funktionen eine lokale Variable in der äußeren Funktion) mit gleichem Namen existiert, wird innerhalb der Funktion die lokale Variable der Funktion verwendet, die lokale Variable überdeckt die globale. Außerhalb der Funktion wird die globale Variable angesprochen.

Beispiel 1

```
<wtoncreatescript>
<!--
var scope = "global";                       // globale Variable
function test_scope()
{
var scope = "local";                        // gleichnamige lokale Variable
document.write("Scope=" + scope + "<br>"); // lokale Variable wird benutzt
}
test_scope();                               // Gibt "Scope=local" aus
//-->
</wtoncreatescript>
```

Wenn der Kontext, in dem eine Funktion verwendet wird, nicht vollständig bekannt ist, empfiehlt es sich, immer das Schlüsselwort `var` zu verwenden. So können Sie ausschließen, dass globale Variablen gegebenenfalls überschrieben werden. Das folgende Beispiel zeigt, was passiert, wenn Sie das Schlüsselwort `var` nicht verwenden.

Beispiel 2

```
<wtoncreatescript>
<!--
scope = "global";                          // globale Variable
function test_scope()
{
   scope = "local";                           // globale Varibale wird
                                              // überschrieben !!!
   document.write("Scope=" + scope + "<br>");// globale Variable wird benutzt
   newScope = "local";                        // Deklariert eine weitere
                                              // globale Variable
}
test_scope() ;                                 // Gibt "Scope=local" aus
document.write("Scope=" + scope + "<br>");     // Gibt "Scope=local" aus
document.write("Scope=" + newScope + "<br>"); // Gibt "Scope=local" aus

//-->
</wtoncreatescript>
```

Sie können Funktionsaufrufe schachteln: Da jede Funktion ihren eigenen lokalen Gültigkeitsbereich hat, kann es mehrere geschachtelte lokale Gültigkeitsbereiche geben. Wenn die gerufene Funktion innerhalb der rufenden Funktion definiert ist, hat die gerufene Funktion Zugriff auf die globalen und lokalen Variablen (und ebenso auf die Argumente) der rufenden Funktion. Wenn die Funktionen unabhängig voneinander definiert sind, bleiben die lokalen Variablen der rufenden Funktion der aufgerufenen Funktion verborgen.

Beispiel 3

```
<wtoncreatescript>
<!--
scope = "global";                                    // globale Variable
function test_scope()
{
   var scope = "local";                              // gleichnamige lokale
                                                     // Variable
   function nested()
   {
       var scope="nested";                           // lokal in geschachtelter
                                                     // Funktion
       document.write("Scope=" + scope + "<br>");    // Gibt "Scope=nested" aus
   }
   nested();
   document.write("Scope=" + scope + "<br>");        // Gibt "Scope=local" aus

}
test_scope();
document.write("Scope=" + scope + "<br>");           // Gibt "Scope=global" aus

//-->
</wtoncreatescript>
```

4.3 Lebensdauer von Variablen

Bei WebTransactions lassen sich wie bei JavaScript vier Regeln für die Lebensdauer von Variablen aufstellen:

- **globale Variable:**
  Globale Variable, die in Scriptbereichen des Templates angelegt werden, leben vom Moment des Anlegens bis das letzte `OnReceiveScript`-Script dieses Templates abgearbeitet ist oder bis sie explizit durch den `delete`-Operator zerstört werden.
- **Objektattribute**:
  Attribute leben so lange wie das enthaltende Objekt oder bis sie explizit zerstört werden.
- **Aktualparameter**:
  Die aktuellen Parameter eines Funktionsaufrufs leben vom Aufruf der Funktion bis zum Ende der Funktion, oder bis sie explizit gelöscht werden.
- **Lokale Variable einer Funktion**:
  Alle Variablen, die innerhalb einer Funktion mit dem Schlüsselwort `var` definiert werden, sind lokale Variablen dieser Funktion. Sie leben vom Moment des Anlegens bis zum Ende der Funktion, oder bis sie explizit gelöscht werden.

Lebensdauer vordefinierter Objekte

Mit WebTransactions stehen Ihnen neben selbst definierten Variablen auch einige vordefinierte Objekte zur Verfügung, für die Sonderregeln gelten:

- **Systemobjekt**:
  Das Systemobjekt wird von WebTransactions als globales Objekt vom Typ `object` mit dem Namen `WT_SYSTEM` angelegt und lebt für die Dauer einer Sitzung. Dieses Objekt hat einige Attribute, die für die Steuerung von WebTransactions eine Bedeutung haben.
- **Postedobjekt**:
  Das Postedobjekt wird von WebTransactions als globales Objekt vom Typ `object` mit dem Namen `WT_POSTED` angelegt und lebt für die Dauer einer Sitzung. Dieses Objekt hat als Attribute die jeweils zuletzt vom Browser übermittelten Daten.
- **Host-Wurzelobjekt**:
  Das vordefinierte Objekt `WT_HOST` ist ein Behälter für alle Kommunikationsobjekte. Es lebt für die Dauer einer Sitzung.

- **Kommunikationsobjekte**:
  Ein Kommunikationsobjekt wird durch den Konstruktor-Aufruf `WT_Communication` als Attribut von `WT_HOST` angelegt. Kommunikationsobjekte leben bis zum Ende der Sitzung.

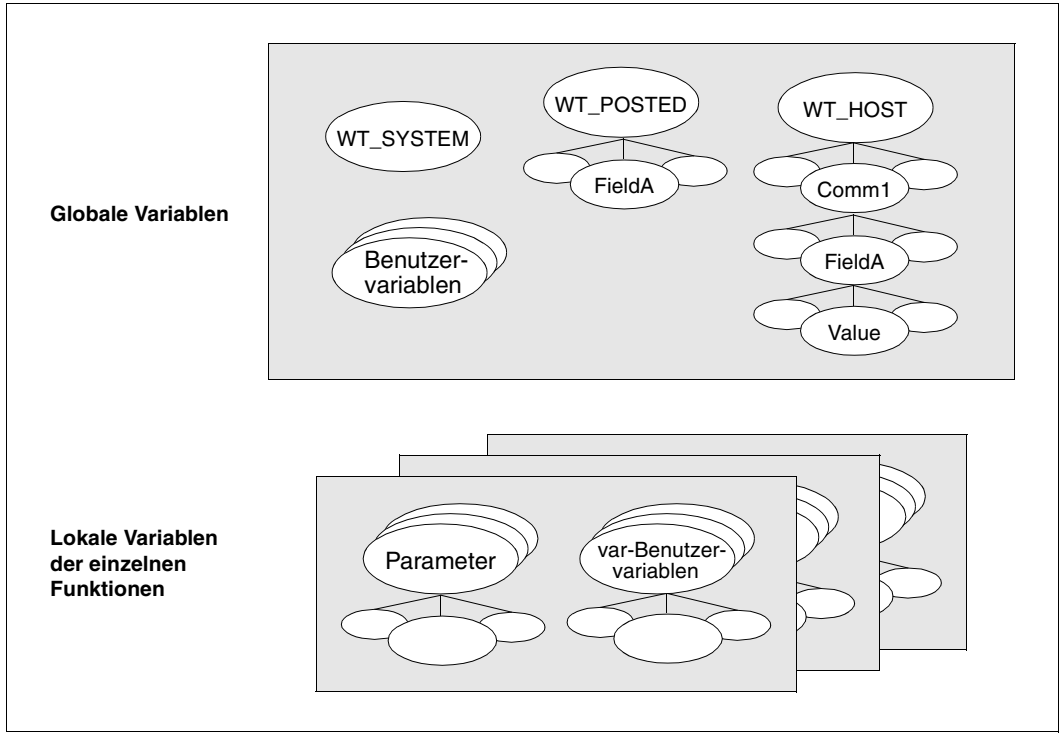
  Kommunikationsobjekte können also mehrere Dialogschritte überleben. Kommunikationsobjekte ermöglichen Ihnen das Handling von parallelen Verbindungen und somit die Integration von mehreren Host-Anwendungen innerhalb einer WebTransactions-Anwendung.

- **Host-Datenobjekte:**
  Host-Datenobjekte stehen für die Teile (z.B. Felder) der eigentlichen Nutznachricht, die WebTransactions mit dem Host austauscht. Sie werden durch die Methode `receive` angelegt. Dabei werden ältere Hostobjekte zerstört.

Ausführliche Informationen zu diesen vordefinierten Objekten finden Sie im WebTransactions-Handbuch „Konzepte und Funktionen“.

Übersicht: Namensräume von Variablen

Die folgende Grafik veranschaulicht die Namensräume: Die vordefinierten Objekte befinden sich zusammen mit den globalen Variablen, die im Script angelegt werden, in einem Raum globaler Variablen. Für jede aktuell aufgerufene Funktion gibt es einen Raum lokaler Variablen, in dem sich neben den mit dem Schlüsselwort `var` deklarierten lokalen Benutzervariablen auch die Aktualparameter befinden.



4.4 Initialisierung

Bei WebTransactions hat jede Variable auch einen Wert. Einige Attribute des Systemobjekts werden von WebTransactions beim Start initialisiert. Kommunikations- und Host-Datenobjekte werden durch die Kommunikationsmodule initialisiert. Selbst angelegte Variablen können Sie durch eine Zuweisung initialisieren. Eine Variable, der noch kein Wert zugewiesen wurde, hat den Typ `undefined` und den Wert `undefined`. Ein Parameter eines Funktionsaufrufs wird mit einem aktuellen Wert aus dem Funktionsaufruf versorgt. Enthält der Aufruf kein entsprechendes Argument, hat der Aktualparameter den Typ und den Wert `undefined`.

4.5 Aufbau von Namen

Namen bezeichnen Variablen und deren Unterstrukturen sowie Funktionen. Es gibt einfache und zusammengesetzte Namen. Ein einfacher Name besteht aus einem einzelnen Bezeichner. Ein zusammengesetzter Name besteht aus einer Folge von Bezeichnern, die durch Punktoperatoren oder Indexoperatoren abgetrennt sind.

Punktoperator

Da Bezeichner nicht mit einer Ziffer beginnen dürfen, sind nach dem Punktoperator keine Indexangaben möglich.

---

bezeichner.bezeichner

Beispiele

```
myarray.length
WT_HOST.KOMM1.Command.Value
```

Indexoperator

Der Indexoperator `[ ]` erlaubt den Zugriff auf alle Attribute eines Objekts.

---

bezeichner`[`*expression*`]`

Steht in den eckigen Klammern ein Ausdruck, der eine ganze Zahl zurückliefert, so wird auf den entsprechenden Index verwiesen:

bezeichner`[`*index*`]`

Steht in den eckigen Klammern ein Ausdruck, der einen String zurückliefert, so wird auf das entsprechende Attribut verwiesen:

bezeichner1`["`*bezeichner2*`"]` und *bezeichner1*`.`*bezeichner2* sind also gleichbedeutend.

Beispiel

```
for (i=0 ; i < myarray["length"] ; i++) document.writeln(myarray[i]);
```

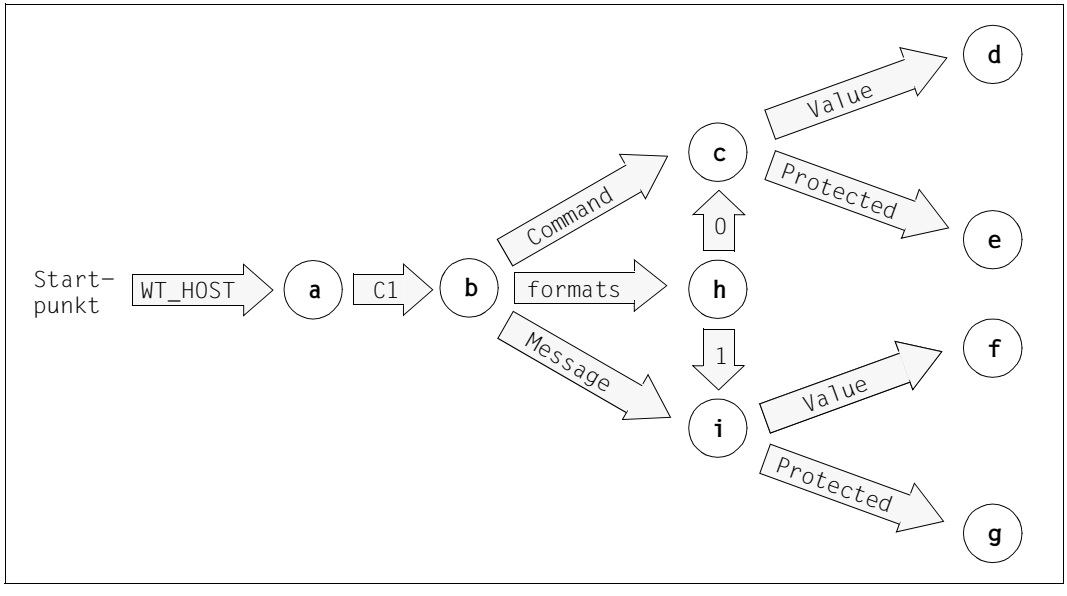
Diese for-Schleife gibt alle Elemente des Arrays `myarray` aus.

4.5.1 Vollqualifizierte Angabe

Bei vollqualifizierter Angabe werden alle Bestandteile des Namens angegeben. Durch einen vollqualifiziert angegebenen Namen wird das bezeichnete Objekt unabhängig vom Kontext eindeutig identifiziert. Ähnlich wie bei vollqualifizierten Dateinamen in einem hierarchisch aufgebauten Dateiverzeichnis wird jedes Element durch eine „Pfadangabe“ identifiziert, die an der „Wurzel“ beginnt.

Beispiel

Sie können sich beispielsweise einen Bezeichner als Wegweiser vorstellen, der auf das bezeichnete Element zeigt. Dann erhalten Sie den vollqualifizierten Namen eines Elements, indem Sie alle Wegweiser des Pfades, über den das Element erreichbar ist, mittels Punkt oder Index-Operator aneinanderfügen.



Im dargestellten Fall ergeben sich z.B. folgende vollqualifizierte Namen:

```
Für a:  WT_HOST

Für b:  WT_HOST.C1
        WT_HOST["C1"]

Für c:  WT_HOST.C1.Command
        WT_HOST.C1["Command"]
        WT_HOST.C1.formats[0]

Für d: WT_HOST.C1.Command.Value
       WT_HOST.C1["Command"]["Value"]
       WT_HOST.C1.formats[0].Value

Für g: WT_HOST.C1.formats[1].Protected
       WT_HOST.C1["formats"][1]["Protected
       "]
```

4.5.2 Relative Angabe

Namen können Sie nicht nur voll qualifiziert sondern auch unvollständig angeben.

Dabei wird der Pfad nicht von der Wurzel beginnend angegeben, sondern von einem anderen Ausgangspunkt aus. Die Angabe bezeichnet das Objekt somit relativ zu diesem Ausgangspunkt.

Im auf der vorhergehenden Seiten dargestellten Fall beispielsweise bezeichnet `Command.Value` das Objekt `d` relativ zu `b`.
`Value` ist relativ zu `c` ein Name für `d` und relativ zu `i` ein Name für `f`.

4.5.3 Zuordnung: Name - bezeichnetes Objekt

WebTransactions verfährt nach folgenden Regeln, um dem angegeben Namen ein Objekt zuzuordnen:

1. Steht ein Name innerhalb eines `with` Statements (siehe Abschnitt „with-Anweisung“ auf Seite 321), so wird für jedes einschließende `with` Statement von innen nach außen überprüft, ob der Name relativ zum jeweiligen Objekt ein existierendes Objekt bezeichnet. Wenn dieses der Fall ist, ist die Suche erfolgreich abgeschlossen.
2. Steht der Name innerhalb einer Funktion, so wird überprüft, ob der Name relativ zum Funktionsobjekt ein Objekt bezeichnet (lokale Variable oder Parameter der Funktion). Wenn dieses der Fall ist, ist die Suche erfolgreich abgeschlossen.
3. Falls die Schritte 1. und 2. zu keinem Ergebnis führen, wird der Name nun als voll qualifizierter Name aufgefasst und überprüft, ob ein entsprechendes globales Objekt existiert. Wenn dieses der Fall ist, ist die Suche erfolgreich abgeschlossen.
4. Abschließend wird überprüft, ob der Name relativ zum vordefinierten Kommunikationsobjekt `WT_HOST`.*handle* ein Objekt bezeichnet (*handle* ist hierbei der Inhalt des Attributs `HANDLE` des globalen Systemobjekts). Wenn dieses der Fall ist, ist die Suche erfolgreich abgeschlossen.

Wird nach diesen Regeln kein Objekt gefunden, so ist der Name ungültig:

– Bei einem lesenden Zugriff wird `undefined` zurückgeliefert.

– Beim schreibenden Zugriff wird bei einteiligen Namen (wie z.B. `x` oder `result`) implizit ein Objekt dieses Namens definiert.

  Bei mehrteiligen Namen (wie z.B. `result.value`) wird zunächst geprüft, ob der um den letzten Namensteil reduzierte Name ein existierendes Objekt vom Typ `object` bezeichnet. Wenn ja, wird wie bei einteiligen Namen ein neues Objekt erzeugt - wenn nein, wird ein Fehler gemeldet.

Beispiel

```
document.writeln("type of  x: " + typeof(x) );   //Ausgabe: undefined
x.colour = "red";   //Fehler, weil x nicht als Objekt definiert ist

x=new Object("car");
x.colour="red";
document.writeln("now there's a car and its colour is " + x.colour);
```

4.6 Selbstdefinierte Klassen

Für die Anlage eigener Klassen zur einfachen Definition von ähnlichen Objekten in WebTransactions stehen Ihnen entsprechende Sprachmittel zur Verfügung, mit deren Hilfe Sie neue Klassen, Attribute und Methoden definieren können.

Die Definition einer eigenen Klasse erfolgt in zwei Schritten:

1. Beschreibung der Klasse und der Attribute
2. Definition der Methoden

Beschreibung der Klasse und der Attribute

Um eine neue Klasse zu definieren, muss der Konstruktor für Objekte dieser Klasse als Script-Funktion definiert werden, z.B.:

```
// Konstruktor fuer Klasse "Employee":
function Employee() {
   // Definition der Klassen-Attribute:
   this.name      = "";
   this.division  = "development";
   this.machine   = "computer";
   this.worktime  = 35;

}
```

Mit dieser Definition wurde bis jetzt ein Konstruktor für Objekte des selbstdefinierten Typs `Employee` definiert. Der Name des Konstruktors ist zugleich der Name der Klasse. Über das Schlüsselwort `this` werden die Attribute dieser Klasse festgelegt und mit voreingestellten Werten versorgt (im Beispiel: `name`, `division`, `machine` und `worktime`).

Objekte dieser Klasse können jetzt erzeugt werden, wobei die Attribute mit den voreingestellten Werten des Konstruktors versorgt werden. Diese Voreinstellung kann überschrieben werden:

```
// Liefert ein Objekt der Klasse Employee mit den Attributen
// name = "Manuella Mueller", division = "development",
// machine = "computer" und worktime = 30
angest = new Employee();
angest.name = "Manuela Mueller";
angest.worktime = 30;
```

Zusätzlich können jederzeit weitere Attribute für einzelne Instanzen der neuen Klasse definiert werden, wenn dies für die WebTransactions-Anwendung notwendig sein sollte. Beachten Sie bitte, dass solche Attribute nur für die jeweilige Instanz existieren.

```
angest.homework = true;
```

Definition der Methoden

Um für eine selbstdefinierte Klasse eine Methode zu definieren, müssen Sie zunächst eine Script-Funktion für diese Methode erstellen und dann im Konstruktor der Klasse eine Referenz auf diese Funktion anlegen:

```
// Methode fuer Employee:
function gibName() {
   return (this.name);
}

// Konstruktor fuer Klasse "Employee":
function Employee() {
   // Definition der Klassen-Attribute:
   this.name      = "";
   this.division  = "development";
   this.machine   = "computer";
   this.worktime  = 35;

   // Referenz auf Methode:
   this.gibName   = gibName;
}
```

Die neue Methode (im Beispiel `gibName`) kann jetzt wie gewohnt verwendet werden:

```
j = angest.gibName();
```

Solche selbstdefinerten Datentypen können auch aus definierten Klassen abgeleitet werden, wodurch eine Objekthierarchie entsteht. Bei der Ableitung von Objekten aus Klassen werden Eigenschaften (Attribute und Methoden) vererbt. Einzelheiten dazu finden Sie im folgenden Abschnitt.

4.7 Objekthierarchie und Vererbung

In WebTransactions können neue Klassen aus vorhandenen Klassen oder Objekten abgeleitet werden, wobei die Attribute und Methoden für die Objekte der neuen Klasse aus denen der ursprünglichen Klasse geerbt werden. Dieser Abschnitt beschreibt schrittweise das Vorgehen bei der Ableitung von Klassen und erklärt die dabei zu beachtenden Besonderheiten. Das Beispiel aus dem vorangegangenen Abschnitt wird auch für die folgenden Erklärungen verwendet:

```
// Methode fuer Employee:
function gibName() {
   return (this.name);
}

// Konstruktor fuer Klasse "Employee":
function Employee() {
   // Definition der Klassen-Attribute:
   this.name      = "";
   this.division  = "development";
   this.machine   = "computer";
   this.worktime  = 35;

   // Referenz auf Methode:
   this.gibName   = gibName;
}
```

Diese Klasse definiert einen Angestellten eines Unternehmens mit den grundsätzlichen Eigenschaften jedes Angestellten im Unternehmen. Für verschiedene Angestellten-Typen sind jetzt möglicherweise unterschiedliche weitere Eigenschaften notwendig, die sich von Typ zu Typ unterscheiden. So kann ein Vertriebsmitarbeiter für verschiedene PLZ-Bereiche, ein Ingenieur für verschiedene Projekte zuständig sein. Die grundlegenden Eigenschaften sind aber für alle gleich, sodass es hier sinnvoll ist, diese abzuleiten.

Um z.B. einen Angestellten-Typ `SalesManager` zu definieren, kann man folgendermaßen vorgehen:

```
// Konstruktor für Klasse "SalesManager"
function SalesManager() {
   // Definition der zusaetzlichen Klassen-Attribute:
   this.area = 8;     // PLZ-Bereich, Voreinstellung 8
   this.quota = 100;  // Umsatzvorgabe
}
// Ableitung der uebrigen Attribute aus "Employee"
SalesManager.prototype = new Employee();
```

Das Schlüsselwort `prototype` legt eine Referenz auf ein neues Objekt der Klasse `Employee` an, wodurch in neuen Instanzen der Klasse `SalesManager` alle Klassenattribute aus `Employee` ebenfalls (als Referenz) zur Verfügung stehen. Mit dem Schlüsselwort `prototype` nimmt man Bezug auf ein echtes Objekt. Es ist nicht notwendig, hier mit `new` ein neues Objekt einer vorhandenen Klasse zu definieren, man kann auch auf ein existierendes Objekt verweisen. Das folgende Beispiel zeigt noch einmal die Ableitung der Klasse `SalesMananger`, diesmal aus einem Objekt:

```
proto = new Object;
proto.name      = "";
proto.division  = "development";
proto.machine   = "computer";
proto.worktime  = 35;

// Jetzt folgt die Ableitung der neuen Klasse "SalesManager" aus dem
// Objekt "proto"
function SalesManager() {
   // Definition der zusaetzlichen Klassen-Attribute:
   this.area = 8;     // PLZ-Bereich, Voreinstellung 8
   this.quota = 100;  // Umsatzvorgabe
}
// Ableitung der uebrigen Attribute aus dem Objekt "proto"
SalesManager.prototype = proto;
```

Beachten Sie, dass die abgeleiteten Attribute aus dem Prototyp-Objekt zunächst nur als Referenz angelegt werden. Erst bei einer Wertzuweisung eines solchen abgeleiteten Attributs in einer Instanz wird ein Instanz-Attribut mit dem neuen Wert angelegt:

```
manager1 = new SalesManager();

manager1.worktime = 60;          // Instanz-Attribut
document.write("Abteilung   = " + manager1.division +"<BR>");
document.write("Arbeitszeit = " + manager1.worktime +"<BR>");
proto.division = "marketing";    // Aenderung im Prototyp
proto.worktime = 30;             // Aenderung im Prototyp
document.write("Abteilung   = " + manager1.division +"<BR>");
document.write("Arbeitszeit = " + manager1.worktime +"<BR>");
```

Dies liefert die folgende Ausgabe:

```
Abteilung   = development
Arbeitszeit = 60
Abteilung   = marketing
Arbeitszeit = 60
```

Wird ein Attribut eines abgeleiteten Objekts explizit gelöscht (mit dem Operator `delete`), dann wird es im abgeleiteten Objekt nur als gelöscht gekennzeichnet. Im Prototyp-Objekt bleibt es erhalten.

Neue Klassen können nicht nur aus selbstdefinierten Klassen sondern auch aus vordefinierten Klassen abgeleitet werden. Das folgende Beispiel soll dies verdeutlichen:

```
// Ableitung einer neuen Klasse "NamedArray" aus der Klasse "Array":
function NamedArray (n) {
   this.name = n;
}
NamedArray.prototype = new Array;

MyArray = new NamedArray("first");
```

Die neue Klasse `NamedArray` besitzt alle Methoden der Klasse `Array` und das zusätzliche Attribut `name`. `MyArray` ist jetzt ein `Array` der Länge 0 mit dem Namen `first`.

Analog wie im Beispiel `SalesManager` weiter oben könnte diese Klasse auch direkt aus einem Objekt der Klasse `Array` abgeleitet werden:

```
// Ableitung einer neuen Klasse "NamedArray" aus einem "Array"-Objekt:
a = new Array;
for (i=0; i<= 10; i++)
   a[i] = 0;

function NamedArray (n) {
   this.name = n;
}
NamedArray.prototype = a;

MyArray = new NamedArray("second");
```

Im Gegensatz zur ersten Version ist hier `MyArray` ein `Array` der Länge 11 und mit dem Namen `second`, dessen Elemente mit 0 vorbelegt sind.

5 Ausdrücke und Operatoren

In diesem Kapitel erhalten Sie zunächst einen Überblick über die Ausdrücke der Template-Sprache. In den anschließenden Abschnitten „Arithmetische Operatoren“ auf Seite 71 bis „Spezielle Operatoren“ auf Seite 80 werden die einzelnen Operatoren vorgestellt. Die Auswertungsreihenfolge der Operatoren ist im Anschluss an die Operatoren in Abschnitt „Auswertungsreihenfolge“ auf Seite 90 beschrieben.

Ausdrücke sind Kombinationen aus Literalen, Variablen, Operatoren und Ausdrücken, deren Auswertung jeweils ein bestimmtes Ergebnis liefert.

Bei WebTransactions-Ausdrücken sind folgende Ergebnisse möglich:

– ein Wert
  `43+7` liefert z.B. einen Wert vom Typ `number`

– eine Referenz auf ein Objekt
  Beispielsweise liefert ein Ausdruck, der einen Konstruktor aufruft, eine Referenz auf ein Objekt: `myArray=new Array()`

– undefined
  Ein Ausdruck ist `undefined`, wenn eine nicht initialisierte Variable, ein Aufruf einer Funktion ohne Rückgabewert oder der `void` Operator verwendet wird. `undefined` ist eigentlich auch ein Wert - *alle* Ausdrücke liefern ja ein Ergebnis.

5.1 Verschiedene Arten von Ausdrücken

Von WebTransactions werden alle bei JavaScript möglichen Ausdrücke unterstützt. Diese Ausdrücke können in den WTScript-Bereichen und innerhalb des Auswertungsoperators auftreten.

- Es gibt Ausdrücke, die einer Variable einen Wert zuweisen, und Ausdrücke, die einfach nur einen Wert haben: Der Ausdruck `x=4+5` z.B. weist der Variable x den Wert des Ausdrucks `4+5` zu und repräsentiert selbst diesen Wert. Solche Ausdrücke verwenden Zuweisungsoperatoren. Ein Ausdruck wie `4+5` dagegen enthält keine Zuweisung sondern ergibt einfach nur den Wert 9.
- Es gibt elementare Ausdrücke, die lexikalischen Einheiten entsprechen (wie z.B. eine Variable wie x oder ein Literal wie 42 oder „hello world“), und komplexe Ausdrücke, die sich aus elementaren Ausdrücken zusammensetzen. Dabei gelten die in Abschnitt „Auswertungsreihenfolge“ auf Seite 90 genannten Auswertungsregeln. Sie können aber auch durch Klammerung `()` eine bestimmte Auswertungsreihenfolge erzwingen.
- Oft wird auch je nach der Anzahl der Operanden, die durch einen Operator verbunden werden, zwischen einstelligen und zweistelligen Ausdrücken unterschieden. Mit dem Bedingungsoperator „?:“ (siehe Abschnitt „Bedingungsoperator (?:)“ auf Seite 80) lassen sich sogar dreistellige Ausdrücke bilden.
- Häufig werden auch Ausdrücke mit verwandten Operatoren unter einem Begriff zusammengefasst, z.B. wird zwischen arithmetischen Ausdrücken und Vergleichsausdrücken unterschieden.

Erweiterungen gegenüber JavaScript-Ausdrücken

Darüber hinaus gibt es folgende Erweiterungen:

– Die Bedingungen der WTML-Tags `<wtIf ...>`, `<wtDoWhile ...>` und `<wtUntil...>` können aus Kompatibilitätsgründen zusätzlich die Vergleichsoperatoren `#==`, `#!=`, `#>`, `#<`, `#>=`, `#<=` enthalten.

– Strings innerhalb von WTML-Tags können aus Kompatibilitätsgründen neben festen Zeichen (String-Literalen) auch Auswertungsoperatoren enthalten.

5.2 Arithmetische Operatoren

Arithmetische Operatoren werden auf numerische Werte angewendet und liefern als Ergebnis einen einzelnen numerischen Wert.

Operator	Bedeutung	Beispiel
+	Addition. Addiert zwei `number`-Operanden. Falls einer der Operanden vom Typ string ist oder stringähnlich, wirkt das +-Zeichen dagegen als Verknüpfungsoperator (siehe Seite 79 und Beispiel unten)	`4+x`

++	Inkrement.
	Dieser einstellige Operator erhöht den Wert seines Operanden um 1. Der Operand muss eine Variable sein, deren Wert vom Typ `number` oder auf diesen Typ konvertierbar ist. Der erhöhte Wert wird dem Operanden zugewiesen. Wird der Operator **vor** dem Operanden platziert, wird der erhöhte Wert zurückgeliefert. Wird der Operator dagegen **nach**gestellt, wird der ursprüngliche Wert zurückgeliefert und erst danach erhöht.
		`x++`
		(ergibt 3, falls x ursprünglich gleich 3; neuer Wert von x ist 4)
		`++x`
		(ergibt 4, falls x ursprünglich gleich 3; neuer Wert von x ist 4)

–	Subtraktion oder einstelliges Minus: – Steht das Minus-Zeichen zwischen zwei Operanden, wird der zweite Operand vom ersten subtrahiert. – Als einstelliger Minus-Operator steht das Minus-Zeichen vor dem Operanden. Der Operand wird negiert, d.h. das Vorzeichen wird umgekehrt.	zweistellig: `y-4` einstellig: `-x` (-x ergibt -3, falls x gleich 3; x behält den Wert 3)

--	Dekrement.
	Dieser einstellige Operator vermindert den Wert seines Operanden um 1. Der Operand muss eine Variable sein, deren Wert vom Typ `number` oder auf diesen Typ konvertierbar ist. Der verminderte Wert wird dem Operanden zugewiesen. Wird der Operator **vor** dem Operanden platziert, wird der verminderte Wert zurückgeliefert. Wird der Operator dagegen **nach**gestellt, wird der ursprüngliche Wert zurückgeliefert und erst danach vermindert.
		`x--`
		(ergibt 3, falls x ursprünglich gleich 3; neuer Wert von x ist 2)
		`--x`
		(ergibt 2, falls x ursprünglich gleich 3; neuer Wert von x ist 2)

*	Multiplikation. Multipliziert die beiden Operanden.	`4*x`
/	Division. Dividiert die beiden Operanden.	`17/4` (ergibt 4.25)
%	Modulus. Liefert den ganzzahligen Rest der Division der beiden Operanden zurück.	`17%4` (ergibt 1)

Die Operatoren `*`, `/` und `%` liefern das Ergebnis immer in der maximal möglichen Genauigkeit, auch die Verknüpfung von ganzen Zahlen wird häufig eine Gleitkommazahl als Ergebnis haben (`17/4` ergibt z.B. 4.25).

Ist einer der Operatoren einer arithmetischen Operation `NaN` (Not a Number), so ist das Ergebnis immer `NaN`.

5.3 Vergleichsoperatoren

Ein Vergleichsoperator vergleicht seine Operanden und ergibt einen logischen Wert: `true` wenn der Vergleich zutrifft, ansonsten `false`.

Operator	Bedeutung	Beispiel
==	gleich; ergibt `true`, wenn die Operanden gleich sind	`3 == 3`
!=	ungleich; ergibt `true`, wenn die Operanden ungleich sind	`3 != 4`
>	größer als; ergibt `true`, wenn der linke Operand größer als der rechte Operand ist	`4 > 3`
>=	größer oder gleich; ergibt `true`, wenn der linke Operand größer oder gleich dem rechten Operanden ist	`4 >= 4`
<	kleiner als; ergibt `true`, wenn der linke Operand kleiner als der rechte Operand ist	`3 < 4`
<=	kleiner als oder gleich; ergibt `true`, wenn der linke Operand kleiner oder gleich dem rechten Operanden ist	`3 <= 4`

Der numerische Wert `NaN` liegt außerhalb der Ordnung. Ein Vergleich, bei dem einer der Operanden `NaN` ist, liefert immer `false`, also auch bei `NaN==NaN`.

Auswertung der relationalen Vergleichsoperatoren (>, >=, <, >=)

Sind beide Operanden stringähnlich (siehe Abschnitt „Stringähnliche Datentypen“ auf Seite 49), so werden beide Operanden in Strings konvertiert und das Ergebnis des lexikographischen Vergleichs dieser beiden Operanden zurückgeliefert. Ist einer der Operanden undefiniert oder das `null` Objekt, so ist das Ergebnis `false`. Anderenfalls werden beide Operanden auf den Typ `number` konvertiert und das Ergebnis des numerischen Vergleichs dieser beiden Operanden zurückgeliefert.

Auswertung der Gleichheits-Vergleichsoperatoren (==, !=)

Sind beide Operanden vom Typ `object` oder `function`, wird verglichen, ob beide Operanden dasselbe Objekt referenzieren. Ist einer der Operanden das `null` Objekt, so wird der andere Operand auf den Typ `object` konvertiert und ein Vergleich durchgeführt.

Ist ein Operand ein String und der andere stringähnlich (siehe Abschnitt „Stringähnliche Datentypen“ auf Seite 49), so werden beide Operanden auf den Typ `string` konvertiert und das Ergebnis des Vergleichs zurückgeliefert.

In allen anderen Fällen werden die beiden Operanden auf den Typ `number` konvertiert und numerisch verglichen.

Vergleichsoperatoren, die numerischen Vergleich erzwingen (nur in WTML-Tags)

Die Operatoren `#==`, `#!=`, `#>`, `#<`, `#>=`, `#<=` konvertieren die Operanden auf den numerischen Datentyp und liefern das Ergebnis des entsprechenden numerischen Vergleichs. Die Operatoren `#==`, `#!=`, `#>`, `#<`, `#>=`, `#<=` werden wegen der Kompatibilität zu WebTransactions V1.x unterstützt. Sie sind **nur** in den Bedingungen der WTML-Tags `<wtIf ...>`, `<wtDoWhile ...>` und `<wtUntil ...>` zugelassen und an allen anderer Stelle im Template **nicht** zugelassen.

Beispiele

```
"7" > "10";  //liefert true
"7" > 10;    //liefert false
"7" #> "10"  //liefert false (nur zulässig in WTML-Tag-Bedingungen)
```

5.4 Bitweise Operatoren

Bitweise Operatoren behandeln ihre Operanden als Folge von Bits (Nullen und Einsen). Die Dezimalzahl 9 beispielsweise wird durch die Bitfolge 1001 repräsentiert.

Bitweise Operatoren transformieren zwar Bitfolgen, geben aber das Ergebnis jeweils als normalen numerischen Wert zurück.

Es gibt bitweise logische Operatoren und bitweise Shift-Operatoren. Die Operanden müssen jeweils auf den Typ `number` konvertierbar sein.

5.4.1 Bitweise logische Operatoren (&, |, ^, ~)

Bitweise logische Operatoren arbeiten folgendermaßen:

– Die Operanden werden in 32-bit-Zahlen umgewandelt.

– Die Bits der beiden Operanden werden jeweils paarweise betrachtet: das erste Bit des linken Operanden korrespondiert mit dem ersten Bit des rechten Operanden, das zweite Bit mit dem zweiten Bit usw.

– Der Operand wird auf jedes dieser Bit-Paare angewandt und das Ergebnis wird bitweise aus den einzelnen Teilergebnissen zusammengesetzt.

Eine Sonderstellung nimmt der bitweise NOT-Operator (~) ein, der als Einziger einstellig ist, also nur einen Operanden hat. Dieser Operator kehrt die Bits des Operanden um - aus 0 wird 1 und aus 1 wird 0.

Folgende Tabelle zeigt die Funktionsweise der bitweisen logischen Operatoren:

Operator	Beschreibung	Beispiel
`&`	bitweises AND Gibt für alle die Bit-Paare 1 zurück, bei denen beide Bits 1 sind.	`15 & 9` ergibt `9` `(1111 & 1001 = 1001)`
`|`	bitweises OR (inklusiv). Gibt für alle die Bit-Paare 1 zurück, bei denen mindestens eines der Bits 1 ist.	`15 | 9` ergibt `15` `(1111 | 1001 = 1111)`
`^`	bitweises XOR (exklusiv). Gibt für alle die Bit-Paare 1 zurück, bei denen genau eines der Bits 1 ist.	`15 ^ 9` ergibt `6` `(1111 & 1001 = 0110)`
`~`	bitweises NOT (Komplement) Kehrt jedes Bit des Operanden um.	`~15` ergibt `-16` `(~00...001111 = 11...110000)`

5.4.2 Bitweise Shift-Operatoren (<<, >>, >>>)

Der linke Operand gibt jeweils den Ausgangswert an, der rechte die Anzahl der zu verschiebenden Stellen. Shift-Operatoren wandeln ihre Operanden in 32-bit-Zahlen um und geben einen Wert vom Typ `number` zurück.

Operator	Beschreibung	Beispiel
<<	links shift. Die Binär-Darstellung des ersten Operanden wird um die im zweiten Operanden angegebene Anzahl von Stellen nach links verschoben. Die über die linke Grenze hinaus verschobenen Bits fallen weg, von rechts werden Bits mit dem Wert 0 ergänzt.	`9 << 2` ergibt `36` (`1001` um zwei Bits nach links: `100100`)
>>	rechts shift mit Berücksichtigung des Vorzeichens. Die Binär-Darstellung des ersten Operanden wird um die im zweiten Operanden angegebene Anzahl von Stellen nach rechts verschoben. Von links werden Bits mit dem Wert des Vorzeichens ergänzt (0 für +, 1 für -). Die über die rechte Grenze hinaus verschobenen Bits fallen weg.	`9 >> 2` ergibt `2` (`1001` um zwei Bits nach rechts: `10`) `-9 >> 2` ergibt `-3`
>>>	rechts shift ohne Berücksichtigung des Vorzeichens. Die Binär-Darstellung des ersten Operanden wird um die im zweiten Operanden angegebene Anzahl von Stellen nach rechts verschoben. Die über die rechte Grenze hinaus verschobenen Bits fallen weg, von links werden Bits mit dem Wert 0 ergänzt. Für nicht-negative Zahlen führt der Operator >>> zum gleichen Ergebnis wie der Operator >>.	`19 >>> 2` ergibt `4` (`10011` um zwei Bits nach rechts: `100`) `-9 >>> 2` ergibt `1073741821`

5.5 Boolsche Operatoren (&&, ||, !)

Bei den boolschen (= logischen) Operatoren wird jeweils der erste (bei logischem NOT ist das auch der Einzige) Operand ausgewertet und ggf. auf den Typ boolean konvertiert. Zurückgeliefert wird entweder dieser boolsche Wert oder der Wert eines Operanden:

Operator	Beschreibung	Beispiel

`&&`	logisches AND.
	Gibt `false` zurück, falls die Auswertung des linken Operanden konvertiert auf boolean `false` ergibt. Der rechte Operand wird in diesem Fall nicht mehr ausgewertet. Andernfalls wird der Wert des rechten Operanden zurückgegeben.
		`"boy" && "girl"` ergibt: `"girl"`
		
		`3==4 && "girl"` ergibt: `false`

`||`	logisches OR.
	Gibt den Wert des linken Operanden zurück, falls dessen Auswertung `true` ergibt. Der rechte Operand wird in diesem Fall nicht mehr ausgewertet. Andernfalls wird der Wert des rechten Operanden zurückgegeben.
		`"boy" || "girl"` ergibt: `true`
		
		`3==4 || "girl"` ergibt: `"girl"`

`!`	logisches NOT. Gibt `true` zurück, falls die Auswertung des Operanden `false` ergibt, anderenfalls `false`.	`!"girl"` ergibt: `false` `!(3==4)` ergibt: `true`

Abgekürzte Auswertung

Boolsche Ausdrücke werden von links nach rechts ausgewertet: sobald das Ergebnis feststeht, wird die Auswertung abgebrochen:

– `false &&` *anything*
  Ergibt beim logischen AND bereits die Auswertung des ersten Operanden `false`, wird der zweite Operand nicht mehr ausgewertet.

– `true ||` *anything*
  Ergibt beim logischen OR bereits die Auswertung des ersten Operanden `true`, wird der zweite Operand nicht mehr ausgewertet.

In den angeführten Fällen bleibt also der Ausdruck *anything* unausgewertet. Eventuelle Nebeneffekte, wie z.B. Zuweisungen innerhalb von *anything*, kommen nicht zum Tragen.

Beispiel

```
x=0;
document.write( false  &&  (x=99) ); //ausgegeben wird false
document.write(x);                   //ausgegeben wird 0
```

5.6 Zuweisungsoperatoren

Ein Zuweisungsoperator steht zwischen zwei Operanden. Er weist dem linken Operanden einen Wert zu, der auf dem Wert seines rechten Operanden basiert.

Gleichheitszeichen

Der grundlegende Zuweisungsoperator ist das Gleichheitszeichen (einfache Zuweisung):

operand1 =operand2

Der rechte Operand wird ausgewertet und das Ergebnis wird dem linken Operanden zugewiesen. Der Zuweisungsausdruck selbst repräsentiert diesen Wert. Der Ausdruck x=y+1 beispielsweise, weist x den Wert von y +1 zu und repräsentiert selbst den Wert y+1. Dieser Wert wird im Ausdruck z=(x=y+1) der Variablen z zugewiesen.

Ist das Ergebnis des rechten Operanden vom Typ `object` oder `function`, wird eine Referenz zugewiesen, in allen anderen Fällen ein Wert.

Beachten Sie, dass sich der Typ des linken Operanden durch eine Zuweisung ändern kann.

Beispiel

```
x=7;         // x hat den Typ number
x="otto";    // x hat nun den Typ string
x=[1,2,3];  // x ist jetzt ein Array mit drei Elementen 1, 2 und 3
x={y:1, z:23}; // x ist jetzt ein Objekt mit den Attributen y und z
```

Zuweisungsoperatoren für Standardoperationen

Alle übrigen Zuweisungsoperatoren sind Kurzformen für Standardoperationen, wie folgende beiden Tabellen zeigen:

Operator	Bedeutung
`x += y`	`x = x + y`
`x -= y`	`x = x - y`
`x *= y`	`x = x * y`
`x /= y`	`x = x / y`
`x %= y`	`x = x % y`

Zuweisungsoperatoren für arithmetische Operationen

Operator	Bedeutung
x <<= y	x = x << y
x >>= y	x = x >> y
x >>>= y	x = x >>> y
x &= y	x = x & y
x ^= y	x = x ^ y
x |= y	x = x | y

Zuweisungsoperatoren für bitweise Operationen

5.7 String-Verknüpfungsoperator (+)

Neben den Vergleichsoperatoren, die für Stringoperanden genutzt werden können, gibt es speziell für Strings den Verknüpfungsoperator +:

string1 + *string2*

Mit dem +-Operator werden die Werte der beiden Operanden aneinandergehängt: Das Ergebnis ist **ein** Stringwert.

Beispielsweise ergibt `"good " + "morning"` den String `"good morning"`.

Voraussetzung ist, dass mindestens einer der beiden Operanden **stringähnlich** ist. Als stringähnlich gelten die Datentypen `string` und `function` sowie alle Objekte, die keine eigene `valueOf`-Methode haben, oder deren `valueOf`-Methode einen String zurückliefert.

Auch der Kurzform-Zuweisungsoperator += lässt sich auf String-Operanden anwenden:

Falls beispielsweise `mystring` den Wert `"good "` hat, ergibt `mystring += "evening"` den String `"good evening"`.

Beispiel: arithmetische Addition vs. Stringverknüpfung

```
document.writeln(4 + 4);        //ausgegeben wird: 8

document.writeln(4 + "4");      //ausgegeben wird: 44

myString=new String(4);
document.writeln(4 + myString); //ausgegeben wird: 44
```

Das Objekt `myString` ist „stringähnlich“, da seine Methode `valueOf` einen String zurückliefert.

5.8 Spezielle Operatoren

5.8.1 Bedingungsoperator (?:)

Der Bedingungsoperator wird häufig als kurze Alternative zu einfachen IF-Anweisungen verwendet. Er ist der einzige JavaScript-Operator, der drei Operanden hat:

---

condition ? *expression1* : *expression2*

---

condition wird ausgewertet. Falls *condition,* konvertiert auf `boolean` Typ, den Wert `true` hat, liefert der Bedingungsoperator den Wert von *expression1* zurück, ansonsten den Wert von *expression2*. Der jeweils andere Ausdruck bleibt unausgewertet (siehe Beispiel 2).

Beispiel 1

```
document.write( age >= 30 ? "you're a senior!" : "you're a junior!" );
```

Beispiel 2

```
document.write( true ? x="Peter" : y="Paul"); //ausgegeben wird: Peter
document.write( typeof y );                   //ausgegeben wird: undefined
```

Beispiel 3

Dieses Beispiel erzeugt eine Drop-Down-Liste. Der Bedingungsoperator realisiert eine Vorbelegung: Hat das Attribut `COUNTRY` z.B. gegenwärtig den Wert 2, wird im Browser als Default „USA“ angezeigt.

Der Auswertungsoperator ##...# sorgt dafür, dass der jeweilige Wert des Bedingungsausdrucks (`"SELECTED"` oder `""`) unmittelbar in die HTML-Ausgabe einfließt.

```
<SELECT Name="COUNTRY" Size=1>
  <OPTION ##host.COUNTRY.Value == 1 ? "SELECTED" : ""# Value="1">Belgium
  <OPTION ##host.COUNTRY.Value == 2 ? "SELECTED" : ""# Value="2">USA
  <OPTION ##host.COUNTRY.Value == 3 ? "SELECTED" : ""# Value="3">Germany
</SELECT>
```

5.8.2 Komma-Operator (,)

Der Komma-Operator wertet zuerst den linken und dann den rechten Operanden aus, das Ergebnis des rechten Operanden wird zurückgegeben:

---

expression1 , expression2

Der Komma-Operator ermöglicht es, innerhalb eines Ausdrucks mehrere separate Auswertungen vornehmen zu lassen. Ein häufiger Einsatzfall sind for-Schleifen (siehe Beispiel 2) oder wenn in einem Auswertungsoperator mehrere Operationen durchgeführt werden sollen (siehe Beispiel 3).

Beispiel 1

```
x=((y=5),4)
```

Das Beispiel ist zwar nicht besonders sinnvoll, zeigt aber das Prinzip:
x hat nach dieser Zuweisung den Wert 4, y den Wert 5.

Beispiel 2

Der Komma-Operator wird in der letzten for-Schleife des Beispiels eingesetzt
(die anderen Schleifen dienen lediglich dazu ein zwei-dimensionales Array anzulegen und Werte zuzuweisen).

Der Komma-Operator ermöglicht es, innerhalb der Bedingung der for-Schleife neben dem Schleifenzähler `i` noch eine weitere Variable mitlaufen zu lassen.

```
d=new Array(10);
for (i=0;i<=9;i++) d[i]=new Array(9);
for (i=0;i<=9;i++) {
   for (j=0;j<=9;j++) {
      d[i][j]= i + ":" +j;
   }
}

for (i=0,j=9 ; i<=9 ; i++,j--)
   document.write(d[i][j] + " ; ");
```

Das Beispiel gibt die Werte der „diagonalen“ Array-Elemente aus:

0:9 ; 1:8 ; 2:7 ; 3:6 ; 4:5 ; 5:4 ; 6:3 ; 7:2 ; 8:1 ; 9:0 ;

Beispiel 3

```
##a=1,b=42,...,""#
```

Alle durch Komma getrennten Operationen werden ausgeführt. Durch einen Leerstring als letzten Operand des Komma-Operators kann erreicht werden, dass keine Ausgabe erzeugt wird. Innerhalb eines HTML-Bereichs wäre dieser Auswertungsoperator-Ausdruck also „unsichtbar“.

5.8.3 new-Operator

Mit dem `new`-Operator lassen sich Objekt-Instanzen für vor- und selbstdefinierte Klassen anlegen.

---

objectname = new *objecttype*([*parameter*] ...)

---

objectname
: Name der neuen Objekt-Instanz.

objecttype
: Objekttyp. Dies ist der Name der jeweiligen Konstruktor-Funktion. In server-seitigen Scripts ist hier derzeit einer der folgenden Objekttypen möglich:
`{Object|Boolean|Date|Document|Number|String|Array|RegExp|WT_Communication|WT_Userexit|`selbstdefinierte Objekte`}`

[*parameter*] ...
: Beim Aufruf der Konstruktor-Funktionen können Sie Parameter angeben und damit Attribute des neuen Objekts mit Werten versorgen. Welche Parameter dies jeweils sind und was sie bedeuten, hängt von der jeweiligen Konstruktor-Funktion ab. Einzelheiten hierzu finden Sie im Kapitel „Eingebaute Klassen und Methoden“ auf Seite 127ff.

Beispiel

```
myarray = new Array(20);
```

Dieser Ausdruck legt ein Array-Objekt mit dem Namen `myarray` an, dessen erstes (und derzeit einziges) Element den Wert 20 hat.[1]

[1] Bis WTML Version 2.0 wurde so ein Array mit 20 Elementen angelegt.

5.8.4 delete-Operator

Der `delete`-Operator löscht ein Objekt, ein Objektattribut oder ein Array-Element und gibt den belegten Speicher frei.

Der Operator liefert `undefined` zurück.

5.8.5 in-Operator

Der `in`-Operator liefert einen boolschen Wert zurück, der anzeigt, ob ein bestimmtes Attribut in einem spezifizierten Objekt enthalten ist.

---

attributeNameOrIndex `in` *object*

---

attributeNameOrIndex
: String-Ausdruck oder numerischer Ausdruck, der den Namen des Attributs oder einen Array-Index repräsentiert.

objectname
: Name des Objekts, für das geprüft werden soll, ob das durch *attributeNameOrIndex* spezifizierte Attribut bzw. der durch *attributeNameOrIndex* spezifizierte Array-Index im Objekt enthalten ist.

Beispiel

```
a = new Object();
a.b = "abc";
if ("b" in a )      // liefert true zurück
…
if ("c" in a )      // liefert false zurück
```

5.8.6 instanceof-Operator

Der `instanceof`-Operator liefert einen boolschen Wert zurück, der anzeigt, ob ein bestimmtes Objekt von einer spezifizierten Klasse abgeleitet ist.

---

objectname `instanceof` *objecttype*

---

objectname
: spezifiziert das Objekt, für das geprüft werden soll, ob es von der eingebauten Klasse *objecttype* abgeleitet ist.

objecttype
: spezifiziert die Klasse, für die geprüft werden soll, ob das Object *objectname* von ihr abgeleitet ist.

Beispiel

```
a= new String ("abc");
b = "abc";
if ( a instanceof String ) . . . // liefert true zurück
if ( b instanceof String ) . . . // liefert false zurück, weil kein String-
                                 // Objekt
```

5.8.7 WT_THIS (nur für Klassen-Templates)

Dieses Schlüsselwort liefert innerhalb eines Klassen-Templates eine Referenz auf das aufrufende Host-Datenobjekt. Damit kann im Klassen-Template auf das rufende Host-Datenobjekt zugegriffen werden.

Nähere Informationen zu Klassen-Templates und `WT_THIS` finden Sie im Kapitel „Klassen-Templates (*.clt)“ auf Seite 329.

5.8.8 this

Dieses Schlüsselwort liefert innerhalb eines Konstruktors bzw. einer Methode eine Referenz auf das rufende Objekt.

Beispiel

```
// Methode fuer neue Klasse "Employee":
function gibName() {
   return (this.name);
}

// Konstruktor fuer Klasse "Employee":
function Employee() {
   // Definition der Klassen-Attribute:
   this.name      = "";
   this.division  = "development";
   this.machine   = "computer";
   this.worktime  = 35;

   // Referenz auf Methode:
   this.gibName   = gibName;
}
```

Außerhalb von Konstruktoren und Methoden liefert `this` das globale Objekt zurück, das alle global definierten Variablen enthält.

Die Objekte `WT_SYSTEM`, `WT_POSTED`, `WT_HOST` und global definierte Variablen, die durch Module angelegt wurden, sind keine Attribute des globalen Objekts und werden daher nicht als Attribute von `this` zurückgeliefert.

Beispiel

```
a=1;
b=2;
c=3;
for(i in this)
  document.writeln(i,': ',this[i]);
```

liefert die folgende Ausgabe:

```
a: 1
b: 2
c: 3
```

Sie können das `this`-Literal auch dazu verwenden, innerhalb einer Funktion auf globale Variablen zuzugreifen, die durch gleichnamige lokale Variablen verdeckt sind.

Beispiel

```
a=6;
function f(x)
{
  var a=7;
..return this.a*x;
}
res=f(7);
```

In `res` wird wie immer *42* zurückgeliefert. Mit `this.a` wird die globale Variable `a` referenziert, obwohl eine gleichnamige lokale Variable existiert.

5.8.9 Auswertungsoperator ##...#

Der Auswertungsoperator wertet den in ihm enthaltenen Ausdruck aus und gibt das Ergebnis als String zurück. Einzige Ausnahme: Wenn das Ergebnis `undefined` ist, wird ein Leerstring und nicht der String "`undefined`" zurückgegeben.

expression

expression beliebiger Ausdruck.

Durch die Auswertungsoperatoren können Sie z.B. im Template auf die aktuellen Werte von Objekten oder Objekt-Attributen zugreifen. Dabei werden die Auswertungsoperatoren durch diese aktuellen Werte ersetzt, d.h. sie werden ähnlich wie eine Variable verwendet.

Im Gegensatz zu Variablen können Auswertungsoperatoren jedoch auch in Kontexten eingesetzt werden, in denen sonst nicht mit Variablen gearbeitet werden kann:

– im festen HTML-Text

– innerhalb von HTML-Tags und in bestimmten WTML-Tags, um z.B. Tag-Eigenschaften dynamisch zu setzen. Der Auswertungsoperator kann hier sogar innerhalb der String-Begrenzer " " bzw. ' ' stehen. Solche Strings, die Auswertungsoperatoren enthalten, werden auch **einfache Stringausdrücke** genannt.

Innerhalb von OnCreate- und OnReceive-Scripts ist der Auswertungsoperator nicht zulässig. Sie können jedoch an diesen Stellen die `toString`-Methode des jeweiligen Objekts nutzen, die eine vergleichbare Funktionalität bietet.

Beispiele

```
##WT_SYSTEM.BASEDIR#   //liefert den Wert des Systemobjekt-Attributs BASEDIR

##++index#             //liefert einen um 1 erhöhten Index

##void a+=6*7#         //liefert Leerstring

##WT_HOST.STD.SELECT#  //liefert die Auswertung des Klassen-Templates für das
                      //Host-Datenobjekt SELECT
```

Siehe auch „Beispiel 3“ auf Seite 80.

Objekte im Auswertungsoperator

Auf ein Objekt, das alleine innerhalb eines Auswertungsoperators steht, wird immer die Methode `toString` angewendet. So ist z.B. `##hostobject#` gleichbedeutend mit `##hostobject.toString()#`. Für Host-Datenobjekte wird also ggf. das entsprechende Klassen-Template ausgeführt (siehe Kapitel „Klassen-Templates (*.clt)“ auf Seite 329ff).

Ebenso ist es möglich, z.B. `##WT_SYSTEM#` zu schreiben. Das Ergebnis ist in einem solchen Fall eine Auflistung sämtlicher Attribute von `WT_SYSTEM` und deren Werte (siehe Abschnitt „Methode toString“ auf Seite 188).[1]

5.8.10 typeof-Operator

Der `typeof`-Operator ermittelt den Typ des Operanden und gibt das Ergebnis als String zurück. Mögliche Werte sind: `undefined` , `object` , `function`, `number`, `boolean` oder `string`.

`typeof` *operand*

Beispiel

```
myArray=new Array("Peter", 49, false, null);
document.writeln("type of myArray is: " + typeof myArray +"<BR>");
document.writeln("type of myarray is: " + typeof myarray+"<BR>");
document.writeln("type of myArray.length is: "
                 + typeof myArray.length+"<BR>");

for (i in myArray)
    document.writeln("type of " + myArray[i] + " is: "
                     + typeof myArray[i]+"<BR>");
```

Das Beispiel erzeugt folgende Ausgabe:

```
type of myArray is: object
type of myarray is: undefined
type of myArray.length is: number
type of Peter is: string
type of 49 is: number
type of false is: boolean
type of null is: object
```

[1] Bis WTML Version 2.0 war das Ergebnis gleich dem Ergebnis der Methode `toString`, nämlich `[object Object]`.

5.8.11 void-Operator

Der `void`-Operator wertet den Operanden aus ohne das Ergebnis der Auswertung zurückzuliefern. Er gibt keinen „richtigen“ Wert zurück, sondern `undefined`.

`void` *operand*

Beispiel

```
document.writeln("the 'value' of the void expression is: " + void(x=4));
document.writeln("the value of x is: " + x);
```

Das Beispiel erzeugt folgende Ausgabe:

```
the 'value' of the void expression is: undefined
the value of x is: 4
```

5.9 Auswertungsreihenfolge

Die einzelnen Operatoren verteilen sich auf 16 Vorrangstufen:
– Operatoren gleicher Stufe werden der Reihe nach (meist von links nach rechts) ausgeführt.
– Bei Operatoren unterschiedlicher Stufe wird zunächst der mit höherem Vorrang ausgeführt.

Die Zeilen der folgenden Tabelle stehen für die 16 Rangstufen beginnend mit dem höchsten Vorrang absteigend geordnet bis zum niedrigsten Rang:

Operator-Typ	**individuelle Operatoren**

Aufruf, Zugriff, die Klammern	`()` bei Funktionsaufrufen
	`[]` z.B.: `myArray[2+3]`
	`.` z.B.: `myArray.length`

einstellige Operatoren	`! ~ - ++ -- typeof void`
Multiplikation, Division, Modulus	`* / %`
Addition/String-Verknüpfung, Subtraktion	`+ -`
bitweise Shift-Operatoren	`<< >> >>>`
relationale Vergleichsoperatoren	`< <= > >=`
Gleichheit/Ungleichheit	`== !=`
bitweises AND	`&`
bitweises XOR	`^`
bitweises OR	`|`
logisches AND	`&&`
logisches OR	`||`
Bedingungsoperator	`?:`
Zuweisungsoperatoren	`= += -= *= /= <<= >>= >>>= &= ^= |=`
Komma-Operator	`,`
Auswertungsoperator	`##...#`

Außer bei den Operatoren `&&`, `||` und `?:` werden immer sämtliche Operanden ausgewertet.

Beispiele

```
document.writeln( true && true != true && false );//Ausgabe: false
x = false ? 1 : 2;                                    // x ist 2 (nicht false)
document.writeln(2+3*4);                           //Ausgabe: 14 (nicht 20)
document.writeln(typeof 2+42)                      //Ausgabe: number42 (nicht number)
```

6 Globale Funktionen

Dieses Kapitel beschreibt die klassenunabhängigen, globalen Funktionen. Zur Fehlerbehandlung bei den Funktionen siehe auch Abschnitt „Fehlerbehandlung durch Ausnahmen (Exceptions)“ auf Seite 322.

6.1 Funktion copyFile()

Die globale Funktion `copyFile()` kopiert eine existierende Datei.

`copyFile(`*source*`,` *destination*`)`

---

source
: Pfad und Name der Datei, die kopiert werden soll.

destination
: Pfad und Name der Zieldatei.

Beide Dateien müssen innerhalb des Basisverzeichnisses liegen. *source* und *destination* können relativ zum Basisverzeichnis oder absolut angegeben werden.

Beispiel

```
copyFile("folder/file1","folder/filenew");
```

6.2 Funktion createFolder()

Die Funktion `createFolder()` legt das angegebene Verzeichnis im Basisverzeichnis an. Die Angabe des Verzeichnisses erfolgt immer relativ zum Basisverzeichnis. Sind im Pfad angegebene Elternverzeichnisse nicht vorhanden, kann mit dem Parameter *parents* (wenn er nach `boolean` konvertiert `true` ergibt) angegeben werden, dass diese erzeugt werden sollen.

`createFolder(`*foldername*`[ , `*parents*`])`

---

foldername
: Name des Verzeichnisses relativ zum Basisverzeichnis.

parents
: Anzeige, dass alle nicht existenten Elternverzeichnisse zu diesem Pfad auch angelegt werden sollen.

Ergebnis

Boolean-Wert, der angibt, ob das Verzeichnis angelegt wurde oder nicht.

6.3 Funktion deleteFile()

Die Funktion `deleteFile()` löscht Dateien und Verzeichnisse aus dem Basisverzeichnis. Leere Verzeichnisse werden genauso gelöscht wie Dateien. Nicht leere Verzeichnisse werden inklusive Inhalt gelöscht, wenn der Parameter *recursive* gesetzt (nach `boolean` konvertiert `true`) ist.

`deleteFile(`*filename*`[ , `*recursive*`])`

---

filename
: Name des Verzeichnisses relativ zum Basisverzeichnis.

recursive
: Anzeige, dass nicht leere Verzeichnisse inklusive Inhalt gelöscht werden.

Ergebnis

Boolean-Wert, ob die Datei gelöscht wurde oder nicht.

Beispiel

```
deleteFile("Ablage",true);
```

löscht das Verzeichnis *Ablage* im Basisverzeichnis samt Inhalt.

6.4 Funktion escape()

Die globale Funktion `escape()` konvertiert Sonderzeichen innerhalb eines ASCII-Strings in die hexadezimale Darstellung. Dabei werden alle Sonderzeichen, die nicht aus der Menge { 'A'-'Z', 'a'-'z', '0'-'9', '+', '–', '*', '/', '_', '@', '.' } stammen, in die hexadezimale Darstellung der Form *%nn* umgesetzt.

`escape(`*string*`)`

---

string ASCII-String.

Ergebnis

Die als Argument übergebene Zeichenkette, wobei alle Sonderzeichen in hexadezimale Darstellung konvertiert sind.

Beispiel

```
document.writeln("<BR>" + escape("The_rain. In Spain, Ma'am!"));
```

Das Beispiel erzeugt folgende Ausgabe:

```
The_rain.%20In%20Spain%2C%20Ma%27am%21
```

Siehe auch

„Funktion unescape()“ auf Seite 124.

6.5 Funktion eval()

Die globale Funktion `eval()` überprüft die als Argument angegebene Zeichenkette. Handelt es sich dabei um eine gültige WTScript-Anweisung bzw. eine Folge von gültigen WTScript-Anweisungen, wird diese ausgeführt. Handelt es sich hingegen um einen Ausdruck, so wird dessen Ergebnis berechnet und zurückgegeben.

Die Funktion `eval()` kann dazu verwendet werden, WTScript-Anweisungen oder arithmetische Ausdrücke als Zeichenketten dynamisch zu erzeugen und diese zu einem späteren Zeitpunkt auszuführen oder auszuwerten.

`eval(`*string*`)`

---

string Zeichenkette, die entweder WTScript-Anweisungen oder einen Ausdruck enthält.

Ergebnis

Das Argument *string* wird in eine Zeichenkette konvertiert und dann als WTScript-Programm oder Ausdruck ausgewertet und ausgeführt.

Beispiel

```
document.writeln("<BR>" + eval("3+7"));       // Ausdruck

// WTScript-Anweisung:
x=39;
y=2;
eval("if((x+y+1) == 42) abc='ja'; else abc='nein';");
document.writeln("<BR>" + abc);
```

Das Beispiel erzeugt folgende Ausgabe:

```
10
ja
```

6.6 Funktion evaluate()

Die globale Funktion `evaluate()` ruft das angegebene Template als eine Art Unterprogramm auf und liefert den erzeugten HTML-Text als Zeichenkette zurück. Diese Zeichenkette kann dann beliebig weiterverarbeitet werden.

Die Funktion `evaluate()` verhält sich wie die Funktion `include()`, mit dem Unterschied, dass Ausgaben, die normalerweise in den HTML-Ausgabestrom gehen und später im Browser ausgegeben werden, in eine Zeichenkette geschrieben werden. Insbesondere werden auch OnReceive-Teile, die im evaluierten Template stehen, zum nächsten Receive-Zeitpunkt ausgeführt.

Wenn Sie `evaluate()` innerhalb einer Funktion aufrufen, beachten Sie analog den Abschnitt „Hinweise für die Verwendung der Funktion include() innerhalb einer Funktion“ auf Seite 108.

`evaluate(`*template*`)`

---

template

Zeichenkette mit dem Namen eines Templates. *template* ist ein relativer Dateiname. Das Suffix `.htm` des Dateinamens müssen Sie nicht angeben. WebTransactions sucht das entsprechende Template gemäß eingestellter Sprache und eingestelltem Stil.

Es gilt die allgemein gültige Suchreihenfolge für Templates, siehe hierzu auch WebTransactions-Handbuch „Konzepte und Funktionen“.

Ergebnis

Der Inhalt des Templates wird ausgeführt und das Ergebnis als Zeichenkette zurückgeliefert. Die Zeichenkette kann für Abfragen oder Berechnungen weiterverarbeitet werden.

Beispiel

Template `test.htm`:

```
<H4> Dies ist ein Test von USER </H4>
<wtoncreatescript>
<!--
stringInTest = "Hallo";
//-->
</wtoncreatescript>
```

Im rufenden Template steht:

```
<wtoncreatescript>
<!--
document.write("<br><br><H2>*******evaluate Test********</H2><br>");
str = evaluate("test");
//String verändern und ausgeben
newStr = str.replace("von USER","vom WebTransactions-Team");
document.write(newStr);
// Jetzt kann auf Variablen zugegriffen werden, die in "test" definiert sind,
document.write(stringInTest);
//-->
</wtoncreatescript>

<H1>Ende des Test-Templates</H1>
```

Das Beispiel erzeugt folgende Ausgabe:

*********evaluate Test**********

Dies ist ein Test vom WebTransactions-Team

Hallo

Ende des Test-Templates

Siehe auch

„Funktion include()“ auf Seite 107.

6.7 Funktion exitDialogStep()

Die globale Funktion `exitDialogStep()` bricht die Verarbeitung aller an diesem Dialogschritt beteiligten Templates ab.

Ausführlich beschrieben ist die Dialogsteuerung im WebTransactions-Handbuch „Konzepte und Funktionen“.

`exitDialogStep()`

Beispiel

Template `test1.htm`:

```
<wtOnCreateScript>
<!--
    document.write("vor exitDialogStep<BR>");
    exitDialogStep();
    document.write("nach exitDialogStep<BR>");
//-->
</wtOnCreateScript>
```

Im rufenden Tempate steht:

```
<wtOnCreateScript>
<!--
document.write("<br><br><H2>******exitDialogStep Test******</H2><br>");
include("test1");
document.write("Ausgabe des rufenden Templates<br>");
//-->
</wtOnCreateScript>
```

Das Beispiel erzeugt folgende Ausgabe:

*********exitDialogStep Test**********

vor exitDialogStep

Anmerkung:

Mit `exitDialogStep()` wird in diesem Fall sowohl das includierte als auch das rufende Template abgebrochen.

Siehe auch

„Funktion exitReceiveProcessing()“ auf Seite 99, „Funktion exitScript()“ auf Seite 100, „Funktion exitSession()“ auf Seite 102 und „Funktion exitTemplate()“ auf Seite 103.

6.8 Funktion exitReceiveProcessing()

Die globale Funktion `exitReceiveProcessing()` ist nur in einem ReceiveScript-Bereich gültig. Wenn sie dort aufgerufen wird, beendet sie die aktuelle sowie alle folgenden Receive-Regeln. `exitReceiveProcessing()` liefert kein Ergebnis zurück.

```
exitReceiveProcessing()
```

Siehe auch

„Funktion exitDialogStep()“ auf Seite 98, „Funktion exitScript()“ auf Seite 100, „Funktion exitSession()“ auf Seite 102 und „Funktion exitTemplate()“ auf Seite 103.

6.9 Funktion exitScript()

Die globale Funktion `exitScript()` bricht die Verarbeitung des aktuellen Script-Bereichs ab. Die Weiterverarbeitung setzt bei der ersten Anweisung nach dem Script-Bereich wieder auf.

```
exitScript()
```

Beispiel 1

Template `test1.htm`:

```
<wtOnCreateScript>
<!--
   document.write("vor exitScript<BR>");
    exitScript();
    document.write("nach exitScript<BR>");
//-->
</wtOnCreateScript>
```

Im rufenden Template steht:

```
<wtoncreatescript>
<!--
document.write("<br><br><H2>*******exitScript Test********</H2><br>");
include("test1");
document.write("Ausgabe des rufenden Templates<br>");
//-->
</wtoncreatescript>
```

Das Beispiel erzeugt folgende Ausgabe:

*********exitScript Test**********

vor ExitScript

Beispiel 2

```
<wtOnCreateScript>
<!--
document.write ("Vor exitScript()<br>");
exitScript();
document.write ("Nach exitScript()<br>");
//-->
</wtOnCreateScript>
<wtOnCreateScript>
<!--
document.write ("Neuer Scriptbereich<br>");
//-->
</wtOnCreateScript>
```

Das Beispiel erzeugt folgende Ausgabe:

```
Vor exitScript()
Neuer Scriptbereich
```

Siehe auch

„Funktion exitDialogStep()“ auf Seite 98, „Funktion exitReceiveProcessing()“ auf Seite 99 und „Funktion exitTemplate()“ auf Seite 103.

6.10 Funktion exitSession()

Die globale Funktion `exitSession()` beendet die aktuelle WebTransactions-Sitzung zum nächstmöglichen Zeitpunkt. Dieser richtet sich nach dem Ort des Aufrufs der Funktion `exitSession()`:

– bei Aufruf innerhalb eines `wtOnCreateScript` ist das Ergebnis des aktuellen WTML-Dokuments die letzte an den Browser gesendete Seite.

– bei Aufruf innerhalb eines `wtOnReceiveScript` wird die WebTransactions-Sitzung nach dem Erzeugen der nächsten synchronisierten Ausgabe beendet.

Das Verhalten dieser Funktion ist äquivalent zu `WT_SYSTEM.EXIT_SESSION="TRUE"`.

`exitSession()`

`exitSession()` beendet nach diesem Dialogschritt die WebTransactions-Sitzung. Wenn Sie die Sitzung sofort beenden wollen, müssen Sie zusätzlich die Funktion `exitDialogStep()` aufrufen. Andernfalls werden die hinter `exitSession()` stehenden Anweisungen noch ausgeführt. bevor die Sitzung beendet wird.

Siehe auch

Globales Systemobjekt-Attribut `PREVENT_EXIT_SESSION` (siehe WebTransactions-Handbuch „Konzepte und Funktionen), „Funktion exitDialogStep()“ auf Seite 98, „Funktion exitReceiveProcessing()“ auf Seite 99, „Funktion exitScript()“ auf Seite 100 und „Funktion exitTemplate()“ auf Seite 103.

6.11 Funktion exitTemplate()

Die globale Funktion `exitTemplate()` bricht die Verarbeitung des aktuellen Templates ab. Die Weiterverarbeitung setzt bei der nächsten Anweisung im rufenden Template wieder auf. Das können Sie sich wie einen Rücksprung aus einem Unterprogramm vorstellen.

Ein Aufruf der Funktion `exitTemplate()` auf oberster Template-Ebene entspricht im Ergebnis einem Aufruf der Funktion `exitDialogStep()`.

`exitTemplate()`

Beispiel

Template test1.htm:

```
<wtOnCreateScript>
<!--
   document.write("vor exitTemplate<BR>");
    exitTemplate();
    document.write("nach exitTemplate<BR>");
//-->
</wtOnCreateScript>
```

Im rufenden Template steht:

```
<wtOnCreateScript>
<!--
document.write("<br><br><H2>*****exitTemplate Test********</H2><br>");
include("test1");
document.write("Ausgabe des rufenden Templates<br>");
//-->
</wtOnCreateScript>
```

Das Beispiel erzeugt folgende Ausgabe:

*********exitTemplate Test**********

vor exitTemplate
Ausgabe des rufenden Templates

Anmerkung

Hier wird nur die Verarbeitung des includierten Templates abgebrochen. Die Verarbeitung des rufenden Templates geht weiter.

Siehe auch

„Funktion exitDialogStep()“ auf Seite 98, „Funktion exitReceiveProcessing()“ auf Seite 99, „Funktion exitScript()“ auf Seite 100 und „Funktion exitSession()“ auf Seite 102.

6.12 Funktion forward()

Die globale Funktion `forward()` sucht das als Argument angegebene Template und startet dessen Verarbeitung. Die Kontrolle geht auf dieses Template über und wird nicht mehr an das rufende Template zurückgegeben. Das bedeutet, dass alle Anweisungen im rufenden Template, die nach dem Aufruf von `forward()` stehen, nicht mehr ausgeführt werden. Receive-Regeln, die bis zu diesem Zeitpunkt gelesen wurden, werden zum nächsten Receive-Zeitpunkt ausgeführt.

Wird das angegebene Template nicht gefunden, dann wird eine Fehlermeldung ausgegeben und das WTScript läuft nach `forward()` weiter.

Wenn Sie `forward()` innerhalb einer Funktion aufrufen, beachten Sie analog den Abschnitt „Hinweise für die Verwendung der Funktion include() innerhalb einer Funktion“ auf Seite 108.

`forward(`*template*`)`

---

template

Zeichenkette mit dem Namen eines Templates. *template* ist ein relativer Dateiname. Das Suffix `.htm` des Dateinamens müssen Sie nicht angeben. WebTransactions sucht das entsprechende Template gemäß eingestellter Sprache und eingestelltem Stil.

Es gilt die allgemein gültige Suchreihenfolge für Templates, siehe hierzu auch WebTransactions-Handbuch „Konzepte und Funktionen“.

Ergebnis

Die Verarbeitungs-Kontrolle wird an das angegebene Template übergeben.

Beispiel

Template `test.htm`:

```
<H4> Dies ist ein Test von USER </H4>
<wtoncreatescript>
<!--
stringInTest = "Hallo";
//-->
</wtoncreatescript>
```

Im rufenden Template steht:

```
<wtoncreatescript>
<!--
document.write("<br><br><H2>*******forward Test********</H2><br>");
forward("test");
document.write("Dieser Text wird nur im Fehlerfall ausgegeben");
//-->
</wtoncreatescript>
```

Das Beispiel erzeugt folgende Ausgabe:

*********forward Test**********

Dies ist ein Test von USER

Siehe auch

„Funktion evaluate()“ auf Seite 96 und „Funktion include()“ auf Seite 107.

6.13 Funktion import()

Mit der Funktion `import()` können Sie ein Template als Modul laden. Funktionen und Variablen, die Sie in Modulen definieren, stehen Ihnen in der gesamten WebTransactions-Sitzung zur Verfügung.

Mehr Informationen über Module und was Sie beachten müssen, wenn Sie ein Template als Modul laden wollen, finden Sie im WebTransactions-Handbuch „Konzepte und Funktionen“, Abschnitt „Master-, Klassen- und Modul-Templates“.

`import(`*template*`)`

---

template

Zeichenkette mit dem Namen eines Templates. *template* ist ein relativer Dateiname. Das Suffix `.htm` des Dateinamens müssen Sie nicht angeben. WebTransactions sucht das entsprechende Template gemäß eingestellter Sprache und eingestelltem Stil.

Es gilt die allgemein gültige Suchreihenfolge für Templates, siehe hierzu auch WebTransactions-Handbuch „Konzepte und Funktionen“.

Das Verzeichnis `<WebTA-Installationsverzeichnis>/modules` wird in den Suchpfad für Templates an letzter Stelle aufgenommen, um das explizite Laden optionaler Standard-Module zu ermöglichen.

6.14 Funktion include()

Die globale Funktion `include()` ruft das angegebene Template als eine Art Unterprogramm auf. Die Ergebnisse des includierten Templates werden direkt in den HTML-Ausgabestrom ausgegeben.

Mit dieser globalen Funktion können Sie auch in WTScript-Bereichen Templates inkludieren.

`include(`*template*`)`

---

template

Zeichenkette mit dem Namen eines Templates. *template* ist ein relativer Dateiname. Das Suffix `.htm` des Dateinamens müssen Sie nicht angeben. WebTransactions sucht das entsprechende Template gemäß eingestellter Sprache und eingestelltem Stil.

Es gilt die allgemein gültige Suchreihenfolge für Templates, siehe hierzu auch WebTransactions-Handbuch „Konzepte und Funktionen“.

Ergebnis

Das includierte Template wird komplett eingeschoben. Innerhalb des includierten Templates können die gleichen Sprachmittel verwendet werden wie in jedem anderen Template, auch `wtOnCreate`-Scripts und `wtOnReceive`-Scripts. Die WTML-Tags müssen allerdings in jedem Template syntaktisch vollständig sein, z.B. ist es *nicht* zulässig, eine IF-Kontrollstruktur im includierenden Template beginnen zu lassen und im includierten Template abzuschließen.

Beispiel

Template `test.htm`:

```
<H4> Dies ist ein Test von USER </H4>
<wtoncreatescript>
<!--
stringInTest = "Hallo";
//-->
</wtoncreatescript>
```

Im rufenden Template steht:

```
<wtoncreatescript>
<!--
document.write("<br><br><H2>*******Include Test********</H2><br>");
include("test");
// Jetzt kann auf Variablen, die in "test" definiert sind, zugegriffen werden
document.write(stringInTest);
//-->
</wtoncreatescript>

<H1>Ende des Test-Templates</H1>
```

Das Beispiel erzeugt folgende Ausgabe:

*********Include Test***********

Dies ist ein Test von USER

Hallo

Ende des Test-Templates

Siehe auch

„Funktion evaluate()“ auf Seite 96, „Funktion forward()“ auf Seite 104 und „Funktion import()“ auf Seite 106.

Hinweise für die Verwendung der Funktion include() innerhalb einer Funktion

Wenn Sie `include()` (`forward()` und `evaluate()` entsprechend) innerhalb einer Funktion aufrufen, stehen innerhalb dieser Funktion Funktionen und die Variablen zur Verfügung, die in dem inkludierten Template mit dem Schlüsselwort `var` definiert sind, zur Verfügung; und zwar als lokale Variablen bzw. Funktionen.

Funktionen aus dem inkludierten Template können sich gegenseitig aufrufen. Ebenso können Konstruktoren und Methodendefinitionen in den inkludierten Templates lokal ausgeführt werden.

Wenn mit einem Konstruktor aus dem inkludierten Template ein Objekt erzeugt wird, das länger lebt als die Funktion, in der es definiert wurde, kann folgendes Problem auftreten: Globale Hilfsfunktionen, die in dem inkludierten Template definiert und innerhalb der Methoden verwendet werden, stehen nach Ablauf der äußeren Funktion nicht mehr zur Verfügung.

Definieren und verwenden Sie daher solche Hilfsfunktionen stets als Methode des Objekts.

Beispiel

Die `include`-Funktion zum Inkludieren des Templates `myClass.htm` wird innerhalb der Funktion `outer()` aufgerufen:

```
function outer()
{
  include('myClass.htm');
  return new myClass();
}
myObject= outer();
myObject.myMethod();
```

Probleme treten auf, wenn `myClass.htm` die Klasse folgendermaßen definiert:
Die Hilfsfunktion `twice()` wird in der Funktion `myMethod()` verwendet und ist außerhalb der Funktion `outer()` nicht mehr verfügbar.

```
function myClass()
{
  this.myMethod=myMethod;
}
function myMethod()
{
  document.write( twice(21) );
}
function twice(x)
{
  return 2*x;
}
```

Es empfiehlt sich, auch die Hilfsfunktion `twice()` als Methode zu definieren, um zu vermeiden, dass `twice()` nach Ablauf der Funktion `outer()` nicht mehr verfügbar ist.

```
function myClass()
{
  this.myMethod=myMethod;
  this.twice=twice;
}
function myMethod()
{
  document.write( this.twice(21) );
}
function twice(x)
{
  return 2*x;
}
```

6.15 Funktion isRequestWaiting()

Die globale Funktion `isRequestWaiting()` fragt ab, ob ein neuer Auftrag (z.B. vom Browser gepostete Daten) auf Bearbeitung wartet.

An einen WebTransactions-Prozess können keine neuen Aufträge gestellt werden. solange dieser sich in einer Schleife innerhalb einer Funktion befindet, die durch `setTimeout()` angestoßen wurde. In einer solchen Schleife kann mit `isRequestWaiting()` zyklisch abgefragt werden, ob ein neuer Auftrag wartet.

```
isRequestWaiting()
```

Ergebnis

Boolean-Wert, der angibt, ob ein neuer Auftrag auf Bearbeitung wartet.

Beispiel

```
<html>
<body>
<wtoncreatescript>
 if(typeof WT_SYSTEM.counter != 'number' )
 WT_SYSTEM.counter=0;
 if(WT_POSTED.quit)
 {
  document.write('bye '+WT_SYSTEM.counter);
  exitSession();exitTemplate();
 }
 WT_SYSTEM.ex = new WT_Userexit( "WTSystemExits" );
 function backgroundLoop()
 {
 do
  {
   WT_SYSTEM.ex.WTSleep(1000);WT_SYSTEM.counter++;
  } while(! isRequestWaiting());
 }
</wtoncreatescript>
<form webtransactions>
##WT_SYSTEM.counter++#
<input type="submit" name="step" value="step">
<input type="submit" name="background" value="background">
<input type="submit" name="quit" value="quit">
<wtOnReceiveScript>
 if( WT_Posted.background )
  setTimeout('backgroundLoop()', 500);
</wtOnReceiveScript>
```

```
</form>
</body>
</html>
```

6.16 Funktion listFolder()

Die Funktion `listFolder()` liefert das Inhaltsverzeichnis eines Verzeichnisses aus dem Basisverzeichnis. Sie liefert einen Array von String-Objekten zurück, wobei jedes String-Objekt eine enthaltene Datei beschreibt, sofern ihr Name dem optionalen Parameter *pattern* entspricht. Eventuelle Links werden nicht aufgelöst, aber der Suffix '*.lnk*' unter Windows wird aus dem Dateinamen entfernt. Der Wert des String-Objektes ist der Name der Datei.

Die Verzeichnisse '.' (Punkt) und '..' (PunktPunkt) werden standardmäßig nicht in den Array aufgenommen. Mit dem Parameter *all* (wenn er nach `boolean` konvertiert `true` ergibt) kann angegeben werden, dass auch Dateinamen aufgelistet werden sollen, die mit '.' (Punkt) beginnen.

Das Objekt hat folgende Attribute:

isDir Vom Typ `boolean`. Zeigt an, ob es sich bei der beschriebenen Datei um ein Verzeichnis handelt.

size Vom Typ `number`. Gibt die Dateigröße in Bytes an.

lastAccess
Vom Typ `number`. Datum des letzten Zugriffs in Millisekunden (siehe Beispiel).

lastModified
Vom Typ `number`. Datum der letzten Veränderung in Millisekunden (siehe Beispiel).

`listFolder(`*foldername*`[, pattern [, all]])`

---

foldername
Name des Verzeichnisses relativ zum Basisverzeichnis. Soll das Basisverzeichnis aufgelistet werden, ist für *foldername* ein Schrägstrich ('/') anzugeben.

pattern
Muster, dem der Dateiname entsprechen muss (optional).

Das Muster entspricht den Regeln zur Angabe eines teilqualifizierten Dateinamens der Plattform, auf der WebTransactions eingesetzt wird.

all Anzeige, dass auch Dateinamen aufgelistet werden sollen, die mit '.' (Punkt) beginnen (optional).

Ergebnis

Array von String-Objekten. Existiert das Verzeichnis nicht, so wird `undefined` zurückgeliefert.

Beispiel 1

Ausgeben aller .htm-Dateien im Verzeichnis `config/forms`:

```
dirList=listFolder("config/forms","*.htm");
for (i=0;i<dirList.length;i++)
document.write(dirList[i]+"<br>");
```

Das Beispiel erzeugt folgende Ausgabe:

```
AutomaskOSD.htm
example.htm
StartTemplateHTTP.htm
StartTemplateOSD.htm
wtasync.htm
wtBrowserFunctions.htm
wtKeysOSD.htm
wtPkeyFunctions.htm
wtPKEYS.htm
wtPKeyValues.htm
```

Beispiel 2

Verwendung von *pattern* und der Attribute *lastAccess* und *lastModified* zur Ausgabe als Date-String:

```
fileArr = listFolder('/config/forms',"*.htm");
document.write("Letzter Zugriff auf " + fileArr[0] + ":   ");
document.writeln((new Date(fileArr[0].lastAccess)).toLocaleString());
document.write("<br>Letzte Veränderung von " + fileArr[0] + ":  ");
document.writeln((new Date(fileArr[0].lastModified)).toLocaleString());
```

Das Beispiel erzeugt folgende Ausgabe:

```
Letzter Zugriff auf AutomaskOSD.htm: 04/02/03 13:22:24
Letzte Veränderung von AutomaskOSD.htm: 04/02/03 13:22:24
```

Beispiel 3

```
listFolder("folder");
listFolder("folder","*.txt");
```

Verwendung von *all*, um auch Dateinamen mit '.' (Punkt) aufzulisten:

```
listFolder("folder","*",true);
```

Das Beispiel erzeugt folgende Ausgaben:

```
[(new String("file.txt")),(new String("file1"))]
[(new String("file.txt"))]
[(new String(".invisible")),(new String("file.txt")),(new String("file1"))]
```

6.17 Funktion moveFile()

Die globale Funktion `moveFile()` verschiebt eine Datei in ein anderes Verzeichnis im Basisverzeichnis oder benennt sie um..

`moveFile(`*name_1* , *name_2*`)`

---

name_1
: Pfad und Name der Datei, die verschoben/umbenannt werden soll.

name_2
: Pfad und Name der Zieldatei.

Beide Dateien müssen innerhalb des Basisverzeichnisses liegen. *name_1* und *name_2* können relativ zum Basisverzeichnis oder absolut angegeben werden.

Beispiel

```
moveFile("/folder/filenew","/folder/filerename");
```

6.18 Funktion Number()

Die globale Funktion `Number()` konvertiert einen beliebigen Ausdruck in den Datentyp `number` oder in `NaN`, wenn dies nicht möglich ist.

`Number(`*expression*`)`

---

expression
beliebiger Ausdruck

Ergebnis

Die Konvertierung erfolgt so, wie dies für den Ziel-Datentyp `number` im Abschnitt „Typkonvertierung“ auf Seite 49 beschrieben ist.

Gibt es für *expression* keine numerische Darstellung, so wird `NaN` zurückgeliefert.

Beispiel

```
object1="Hallo Welt!";
document.writeln("<BR>" + "Der Wert von " + object1 + " ist: " +
                 Number(object1));
object2=new String("42");
document.writeln("<BR>" + "Der Wert von " + object2 + " ist: " +
                 Number(object2));
```

Das Beispiel erzeugt folgende Ausgabe:

```
Der Wert von Hallo Welt! ist: NaN
Der Wert von 42 ist: 42
```

6.19 Funktion parseFloat()

Die globale Funktion `parseFloat()` konvertiert einen String in dessen numerischen Wert oder in `NaN`, wenn es sich nicht um eine Zahl handelt.

`parseFloat(`*string*`)`

string String mit einem numerischen Inhalt

Ergebnis

Numerischer Wert von *string*, wenn es sich bei *string* um die Darstellung eines numerischen Werts handelt, `NaN` sonst.

Falls `parseFloat()` im Argument ein Zeichen findet, das nicht zu einem numerischen Wert gehört, wird der Wert bis zu dieser Stelle ausgewertet und zurückgegeben. Der Rest des Arguments wird ignoriert. Beispielsweise liefert "`22a`" `22`, "`2.3.4`" `2.3`.

Führende und am Ende angehängte Blanks sind erlaubt.

Beispiel

```
string1="Hallo Welt!";
document.writeln("<BR>" + "Der Wert von \"" + string1 + "\" ist: " +
                 parseFloat(string1));
string2=" +3.1415927 ";
document.writeln("<BR>" + "Der Wert von \"" + string2 + "\" ist: " +
                 parseFloat(string2));
```

Das Beispiel erzeugt folgende Ausgabe:

```
Der Wert von "Hallo Welt!" ist: NaN
Der Wert von "+3.1415927" ist: 3.1415927
```

6.20 Funktion parseInt()

Die globale Funktion `parseInt()` konvertiert einen String in dessen numerischen Wert oder in `NaN`, wenn es sich nicht um eine ganze Zahl handelt. Zusätzlich kann die Funktion `parseInt()` die Basis der Zahlendarstellung berücksichtigen.

`parseInt(`*string*`[, `*base*`])`

---

string String mit numerischem Inhalt

base Grundzahl, auf deren Basis die Zahl in *string* angegeben ist. *base* muss eine Ganzzahl größer als 1 und maximal 36 sein. Ist *base* nicht angegeben, so wird 10 angenommen.

Ergebnis

Numerischer Wert von *string*, wenn es sich bei *string* um die Darstellung eines ganzzahligen numerischen Werts handelt, `NaN` sonst.

Falls `parseInt()` im Argument ein anderes Zeichen außer einer Ziffer in der angegebenen Basis findet, wird der Wert bis zu dieser Stelle ausgewertet und zurückgegeben. Der Rest des Arguments wird ignoriert. Führende und am Ende angehängte Blanks sind erlaubt.

Beispiel

```
document.writeln("<BR>" + parseInt("F", 16));
document.writeln("<BR>" + parseInt("1111", 2));
document.writeln("<BR>" + parseInt("F", 10));
document.writeln("<BR>" + parseInt("15"));
document.writeln("<BR>" + parseInt("4.2"));
```

Das Beispiel erzeugt folgende Ausgabe:

```
15
15
NaN
15
4
```

6.21 Funktion setNextPage()

Die globale Funktion `setNextPage()` legt das WTML-Dokument fest, das für den nächsten synchronisierten Dialogschritt verwendet werden soll.

Sie besetzt das Attribut `WT_SYSTEM.FORMAT` mit dem Namen des neuen Dokuments.

`setNextPage(`*documentName*`)`

---

documentName

Das Argument wird in einen String konvertiert, der als reiner Dateiname interpretiert wird. Die Dateinamen-Erweiterung `.htm` kann weggelassen werden. Das angegebene Dokument wird entsprechend den Einstellungen im Systemobjekt für Stil (`WT_SYSTEM.STYLE`) und Sprache (`WT_SYSTEM.LANGUAGE`) und der Suchstrategie gemäß `WT_SYSTEM.FORMAT` gesucht und ausgewählt.

Es gilt die allgemein gültige Suchreihenfolge für Templates, siehe hierzu auch WebTransactions-Handbuch „Konzepte und Funktionen“.

6.22 Funktion setSingleStep()

Die Funktion `setSingleStep()` wird im Zusammenhang mit der Offline-Einzelschrittverfolgung in WebLab verwendet. Bei der Offline-Einzelschrittverfolgung wird während der Seitengenerierung und dem Ablauf der Receive-Scripts jede ausgeführte Zeile mit den dazugehörenden Variablen und ihren Werten in einer Datei protokolliert. Nach Beendigung der Generierung kann WebLab diese Datei verarbeiten, und Sie können den Ablauf des Codes verfolgen.

Mit der Funktion `setSingleStep()` können Sie den Umfang der protokollierten Information steuern. Mit `setSingleStep("on")` schalten Sie die Protokollierung ein, mit `setSingleStep("off")` schalten Sie Protokollierung aus. Sie können die Protokollierung in einem Template durch Aufruf der Funktion `setSingleStep()` auch mehrfach ein- und ausschalten. Es werden dann nur die Einzelschritte zwischen den `setSingleStep("on")`- und `setSingleStep("off")`-Aufrufen aufgezeichnet.

`setSingleStep( {"on" | "off"} )`

"on" aktiviert die Aufzeichnung der Einzelschritte.
"off" deaktiviert die Aufzeichnung der Einzelschritte.

Beispiel

In diesem Beispiel werden in der `AutomaskOSD` nur die Aufrufe von `taggedOutput` protokolliert.

```
...
<wtOnCreateScript>
<!--
 ...
  for (element = OSD_0.$FIRST.Name; OSD_0 && element != '$END';
       element = OSD_0.$NEXT.Name)
  {
   ...
    if ( currentHostObject.Type == 'Protected' &&
         currentHostObject.Markable == 'No' )
    {
      setSingleStep ("on");
      taggedOutput( currentHostObject );
      setSingleStep ("off");
    }
   ...
  }
//-->
</wtOnCreateScript>
```

6.23 Funktion setTimeout()

Die globale Funktion `setTimeout()` dient dazu, eine Verarbeitung zeitverzögert zur Ausführung zu bringen. Das Script wird immer ohne Unterbrechung abgearbeitet (siehe jedoch „Funktion isRequestWaiting()“ auf Seite 110).

Das Timeout-Script wird frühestens nach Ablauf der angegebenen Zeit angestartet. In der Wartezeit können Anfragen von außen, z.B. Benutzereingaben am Browser, abgearbeitet werden. Der Start des Timeout-Scripts wird ggf. dann noch verzögert durch "gleichzeitige" Benutzereingaben oder asynchrone WebTransactions-Anfragen an die Sitzung. Eine merkliche Verzögerung tritt dabei aber meist nicht auf. Somit ist es z.B. möglich, die Seitengenerierung und das zeitverzögerte Abarbeiten von Scripten ohne unnötige Wartezeiten zu kombinieren.

Verhindert bzw. unterdrückt wird die zeitverzögerte Abarbeitung noch nicht gestarteter Scriptteile durch ein Beenden der Sitzung.

Die Abarbeitung erfolgt immer im synchronen Kontext.

`setTimeout(`*script*`,` *milliseconds*`)`

---

script Zeichenkette, die entweder einen Ausdruck oder WTScript-Anweisungen enthält, die ausgeführt werden sollen.

milliseconds
Anzahl von Millisekunden, die bis zur Ausführung von *script* mindestens verstreichen sollen.

Ergebnis

Keins.

Beispiel

Beispiel einer endlosen Verarbeitung im 15-Sekunden-Raster:

```
<wtOn...Script>
  function doit ()         { // do anything;};
  WT_SYSTEM.doit           = doit;
  WT_SYSTEM.endlessTask    = "WT_SYSTEM.doit();";
  WT_SYSTEM.endlessTask    += "setTimeout( WT_SYSTEM.endlessTask, 15000 );"
  setTimeout( WT_SYSTEM.endlessTask, 15000 );
</wtOn...Script>
```

In den WTScript-Anweisungen, die Sie zeitverzögert ausführen wollen, sollten Sie nur auf Funktionen und Variablen zugreifen, die in der ganzen Sitzung zur Verfügung stehen, also z.B. unter `WT_SYSTEM` abgelegt sind.

Wenn keine Dialogschritte vom Browser mehr eingehen, beendet die Sitzung sich, wenn `WT_SYSTEM.TIMEOUT_USER` abgelaufen ist.

6.24 Funktion setTraceLevel()

Die globale Funktion `setTraceLevel()` dient dazu, den Trace während einer laufenden Sitzung ein- oder auszuschalten. Dadurch kann der Trace auf bestimmte Teile der Templates beschränkt werden, um die Trace-Datei klein zu halten und auf festgelegte Abschnitte der Sitzung zu beschränken.

`setTraceLevel(`*string*`)`

---

string Der String `"FULL"` (vollständiger Trace eingeschaltet) oder der String `"NONE"` (Trace ausgeschaltet).

Siehe auch

„Funktion writeToTrace()“ auf Seite 125.

6.25 Funktion String()

Die globale Funktion `String()` konvertiert das Ergebnis eines Ausdrucks in einen String.

`String(`*expression*`)`

---

expression
Beliebiger Ausdruck.

Ergebnis

String, der das Objekt darstellt.

Die Konvertierung erfolgt so, wie dies für den Ziel-Datentyp `string` im Abschnitt „Typkonvertierung“ auf Seite 49 beschrieben ist.

Beispiel

```
value1 = true;
document.writeln("<BR>value1 = " + String(value1));

timeNow = new Date ();
document.writeln("<BR>Jetzige Zeit ist " + String(timeNow));
```

Das Beispiel erzeugt folgende Ausgabe:

```
value1 = true
Jetzige Zeit ist Mon Jul 05 16:33:06 1999
```

6.26 Funktion unescape()

Die globale Funktion `unescape()` konvertiert hexadezimale Darstellungen der Form *%nn* von Zeichen innerhalb eines ASCII-Strings in die Zeichen zurück.

`unescape(`*string*`)`

---

string ASCII-String mit hexadezimaler Darstellung von Zeichen.

Ergebnis

Die als Argument übergebene Zeichenkette, wobei alle hexadezimale Darstellungen von Zeichen in die Zeichen selbst konvertiert sind.

Beispiel

```
document.writeln("<BR>" +
                 unescape("The_rain.%20In%20Spain%2C%20Ma%27am%21"));
```

Das Beispiel erzeugt folgende Ausgabe:

```
The_rain. In Spain, Ma'am!
```

Siehe auch

„Funktion escape()“ auf Seite 94.

6.27 Funktion writeToTrace()

Die globale Funktion `writeToTrace()` schreibt einen Eintrag in die WebTransactions-Trace-Datei, sofern die Trace-Option aktiviert ist. Dabei wird diesem Trace-Eintrag zur Kennzeichnung der Text `writeToTrace():` vorangestellt.

`writeToTrace(`*expression*`)`

---

expression
: Beliebiger Ausdruck. Der Ausdruck wird in den Datentyp `String` konvertiert und bei aktivierter Trace-Option in die Trace-Datei geschrieben.

Siehe auch

„Funktion setTraceLevel()“ auf Seite 122.

7 Eingebaute Klassen und Methoden

Jede Variable vom Typ `object` gehört zu einer bestimmten Klasse. Diese Klasse bestimmt die Methoden, die für das Objekt zur Verfügung stehen. So ist z.B. die Methode `replace` auf alle Objekte der Klasse `String` anwendbar. Bei manchen Klassen gibt es auch vordefinierte Attribute, die dann jedes Objekt dieser Klasse automatisch hat, z.B. haben alle Strings das Attribut `length`, das jeweils die Anzahl der enthaltenen Zeichen angibt.

Dieses Kapitel beschreibt die eingebauten Klassen und die vordefinierten Attribute und Methoden, die WebTransactions unterstützt, in alphabetischer Reihenfolge. Das Verhalten dieser Klassen entspricht dem der entsprechenden JavaScript-Klassen.

7.1 Array-Klasse

Arrays sind Objekte, deren Attribute mit Indizes benannt sind. Indizes sind nicht-negative ganze Zahlen beginnend mit 0.

Arrays werden mit einer bestimmten Länge angelegt und können dynamisch erweitert werden. Die Typen der einzelnen Attribute eines Arrays können unterschiedlich sein. Ein Array kann als Attribut wieder ein Array enthalten. Auf diese Weise können Sie mehrdimensionale Arrays anlegen (siehe „Beispiel 2: zwei-dimensionales Array“ auf Seite 129).

7.1.1 Konstruktoren

```
Array()
Array(comp)
...
Array(comp, comp, ...)
```

Rückgabewert

Objekt der Klasse Array

Parameter

comp Komponente des Arrays

Wird der Konstruktor ohne Argument aufgerufen, so wird ein Array der Länge `0` ohne Attribute angelegt.

Wird der Konstruktor mit einem Argument aufgerufen, so wird ein Array der Länge 1 angelegt, für das Argument wird ein Attribut erzeugt und diesem das Argument zugewiesen.[1]

Wird der Konstruktor mit mehreren Argumenten aufgerufen, so wird immer ein Array entsprechender Länge angelegt, für jedes Argument ein Attribut erzeugt und diesem das Argument zugewiesen.

Ein Array kann auch direkt durch Zuweisung in der Literal-Notation angelegt werden (siehe folgendes Beispiel 1).

[1] Bis WTML Version 2.0 wurde mit diesem Konstruktor ein Array mit der im Argument angegebenen Länge angelegt.

Beispiel 1

```
a = new Array();
b = new Array(25);
c = new Array(4,"Peter");
d = [1, "string1"];
```

Erläuterung:

`a` ist ein Array mit keinem Attribut, d.h. mit der Länge 0.

`b` ist ein Array der Länge 1. Das Attribute `b[0]` ist vom Typ `number` und hat den Wert 25.[1]

`c` ist ein Array der Länge 2. Das Attribut `c[0]` ist vom Typ `number` und hat den Wert 4; `c[1]` ist vom Typ `string` und hat den Wert "`Peter`".

`d` ist ein Array der Länge 2. Das Attribut `d[0]` ist vom Typ `number` und hat den Wert 1; `d[1]` ist vom Typ `string` und hat den Wert "`string1`".

Beispiel 2: zwei-dimensionales Array

```
d=new Array();
for (i=0;i<=2;i++) d[i]=new Array();

for (i=0;i<=2;i++) {
   str=i+1 + ".Row: | ";
   for (j=0;j<=4;j++) {
      d[i][j]=i+","+j+" | ";
      str+=d[i][j];
   }
   document.write(str+"<BR>");
}
```

Erläuterung:

Die ersten beiden Zeilen dieses Beispiels definieren ein zweidimensionales Array.
In den darauf folgenden geschachtelten `for`-Schleifen werden den Array-Elementen Werte zugewiesen (jeweils ein String, der die Indizes enthält). Die Werte werden pro Reihe im String `str` zusammengefasst und ausgegeben.

Das Beispiel erzeugt folgende Ausgabe:

```
1.Row: | 0,0 | 0,1 | 0,2 | 0,3 | 0,4 |
2.Row: | 1,0 | 1,1 | 1,2 | 1,3 | 1,4 |
3.Row: | 2,0 | 2,1 | 2,2 | 2,3 | 2,4 |
```

[1] Bis WTML Version 2.0 wurde `b` als ein Array der Länge 25 angelegt.

7.1.2 Attribute

`length`

Das Attribut hat einen ganzzahligen Wert, der die Anzahl der Elemente des Arrays angibt. Durch den Konstruktor wird bereits implizit eine Anfangslänge festgelegt.

Werden neue Indizes durch Zuweisung angelegt, so ist die Länge immer um 1 größer als der größte verwendete Index, da die Indizes bei 0 und nicht bei 1 beginnen.

Bei mehrdimensionalen Arrays werden jeweils nur die Elemente der jeweils ersten Dimension berücksichtigt, da die Array-Objekte weiterer Dimensionen jeweils eigenständige Objekte sind (vgl. im Beispiel unten das zweidimensionale Array `g`).

Beispiel

```
e=new Array();
document.write("Length of array e is: " + e.length + "<BR>");

f=new Array(25);
document.write("Length of array f is: " + f.length + "<BR>");

f[99]="last element";
document.write("New length of array f is: " + f.length + "<BR>");

g=new Array();
for (i=0;i<=2;i++) g[i]=new Array(6);
document.write("Length of array g is: " + g.length + "<BR>");
```

Das Beispiel erzeugt folgende Ausgabe:

```
Length of array e is: 0
Length of array f is: 1
New length of array f is: 100
Length of array g is: 3
```

7.1.3 Methode concat

Die Methode `concat` fügt das aufrufende Array und ein weiteres Array *array* zu einem Ergebnis-Array zusammen. Das aufrufende Array und *array* selbst werden nicht verändert.

`concat(`*array*`)`

Rückgabewert

Array, zusammengesetzt aus dem aufrufenden Array und dem als Argument übergebenen Array.

Parameter

array spezifiziert ein Array, das nach dem aufrufenden Array an das Ergebnis-Array angehängt wird.

Beispiel

```
h=new Array("Maria","Lena","Hilde");
i=new Array("Franz","Oskar","Hans");
j=h.concat(i);
document.write(j.length + " ");
document.write(j);
```

Das Beispiel erzeugt folgende Ausgabe:

```
6 ["Maria","Lena","Hilde","Franz","Oskar","Hans"]
```

7.1.4 Methode equals

Die Methode `equals` vergleicht das aufrufende Array-Objekt mit dem als Argument übergebenen Objekt auf Gleichheit hinsichtlich der Klasse, Attribute und Werte.

Zwei Arrays sind gleich, wenn sie gleich viele Elemente haben und außerdem die Werte von korrespondierenden Array-Elementen (Elementen mit gleichem Index oder gleichem Attributnamen bei assoziativen Arrays) übereinstimmen.

`equals(`*object*`)`

Rückgabewert

Rückgabewert vom Typ `boolean`: `true` bei Gleichheit, `false` bei Ungleichheit

Parameter

object spezifiziert das Objekt, mit dem das aufrufende Objekt verglichen werden soll.

Beispiel

```
myArray1 = new Array("a","b","c");
myArray2 = new Array("a","b","c");

document.write(myArray1.equals(myArray2)? "Arrays gleich": "Arrays verschieden");
```

Das Beispiel erzeugt folgende Ausgabe:

```
Arrays gleich
```

7.1.5 Methode getClassName

Die Methode liefert den Namen der Klasse, der das aufrufende Objekt angehört, als Zeichenkette zurück.

```
getClassName()
```

Rückgabewert

Zeichenkette, die die Klasse des aufrufenden Objekts angibt, hier "Array".

Parameter

keine

Beispiel

```
arr = new Array();
arrClass = arr.getClassName(); // arrClass ist vom Typ String und
                               // enthält den Klassennamen "Array"
```

7.1.6 Methode join

Die Methode `join` konvertiert alle Elemente des Arrays zu Strings und fasst sie zu einem String zusammen. Nicht definierte Elemente werden zu Leerstrings konvertiert.

```
join()
join(separator)
```

Rückgabewert

String, der aus den Elementen des aufrufenden Array-Objekts gebildet ist.

Parameter

separator

spezifiziert einen String-Ausdruck, durch den die einzelnen Elemente getrennt werden sollen. Falls *separator* nicht bereits vom Typ `string` ist, wird es in einen solchen konvertiert. Wird *separator* nicht angegeben, werden die Elemente durch Kommata getrennt (optional).

Beispiel 1

```
h=new Array("Maria","Lena","Hans");
document.write(h.join(" - ") + ".");
```

Das Beispiel erzeugt folgende Ausgabe:

```
Maria - Lena - Hans.
```

Beispiel 2

```
d=new Array();
for (i=0;i<=2;i++) d[i]=new Array();
for (i=0;i<=2;i++) {
   for (j=0;j<=4;j++) {
      d[i][j]=i+":"+j;
   }
}
document.write(d.join("<BR>"));
```

Das Beispiel erzeugt folgende Ausgabe:

```
["0:0","0:1","0:2","0:3","0:4"]
["1:0","1:1","1:2","1:3","1:4"]
["2:0","2:1","2:2","2:3","2:4"]
```

7.1.7 Methode pop

Die Methode `pop` löscht das letzte Array-Element und liefert dessen Wert zurück. Die Methode verändert das aufrufende Objekt.

```
pop()
```

Rückgabewert

Wert des letzten Array-Elements

Parameter

keine

Beispiel

Siehe bei der Methode `push`.

7.1.8 Methode push

Die Methode `push` hängt die angegebenen Elemente *elem1*, *elem2*, usw. an das aufrufende Array an und liefert die neue Länge des Arrays zurück. Die Methode verändert das aufrufende Objekt.

```
push(elem1)
push(elem1, elem2)
...
push(elem1, elem2, ...)
```

Rückgabewert

Länge des Arrays nach Aufruf von `push`

Parameter

elem1, *elem2*, ...
spezifizieren die Elemente, die zu dem Array hinzugefügt werden sollen.

Beispiel

```
j=new Array();
k=j.push("a", "b", "c");
l=j.pop();
document.write(k + "<BR>");
document.write(l + "<BR>");
document.write(j);
```

Das Beispiel erzeugt folgende Ausgabe:

```
3
c
["a","b"]
```

7.1.9 Methode reverse

Die Methode `reverse` kehrt die Reihenfolge der Array-Elemente um. Das erste Element wird das letzte Element, und das letzte Element wird das erste Element. Die Methode verändert das aufrufende Objekt.

Wird die Methode `reverse` auf ein Objekt einer aus der `Array`-Klasse abgeleiteten Klasse angewendet, so wird implizit die Vererbung aufgelöst (siehe Beispiel). Der Grund dafür ist, dass diese Methode für alle Elemente des Arrays vor der Ausführung zunächst eine Zuweisung des jeweiligen Werts vom Prototyp an das entsprechende Instanz-Attribut ausführt.

```
reverse()
```

Rückgabewert

Es wird eine Referenz auf das aufrufende Objekt zurückgeliefert.

Parameter

keine

Beispiel

```
// bei "normalen" Arrays:
j1=new Array("a","b","c");
k1=j1.reverse();
document.write("k1: " + k1 + "<BR>");
document.write("j1: " + j1 + "<BR>");

// bei abgeleiteten Klassen:
j2=new Array("a","b","c");
function MyArray() {}
MyArray.prototype = j2;
k2=new MyArray();
k2.reverse();
j2[0]="d";
document.write("k2: " + k2 + "<BR>");
document.write("j2: " + j2);
```

Das Beispiel erzeugt folgende Ausgabe:

```
k1: ["c","b","a"]
j1: ["c","b","a"]
k2: ["c","b","a"]
j2: ["d","b","c"]
```

7.1.10 Methode shift

Diese Methode löscht das erste Array-Element.

```
shift()
```

Rückgabewert

gelöschtes Array-Element

Parameter

keine

Beispiel

siehe bei „Methode unshift“ auf Seite 145.

7.1.11 Methode slice

Diese Methode liefert das Teil-Array von Index *index1* bis Index *index2*-1 des aufrufenden Array als Ergebnis zurück. Die Methode verändert das aufrufende Objekt nicht.

```
slice(index1)
slice(index1, index2)
```

Rückgabewert

Teil-Array von *index1* bis *index2*-1 des aufrufenden Arrays.

Parameter

index1 Index des Elements im aufrufenden Array, das zum ersten Element des Ergebnis-Arrays wird.
Ist *index1* < 0 oder größer als die Länge des Arrays, so wird ein leeres Array zurückgeliefert.

index2 Index des ersten Elements im aufrufenden Array, das im Ergebnis-Array nicht mehr zurückgeliefert wird.

Ist *index2* nicht angegeben, so wird das Array bis zum Ende zurückgeliefert (entspricht `slice(`*index1, callingArray*`.length)`).
Ist *index2* kleiner als *index1*, so wird ein leeres Array zurückgeliefert.
Ist *index2* < 0, so gibt dies den Versatz vom Array-Ende her an.
(entspricht `slice(`*index1, callingArray*`.length+`*index2*-1). Ist dieser Index kleiner als *index1*, so wird ein leeres Array zurückgeliefert.

Beispiel

```
j=new Array("a", "b", "c", "d", "e");
k=j.slice(1,3);
l=j.slice(2,-1);
document.write(j + "<BR>");
document.write(k + "<BR>");
document.write(l + "<BR>");
```

Das Beispiel erzeugt folgende Ausgabe:

```
["a","b","c","d","e"]
["b","c"]
["c","d"]
```

7.1.12 Methode sort

Die Methode sortiert ein Array in aufsteigender Reihenfolge. Das aufrufende Array wird dabei verändert. Wird die Methode ohne Argument aufgerufen, so werden die Werte aller definierten Indizes in Strings konvertiert, und die lexikographische Ordnung wird verwendet (beispielsweise wird „14“ vor „3“ eingeordnet). Undefinierte Array-Elemente werden zu Leerstrings konvertiert und ans Ende des Arrays gestellt (siehe „Beispiel 2“ auf Seite 141).

```
sort()
sort(compareFunction)
```

Rückgabewert

Aufrufendes Array, in aufsteigender Reihenfolge sortiert

Parameter

compareFunction

Soll nach einer anderen Ordnung sortiert werden, so ist der Name einer Funktion anzugeben, die wie folgt definiert ist:
`function compareFunction(a,b){...return ..}`

Der Returnwert steuert den Vergleich:

- Soll `a` vor `b` eingeordnet werden, so muss die Funktion einen numerischen Wert kleiner `0` zurückgeben.
- Soll `b` vor `a` eingeordnet werden, so muss die Funktion einen numerischen Wert größer `0` zurückgeben.
- Spielt es keine Rolle, ob `a` und `b` eine bestimmte Reihenfolge haben, so muss die Funktion `0` zurückgeben.

Um Zahlen an Stelle von Strings zu vergleichen, muss die Vergleichsfunktion lediglich `b` von `a` subtrahieren (siehe „Beispiel 3“ auf Seite 141).

Wird die Methode `sort` auf ein Objekt einer aus der Array-Klasse abgeleiteten Klasse angewendet, so wird implizit die Vererbung aufgelöst (siehe Beispiel 5). Der Grund dafür ist, dass diese Methode für alle Elemente des Arrays vor der Ausführung zunächst eine Zuweisung des jeweiligen Werts vom Prototyp an das entsprechende Instanz-Attribut ausführt.

Beispiel 1

```
myArray=new Array("Pit","Zoe","Adam");
document.write(myArray.sort());
```

Das Beispiel erzeugt folgende Ausgabe:

```
["Adam","Pit","Zoe"]
```

Beispiel 2

```
a=new Array();
a[0]="Zoe";
a[4]="Adam";
b=a.sort();
document.write(b[1]);
```

Das Beispiel erzeugt folgende Ausgabe:

```
Zoe
```

Beispiel 3

```
function compareNumber(a,b) {
   return a-b;
}

myArray=new Array(14,3,9,"-2",-8);
sortedArray=myArray.sort();
document.write(sortedArray + " (ohne Vergleichsfunktion) <BR>");
sortedArray=myArray.sort(compareNumber);
document.write(sortedArray + " (mit Vergleichsfunktion)");
```

Das Beispiel erzeugt folgende Ausgabe. Die Ausgabe zeigt, dass Zahlen numerisch korrekt sortiert werden, falls die Vergleichsfunktion genutzt wird - auch dann, wenn es sich um numerische Strings handelt (`"-2"`). Das liegt daran, dass die Operanden des zweistelligen Minus-Operators auf den Typ `number` konvertiert werden.

```
["-2",-8,14,3,9] (ohne Vergleichsfunktion)
[-8,"-2",3,9,14] (mit Vergleichsfunktion)
```

Beispiel 4

```
function CompareLength(a,b) {
   if (a.length < b.length)
      return -1;
   if (a.length > b.length)
      return 1;
   return 0;
}
myarray=new Array("Sebastian","Eva","Roberta","Mike","Martin","Ute");
sorted_array=myarray.sort(CompareLength);
document.write(sorted_array.join("<BR>"));
```

Das Beispiel erzeugt folgende Ausgabe:

```
Eva
Ute
Mike
Martin
Roberta
Sebastian
```

Beispiel 5

```
a = [5, 7, 4];

function MyArray() {}
MyArray.prototype = a;

b = new MyArray();    // b = [5, 7, 4]
a[3] = 1;
a[1] = 9;            // b = [5, 9, 4, 1] (wg. Vererbung)

b.sort();             // b = [1, 4, 5, 9]

a[1] = 7;             // b = [1, 4, 5, 9] (Vererbung aufgeloest)
```

7.1.13 Methode splice

Diese Methode entfernt ein Teil-Array des aufrufenden Arrays und ersetzt es ggf. durch die optional angegebenen Elemente. Diese Methode verändert das aufrufende Objekt.

```
splice(index, count)
splice(index, count, elem1)
splice(index, count, elem1, elem2)
...
splice(index, count, elem1, elem2, ...)
```

Rückgabewert

Array, das die aus dem aufrufenden Array entfernten Elemente enthält.

Parameter

index Index des Elements im aufrufenden Array, das als erstes Element gelöscht wird.

Ist *index*<0 so wird *index*=0 angenommen.

count Anzahl der Elemente im aufrufenden Array, die gelöscht werden sollen.

Ist *count*=0, so werden keine Elemente des Arrays gelöscht.
Ist *count* größer als restliche Elemente im Array vorhanden sind, werden nur soviel Elemente gelöscht, wie vorhanden sind.
Ist *count* < 0, werden keine Elemente gelöscht.

elem1, *elem2*, ...
(optional) wird mindestens ein Element angegeben, so wird dieses zusammen mit allen weiteren angegebenen Elementen an der Position in das aufrufende Array eingefügt, an der zuvor ggf. Elemente gelöscht wurden.

Beispiel

```
j=new Array("a", "b", "c", "d");
document.write(j + "<BR>");
k=j.splice(2,1,"z");
document.write("entfernt: " + k + "!" + "<BR>");
document.write(j + "<BR>");
```

Das Beispiel erzeugt folgende Ausgabe:

```
["a","b","c","d"]
entfernt: ["c"]!
["a","b","z","d"]
```

7.1.14 Methode toString

Die Methode transformiert das aufrufende Array in einen String, der, durch Kommas getrennt, die Werte der Array-Elemente enthält:

["*element0*", "*element1*", ...]

Um Endlosverkettungen zu vermeiden, beendet die Methode `toString` die Ausgabe bei Rekursion: Die Ausgabe wird dort abgebrochen, wo dieselbe Objektreferenz ein zweites Mal ausgegeben würde. Wegen Rekursion siehe

```
toString()
```

Rückgabewert

String, der, durch Kommas getrennt, die Werte der Array-Elemente des aufrufenden Arrays enthält.

Parameter

keine

Beispiel

```
myArray = new Array("Dies", "ist", "ein", "String");
document.write(myArray.toString());
myArray = new Array("Dies", "ist", "ein", "String");
document.write(myArray.toString());
myArray[4] = new String("und ein Object");
document.write(myArray.toString());
```

Das Beispiel erzeugt folgende Ausgabe:[1]

```
["Dies","ist","ein","String"]
["Dies","ist","ein","String",(new String("und ein Object"))]
```

Ein Beispiel zur Verdeutlichung der Rekursion finden Sie bei der Beschreibung von Object.toString() auf Seite 188.

[1] Bis WTML Version 2.0 fehlten in der Ausgabe die eckigen Klammern [ ].

7.1.15 Methode unshift

Diese Methode fügt die angegebenen Elemente *elem1*, *elem2*, usw. am Anfang des aufrufenden Arrays ein und liefert die neue Länge des Arrays zurück.

```
unshift(elem1)
unshift(elem1, elem2)
...
unshift(elem1, elem2, ...)
```

Rückgabewert

Länge des aufrufenden Arrays nach Ausführung von `unshift`

Parameter

elem1, elem2, ...
Werte, die am Anfang des aufrufenden Arrays eingefügt werden sollen.

Die Methode verändert das aufrufende Objekt.

Wird die Methode `unshift` auf ein Objekt einer aus der Array-Klasse abgeleiteten Klasse angewendet, so wird implizit die Vererbung aufgelöst. Der Grund dafür ist, dass diese Methode für alle Elemente des Arrays vor der Ausführung zunächst eine Zuweisung des jeweiligen Werts vom Prototyp an das entsprechende Instanz-Attribut ausführt.

Beispiel

```
j=new Array();

k=j.unshift("a", "b", "c");
document.write(j + "<BR>");
document.write(k + "<BR>");

l=j.shift();
document.write(l + "<BR>");
document.write(j);
```

Das Beispiel erzeugt folgende Ausgabe:

```
["a","b","c"]
3
a
["b","c"]
```

7.1.16 Methode valueOf

Die Methode `valueOf` liefert eine Referenz auf das aufrufende Array-Objekt zurück.

```
valueOf()
```

Rückgabewert

Referenz auf das aufrufende Array-Objekt.

Parameter

keine

Beispiel

```
myArray = new Array("1","2","3");
document.write(myArray.valueOf()[1]);
```

Das Beispiel erzeugt die folgende Ausgabe:

```
2
```

7.2 Boolean-Klasse

Ein Objekt der Klasse Boolean repräsentiert einen logischen Wert. Vordefinierte Attribute gibt es bei dieser Klasse nicht.

7.2.1 Konstruktoren

```
Boolean(expression)
Boolean()
```

Rückgabewert

Objekt der Klasse Boolean

Parameter

expression

Der Ausdruck *expression* wird ausgewertet und ggf. in den Typ `boolean` umgewandelt. Es wird ein Objekt mit diesem Wert angelegt. Wird der Konstruktor ohne Argument aufgerufen, wird ein Objekt mit dem Wert `false` angelegt.

7.2.2 Methode equals

Die Methode `equals` vergleicht das aufrufende Boolean-Objekt mit dem als Argument übergebenen Objekt auf Gleichheit hinsichtlich Klasse und Wert.

`equals(`*object*`)`

Rückgabewert

Rückgabewert vom Typ `boolean`: `true` bei Gleichheit, `false` bei Ungleichheit

Parameter

object spezifiziert das Boolean-Objekt, mit dem das aufrufende Boolean-Objekt verglichen werden soll.

7.2.3 Methode getClassName

Die Methode liefert den Namen der Klasse, der das aufrufende Objekt angehört, als Zeichenkette zurück.

`getClassName()`

Rückgabewert

Zeichenkette, die die Klasse des aufrufenden Objekts angibt, hier "Boolean".

Parameter

keine

Beispiel 1

```
boolobj = new Boolean();
boolClass = boolobj.getClassName(); // boolClass ist vom Typ string
                                    // und enthält den Klassennamen "Boolean"
```

7.2.4 Methode setValue

Die Methode weist dem aufrufenden Boolean-Objekt einen neuen Wert zu.

```
setValue(value)
```

Rückgabewert

kein Rückgabewert

Parameter

value spezifiziert den neuen Wert für das aufrufende Boolean-Objekt.

7.2.5 Methode toString

Die Methode konvertiert die Boolean-Werte `true` und `false` in die korrespondierenden `string`-Werte `"true"` bzw. `"false"`.

```
toString()
```

Rückgabewert

Die Methode liefert abhängig vom Wert des aufrufenden Boolean-Objekts den String "`true`" oder "`false`" zurück.

Parameter

keine

Beispiel

```
boolVar = new Boolean(true);
document.write(boolVar.toString());
```

Das Beispiel erzeugt folgende Ausgabe:

```
true
```

7.2.6 Methode valueOf

Die Methode liefert abhängig vom Wert des Objekts einen boolschen Wert `true` oder `false` zurück.

```
valueOf()
```

Rückgabewert

boolscher Wert `true` bzw. `false`

Parameter

keine

Beispiel

```
boolVar = new Boolean(true);
document.write(boolVar.valueOf());
```

Das Beispiel erzeugt folgende Ausgabe:

```
true
```

7.3 Date-Klasse

Ein Objekt der Klasse `Date` repräsentiert ein Datum.

7.3.1 Konstruktoren

```
Date()
Date(milliseconds)
Date(year, month, day)
Date(year, month, day, hour)
Date(year, month, day, hour, minute)
Date(year, month, day, hour, minute, second)
```

Rückgabewert

Objekt der Klasse `Date`

Parameter

Wird der Konstruktor ohne Argument verwendet, so wird das neue Objekt mit der aktuellen Zeit initialisiert.

Wird ein Argument (*milliseconds*) verwendet, so wird dieses in den Typ `number` konvertiert und als Anzahl der Millisekunden seit dem 1.1.1970 00:00:00 in Greenwich interpretiert. Mit dieser Zeit wird das neue Objekt initialisiert. Liegt der angegebene Zeitpunkt außerhalb des erlaubten Bereichs (1970-9999), dann wird das Objekt mit der aktuellen Zeit initialisiert.

Für die Argumente *month*, *day*, *hour*, *minute* `und` *second* gelten folgende Gültigkeitsbereiche:

month: 0–11
day: 1–31
hour: 0–23
minute, *second*: 0–59

Werden drei oder mehrere Argumente angegeben, so werden diese in den Typ `number` konvertiert und als Angabe der lokalen Zeit in Jahr, Monat, Tag, Stunde, Minute, Sekunde interpretiert. Für fehlende Argumente wird 0 angenommen.

7.3.2 Methode equals

Die Methode `equals` vergleicht das aufrufende Date-Objekt mit dem als Argument übergebenen Objekt auf Gleichheit hinsichtlich der Klasse, Attribute und Werte.

`equals(`*object*`)`

Rückgabewert

Rückgabewert vom Typ `boolean`: `true` bei Gleichheit, `false` bei Ungleichheit

Parameter

object spezifiziert das Objekt, mit dem das aufrufende Objekt verglichen werden soll.

7.3.3 Methode getClassName

Die Methode liefert den Namen der Klasse, der das aufrufende Objekt angehört, als Zeichenkette zurück.

`getClassName()`

Rückgabewert

Zeichenkette, die die Klasse des aufrufenden Objekts angibt, hier "Date".

Parameter

keine

7.3.4 Methode get...

```
getDate()
getDay()
getHours()
getMinutes()
getMonth()
getSeconds()
getYear()
```

Rückgabewerte

`getDate()` liefert den Tag des Monats.

`getDay()` liefert den Tag der Woche (Null-basiert, d.h. Sonntag = 0, Montag=1, etc.).

`getHours()` liefert die Stunde.

`getMinutes()` liefert die Minute.

`getSeconds()` liefert die Sekunde.

`getMonth()` liefert den Monat (Null-Basiert, Januar = 0, Februar =1, etc).

`getYear()` liefert das Jahr

Die Methoden liefern den Wert des jeweils zugehörigen Attributs in lokaler Zeit zurück.

Parameter

keine

7.3.5 Methode getTimezoneOffset

Die Methode zeigt an, welcher Zeitunterschied zwischen der lokalen Zeit und GMT besteht. Der Unterschied wird in Minuten angegeben.

```
getTimezoneOffset()
```

Rückgabewert

Wert vom Datentyp `number` mit der Anzahl Minuten, die der Unterschied beträgt.

Parameter

keine

Beispiel

```
christmasTime = new Date(2002, 11, 25);
output = 'Weihnachten 2002 hat einen Unterschied zu GMT von ';
output += christmasTime.getTimezoneOffset();
output += ' Minuten!';
document.writeln(output);
```

Das Beispiel erzeugt in der mitteleuropäischen Zeitzone folgende Ausgabe:

```
Weihnachten 2002 hat einen Unterschied zu GMT von -60 Minuten!
```

7.3.6 Methode set...

```
setDate(Tag)
setHours(Stunde)
setMinutes(Minute)
setMonth(Monat)
setSeconds(Sekunde)
setYear(Jahr)
```

Die Methoden setzen den Wert des jeweils zugehörigen Attributs auf den übergebenen Wert.

Rückgabewert

keine Rückgabewert

Parameter

Ist der Wert für die Argumente *Stunde*, *Minute*, *Sekunde* und *Monat* nicht gültig, so wird 0 angenommen. Ist der Wert für das Argument *Tag* nicht gültig, so wird 1 angenommen. Es wird nicht geprüft, ob *Tag* für den entsprechenden *Monat* gültig ist. Liegt das angegebene *Jahr* außerhalb des erlaubten Bereichs (1970-9999), dann wird das Objekt nicht verändert.

Das Argument ist in lokaler Zeit anzugeben.

7.3.7 Methode toGMTString

Die Methode liefert für das aufrufende `Date`-Objekt einen String der Form `"Mon, Dec 20 1999 17:23:45 GMT"` zurück. Dabei werden Datum und Uhrzeit ggf. aus der lokalen Zeit in GMT umgerechnet.

```
toGMTString()
```

Rückgabewert

String, der die Daten des aufrufenden Objekts gemäß Formatierung enthält.

Parameter

keine

7.3.8 Methode toLocaleString

Die Methode liefert für das `Date`-Objekt einen String der Form `"12/20/99 17:23:45"` in der lokalen Zeit zurück. Jahreszahlen kleiner 2000 werden zweistellig, Jahreszahlen ab 2000 werden vierstellig dargestellt.

```
toLocaleString()
```

Rückgabewert

String, der die Daten des aufrufenden Objekts gemäß Formatierung enthält.

Parameter

keine

7.3.9 Methode toString

Die Methode liefert für das `Date`-Objekt einen String der Form `"Mon Dec 20 17:23:45 1999"` in der lokalen Zeit zurück.

```
toString()
```

Rückgabewert

String, der die Daten des aufrufenden Objekts gemäß Formatierung enthält.

Parameter

keine

Beispiel

```
<wtoncreatescript>
<!--
d=new Date();
o=new Object();
o.d=d;
document.writeln('<br>',d.toString());
document.writeln('<br>',o.toString());
//-->
</wtoncreatescript>
```

Das Beispiel erzeugt folgende Ausgabe:

```
Tue Jun 15 20:26:41 2010 {d: (new Date(1276626401171))}
```

7.3.10 Methode valueOf

Die Methode liefert für das `Date`-Objekt einen Wert vom Datentyp `number` zurück. Der Wert beziffert die Anzahl von Millisekunden seit 01.01.1970 00:00:00h in Greenwich.

```
valueOf()
```

Rückgabewert

Wert vom Datentyp `number`, der den Wert des aufrufenden Date-Objekts repräsentiert.

Parameter

keine

7.4 Document-Klasse

Die Klasse `Document` erlaubt den Zugriff auf den HTML-Ausgabestrom bzw. auf Dateien.

Innerhalb von WTScript hat `Document` keine vordefinierten Attribute.

Innerhalb von client-seitigem JavaScript (das im Browser und nicht auf dem WebTransactions-Rechner ausgeführt wird) können Sie selbstverständlich sämtliche von JavaScript definierten Attribute und Methoden des `Document`-Objekts nutzen.

7.4.1 Konstruktor

Eine Instanz `document` der Klasse `Document` steht automatisch zur Verfügung. Mit dieser Instanz kann man auf den HTML-Ausgabestrom zugreifen.

Zusätzlich erlaubt der Konstruktor, Objekte der Klasse `Document` für den Zugriff auf Dateien zu erzeugen.

`Document(`*filename*`)`

Rückgabewert

Instanz der Klasse `Document` für den Zugriff auf Dateien

Parameter

filename

Dateiname der Datei, auf die zugegriffen werden soll. Sie können nur auf Dateien zugreifen, die im Basisverzeichnis liegen. Relative Dateinamen beziehen sich auf das Session-Verzeichnis. Dateien werden im Verzeichnis `Session` gesucht (`tmp/`*sitzungsnummer*). Sie können auch absolute Pfadnamen angeben.

Der Konstruktor legt ein Objekt der Klasse `Document` an. Die angegebene Datei wird **nicht** implizit geöffnet (siehe Abschnitt „Methode open“ auf Seite 162).

7.4.2 Methode clear

Diese Methode löscht den bisherigen Inhalt der Ausgabedatei oder des HTML-Ausgabestroms. Sie liefert nichts zurück.

Diese Methode kann angewendet werden auf den HTML-Ausgabestrom (`document.clear()`) und auf Objekte, die mit `new Document(`*file*`)` erzeugt und mit `open()` zum Schreiben geöffnet wurden (d.h ohne den Parameter `READ`).

```
clear()
```

Rückgabewert

kein Rückgabewert

Parameter

keine

7.4.3 Methode close

Diese Methode schließt die aktuell geöffnete Ausgabedatei und liefert eine Referenz auf das Objekt zurück. Diese Methode kann nur auf solche Objekte angewendet werden, die mit `new Document(`*file*`)` erzeugt worden sind, nicht jedoch auf den HTML-Ausgabestrom.

```
close()
```

Rückgabewert

Referenz auf das aufrufende Objekt

Parameter

keine

7.4.4 Methode equals

Die Methode `equals()` vergleicht das aufrufende `Document`-Objekt mit dem als Argument übergebenen Objekt auf Gleichheit hinsichtlich der Klasse, Attribute und Werte.

`equals(`*object*`)`

Rückgabewert

Rückgabewert vom Typ `boolean`: `true` bei Gleichheit, `false` bei Ungleichheit

Parameter

object spezifiziert das Objekt, mit dem das aufrufende Objekt verglichen werden soll.

7.4.5 Methode getClassName

Die Methode liefert den Namen der Klasse, der das aufrufende Objekt angehört, als Zeichenkette zurück.

`getClassName()`

Rückgabewert

Zeichenkette, die die Klasse des aufrufenden Objekts angibt, hier "Document".

Parameter

keine

7.4.6 Methode open

Diese Methode öffnet die beim Konstruktor angegebene Datei und liefert eine Referenz auf das Objekt. Wird sie ohne expliziten vorherigen Aufruf von `close()` verwendet, dann wird die Methode `close()` implizit angewendet.

```
open()
open(openmode)

openmode ::= "WRITE"[,"APPEND"]|"READ"|"R[EAD]W[RITE]"[,"APPEND"]|"APPEND"
```

Rückgabewert

Referenz auf das `Dokument`-Objekt

Parameter

`WRITE` (Voreinstellung)
Die Datei wird nur zum Schreiben geöffnet. Mit den Methoden `write()` und `writeln()` tragen Sie den Ausgabetext in die Datei ein. Existiert die Datei noch nicht, wird sie erzeugt.

`WRITE, APPEND`
Die Datei wird nur zum Schreiben geöffnet. Mit den Methoden `write()` und `writeln()` tragen Sie den Ausgabetext in die Datei ein. Existiert die Datei noch nicht, wird sie erzeugt. Eine bestehende Datei wird fortgeschrieben.

`READ` Die Datei wird nur zum Lesen geöffnet. Die Datei muss vorhanden sein.

`READWRITE`
Die Datei muss vorhanden sein und wird zum Schreiben geöffnet. Mit den Methoden `write()` und `writeln()` tragen Sie den Ausgabetext in die Datei ein. Der bestehende Inhalt wird beim ersten `write[ln]()` gelöscht.

`READWRITE, APPEND`
Die Datei wird neu angelegt, wenn sie noch nicht existiert. Eine bestehende Datei wird fortgeschrieben.

`APPEND`
Die Datei wird nur zum Schreiben geöffnet. Mit den Methoden `write()` und `writeln()` hängen Sie den Ausgabetext an die Datei an. Existiert die Datei noch nicht, wird sie erzeugt. Eine bestehende Datei wird fortgeschrieben.

7.4.7 Methode read

Die Methode `read()` liest den kompletten Inhalt der Datei, die dem `Document`-Objekt zugeordnet ist und liefert diesen zurück. Diese Methode kann nur auf solche Objekte angewendet werden, die mit `new Document(`*file*`)` erzeugt und mit `open()` mit dem Parameter `READ` oder `READWRITE` zum Lesen geöffnet wurden.

```
read()
```

Rückgabewert

Inhalt der Datei, die dem `Dokument`-Objekt zugeordnet ist.

Parameter

keine

Beispiel

```
file = new Document("../greetings.txt");
file.open("READ");
if (WT_SYSTEM.ERROR == "")
   str = file.read();

document.write(str);
```

7.4.8 Methode valueOf

Die Methode `valueOf()` liefert den Namen der Datei zurück, die dem `Document`-Objekt zugeordnet ist. Wird die Methode auf das vordefinierte `Document`-Objekt angewendet, wird die Zeichenkette `"(null)"` zurückgeliefert.

```
valueOf()
```

Rückgabewert

Name der Datei, die dem `Dokument`-Objekt zugeordnet ist.

Parameter

keine

7.4.9 Methode write / writeln

Diese beiden Methoden geben die Argumente konvertiert in Strings in den Ausgabestrom an den Browser oder in eine geöffnete Ausgabedatei aus. Damit können Sie innerhalb von `OnCreate`- und `OnReceive`-Scripts Ausgaben erzeugen. `writeln` ergänzt zusätzlich einen Zeilenvorschub am Ende. Da HTML jedoch Zeilenvorschübe standardmäßig lediglich als Trennzeichen behandelt, führt `writeln` nur bei vorformatiertem Text zu Zeilenumbrüchen, z.B. innerhalb von `<PRE>`-Tags.

```
write()
write(expression)
write(expression, expression)
...
write(expression, expression, ...)
```

```
writeln()
writeln(expression)
writeln(expression, expression)
...
writeln(expression, expression, ...)
```

Rückgabewert

kein Rückgabewert

Parameter

expression

Ausdruck bzw. Ausdrücke, die als Strings in den Ausgabestrom geschrieben werden sollen.
`write` und `writeln` können mit beliebig vielen Argumenten aufgerufen werden.

Beispiel

```
document.write("Good");
document.write("Morning");
document.write("<BR>");
document.writeln("Good");
document.writeln("Morning");
```

Das Beispiel erzeugt folgende Ausgabe:

(innerhalb von vorformatierten Text:)

```
GoodMorning
Good
Morning
```

(außerhalb von vorformatierten Text:)

```
GoodMorning
Good Morning
```

7.5 Host-Datenobjekt-Klasse

Alle Daten, die WebTransactions von der Host-Anwendung empfängt oder an die Host-Anwendung sendet, werden in Host-Datenobjekten abgelegt (siehe hierzu auch das WebTransactions-Handbuch „Konzepte und Funktionen“).

Host-Datenobjekte werden durch den Empfang von Nachrichten vom Host als Attribute des entsprechenden Kommunikationsobjekts angelegt. Explizite Konstruktoren gibt es nicht.

7.5.1 Methode getClassName

Die Methode liefert den Namen der Klasse, der das aufrufende Objekt angehört, als Zeichenkette zurück.

```
getClassName()
```

Rückgabewert

Zeichenkette, die die Klasse des aufrufenden Objekts angibt, hier "WT_OldHostobject".

Parameter

keine

7.5.2 Methode toString

Die Methode wertet das aufrufende Objekt gemäß des Klassen-Templates aus und liefert das Ergebnis als String zurück.

```
toString()
toString(expression)
```

Rückgabewert

Zeichenkette, die das Ergebnis der Auswertung des Klassen-Templates enthält.

Parameter

expression

Es wird das Klassen-Template *expression*`.clt` verwendet. Sie haben so die Möglichkeit, unabhängig vom jeweiligen Typ des Host-Datenobjekts zusätzliche Klassen-Templates zu definieren und diese gezielt aufzurufen.

Wird kein Argument angegeben, so verwendet WebTransactions das Klassen-Template, das dem Typ des aufrufenden Host-Datenobjekts, entspricht (*type*`.clt`). Welche Typen möglich sind, hängt vom Kommunikationsmodul ab: Bei openUTM wird das Attribut `IOTYPE` ausgewertet, bei OSD und MVS wird auch das Attribut `Type` ausgewertet.

Weitere Informationen zu Klassen-Templates finden Sie im Kapitel „Klassen-Templates (*.clt)“ auf Seite 329.
Der Auswertungsoperator ist im Abschnitt „Auswertungsoperator ##...#“ auf Seite 87 beschrieben.

7.5.3 Methode valueOf

Die Methode `valueOf` liefert eine Referenz auf das Objekt selbst zurück.

```
valueOf()
```

Rückgabewert

Referenz auf das aufrufende Host-Objekt.

Parameter

keine

7.6 Function-Klasse

Die Klasse `Function` dient dazu, Funktionen als Objekte zu definieren und zu behandeln. Sie kennt einen Konstruktor und Attribute, nicht aber Methoden für `Function` Objekte.

Dynamisch erzeugte Funktionen haben gegenüber interpretierten Funktionen zwei Vorteile:

- Sie können schneller ablaufen, weil etliche Operationen nur einmal nötig sind.
- Der Funktionsrumpf kann dynamisch, d.h. abhängig von vorhandenen Daten, generiert werden.

7.6.1 Konstruktoren

```
Function(body)
Function(arg1, body)
Function(arg1, arg2, body)
...
Function(arg1, ... , argn, body)
```

Rückgabewert

Instanz der Klasse `Function`

Parameter

arg1, ..., *argn*
: Formalargumente der neuen Funktion

body
: String mit WTScript-Anweisungen

Der Konstruktor legt ein neues Objekt der Klasse `Function` mit den optionalen Formalargumenten *arg1* bis *argn* und den Anweisungen in *body* an.

Beispiel

Folgende Funktion soll nicht per fester Definition, sondern über den Konstruktor aufgebaut werden:

```
function f1(no) {
   document.writeln( no.getDay()+' '.substring(0,4),
                     no.getMonth()+' '.substring(0,4),
                     no.getYear()+' '.substring(0,6),
                     no );
}
```

Eine entsprechende Funktion, die über einen Konstruktor aufgebaut ist, könnte dann so aussehen:

```
f1 = new Function("no", "{document.writeln ( no.getDay()+' '.substring(0,4)
                                           + no.getMonth()+' '.substring(0,4)
                                           + no.getYear()+' '.substring(0,6)
                                           , no );}");
```

7.6.2 Attribute

```
arity
caller
prototype
arguments
callee
```

`arity`
: Dieses Attribut enthält die Anzahl der definierten Argumente.

`caller`
: Die lokale Variable `caller` enthält eine Referenz auf die aufrufende Funktion. Sie ist nur im Funktionskörper gültig. Wurde die Funktion aus der obersten Skript-Ebene heraus aufgerufen, so hat `caller` den Wert `null`.

`prototype`
: Das Attribut `prototype` enthält alle Klasseneigenschaften. Diese werden an alle Instanzen weiter vererbt.

`arguments`
: Bei der Abarbeitung des Konstruktors steht mit dem Attribut `arguments` ein Array zur Verfügung, über das auf die Aktual-Parameter der Funktion zugegriffen werden kann.

`callee`
: Das Attribut `callee` enthält eine Referenz auf die Funktion selbst. Es ist nur im Funktionskörper gültig und wird dazu verwendet, rekursive Aufrufe zu realisieren.

Beispiel

Das folgende Beispiel demonstriert die Verwendung der Klasse `Function` zur Definition einer rekursiven Funktion:

```
functStr = "{if (a == 3) { b += a;}
             calleeTyp = typeof callee;
             functCallee = callee;
             functCalleeString = callee.toString();
             if (caller)
             {  callerTyp = typeof caller;
                functCaller = caller;
                functCallerString = caller.toString();
             }
             if (a == 0 || b == 0)
             {  document.writeln(\"<TR><TD>vorzeitiges Funktionsende, weil
                   Argument = 0!</TD></TR>\");
                return 0;
             }
             document.writeln(\"<TR><TD>rekursiver Aufruf von
                 callee!</TD><TD>\" + callee(0, 0) + \"</TD></TR>\");
            return b;
            }";
functPar1   = "a";
functPar2   = "b";
funct01     = new Function (functPar1, functPar2, functStr);
document.writeln("<TABLE>");
document.writeln("<TR><TD>Anzahl Funktionsparameter an funct01():</TD><TD>" +
   funct01.arity + "</TD></TR>");
document.writeln("<TR><TD>Namen der Funktionsparameter:</TD></TR>");
document.writeln("<TR><TD>1. ," + functPar1.toString() + "';</TD><TD>2. ," +
   functPar2.toString() + "';</TD></TR>");
document.writeln("<TR><TD> Jetzt kommen die <BIG>Aufrufe</BIG> von
   funct01()!</TD></TR>");
document.writeln("<TR><TD> funct01(4, 2) liefert:</TD><TD>" + funct01(4, 2) +
   "</TD></TR>");
document.writeln("<TR><TD> funct01(3, 2) liefert:</TD><TD>" + funct01(3, 2) +
   "</TD></TR>");
```

Das Beispiel erzeugt folgende Ausgabe:

```
Anzahl Funktionsparameter an funct01():    2
Namen der Funktionsparameter:
1. ,a';      2. ,b';
Jetzt kommen die Aufrufe von funct01()!
vorzeitiges Funktionsende, weil Argument = 0!
rekursiver Aufruf von callee!    0
funct01(4, 2) liefert:    2
vorzeitiges Funktionsende, weil Argument = 0!
rekursiver Aufruf von callee!    0
funct01(3, 2) liefert:    5
```

7.6.3 Methode equals

Die Methode `equals` vergleicht das aufrufende Function-Objekt mit dem als Argument übergebenen Objekt auf Gleichheit hinsichtlich der Klasse, Attribute und Werte.

`equals(`*object*`)`

Rückgabewert

Wert vom Typ `boolean`: `true` bei Gleichheit, `false` bei Ungleichheit

Parameter

object spezifiziert das Objekt, mit dem das aufrufende Objekt verglichen werden soll.

7.6.4 Methode getClassName

Die Methode liefert den Namen der Klasse, der das aufrufende Objekt angehört, als Zeichenkette zurück.

`getClassName()`

Rückgabewert

Zeichenkette, die die Klasse des aufrufenden Objekts angibt, hier `"Function"`.

Parameter

keine

7.7 Math-Klasse

Die Klasse `Math` dient mathematischen Operationen und lässt keine Instanziierung zu. Deshalb gibt es keinen Konstruktor. Die Methoden sind nur als Klassenmethoden verfügbar, d.h. sie werden direkt mit dem Klassennamen aufgerufen: `Math.`*function()*.

7.7.1 Klassenattribute

Es gibt folgende Klassenattribute:

`E`	die Basis des natürlichen Logarithmus (Eulersche Zahl `e`, ca. 2.718281828)
`LN10`	der natürliche Logarithmus von 10 (ca. 2.302585)
`LN2`	der natürliche Logarithmus von 2 (ca. 0.693147)
`LOG2E`	der 2-er Logarithmus von E (ca. 1.442695)
`LOG10E`	der 10-er Logarithmus von E (ca. 0.434294)
`PI`	der Wert der Zahl π (ca. 3.141592)
`SQRT2`	die Quadratwurzel von 2 (ca. 1.414213)
`SQRT1_2`	die Quadratwurzel von ½ (ca. 0.707106)

7.7.2 Methode abs

Die Methode liefert den Absolutbetrag einer Zahl *number* zurück.

`abs(`*number*`)`

Rückgabewert

Absolutbetrag von *number*

Parameter

number beliebige Gleitkommazahl, deren Absolutbetrag zurückgeliefert werden soll.

7.7.3 Methode acos

Die Methode liefert den Arcus Cosinus einer Zahl *number*.

```
acos(number)
```

Rückgabewert

Liegt das Argument zwischen -1 und +1, dann wird das Ergebnis im Bogenmaß zwischen 0 und +π zurückgeliefert.

Für andere Argumente wird `NaN` zurückgeliefert.

Parameter

number Gleitkommazahl zwischen -1 und +1

7.7.4 Methode asin

Die Methode liefert den Arcus Sinus einer Zahl *number*.

```
asin(number)
```

Rückgabewert

Liegt das Argument zwischen -1 und +1, dann wird das Ergebnis im Bogenmaß zwischen -π/2 und +π/2 zurückgeliefert.

Für andere Argumente wird `NaN` zurückgeliefert.

Parameter

number Gleitkommazahl zwischen -1 und +1

7.7.5 Methode atan

Die Methode liefert den Arcus Tangens einer Zahl *number*.

`atan(`*number*`)`

Rückgabewert

Das Ergebnis (Arcus Tangens) wird im Bogenmaß zwischen -π/2 und +π/2 zurückgeliefert.

Parameter

number Gleitkommazahl zwischen `-Infinity` und `+Infinity`

7.7.6 Methode ceil

Die Methode liefert die kleinste ganze Zahl, die nicht kleiner als das Argument *number* ist.

`ceil(`*number*`)`

Rückgabewert

Ist *number* eine ganze Zahl, so ist das Ergebnis gleich dem Argument. Andernfalls wird die zu *number* nächsthöhere ganze Zahl zurückgeliefert.

Parameter

number beliebige Gleitkommazahl

Beispiel

```
document.writeln(("<BR>ceil(-3.14) ist " + Math.ceil(-3.14) + "!"));
document.writeln(("<BR>ceil(3.14) ist " + Math.ceil(3.14) + "!"));
```

Das Beispiel erzeugt folgende Ausgabe:

```
ceil(-3.14) ist -3!
ceil(3.14) ist 4!
```

7.7.7 Methode cos

Die Methode liefert den Cosinus einer Zahl *number*.

`cos(`*number*`)`

Rückgabewert

Das zurückgelieferte Ergebnis ist ein Wert zwischen -1 und 1.

Parameter

number Gleitkommazahl im Bogenmaß

7.7.8 Methode exp

Die Methode liefert e^{number} (die Potenz von *number* zur Basis e).

`exp(`*number*`)`

Rückgabewert

e^{number} (die Potenz von *number* zur Basis e)

Parameter

number Gleitkommazahl

7.7.9 Methode floor

Die Methode liefert die größte ganze Zahl, die nicht größer als das Argument *number* ist.

```
floor(number)
```

Rückgabewert

Ist *number* eine ganze Zahl, so ist das Ergebnis gleich dem Argument. Andernfalls wird die nächstniedrigere ganze Zahl zurückgeliefert.

Parameter

number beliebige Gleitkommazahl

Beispiel

```
document.writeln(("<BR>floor(-3.14) ist " + Math.floor(-3.14) + "!"));
document.writeln(("<BR>floor(3.14) ist " + Math.floor(3.14) + "!"));
```

Das Beispiel erzeugt folgende Ausgabe:

```
floor(-3.14) ist -4!
floor(3.14) ist 3!
```

7.7.10 Methode log

Die Methode liefert den natürlichen Logarithmus einer Zahl *number*.

```
log(number)
```

Rückgabewert

natürlicher Logarithmus des Arguments *number*. Wenn *number* kleiner als 0 ist, wird `NaN` zurückgeliefert.

Parameter

number Gleitkommazahl, deren natürlicher Logarithmus zurückgeliefert werden soll.

7.7.11 Methode max

Die Methode bestimmt das Maximum zweier Gleitkommazahlen.

max(*number1*, *number2*)

Rückgabewert

Wert des Größeren der beiden Argumente.

Parameter

number1
number2
Gleitkommazahlen, deren Maximum zurückgeliefert werden soll.

7.7.12 Methode min

Die Methode bestimmt das Minimum zweier Gleitkommazahlen.

min(*number1*, *number2*)

Rückgabewert

Wertes des Kleineren der beiden Argumente.

Parameter

number1
number2
Gleitkommazahlen, deren Minimum zurückgeliefert werden soll.

7.7.13 Methode pow

Die Methode liefert die Potenz zweier Gleitkommazahlen.

```
pow(number1, number2)
```

Rückgabewert

Die Methode liefert $number1^{number2}$ (die Potenz von *number2* zur Basis *number1*) als Ergebnis zurück, sofern die Argumente gültig sind. Andernfalls wird `NaN` zurückgeliefert.

Parameter

number1
beliebige Gleitkommazahl

number2
beliebige Gleitkommazahl

7.7.14 Methode random

Die Methode `random()` generiert eine Pseudozufallszahl.

```
random()
```

Rückgabewert

Die Methode `random()` liefert einen Zufallswert vom Datentyp `number` zurück. Der Zufallswert wird von der aktuellen Zeit abgeleitet.

Der Wert ist größer oder gleich 0 und kleiner 1.

Parameter

keine

7.7.15 Methode round

Diese Methode liefert die gerundete ganze Zahl von *number*. Das ist die nächstgelegene ganze Zahl. Der Wert .5 wird immer auf die nächste größere Zahl gerundet.

round(*number*)

Rückgabewert

Gerundete ganze Zahl von *number*

Parameter

number beliebige Gleitkommazahl

Beispiel

```
document.writeln(("<BR>round(0.500) ist " + Math.round(0.5) + "!"));
document.writeln(("<BR>round(0.499) ist " + Math.round(0.499) + "!"));
document.writeln(("<BR>round(-0.500) ist " + Math.round(-0.5) + "!"));
document.writeln(("<BR>round(-0.501) ist " + Math.round(-0.501) + "!"));
```

Das Beispiel erzeugt folgende Ausgabe:

```
round(0.500) ist 1!
round(0.499) ist 0!
round(-0.500) ist 0!
round(-0.501) ist -1!
```

7.7.16 Methode sin

Die Methode liefert den Sinus einer Gleitkommazahl.

```
sin(number)
```

Rückgabewert

Sinus des Arguments *number*. Das Ergebnis ist ein Wert zwischen -1 und 1.

Parameter

number beliebige Gleitkommazahl

7.7.17 Methode sqrt

Die Methode liefert die Quadratwurzel einer Gleitkommazahl *number*.

```
sqrt(number)
```

Rückgabewert

Quadratwurzel von *number,* wenn *number* >=0 ist. Andernfalls wird NaN zurückgeliefert.

Parameter

number Gleitkommazahl >= 0

7.7.18 Methode tan

Die Methode liefert den Tangens einer Gleitkommazahl.

```
tan(number)
```

Rückgabewert

Tangens von *number*

Parameter

number beliebige Gleitkommazahl

7.8 Number-Klasse

Ein Objekt der Klasse `Number` repräsentiert einen numerischen Wert.

7.8.1 Konstruktoren

`Number(`*expression*`)`
`Number()`

Rückgabewert

Objekt vom Typ `Number`

Parameter

expression
: wird ausgewertet und ggf. auf den Typ `number` konvertiert. Es wird ein Objekt mit dem entsprechenden Wert erzeugt.
Ist *expression* nicht angegeben, so wird ein Objekt mit dem Wert 0 erzeugt.

7.8.2 Klassenattribute

`MAX_VALUE`	die größte in WebTransactions darstellbare Zahl
`MIN_VALUE`	die betragsmäßig kleinste positive in WebTransactions darstellbare Zahl
`NaN`	der Wert `NaN` (not a number)

Diese Attribute sind Klassenattribute, Objekte der Klasse `Number` haben diese Attribute nicht.

Beispiel

```
my_number=new Number();
document.write(my_number.NaN); //ausgegeben wird: undefined
document.write(Number.NaN);    //ausgegeben wird: NaN
```

7.8.3 Methode equals

Die Methode `equals()` vergleicht das aufrufende `Number`-Objekt mit dem als Argument übergebenen Objekt auf Gleichheit hinsichtlich Klasse und Wert.

`equals(`*object*`)`

Rückgabewert

Wert vom Typ `boolean`: `true` bei Gleichheit, `false` bei Ungleichheit

Parameter

object spezifiziert das Objekt, mit dem das aufrufende `Number`-Objekt verglichen werden soll.

7.8.4 Methode getClassName

Die Methode liefert den Namen der Klasse, der das aufrufende Objekt angehört, als Zeichenkette zurück.

`getClassName()`

Rückgabewert

Zeichenkette, die die Klasse des aufrufenden Objekts angibt, hier "Number".

Parameter

keine

Beispiel

```
myNumber = new Number();
document.write(myNumber.getClassName());
```

Das Beispiel erzeugt folgende Ausgabe:

```
Number
```

7.8.5 Methode setValue

Die Methode weist dem aufrufenden `Number`-Objekt einen neuen Wert zu.

```
setValue(value)
```

Rückgabewert

kein Rückgabewert

Parameter

value spezifiziert den neuen Wert für das aufrufende `Number`-Objekt.

7.8.6 Methode toString

Die Methode konvertiert den Wert des aufrufenden Objekts in einen String.

```
toString()
```

Rückgabewert

String, der den Wert des aufrufenden Objekts repräsentiert.

Parameter

keine

7.8.7 Methode valueOf

Die Methode liefert den Wert des aufrufenden Objekts als Wert vom Datentyp `number` zurück.

```
valueOf()
```

Rückgabewert

Wert vom Datentyp number, der den Wert des aufrufenden Number-Objekts repräsentiert.

Parameter

keine

7.9 Object-Klasse

Objekte der Klasse Object sind Behälter für benannte Attribute. Sie können also beliebige Attribute anlegen, vordefinierte Attribute gibt es allerdings nicht.

7.9.1 Konstruktoren

`Object()`
`Object(`*expression*`)`

Wird der Konstruktor ohne Angabe von *expression* aufgerufen, so legt er ein Objekt der Klasse `Object` an. Dieses ist ein leerer Behälter, für den nun Attribute angelegt werden können.

Wird der Konstruktor mit einem Ausdruck *expression* aufgerufen, so wird *expression* ausgewertet. Ist der Ausdruck eine Referenz auf ein Objekt, so wird eine neue Referenz auf dieses Objekt erzeugt. Anderenfalls wird abhängig vom Typ des Ausdrucks ein neues Objekt der Klasse `Boolean`, `Number` oder `String` angelegt.

Beispiel

```
a = new Object();       //legt ein neues Objekt der Klasse Object an
b = new Object(false);  //legt ein neues Objekt der Klasse Boolean an
c = new Object(a);      //liefert eine Referenz auf das Objekt a,
                        //kein neues Objekt!
```

Objekte der Klasse `Object` können auch direkt angelegt werden:

`d = {`*attr:wert, attr:wert*`};`

7.9.2 Methode equals

Die Methode `equals()` vergleicht das aufrufende Objekt mit dem als Argument übergebenen Objekt auf Gleichheit hinsichtlich der Klasse, Attribute und Werte.

`equals(`*object*`)`

Rückgabewert

Rückgabewert vom Typ `boolean`: `true` bei Gleichheit, `false` bei Ungleichheit

Parameter

object spezifiziert das Objekt, mit dem das aufrufende Objekt verglichen werden soll.

Beispiel

```
obj01 = new Object();
obj01.num = 42;
obj01.str = "zweiundvierzig";
obj01.bool = true;
obj02 = new Object();
obj02.num = 21+21;
obj02.str = "zweiundvierzig";
obj02.bool = true;
if ( obj01.equals( obj02 ) ) // liefert true zurück
. . .
```

7.9.3 Methode getClassName

Die Methode liefert den Namen der Klasse, der das aufrufende Objekt angehört, als Zeichenkette zurück.

`getClassName()`

Rückgabewert

Zeichenkette, die die Klasse des aufrufenden Objekts angibt, hier "Object".

Parameter

keine

7.9.4 Methode toString

Die Methode `toString()` liefert eine Zeichenkette zurück, in der jedes Attribut mit seinem aktuellen Wert angeführt wird. Wenn ein Attribut ein Objekt vom Typ `function` ist, wird nicht der Wert zurückgeliefert, sondern die Definition der Funktion.

Diese Methode können Sie verwenden, um ein neues Objekt mit identischen Attributwerten zu erzeugen.

Um Endlosverkettungen zu vermeiden, beendet die Methode `toString` die Ausgabe bei Rekursion. Die Ausgabewird dort abgebrochen, wo dieselbe Objektreferenz ein zweites Mal ausgegeben würde.

```
toString()
```

Rückgabewert

Zeichenkette, in der jedes Attribut mit seinem Wert angeführt ist.

Parameter

keine

Beispiel 1: Ausgabe bei Rekursion

```
a = new Object();
b = new Object();
a.str = "String a";
a.b = b;
b.str = "String b";
b.a = a;

document.write( a.toString() );
// Ausgabe ist: {b: {a: {}, str: "String b"}, str: "String a"}
```

Beispiel 2

```
<wtoncreatescript>
<!--

document.write("<br><br><h3>*******Object Test********</h3><br>");

obj = new Object();
obj.b = "hello \"World!\"";
obj.c = new String ("hello World!");
obj.d = new String ("hello \"World\"!");
obj.e = new Number(21+21);
obj.f = new Boolean(true);
obj.g = false;
obj.o = new Object();
obj.o.text="Hallo";
obj.h = new Function("param", "{if (wt_system.abc)
wt_system.abc+=42; else wt_system.abc=0;}");
obj.i = new Document("file");
obj.j = 84/2;
obj.k = new Date(2010,1,1);
obj.l = this_variable_does_not_exist;
obj.m = null;
objString = obj.toString();
document.writeln(objString);
//Speichern des Objekts
file = new Document("../../objsave.txt");
file.open("WRITE");
file.write(objString);
file.close();
//Wiederherstellen des Objekts
file = new Document("../../objsave.txt");
file.open("READ");
newStr= file.read();
eval("neuesObj = " + newStr + ";");
//Jetzt kann auf das neue Objekt zugegriffen werden, wie auf das alte
document.write("<br><h3>*******Zugriff auf neues Objekt********</h3>");
document.write(neuesObj.o.text);
//-->
</wtoncreatescript>
<wtoncreatescript>
<!--
// Jetzt noch ein Beispiel nur für Funktionen
obj1 = new Object();
obj1.f = new Function("param", "{if (wt_system.abc) \
wt_system.abc+=42; else wt_system.abc=0;}");
objString = obj1.toString();
document.write("<br><h3>*******Ausgabe Funktions-Objekt********</h3>");
document.writeln(objString);
```

```
wt_system.abc=42;
neuesObj.h();
document.write("<br><h3>*******Ausgabe Aufruf Funktionsaufruf********</h3>");
document.write(wt_system.abc);
//-->
</wtoncreatescript>
```

Das Beispiel erzeugt folgende Ausgabe:

*********Object Test***********

```
{b: "hello \"World!\"", c: (new String("hello World!")), d: (new
String("hello \"World\"!")), e: (new Number(42)), f: (new Boolean(true)), g:
false, h: function (param){if (wt_system.abc)
wt_system.abc+=42; else wt_system.abc=0;}, i: (new Document("file")), j: 42,
k: (new Date(1264978800000)), l: undefined, m: null, o: {text: "Hallo"}}
```

*********Zugriff auf neues Objekt***********

```
Hallo
```

*********Ausgabe Funktions-Objekt***********

```
{f: function (param){if (wt_system.abc) wt_system.abc+=42; else
wt_system.abc=0;}}
```

*********Ausgabe Aufruf Funktionsaufruf***********

```
84
```

Beispiel 3

```
o=new Object();
  o.s='abc'; o.n=42; o.b=true;
  o.S=new String('ABC'); o.N=new Number(43); o.B=new Boolean(false);
  o.D=new Date();
  o.u=a;          // a ist nicht definiert
  o.nu=null;
  o.O={n:1, m:2};
  o.A=[1,2,,3,,];
  document.writeln(o.toString());
```

Das Beispiel erzeugt folgende Ausgabe:

```
{A: [1,2,,3,,], B: (new Boolean(false)), D: (new Date(1275668431340)), N:
(new Number(43)), O: {m: 2, n: 1}, S: (new String("ABC")), b: true, n: 42,
nu: null, s: "abc", u: undefined}
```

7.9.5 Methode valueOf

Die Methode `valueOf()` des Objekts liefert eine Referenz auf das Objekt selbst zurück.

```
valueOf()
```

Rückgabewert

Referenz des aufrufenden Objekts auf sich selbst.

Parameter

keine

7.10 RegExp-Klasse

Ein Objekt der Klasse `RegExp` enthält die Angabe eines regulären Ausdrucks. Reguläre Ausdrücke sind Suchmuster (Patterns), die dazu dienen, bestimmte Zeichenkombinationen in Strings zu finden.

7.10.1 Konstruktoren

```
RegExp()
RegExp(expression)
RegExp(expression, mode)
variable = RegExpLiteral
```

Rückgabewert

Objekt vom Typ `RegExp`

expression
: Ausdruck vom Typ `string`, der den regulären Ausdruck angibt (siehe Abschnitt „Literale für reguläre Ausdrücke“ auf Seite 40).

mode
: Ausdruck vom Typ `string`, der den Modus angibt:

  `i` (ignore) Groß-/Kleinschreibung ignorieren.

  `g` (global) alle passenden Stellen finden.

  `ig` oder `gi` Groß-/Kleinschreibung ignorieren und alle passenden Stellen finden.

RegExpLiteral
: Die Zuweisung eines Literals für einen regulären Ausdruck legt ebenfalls ein Objekt der Klasse `RegExp` an.

Beachten Sie, dass bei der Konstruktor-Funktion `RegExp` ein String angegeben wird, während bei der unmittelbaren Zuweisung mit *RegExpLiteral* ein regulärer Ausdruck anzugeben ist (siehe Beispiele auf der nächsten Seite). String-Literale sind in Anführungszeichen zu setzen, RegExp-Literale dagegen haben Schrägstriche als Begrenzungszeichen.

Gegenschrägstriche haben in String-Literalen außer bei Escape-Sequenzen keine Bedeutung, d.h. `"\char"` bedeutet das Gleiche wie `"char"`; z.B. sind `"a*"` und `"a\*"` gleichwertig (siehe Abschnitt „Escape-Sequenzen in Strings“ auf Seite 37).

In `RegExp`-Literalen dagegen entwertet der Gegenschrägstrich das darauffolgende Metazeichen. So können Sie beispielsweise mit `/a\*/` nach der Zeichenfolge `"a*"` suchen, aber nicht nach einer Folge von `a`'s. Wollen Sie ein solches Suchmuster mit der Konstruktor-Funktion definieren, müssen Sie als Argument die Zeichenkette `"a\\*"` angeben.

Sollen in String-Literalen Metazeichen verwendet werden, die aus zwei Zeichen bestehen von denen das erste ein Gegenstrich ist (z.B. `\d` als Metazeichen für eine beliebige Ziffer), so muss der Gegenschrägstrich doppelt angegeben werden, da ja in String-Literalen `\`*`char`* und *`char`* i.a. gleichbedeutend sind. `/\d/` ist also gleichbedeutend mit `new RegExp("\\d")`.

Beispiele

Die Zuweisung des Literals erzeugt jeweils den gleichen regulären Ausdruck wie der darauf folgende Aufruf der Konstruktor-Funktion:

```
re_1 = /abc/i;
re_1 = new RegExp("abc","i");

re_2 = /ab+c/;
re_2 = new RegExp("ab\+c");

re_3 = /\w+/;
re_3 = new RegExp("\\w+");

re_4 = /\\d/;
re_4 = new RegExp("\\\\d");
```

7.10.2 Attribute von Objekten der Klasse RegExp

Jedes Objekt der Klasse `RegExp` hat nach dem Anlegen die folgenden Attribute:

Attribut	Datentyp	Bedeutung
`source`	`string`	Gibt das Suchmuster an, das als Quelle angegeben war, jeweils ohne umgebende Schrägstriche und ohne Flags (i oder g).
`ignoreCase`	`boolean`	Gibt an, ob das Flag i für Groß-Kleinschreibung ignorieren gesetzt ist (`true`) oder nicht (`false`).
`global`	`boolean`	Gibt an, ob das Flag g für globale Suche gesetzt ist (`true`) oder nicht (`false`).
`lastIndex`	`number`	Index des Zeichens, ab dem die nächste Suche beginnt. `lastIndex` wird nur bei globaler Suche gesetzt. Vor der ersten Suche ist `lastIndex=0`. Wurde bereits eine Suche durchgeführt und soll bei der erneuten Suche wieder am Anfang des Strings begonnen werden, so setzen Sie `lastIndex` vorher auf `0` zurück. Für `lastIndex` gelten die unten aufgeführten Regeln.

Die Attribute `source` und `ignoreCase` können Sie nicht durch direkte Zuweisung eines neuen Wertes ändern. Verwenden Sie hierfür die Methode `compile` (siehe Abschnitt „Methode compile“ auf Seite 197).

Regeln für lastIndex

Für `lastIndex` gelten die folgenden Regeln:

– Ist `lastIndex` größer als die Länge des zu durchsuchenden Strings, so liefert die Methode *regexp*`.test` den Wert `false` und `regexp.exec` den Wert `null`. `lastIndex` wird auf `0` gesetzt.

– Ist `lastIndex` gleich der Länge des zu durchsuchenden Strings, und der reguläre Ausdruck entspricht einem Leerstring, dann liefert die Methode *regexp*`.test` den Wert `true` und *regexp*`.exec` ein Ergebnis-Array. `lastIndex` bleibt unverändert.

– Ist `lastIndex` gleich der Länge des zu durchsuchenden Strings und der reguläre Ausdruck entspricht **nicht** einem Leerstring, so liefert *regexp*`.test` den Wert `false` und *regexp*`.exec` den Wert `null`. `lastIndex` wird auf 0 gesetzt.

– In allen anderen Fällen wird `lastIndex` auf die Indexposition gesetzt, die auf die zuletzt gefundene Entsprechung folgt.

7.10.3 Vordefiniertes Objekt RegExp

In jedem Template ist ein vordefiniertes Objekt `RegExp` verfügbar, in dessen Attributen WebTransactions nach folgenden Methodenaufrufen Teilergebnisse ablegt:

– `RegExp`-Methodenaufrufe `exec` und `test`
– `String`-Methodenaufrufe `match` und `replace`

Dieses vordefinierte Objekt hat zwar den Namen `RegExp`, gehört aber zur Klasse `Object`. Es hat folgende Attribute (alle vom Typ `string`):

Attribut	Bedeutung
`lastMatch`	Zeichenfolge der zuletzt gefundenen Entsprechung (= Fundstelle)
`leftContext`	Teilzeichenkette links der Fundstelle
`rightContext`	Teilzeichenkette rechts der Fundstelle
`$1,$2..$9`	In $$n$ ist diejenige Zeichenfolge der Fundstelle abgelegt, die dem n-ten geklammerten Teilausdruck des Suchmusters entspricht (siehe „Beispiel 2“ auf Seite 216) - vorausgesetzt das Suchmuster enthält nicht mehr als neun geklammerte Teilausdrücke. Ein Suchmuster kann zwar beliebig viele geklammerte Teilausdrücke enthalten, das vordefinierte Objekt `RegExp` jedoch kann sich nur neun „merken“. Hat das Suchmuster mehr als neun geklammerte Teilausdrücke, dann sind in $1-$9 die Entsprechungen der letzten neun geklammerten Teilausdrücke abgelegt. Auf die Entsprechungen aller geklammerten Teilausdrücke können Sie über die Indizes des jeweiligen Ergebnis-Arrays zugreifen.
`lastParen`	In `lastParen` ist diejenige Zeichenfolge der Fundstelle abgelegt, die dem letzten geklammerten Teilausdruck des Suchmusters entspricht.
`input`	Default-String, der verwendet wird, falls beim Aufruf der RegExp-Methoden `exec` oder `test` kein zu durchsuchender String angegeben ist. Falls Sie das Attribut `input` verwenden wollen, weisen Sie dem Attribut einen String-Ausdruck zu, der das Default-Suchmuster spezifiziert.

7.10.4 Methode compile

Die Methode `compile` übersetzt ein Suchmuster, das durch einen String-Ausdruck angegeben ist. Die Argumente entsprechen denen der Konstruktor-Funktion.

```
compile(expression)
compile(expression,mode)
```

Rückgabewert

Referenz des aufrufenden Objekts auf sich selbst.

Parameter

expression
: Ausdruck vom Typ `string`, der den regulären Ausdruck angibt (siehe Abschnitt „Literale für reguläre Ausdrücke“ auf Seite 40).

mode Ausdruck vom Typ `string`, der den Modus angibt:

`i` (ignore) Groß-/Kleinschreibung ignorieren.

`g` (global) alle passenden Stellen finden.

`ig` oder `gi`
: Groß-/Kleinschreibung ignorieren und alle passenden Stellen finden.

Mit der Methode `compile` können Sie einem vorhandenen `RegExp`-Objekt einen neuen regulären Ausdruck zuweisen. Die Argumente entsprechen denen der Konstruktoren. Folgende Attribute des Objekts werden neu gesetzt: `source`, `ignore`, `global` und `lastIndex`. Der reguläre Ausdruck wird in übersetzter Form am aufrufenden Objekt hinterlegt.

7.10.5 Methode equals

Die Methode `equals()` vergleicht das aufrufende Objekt mit dem als Argument übergebenen Objekt auf Gleichheit hinsichtlich der Klasse, Attribute und Werte.

`equals(`*object*`)`

Rückgabewert

Rückgabewert vom Typ `boolean`: `true` bei Gleichheit, `false` bei Ungleichheit

Parameter

object spezifiziert das Objekt, mit dem das aufrufende Objekt verglichen werden soll.

7.10.6 Methode exec

Die Methode `exec` durchsucht den angegebenen String entsprechend dem aufrufenden regulären Ausdruck

Jeder Aufruf der Methode `exec` liefert höchstens eine Entsprechung:

– Ist das Flag `global` des regulären Ausdrucks nicht gesetzt (d.h. *regexp*`.global=false`), so liefert die Methode `exec` immer die erste Entsprechung. In diesem Fall entspricht der Aufruf *regExp*`.exec(`*searchStr*`)` dem Aufruf *searchStr*`.match(`*regExp*`)`.

– Ist das Flag `global` des regulären Ausdrucks gesetzt (`=true`) und wird `exec` mehrfach hintereinander aufgerufen, so wird jeweils die nächste Entsprechung gefunden. Die Suche beginnt immer hinter der letzten Fundstelle. Auch der Linkskontext beginnt jeweils hinter der vorangegangenen Fundstelle und nicht am Anfang der Zeichenkette.

Ist das Flag `i` des regulären Ausdrucks gesetzt (`=true`), wird bei dem Vergleich Groß-/Kleinschreibung ignoriert.

---

regexp`.exec()`
regexp`.exec(`*string*`)`

Rückgabewert

Falls eine Entsprechung gefunden wird, liefert `exec` ein Ergebnis-Array zurück und belegt die Attribute des vordefinierten Objekts `RegExp` mit den aktuellen Werten. Wenn das Flag `g` gesetzt ist (d.h. *regexp*`.gloabl=true`), wird auch das Attribut `lastIndex` des aufrufenden Objekts *regexp* neu gesetzt.

Falls keine Entsprechung gefunden wird, so wird `null` zurückgeliefert und die Attribut-Werte des vordefinierten Objekts `RegExp` werden gelöscht.

Parameter

regexp regulärer Ausdruck. Dieser kann der Name eines Objekts oder ein Literal sein.

string (optional) spezifiziert den String-Ausdruck, der durchsucht werden soll. Falls *string* nicht bereits vom Typ `string` ist, wird es in einen solchen konvertiert. Fehlt die Angabe von *string* wird `RegExp.input` verwendet.

Ergebnis-Array der Methode exec

Die Methode `exec` liefert ein Array mit folgenden Attributen zurück:

`[0]`	zum regulären Ausdruck passende Teilzeichenkette von *string*
`[n]`	konkreter Inhalt der *n*=`1,2,...` -ten Klammer des regulären Ausdrucks
`index`	Index des ersten Zeichens der Fundstelle
`input`	Zeichenkette *string*

Beispiel: Aufruf von exec zunächst ohne, dann mit g-Flag

Der folgende reguläre Ausdruck sucht nach doppelt auftretenden Buchstaben.

```
regTest = /((\w)\2)/i;
```

Die äußere Klammer ist nicht unbedingt nötig, macht aber das Suchmuster übersichtlicher. Das Wort „Dampfschifffahrtsgesellschaft“ soll durchsucht werden:

```
text   = "Dampfschifffahrtsgesellschaft";
result = regTest.exec(text);
```

Wir erhalten das folgende Ergebnis :

– Attribute des RegExp-Objekts `regTest`:

```
global: false
ignoreCase: true
lastIndex: 11
source: "((\w)\2)"
```

– Attribute des Ergebnis-Arrays `result`:

```
index: 9
input: "Dampfschifffahrtsgesellschaft"
0: "ff"
1: "ff"
2: "f"
```

– Attribute des vordefinierten Objekts `RegExp`:

```
$1: "ff"
$2: "f"
lastMatch: "ff"
lastParen: "f"
leftContext: "Dampfschi"
rightContext: "fahrtsgesellschaft"
```

Ruft man `regTest.exec(text)` wiederholt auf, erhält man immer das gleiche Ergebnis, da das Flag `g` nicht gesetzt ist.

Definiert man dagegen für **globale Suche**:

```
regTest = /((\w)\2)/ig;
```

so liefert nur der erste Aufruf von `exec` das oben gezeigte Ergebnis. Nach dem zweiten Aufruf erhält man:

– Attribute des RegExp-Objekts `regTest`:

```
global: true
ignoreCase: true
lastIndex: 23
source: "((\w)\2)"
```

– Attribute des Ergebnis-Arrays `result`:

```
index: 21
input: "Dampfschifffahrtsgesellschaft"
0: "ll"
1: "ll"
2: "l"
```

– Attribute des vordefinierten Objekts `RegExp`:

```
$1: "ll"
$2: "l"
lastMatch: "ll"
lastParen: "l"
leftContext: "fahrtsgese"
rightContext: "schaft"
```

und beim dritten Aufruf:

– Attribute des RegExp-Objekts `regTest`:

```
global: true
ignoreCase: true
lastIndex: 0
source: "((\w)\2)"
```

- `result = null`

– Attribute des vordefinierten Objekts `RegExp`:

```
lastMatch: ""
lastParen: ""
leftContext: ""
rightContext: ""
```

7.10.7 Methode getClassName

Die Methode liefert den Namen der Klasse, der das aufrufende Objekt angehört, als Zeichenkette zurück.

```
getClassName()
```

Rückgabewert

Zeichenkette, die die Klasse des aufrufenden Objekts angibt, hier "RegExp".

Parameter

keine

7.10.8 Methode test

Die Methode prüft, ob die Zeichenkette *string* eine zum regulären Ausdruck passende Teilzeichenkette hat. Falls das der Fall ist wird `true` sonst `false` zurückgeliefert. Wenn beim aufrufenden Objekt das Flag `g` (global) gesetzt ist, ist beginnt die Suche bei `lastIndex`, sonst am Anfang von *string*. Ist kein Argument angegeben, so wird `RegExp.input` verwendet.

```
test(string)
test()
```

Rückgabewert

Falls die Zeichenkette *string* eine zum regulären Ausdruck passende Teilzeichenkette hat, wird `true`, sonst `false` zurückgeliefert.

Parameter

string

spezifiziert die Zeichenkette, für die überprüft werden soll, ob sie eine Teilzeichenkette enthält, die zum aufrufenden Ausdruck passt. Das Argument *string* wird in den Typ `string` konvertiert.

Beispiel

```
function check(str,re) {
   if(re.test(str))
      result=" contains the pattern: ";
   else
      result=" does not contain the pattern: ";
   document.write(str+result+re.source);
}

check("WebTransactions",/bt/i);
```

Der Aufruf prüft, ob WebTransactions eine Entsprechung zum regulären Ausdruck `/bt/i` enthält. Da das Flag `i` gesetzt ist, wird Groß-/Kleinschreibung ignoriert und eine Entsprechung gefunden. Die Ausgabe zeigt außerdem, dass in der Eigenschaft `re.source` Flags nicht enthalten sind:

```
WebTransactions contains the pattern: bt
```

7.11 String-Klasse

Ein String-Objekt repräsentiert eine Zeichenkette.

Ein String kann als eine Art Zeichen-Array betrachtet werden: Sie können jedes einzelne Zeichen über seinen Index ansprechen.

Der String-Konstruktor bietet die Möglichkeit, die Konvertierung eines Ausdrucks auf den Typ String zu veranlassen. Zusätzliche individuelle Attribute können Sie nur für ein mit Konstruktor erzeugtes String-Objekt anlegen.

7.11.1 Konstruktoren

`String(`*expression*`)`
`String()`

Rückgabewert

Objekt der Klasse `String`

Der Ausdruck *expression* wird ausgewertet und auf den Typ `string` konvertiert. Es wird ein Objekt mit dem entsprechenden Wert erzeugt. Ist *expression* nicht angegeben, so wird ein Objekt mit dem Wert Leerstring erzeugt.

7.11.2 Attribute

`length`
`[`*number*`]`

`length` liefert die aktuelle Länge der Zeichenkette.

Mit dem Indexausdruck `[`*number*`]` können Sie auf einzelne Zeichen der Zeichenkette zugreifen. So liefert z.B. `a[7]` das 8. Zeichen des Stringobjekts `a`.

7.11.3 Methode charAt

Die Methode liefert das Zeichen zurück, das sich im aufrufenden String-Objekt an der durch den Ausdruck *Index* spezifizierten Position befindet.

```
charAt(Index)
```

Rückgabewert

Das Ergebnis wird als Zeichenkette zurückgeliefert. Liegt *Index* nicht zwischen 0 und *callingString*`.length` - 1, so wird eine leere Zeichenkette zurückgeliefert.

Parameter

Index Ausdruck, der im aufrufenden String-Objekt die Position des Zeichens spezifiziert, das zurückgeliefert werden soll.

Beispiel

```
document.write("WebTransactions".charAt(0));    // 1. Zeichen
document.write("WebTransactions".charAt(16));   // Index groesser als
                                                // Stringlaenge
document.write("WebTransactions".charAt(4-1));  // 4. Zeichen
```

Das Beispiel erzeugt folgende Ausgabe:

```
WT
```

7.11.4 Methode charCodeAt

Die Methode liefert den numerischen Code des Zeichens zurück, das sich im aufrufenden String-Objekt an der durch den Ausdruck *n* spezifizierten Position befindet.

```
charCodeAt(n)
```

Rückgabewert

Das Ergebnis wird als Zahl oder numerischer Wert zurückgeliefert. Liegt *n* nicht zwischen 0 und *callingString*`.length` - 1, so wird `NaN` (not a number) zurückgeliefert.

Parameter

n Ausdruck, der im aufrufenden String-Objekt die Position des Zeichens spezifiziert, das zurückgeliefert werden soll.

Beispiel

```
NL = String.fromCharCode(015,012);     // octal
document.writeln("<br>NL[0]=",NL.charCodeAt(0), ',
',"NL[1]=",NL.charCodeAt(1) );
```

Das Beispiel erzeugt folgende Ausgabe:

```
NL[0]=13, NL[1]=10
```

7.11.5 Methode concat

Die Methode konkateniert den aufrufenden String mit *string*

```
concat(string)
```

Rückgabewert

Zurückgeliefert wird die konkatenierte Zeichenkette. Das aufrufende String-Objekt und die durch *string* übergebene Zeichenkette werden nicht verändert.

Parameter

string spezifiziert die Zeichenkette, mit der das aufrufende String-Objekt konkateniert werden soll.

Beispiel

```
strObj =  new String ("Good Morning");
str = "Good Bye";
resStr1 = strObj.concat(", Mr. President!");
resStr2 = str.concat(", Mr. President!");
document.write("<BR>" + resStr1);
document.write("<BR>" + resStr2);
```

Das Beispiel erzeugt folgende Ausgabe:

```
Good Morning, Mr. President!
Good Bye, Mr. President!
```

7.11.6 Methode equals

Die Methode `equals()` vergleicht das aufrufende `String`-Objekt mit dem als Argument übergebenen Objekt auf Gleichheit hinsichtlich Klasse und Wert.

`equals(`*object*`)`

Rückgabewert

Rückgabewert vom Typ `boolean`: `true` bei Gleichheit, `false` bei Ungleichheit

Parameter

object spezifiziert das Objekt, mit dem das aufrufende Objekt verglichen werden soll.

7.11.7 Methode fromCharCode

Die Methode liefert einen String zurück, der aus den Kodierungen der als Argument übergebenen numerischen Kodierungen gebildet wird. `fromCharCode()` ist eine statische Methode des Konstruktors `String`.

`String.fromCharCode(`*code1* `[, ` *code2*`, […,] ] );`

Rückgabewert

Das Ergebnis wird als Zeichenkette zurückgeliefert.

Parameter

code1, code2, ...
Numerische Kodierungen, aus denen der Ausgabe-String gebildet werden soll.

Beispiel

```
NL = String.fromCharCode(13,10);   //decimal
NL = String.fromCharCode(0x0d,0x0a);   //hex
NL = String.fromCharCode(015,012);   //octal
WT_SYSTEM.HTTP_HEADER = 'Content-type: text/html'+NL+'Connection:
close'+NL+NL;
```

Das Beispiel erzeugt folgende Ausgabe:

```
HTTP_HEADER:  Content-type: text/html
              Connection: close
```

7.11.8 Methode getClassName

Die Methode liefert den Namen der Klasse, der das aufrufende Objekt angehört, als Zeichenkette zurück.

```
getClassName()
```

Rückgabewert

Zeichenkette, die die Klasse des aufrufenden Objekts angibt, hier "String".

Parameter

keine

7.11.9 Methode indexOf

Die Methode durchsucht das aufrufende String-Objekt von vorn nach hinten nach der Zeichenkette *searchValue*. Wird *fromIndex* angegeben, so beginnt die Suche bei diesem Index, anderenfalls bei 0. Bei der Suche wird Groß- und Kleinschreibung unterschieden.

```
indexOf(searchValue)
indexOf(searchValue, fromIndex)
```

Rückgabewert

Zurückgeliefert wird der Index des ersten Zeichens der dem Anfang nahesten Entsprechung oder -1 wenn der *searchValue* nicht vorkommt.

Parameter

searchValue
: spezifiziert die Zeichenkette, nach der gesucht werden soll.

fromIndex
: spezifiziert im aufrufenden String-Objekt die Position, ab der nach der durch *searchValue* spezifizierten Zeichenkette gesucht werden soll.
Default-Wert: 0

Beispiel

```
str = "The a the b and the c.";
document.write(str.indexOf("the") + ",");
document.write(str.indexOf("the",7));
```

Das Beispiel erzeugt folgende Ausgabe:

```
6,16
```

Da Groß- und Kleinschreibung unterschieden wird, ist der Index der ersten Fundstelle nicht 0 sondern 6. Beginnt die Suche ab Index 7 wird bei der Suche das zweite kleingeschriebene `"the"` gefunden (Index 16).

7.11.10 Methode lastIndexOf

Die Methode durchsucht den aufrufenden String von hinten nach vorn nach der Zeichenkette *searchValue*. Wird *fromIndex* angegeben, so beginnt die Suche bei diesem Index, anderenfalls bei *string*`.length` -1. Bei der Suche wird Groß- und Kleinschreibung unterschieden.

`lastIndexOf(`*searchValue*`)`
`lastIndexOf(`*searchValue*`, `*fromIndex*`)`

Rückgabewert

Index des ersten Zeichens der dem Ende nahesten Entsprechung oder -1 wenn der Suchstring nicht vorkommt.

Parameter

searchValue
: spezifiziert die Zeichenkette, nach der gesucht werden soll.

fromIndex
: spezifiziert im aufrufenden String-Objekt die Position, ab der nach der durch *searchValue* spezifizierten Zeichenkette gesucht werden soll.
Default-Wert: string.length-1

Beispiel

```
str = "The a the b and the c.";
document.write(str.lastIndexOf("the") + ",");
document.write(str.lastIndexOf("the",6) + ",");
document.write(str.lastIndexOf("the",5));
```

Das Beispiel erzeugt folgende Ausgabe:

```
16,6,-1
```

Der Index der dem Ende nahesten Fundstelle ist 16.
6 ist der niedrigste Index, ab dem bei Suche von rechts der String `"the"` gefunden wird.
Dies zeigt, dass Groß- und Kleinschreibung unterschieden wird.

7.11.11 Methode match

Die Methode `match` durchsucht den aufrufenden String gemäß dem in *pattern* angegebenen Suchmuster.

Liefert der Ausdruck *pattern* ein `RegExp`-Objekt, so werden dazu passende Teilzeichenketten des aufrufenden Strings gesucht. Anderenfalls wird `pattern` auf den Typ `string` konvertiert. In diesem Fall wird ein zweiter Parameter *mode* (vom Typ `string`) berücksichtigt.

Ist *pattern* vom Typ `string` und enthält *mode* das Zeichen `g` oder ist *pattern* ein regulärer Ausdruck mit g-Flag (globale Suche), so werden alle zum Muster passenden Teilstrings des aufrufenden Strings in jeweils ein Element des Ergebnis-Arrays eingetragen. Anderenfalls arbeitet `match` wie die Methode `exec` für reguläre Ausdrücke, d.h. es wird nur die erste Entsprechung berücksichtigt.

`match(`*pattern*`)`
`match(`*pattern, mode*`)`

Rückgabewert

Falls eine dem Muster *pattern* entsprechende Teilzeichenkette im aufrufenden String enthalten ist, wird ein Ergebnis-Array zurückgeliefert, das gemäß der obigen Beschreibung gebildet ist. Andernfalls wird `null` zurückgeliefert.

Parameter

pattern
: gibt ein Suchmuster an.
pattern kann ein Ausdruck vom Typ `string` oder ein `RegExp`-Objekt sein (siehe Abschnitt „RegExp-Klasse“ auf Seite 193).

mode
: Ausdruck vom Typ `string`, der den Modus angibt (diese Form der Modus-Angabe wird nur berücksichtigt, wenn `pattern` kein `RegExp`-Objekt ist):

  `i` (ignoreCase) Groß-/Kleinschreibung ignorieren.

  `g` (global) alle passenden Stellen finden.

  `ig` oder `gi`
  Groß-/Kleinschreibung ignorieren und alle passenden Stellen finden.

Beispiel 1 (mit g-Flag)

```
str="Gabi, Alissa, A,Karla, another, Andreas, Martin, Anke, Pit";
re=/(A(\w+))/g;
resultArray=str.match(re);

for (attr in resultArray)
  document.write("resultArry." + attr + ": " + resultArray[attr] + "<BR>");

for (attr in re)
  document.write("re."+attr + ": " + re[attr] + "<BR>");
```

Das Beispiel erzeugt folgende Ausgabe:

```
resultArry.0: Alissa
resultArry.1: Andreas
resultArry.2: Anke
re.global: true
re.ignoreCase: false
re.lastIndex: 0
re.source: (A(\w+))
```

Beispiel 2 (ohne g-Flag)

```
str="Gabi, Alissa, A,Karla, another, Andreas, Martin, Anke, Pit";
re=/(A(\w+))/;
resultArray=str.match(re);

for (attr in resultArray)
  document.write("resultArry." + attr + ": " + resultArray[attr] + "<BR>");

for (attr in re)
  document.write("re."+attr + ": " + re[attr] + "<BR>");

for (attr in RegExp)
  document.write("RegExp." + attr + ": " + RegExp[attr] + "<BR>");
```

Das Beispiel erzeugt folgende Ausgabe:

```
resultArry.0: Alissa
resultArry.1: Alissa
resultArry.2: lissa
resultArry.index: 6
resultArry.input: Gabi, Alissa, A,Karla, another, Andreas, Martin, Anke, Pit
re.global: false
re.ignoreCase: false
re.lastIndex: 12
re.source: (A(\w+))
RegExp.$1: Alissa
RegExp.$2: lissa
RegExp.input:
RegExp.lastMatch: Alissa
RegExp.lastParen: lissa
RegExp.leftContext: Gabi,
RegExp.multiline: false
RegExp.rightContext: , A,Karla, another, Andreas, Martin, Anke, Pit
```

7.11.12 Methode replace

Die Methode ersetzt Teilstrings des aufrufenden Strings durch das Ergebnis des Ausdrucks *ReplaceString*. Diese modifizierte Zeichenkette wird zurückgeliefert, der aufrufende String selbst bleibt unverändert.

`replace(`*Pattern*`, `*ReplaceString*`)`

Rückgabewert

Modifizierte Zeichenkette, die aus dem aufrufenden String durch Ersetzungen hervorgeht.

Parameter

Pattern

Der Ausdruck *Pattern* darf vom Typ `string` oder ein `RegExp` Objekt sein und gibt ein Muster für die Teilzeichenketten an, die gesucht werden sollen.

ReplaceString

Der Ersetzungsstring *ReplaceString* kann auch folgende $-Variablen erhalten, die aus der passenden und zu ersetzenden Teilzeichenkette bzw. deren Kontext ermittelt werden:

`$&`	gesamte passende Teilzeichenkette
`$+`	Entsprechung für letzten Klammerausdruck
`$1,$2,…,$9`	Entsprechung für 1., 2., ... 9. Klammerausdruck
`$``	linker Kontext
`$´`	rechter Kontext

Beispiel 1

```
re = /monday/gi;
str = "Monday morning and monday evening";
newstr = str.replace(re, "tuesday");
document.write(newstr);
```

Bei der Angabe des regulären Literals `re` sind die Flags `g` (global) and `i` (ignoreCase) gesetzt. In diesem Beispiel sind beide Flags notwendig, um die Mondays zu ersetzen. Das Beispiel liefert die Ausgabe: "`tuesday morning and tuesday evening`"

Beispiel 2

```
re = /(\w+)\s(\w+)/;
str = "today not!John Smith";
newstr = str.replace(re, "$2 $1");
document.write(newstr);
```

Als regulärer Ausdruck `re` wird ein Literal angegeben. Dieses sucht das erste Zwischenraum-Zeichen, das zwischen zwei Zeichenfolgen mit jeweils mindestens einem alphanumerischen Zeichen steht. Durch die Klammerung wird erreicht, dass die Zeichenfolge links dieses Zwischenraums ("`today`") in der Eigenschaft $1 und die Zeichenfolge rechts davon ("`not`") in $2 abgelegt wird.

Das Beispiel liefert die folgende Ausgabe: `"not today!John Smith"`.

7.11.13 Methode search

Die Methode durchsucht den aufrufenden String nach einem regulären Ausdruck.

`search(`*pattern*`)`

Rückgabewert

Position des ersten passenden Teil-Strings. Falls kein passender Teil-String gefunden wurde, wird -1 zurückgeliefert.

Parameter

pattern

gibt einen regulläreren Ausdruck an, nach dem der aufrufende String durchsucht wird. *pattern* kann ein Ausdruck vom Typ `string` oder ein `RegExp`-Objekt sein (siehe Abschnitt „RegExp-Klasse“ auf Seite 193).

Wenn Sie nur wissen wollen, ob eine bestimmte Zeichenkombination in einem String vorkommt oder nicht, verwenden Sie die Methode `search`. Mehr Informationen liefert die Methode `match()`, die aber auch mehr Ausführungszeit benötigt.

Beispiel

```
function check(str,re) {
   if(str.search(re) != -1)
      result=" contains the pattern: ";
   else
      result=" does not contain the pattern: ";
   document.write(str+result+re.source);
}
check("WebTransactions",/bt/i);
```

Der Aufruf prüft, ob WebTransactions eine Entsprechung zum regulären Ausdruck `/bt/i` enthält. Da das Flag `i` gesetzt ist, wird Groß-/Kleinschreibung ignoriert und eine Entsprechung gefunden. Die Ausgabe zeigt außerdem, dass in der Eigenschaft `re.source` eventuell angegebene Flags nicht enthalten sind:

```
WebTransactions contains the pattern: bt
```

7.11.14 Methode setValue

Die Methode weist dem aufrufenden String-Objekt einen neuen Wert zu.

```
setValue(value)
```

Rückgabewert

kein Rückgabewert

Parameter

value spezifiziert den neuen Wert für das aufrufende String-Objekt.

7.11.15 Methode slice

Diese Methode liefert denjenigen Teil des aufrufenden Strings zurück, der zwischen den beiden angegebenen Indizes liegt.

```
slice()
slice(indexA)
slice(indexA,indexB)
```

Rückgabewert

Teil-String des aufrufenden Strings, der zwischen zwei Indizes liegt (einschließlich des durch den kleineren Index spezifizierten Zeichens aber ausschließlich des durch den größeren Index spezifizierten Zeichens).

Parameter

indexA Ausdruck, der den Index für den Start des Teilstrings angibt.

Ist *indexA* kleiner 0, so wird 0 angenommen.
Ist *indexA* größer als *indexB*, wird ein Leerstring zurückgeliefert.
Ist *indexA* größer als *callingString*`.length`, so wird ein Leerstring zurückgeliefert.
Default-Wert: 0

indexB Ausdruck, der den Index für das Ende des Teilstrings angibt.

Ist *indexB* kleiner 0, so wird für *indexB* ein Versatz vom Stringende angenommen (*callingString*`.length`+*indexB*).
Ist *indexB* größer als *callingString*`.length`, so wird *callingString*`.length` angenommen.
Default-Wert: *callingString*.length
Ist *indexB* nicht angegeben, so wird für *indexB* der Wert *callingString*`.length` angenommen, d.h. es wird der Substring von *indexA* bis zum Ende von *callingString* zurückgeliefert.

Sind die beiden Indizes gleich, wird ein Leerstring zurückgeliefert.

Beispiel

```
document.write("<BR>" + "WebTransactions".slice(0,3));
document.write("<BR>" + "WebTransactions".slice(4,3));
document.write("<BR>" + "WebTransactions".slice(3));
document.write("<BR>" + "WebTransactions".slice(0,-12));
```

Das Beispiel erzeugt folgende Ausgabe:

```
Web

Transactions
Web
```

7.11.16 Methode split

Die Methode gibt ein Array von Teilstrings des aufrufenden String zurück.

```
split()
split(pattern)
split(pattern, limit)
```

Rückgabewert

Ergebnis-Array, das wie folgt gebildet wird: Die Zeichen vom Anfang bis zum ersten Trenner, jeweils zwischen zwei Trennern und vom letzten Trenner bis zum Ende bilden jeweils ein Element des Ergebnis-Arrays. Die Trenner selbst sind im Ergebnis nicht enthalten.

Parameter

pattern Objekt vom Typ `RegExp` oder ein `String`. *pattern* gibt ein Muster für eine Zeichenfolge an, die als Trenner gelten soll.

Fehlt die Angabe für *pattern*, so liefert `split` genau einen Eintrag für den gesamten String.

Ist der Trenner ein Leerstring, so liefert `split` für jedes Zeichen der aufrufenden Zeichenkette einen einzelnen Eintrag.

limit Mit *limit* kann eine maximale Anzahl von Trennungen vorgegeben werden.

Beispiel

```
numberString = ";7.1;687.1;32.634;.56;";
document.write ("input string: " + numberString + "<BR>" );
numberArray = numberString.split(";");
document.write("resulting array: |");
for (i in numberArray)
   document.write(numberArray[i] + "|");
```

Der Aufruf von `split` entfernt die trennenden Strichpunkte und gliedert die Zahlen in ein Array ein. Ausgabe des Beispiels:

```
input string: ;7.1;687.1;32.634;.56;
resulting array: ||7.1|687.1|32.634|.56||
```

7.11.17 Methode substr

Diese Methode liefert denjenigen Teil-String des aufrufenden Strings zurück, der beim angegebenen Index beginnt und die angegebene Länge hat.

```
substr(index)
substr(index,length)
```

Rückgabewert

Teil-String der Länge *length* des aufrufenden Strings, der beim Index *index* beginnt (einschließlich des durch den Index spezifizierten Zeichens).

Parameter

index Ausdruck, der den Index für den Start des Teilstrings angibt.

Ist *index* kleiner 0, so wird für *index* ein Versatz vom Stringende angenommen (*callingString*`.length`+*index*). Liegt dieser Versatz vor dem Anfang des aufrufenden String, so wird 0 verwendet.
Ist *index* größer oder gleich *callingString*`.length`, so wird ein Leerstring zurückgeliefert.

length Ausdruck, der die Länge des Teilstrings angibt.

Ist *length* nicht angegeben, so wird der Substring von *index* bis zum Ende von *callingString* zurückgeliefert.
Ist *length* kleiner oder gleich 0, so wird ein Leerstring zurückgeliefert.

Beispiel

```
document.write("<BR>" + "WebTransactions".substr(3,5));
document.write("<BR>" + "WebTransactions".substr(4,0));
document.write("<BR>" + "WebTransactions".substr(3));
document.write("<BR>" + "WebTransactions".substr(-12,2));
```

Das Beispiel erzeugt folgende Ausgabe:

```
Trans

Transactions
Tr
```

7.11.18 Methode substring

Diese Methode liefert denjenigen Teil des aufrufenden Strings zurück, der zwischen den beiden angegebenen Indizes liegt.

```
substring(indexA)
substring(indexA,indexB)
```

Rückgabewert

Teil-String des aufrufenden Strings, der zwischen den beiden angegebenen Indizes liegt (einschließlich des durch den kleineren Index spezifizierten Zeichens aber ausschließlich des durch den größeren Index spezifizierten Zeichens).

Parameter

indexA, *indexB*
: Ausdrücke, die Indizes spezifizieren.

Ist *indexB* nicht angegeben, so wird für *indexB* der Wert *callingString*`.length` angenommen, d.h. es wird der Substring von *indexA* bis zum Ende von *callingString* zurückgeliefert.

Ist ein Index kleiner 0, so wird 0 angenommen. Ist ein Index größer oder gleich *callingString*`.length`, so wird *callingString*`.length` angenommen.

Sind die beiden Indizes gleich, wird ein Leerstring zurückgeliefert. Ist *indexA* größer als *indexB*, werden die beiden Argumente vertauscht, und dann wird der dazwischenliegende Teil-String zurück gegeben.

Beispiel

```
document.writeln("<BR>" + "WebTransactions".substring(0,3));
document.writeln("<BR>" + "WebTransactions".substring(99,3));
document.writeln("<BR>" + "WebTransactions".substring(0));
document.writeln("<BR>" + "WebTransactions".substring(-99)+"!!!");
```

Das Beispiel erzeugt folgende Ausgabe:

```
Web
Transactions
WebTransactions
WebTransactions!!!
```

7.11.19 Methode toLowerCase

Die Methode gibt den Wert des aufrufenden String zurück, wobei alle Großbuchstaben (A–Z) in die entsprechenden Kleinbuchstaben umgewandelt werden. Der aufrufende String wird nicht verändert.

```
toLowerCase()
```

Rückgabewert

String, der aus dem aufrufenden String durch Umwandlung aller Großbuchstaben (A–Z) in die entsprechenden Kleinbuchstaben hervorgeht.

Parameter

keine

Beispiel

```
a="DaDa";
b=a.toLowerCase();
document.write(b+a);
```

Das Beispiel erzeugt folgende Ausgabe:

```
dadaDaDa
```

7.11.20 Methode toString

Die Methode liefert den Wert des aufrufenden Objekts als Wert vom Typ `string` zurück.

```
toString()
```

Rückgabewert

Wert vom Typ `string`

Parameter

keine

7.11.21 Methode toUpperCase

Die Methode gibt den Wert des aufrufenden String zurück, wobei alle Kleinbuchstaben (`a-z`) in die entsprechenden Großbuchstaben umgewandelt werden. Der aufrufende String wird nicht verändert.

```
toUpperCase()
```

Rückgabewert

String, der aus dem aufrufenden String durch Umwandlung aller Kleinbuchstaben (`a-z`) in die entsprechenden Großbuchstaben hervorgeht.

Parameter

keine

Beispiel

```
a="DaDa";
b=a.toUpperCase();
document.write(b+a);
```

Das Beispiel erzeugt folgende Ausgabe:

```
DADADaDa
```

7.11.22 Methode valueOf

Die Methode liefert den Wert des aufrufenden Objekts als Wert vom Typ `string` zurück.

```
valueOf()
```

Rückgabewert

Wert vom Typ `string`, der dem Wert des aufrufenden Objekt entspricht.

Parameter

keine

7.12 WT_Communication-Klasse

Objekte der Klasse `WT_Communication` werden Kommunikationsobjekte genannt. Diese Kommunikationsobjekte ermöglichen Ihnen das Handling von parallelen Verbindungen und somit die Integration von mehreren Host-Anwendungen innerhalb einer WebTransactions-Anwendung. Die Kommunikationsobjekte enthalten Informationen über das jeweils verwendete Kommunikationsmodul, den aktuellen Zustand der Verbindung, die zuletzt von der Host-Anwendung empfangenen Daten etc. (siehe hierzu auch das WebTransactions-Handbuch „Konzepte und Funktionen“).

7.12.1 Konstruktoren

Die Konstruktoren legen ein neues Kommunikationsobjekt an. Es wird ein Verweis auf dieses Objekt zurückgeliefert, das für eine weitere Referenz benutzt werden kann.

`WT_Communication()`
`WT_Communication(`*handle*`)`

Rückgabewert

Kommunikationsobjekt der Klasse `WT_Communication`

Parameter

handle

Durch die Angabe von *handle* kann der Name des Kommunikationsobjekts angegeben werden. WebTransactions legt das Kommunikationsobjekt im Host-Wurzelobjekt `WT_HOST` an. Zusätzlich wird in diesem neu angelegten Kommunikationsobjekt ein verbindungsspezifisches Systemobjekt angelegt, dessen Attribute zur Steuerung der individuellen Verbindung dienen.

Wird der Konstruktor ohne Argument aufgerufen, so ermittelt WebTransactions den Namen aus dem Attribut `HANDLE` des globalen Systemobjekts. Ist auch das nicht gesetzt, kommt es zu einem Fehler.

7.12.2 Methode close

Die Methode `close` schließt die Verbindung zur Host-Anwendung, löscht das Kommunikationsobjekt aber nicht: Mit einem folgenden `open`-Aufruf kann die Verbindung erneut geöffnet werden. Kommunikationsobjekte leben bis zum Ende der Sitzung.

```
close()
```

Rückgabewert

kein Rückgabewert

Parameter

keine

7.12.3 Methode equals

Die Methode `equals()` vergleicht das aufrufende WT_Communication-Objekt mit dem als Argument übergebenen Objekt auf Gleichheit hinsichtlich der Klasse, Attribute und Werte.

```
equals(object)
```

Rückgabewert

Rückgabewert vom Typ `boolean`: `true` bei Gleichheit, `false` bei Ungleichheit

Parameter

object spezifiziert das Objekt, mit dem das aufrufende Objekt verglichen werden soll.

7.12.4 Methode getClassName

Die Methode liefert den Namen der Klasse, der das aufrufende Objekt angehört, als Zeichenkette zurück.

```
getClassName()
```

Rückgabewert

Zeichenkette, die die Klasse des aufrufenden Objekts angibt, hier `"WT_Communication"`.

Parameter

keine

7.12.5 Methode getModule

Die Methode liefert den Namen des Kommunikationsmoduls in einem String zurück.

```
getModule()
```

Rückgabewert

String, der den Namen des Kommunikationsmoduls enthält

Parameter

keine

7.12.6 Methode open

Die Methode stellt eine Verbindung mit einer Host-Applikation her. Um die Methode `open()` aufrufen zu können, muss das betreffende Kommunikationsobjekt bereits existieren.

Liefert die Methode einen Fehlercode zurück, so wird dieser im globalen Systemobjekt als Attribut `ERROR` abgelegt.

`open()`
`open(`*protocol*`)`

Rückgabewert

Referenz auf Objekt der WT_Communication-Klasse

Parameter

protocol

Durch die Angabe von *protocol* kann einer der zu WebTransactions gebundenen Kommunikationsmodule ausgewählt werden. Wird `open` ohne Argument aufgerufen, so ermittelt WebTransactions den Kommunikationsmodul aus dem Attribut `PROTOCOL` des globalen Systemobjekts. Ist auch dieses Attribut nicht gesetzt, liefert WebTransactions einen Fehler.

7.12.7 Methode receive

Die Methode empfängt vom Host eine Nachricht und hinterlegt diese in dem aufrufenden Objekt. Über die Host-Datenobjekte kann anschließend auf die empfangenen Daten zugegriffen werden.

```
receive()
```

Rückgabewert

Nach erfolgreicher Ausführung der Funktion wird das aufrufende Objekt zurückgeliefert, anderenfalls `null`. Eine entsprechende Fehlermeldung wird im Attribut `ERROR` des globalen Systemobjekts abgelegt.

Parameter

keine

7.12.8 Methode send

Die Methode schickt die in den Host-Datenobjekten des aufrufenden Kommunikationsobjekts hinterlegte Nachricht an den Host.

```
send()
```

Rückgabewert

Nach erfolgreicher Ausführung der Funktion wird das aufrufende Objekt zurückgeliefert, andernfalls `null`. Eine entsprechende Fehlermeldung wird im Attribut `ERROR` des globalen Systemobjekts abgelegt.

Parameter

keine

7.13 WT_Filter-Klasse

Die Klasse `WT_Filter` hat als universelle Filterklasse mehrere Aufgaben:

- die portable Darstellung von Daten für die Kommunikation mit externen Anwendungen über XML-Nachrichten (XML=eXtended Markup Language).

  Damit ist es möglich, Daten aus WTScript-Programmen für die Weiterverarbeitung mit externen Anwendungen in XML-Dokumente zu konvertieren, bzw. Daten, die aus externen Quellen als XML-Dokumente geliefert werden, in WebTransactions weiter zu verarbeiten.

  Eine detaillierte Beschreibung dieser Möglichkeit finden Sie im Abschnitt „XML-Dokumente importieren und exportieren“ auf Seite 389.

- die Konvertierung von WTScript-Datenstrukturen in XML-Dokumente und umgekehrt.

  Damit steht z.B. eine komfortable Möglichkeit zur Verfügung, interne Daten über die Lebensdauer einer Sitzung hinaus zur erhalten, indem diese über den Aufruf von `WT_Filter`-Methoden konvertiert und dann über `document`-Methoden gespeichert werden.

  Eine detaillierte Beschreibung dieses Konzepts finden Sie im Abschnitt „Datenstrukturen exportieren“ auf Seite 394.

- Die Konvertierung von WTScript-Methodenaufrufen in XML-Dokumente und umgekehrt, um auf diese Weise die Kommunikation zwischen WebTransactions-Anwendungen zu ermöglichen.

  Dieses Konzept wird ausführlich im WebTransactions-Handbuch „Client-APIs für WebTransactions“ beschrieben.

- Verarbeitung eines beliebigen XML-Dokumentes mit frei programmierbaren Callback-Funktionen. Ein Beispiel dazu finden Sie auf Seite 245.

Die nachfolgend beschriebenen Methoden stehen Ihnen dazu als Klassenmethoden zur Verfügung.

7.13.1 Methode dataObjectToXML

Die Methode erzeugt aus einem WTScript-Datenobjekt ein XML-Dokument entsprechend der in Abschnitt „Datenstrukturen exportieren“ auf Seite 394 beschriebenen XML-Darstellung.

```
dataObjectToXML(objectPattern)
dataObjectToXML(objectPattern, objectName, ...)
```

Rückgabewert

XML-Dokument, das aus einem WTScript erzeugt wurde.

Parameter

objectPattern

Dieses Argument gibt die Namen von WTScript-Daten an, aus deren Strukturen ein XML-Dokument erzeugt und zurückgeliefert werden soll.

object	Ein beliebiges Objekt. Das Datenobjekt *object* sowie alle seine Attribute. Sind Attribute selbst wieder Objekte, so wird die Konvertierung für diese Objekte rekursiv fortgeführt.
object.	Ein beliebiges Objekt ohne Attribute. Das Datenobjekt *object*. Allerdings werden auf Grund des abschließenden Punkts keine Attribute dieses Objekts konvertiert.
object..	Ein beliebiges Objekt ohne Unterobjekte. Das Datenobjekt *object* sowie alle Attribute dieses Datenobjekts (weil zwischen den beiden folgenden Punkten keine Angabe erfolgt). Der abschließende Punkt sorgt dafür, dass keine Unterattribute von Unterobjekten in *object* konvertiert werden
object..wert	Alle namensgleichen Attribute eine Ebene unterhalb eines Objekts. Alle Attribute mit dem Namen *wert*, die in Objekten direkt unterhalb des Datenobjekts *object* enthalten sind.
object..wert1|wert2	Mehrere namensgleichen Attribute eine Ebene unterhalb eines Objekts. Alle Attribute mit den Namen *wert1* oder *wert2*, die in Objekten direkt unterhalb des Datenobjekts *object* enthalten sind.

Wenn Sie innerhalb einer Methode das aufrufende Objekt konvertieren wollen, dürfen Sie nicht die Zeichenkette "this" als Pattern angeben. Sie können aber eine lokale Variable auf das aufrufende Objekt zeigen lassen und diese verwenden:

```
dummy1 = this;
text=dataObjectToXML("dummy1");
```

Genauso lässt sich ein XML-Dokument auf das aufrufende Objekt entpacken:

```
dummy2=this;
XMLToDataObject(text,"dummy2");
```

objectName

(optional) Mit Hilfe dieser Argumente können die Namen der obersten Ebene überschrieben werden.

Methoden von Objekten werden durch `WT_Filter.dataObjectToXML` nicht mit serialisiert. Lediglich die Namen werden serialisiert und das Wiederherstellen erzeugt nicht mehr aufrufbare (als obsolet bezeichnete) Methoden. `WT_Filter.dataObjectToXML` und `WT_Filter.XMLToDataObject` sind nur für Datenobjekte gedacht.

7.13.2 Methode dataObjectToFormattedXML

Die Methode erzeugt aus einem WTScript-Datenobjekt ein formatiertes XML-Dokument entsprechend der in Abschnitt „Datenstrukturen exportieren“ auf Seite 394 beschriebenen XML-Darstellung.

```
dataObjectToFormattedXML(objectPattern, format)
dataObjectToFormattedXML(objectPattern, format, objectName, ...)
```

Rückgabewert

XML-Dokument, das aus einem WTScript erzeugt wurde.

Parameter

objectPattern

Dieses Argument gibt die Namen von WTScript-Daten an, aus deren Strukturen ein XML-Dokument erzeugt und zurückgeliefert werden soll.

object	Ein beliebiges Objekt. Das Datenobjekt *object* sowie alle seine Attribute. Sind Attribute selbst wieder Objekte, so wird die Konvertierung für diese Objekte rekursiv fortgeführt.
object.	Ein beliebiges Objekt ohne Attribute. Das Datenobjekt *object*. Allerdings werden auf Grund des abschließenden Punkts keine Attribute dieses Objekts konvertiert.
object..	Ein beliebiges Objekt ohne Unterobjekte. Das Datenobjekt *object* sowie alle Attribute dieses Datenobjekts (weil zwischen den beiden folgenden Punkten keine Angabe erfolgt). Der abschließende Punkt sorgt dafür, dass keine Unterattribute von Unterobjekten in *object* konvertiert werden
object..wert	Alle namensgleichen Attribute eine Ebene unterhalb eines Objekts. Alle Attribute mit dem Namen *wert*, die in Objekten direkt unterhalb des Datenobjekts *object* enthalten sind.
object..wert1|wert2	Mehrere namensgleichen Attribute eine Ebene unterhalb eines Objekts. Alle Attribute mit den Namen *wert1* oder *wert2*, die in Objekten direkt unterhalb des Datenobjekts *object* enthalten sind.

Wenn Sie innerhalb einer Methode das aufrufende Objekt konvertieren wollen, dürfen Sie nicht die Zeichenkette "this" als Pattern angeben. Sie können aber eine lokale Variable auf das aufrufende Objekt zeigen lassen und diese verwenden:

```
dummy1 = this;
text=dataObjectToXML("dummy1");
```

Genauso lässt sich ein XML-Dokument auf das aufrufende Objekt entpacken:

```
dummy2=this;
XMLToDataObject(text,"dummy2");
```

format

Zeichenkette mit Formatierungsangaben. Groß- und Kleinschreibung wird nicht unterschieden.

WebTransactions V7.5 unterstützt für *format* aktuell nur den Wert `NL`:

`NL`	Nach jedem Tag-Ende, also nach der schließenden spitzen Klammer (>) wird zusätzlich eine neue Zeile erzeugt. Dadurch ist die XML-Ausgabe besser lesbar und in unterschiedlichen Versionen vergleichbar.

Wenn Sie *format* nicht angeben, besteht die XML-Ausgabe aus einer einzigen Zeile.

objectName

(optional) Mit Hilfe dieser Argumente können die Namen der obersten Ebene überschrieben werden.

Methoden von Objekten werden durch `WT_Filter.dataObjectToXML` nicht mit serialisiert. Lediglich die Namen werden serialisiert und das Wiederherstellen erzeugt nicht mehr aufrufbare (als obsolet bezeichnete) Methoden.
`WT_Filter.dataObjectToXML` und `WT_Filter.XMLToDataObject` sind nur für Datenobjekte gedacht.

Beispiel

```
o=new Object();
  o.s='abc'; o.n=42; o.b=true;
  o.S=new String('ABC'); o.N=new Number(43); o.B=new Boolean(false);
  o.D=new Date();
  o.u=a; // a ist nicht definiert
  o.nu=null;
  o.O={n:1, m:2};
  o.A=[1,2,,3,,];
  xml=WT_Filter.dataObjectToFormattedXML('o','nl');
```

Das Beispiel erzeugt folgende Ausgabe:

```
<data>
<object name="o" class="Object"><object name="A" class="Array"><number name=
"0">1</number>
<number name="1">2</number>
<number name="3">3</number>
</object>
<object name="B" class="Boolean">false</object>
<object name="D" class="Date">1275668587840</object>
<object name="N" class="Number">43</object>
<object name="O" class="Object"><number name="m">2</number>
<number name="n">1</number>
</object>
<object name="S" class="String">ABC</object>
<boolean name="b">true</boolean>
<number name="n">42</number>
<object name="nu" class="Undefined"></object>
<string name="s">abc</string>
<undefined name="u"/></object>
</data>
```

7.13.3 Methode methodCallToXML

Diese Methode erzeugt ein XML-Dokument für den Aufruf einer Methode. Sie wird dazu verwendet, den Aufruf einer WebTransactions-Methode einer anderen WebTransactions-Anwendung für die Übermittlung an die Schnittstelle `WT_REMOTE` vorzubereiten (siehe auch WebTransactions-Handbuch „Client-APIs für WebTransactions“).

```
methodCallToXML(methodName)
methodCallToXML(methodName, argArray)
methodCallToXML(methodName, argArray, codeBase)
```

Rückgabewert

Zeichenkette, die das XML-Dokument für den Aufruf der Methode enthält.

Parameter

methodName
: Name einer Methode

argArray
: Array mit den Argumenten für die Methode (falls erforderlich)

codeBase
: WTML-Dokument mit der Funktionsdefinition in der fernen WebTransactions-Anwendung

7.13.4 Methode objectTreeToXML

Die Methode erwartet als Argument eine WTScript-Datenstruktur, die bestimmten Konventionen genügt, und transformiert diese in ein XML-Dokument, das als Ergebnis zurückgeliefert wird.

`objectTreeToXML(`*xmlObject*`)`

Rückgabewert

XML-Dokument, das aus der übergebenen WTScript-Datenstruktur hervorgegangen ist.

Parameter

xmlObject

WTScript-Datenstruktur gemäß der Beschreibung in Abschnitt „XML-Dokumente importieren und exportieren“ auf Seite 389, bei der das Attribut `child` weggelassen werden kann.

Ist *xmlobject* nicht syntaktisch korrekt, so wird eine Fehlermeldung ausgegeben. In diesem Fall ist das Ergebnis der Methode ein XML-Dokument, das so vollständig wie möglich erzeugt wurde.

7.13.5 Methode XMLToDataObject

Diese Methode ist die Umkehrmethode zu `dataObjectToXML` (siehe Abschnitt „Methode dataObjectToXML“ auf Seite 232). Sie erzeugt aus XML-Text wieder eine entsprechende WT-Script-Datenstruktur (siehe auch Abschnitt „Datenstrukturen exportieren“ auf Seite 394).

Diese Methode arbeitet additiv, das heißt, wenn das in XML serialisierte Objekt vorher bereits existiert, wird es durch `WT_Filter.XMLToDataObject` nicht komplett ersetzt, sondern die im XML-Dokument enthaltenen Attribute werden ergänzt. Andere bereits vorhandene Attribute (und auch Methoden) bleiben unverändert.

`XMLToDataObject(`*xmlObjectText*`[, `*suppressWhitespace*`])`
`XMLToDataObject(`*xmlObjectText*`, `*objectName*`, ...[, `*suppressWhitespace*`])`

Rückgabewert

Einen Rückgabewert im eigentlichen Sinne gibt es nicht. Die Funktion erzeugt das Objekt, das in *xmlObjectText* gesichert wurde (gegebenenfalls unter dem Namen *objectName*).

Parameter

xmlObjectText
: XML-Text, der ein WTScript-Datenobjekt beschreibt.

objectName
: (optional) Mit Hilfe dieser Argumente können die Namen der obersten Ebene überschrieben werden.

suppressWhitespace
: (optional) Behandlung der freien Daten des XML-Dokuments. Dieses Attribut wird gegebenenfalls nach Typ `boolean` konvertiert. Liefert dieses `true`, so werden alle führenden und abschließenden Whitespace-Zeichen des Data-Strings unterdrückt. Defaultwert ist `false`.

Wenn Sie innerhalb einer Methode das aufrufende Objekt konvertieren wollen, dürfen Sie nicht die Zeichenkette "this" als Pattern angeben. Sie können aber eine lokale Variable auf das aufrufende Objekt zeigen lassen und diese verwenden:

```
dummy1 = this;
text=WT_Filter.dataObjectToXML("dummy1");
```

Genauso lässt sich ein XML-Dokument auf das aufrufende Objekt entpacken:

```
dummy2=this;
WT_Filter.XMLToDataObject(text,"dummy2");
```

7.13.6 Methode XMLToMethodCall

Diese Methode interpretiert ein XML-Dokument, das einen Methodenaufruf beschreibt, und führt diesen ggf. aus (siehe auch WebTransactions-Handbuch „Client-APIs für WebTransactions“).

`XMLToMethodCall(`*xmlInvokeText*`)`

Rückgabewert

Rückgabewert der Funktion, die durch das interpretierte XML-Dokument beschrieben ist.

Parameter

xmlInvokeText

Dieses Argument wird in den Typ `string` konvertiert und als XML-Dokument interpretiert. Handelt es sich dabei um einen Methodenaufruf, so wird die Methode ggf. im angegebenen WTML-Dokument gesucht und anschließend ausgeführt. Dabei werden die zuvor beim Aufruf der Umkehrmethode `methodCallToXML` eventuell angegebenen Parameter berücksichtigt.

7.13.7 Methode XMLToObjectTree

Die Methode ist die Umkehrmethode zu `objecTreeToXML` (siehe Abschnitt „Methode objectTreeToXML“ auf Seite 238). Sie erwartet als Argument einen syntaktisch korrekten XML-Text und transformiert ihn in eine entsprechende WTScript-Datenstruktur, die als Ergebnis zurückgeliefert wird (siehe auch Abschnitt „XML-Dokumente importieren und exportieren“ auf Seite 389).

`XMLToObjectTree(`*xmlText*`[, `*suppressWhitespace*`])`

Rückgabewert

WTScript-Datenstruktur, die aus dem übergebenen XML-Text hervorgegangen ist.

Parameter

xmlText
: syntaktisch korrekter XML-Text als String

suppressWhitespace
: (optional) Behandlung der freien Daten des XML-Dokuments. Dieses Attribut wird gegebenenfalls nach Typ `boolean` konvertiert. Liefert dieses `true`, so werden alle führenden und abschließenden Whitespace-Zeichen des Data-Strings unterdrückt. Defaultwert ist `false`.

Ist *xmlText* nicht syntaktisch korrekt oder enthält *xmlText* XML-Entities oder -Verarbeitungsanweisungen, so wird eine Fehlermeldung ausgegeben. In diesem Fall ist das Ergebnis der Methode nur ein Fragment einer WTScript-Datenstruktur, das so weit wie möglich mit der Semantik des XML-Texts übereinstimmt.

7.13.8 Methode XML_SAXParse

Mit dieser Methode kann ein XML-Dokument interpretiert werden. Die Methode analysiert das XML-Dokument und startet bei Erkennen von Start-, Ende-Tags, Daten oder Verarbeitungsanweisungen eine entsprechende Callback-Funktion. Diese Callback-Funktionen müssen in Form von WTScript-Funktionen bzw. -Funktionsobjekten zur Verfügung gestellt werden. Die Funktionen werden der Methode mittels eines Handlerobjekts übergeben.

`XML_SAXParse (`*xmlDocument*`,` *HandlerObject* `[,` *suppressWhitespace*`])`

Rückgabewert

Rückgabewert des integrierten XML-Parsers als Typ `object` mit den Attributen `ErrorCode` vom Typ `number` und `ErrorText` vom Typ `string` mit den folgenden Inhalten:

ErrorCode	ErrorText	Bedeutung
0	XML_ERROR_NONE	Alles ok
1	XML_ERROR_NO_MEMORY	Kein Speicher mehr verfügbar
2	XML_ERROR_SYNTAX	Syntaktischer Fehler im Dokument
3	XML_ERROR_NO_ELEMENTS	Kein Element gefunden
4	XML_ERROR_INVALID_TOKEN	Nicht wohlgeformt
5	XML_ERROR_UNCLOSED_TOKEN	Nicht geschlossenes Token
6	XML_ERROR_PARTIAL_CHAR	Nicht geschlossenes Token
7	XML_ERROR_TAG_MISMATCH	Unpaarigkeit der Tags
8	XML_ERROR_DUPLICATE_ATTRIBUTE	Doppeltes Attribut
9	XML_ERROR_JUNK_AFTER_DOC_ELEMENT	Etwas Unzulässiges nach document-Element
10	XML_ERROR_PARAM_ENTITY_REF	Unzulässige Referenz auf Entität
11	XML_ERROR_UNDEFINED_ENTITY	Nicht definierte Entität
12	XML_ERROR_RECURSIVE_ENTITY_REF	Rekursive Entitäten-Referenz
13	XML_ERROR_ASYNC_ENTITY	Aynchrone Entität
14	XML_ERROR_BAD_CHAR_REF	Referenz auf ungültiges Zeichen
15	XML_ERROR_BINARY_ENTITY_REF	Referenz auf binäre Entität
16	XML_ERROR_ATTRIBUTE_EXTERNAL_ENTITY_REF	Referenz auf externe Entität in Attribut
17	XML_ERROR_MISPLACED_XML_PI	Verarbeitungsanweisungen nicht zu Beginn einer externen Entität
18	XML_ERROR_UNKNOWN_ENCODING	Unbekannte Kodierung
19	XML_ERROR_INCORRECT_ENCODING	Definierte Kodierung ist falsch

ErrorCode	ErrorText	Bedeutung
20	XML_ERROR_UNCLOSED_CDATA_SECTION	CDATA nicht abgeschlossen
21	XML_ERROR_EXTERNAL_ENTITY_HANDLING	Fehler bei der Verarbeitung einer externen Entität

Parameter

xmlDocument

Das zu interpretierende XML-Dokument. Ist dies ein Objekt der Klasse Document, wird der Inhalt der damit bezeichneten Datei interpretiert. Jeder andere Typ wird gegebenenfalls nach Typ `string` konvertiert und dann interpretiert.

HandlerObject

Bekanntgabe der Callback-Funktionen. Dies ist ein Objekt der Klasse Object und hat folgendes Layout:

`StartElementHandler`:
Referenz auf die Callback-Funktion, die die Start-Tags bearbeitet.

`EndElementHandler`:
Referenz auf die Callback-Funktion, die die Ende-Tags bearbeitet.

`CharacterDataHandler`:
Referenz auf die Callback-Funktion, die die Daten bearbeitet.

`ProcessingInstructionHandler`:
Referenz auf die Callback-Funktion, die die Verarbeitungsanweisungen bearbeitet.

Die Attribute sind optional. Sind sie nicht definiert bzw. referenzieren keine Funktion oder ein Funktionsobjekt, wird kein entsprechender Handler gestartet.

suppressWhitespace

(optional) Behandlung der freien Daten des XML-Dokuments. Dieses Attribut wird gegebenenfalls nach Typ `boolean` konvertiert. Liefert dieses `true`, so werden alle führenden und abschließenden Whitespace-Zeichen des Data-Strings unterdrückt. Defaultwert ist `false`.

Prototyp der einzelnen Callback-Funktion

Die Callback-Routinen für den XML-Parser müssen einer festen Schnittstelle genügen. Auf Gültigkeit wird allerdings nicht geprüft.

```
StartTagHandler  ( string tagname, Object[] nameValue[, string namespace] )
```

tagname
: Hier wird der Name des geöffneten Tags übergeben (**ohne** Namespace-Präfix).

nameValue
: In diesem Array werden als einzelne Objekte die Name-Value-Paare der Tag-Attribute übergeben. Die Attributnamen der Objekte sind `name` und `value`.

namespace
: Hier wird der Name des evtl. geltenden Namespaces oder ein Leerstring übergeben.

```
EndTagHandler  ( string tagname[, string namespace] )
```

tagname
: Hier wird der Name des schließenden Tags übergeben (**ohne** Namespace-Präfix).

namespace
: Hier wird der Name des evtl. geltenden Namespaces oder ein Leerstring übergeben.

```
CharacterDataHandler  ( string data )
```

data Hier werden die freien Daten übergeben.

```
ProcessingInstructionHandler  ( string target, string data )
```

target Hier wird der Name der Verarbeitungsanweisung als String übergeben.

data Hier wird die gefundene Verarbeitungsanweisung als String übergeben.

Beispiel

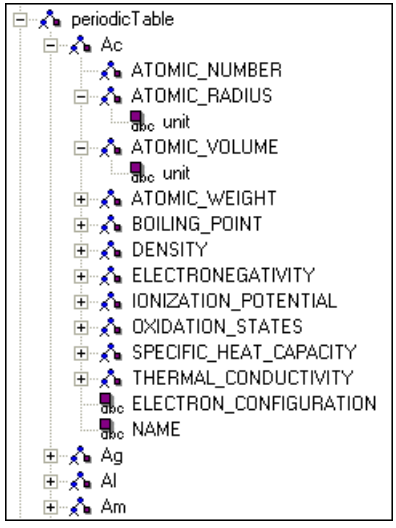
Grundlage für die folgenden Beispiele ist die XML-Datei `allelements.xml`. Diese Datei enthält eine Auflistung aller chemischen Elemente und deren Eigenschaften.

```
<PERIODIC_TABLE>
  <ATOM>
    <NAME>Actinium</NAME>
    <ATOMIC_WEIGHT>227</ATOMIC_WEIGHT>
    <ATOMIC_NUMBER>89</ATOMIC_NUMBER>
    <OXIDATION_STATES>3</OXIDATION_STATES>
    <BOILING_POINT UNITS="Kelvin">3470</BOILING_POINT>
    <SYMBOL>Ac</SYMBOL>
    <DENSITY UNITS="grams/cubic centimeter"><!-- At 300K -->
      10.07
    </DENSITY>
    <ELECTRON_CONFIGURATION>[Rn] 6d1 7s2 </ELECTRON_CONFIGURATION>
    <ELECTRONEGATIVITY>1.1</ELECTRONEGATIVITY>
    <ATOMIC_RADIUS UNITS="Angstroms">1.88</ATOMIC_RADIUS>
    <ATOMIC_VOLUME UNITS="cubic centimeters/mole">
      22.5
    </ATOMIC_VOLUME>
    <SPECIFIC_HEAT_CAPACITY UNITS="Joules/gram/degree Kelvin">
      0.12
    </SPECIFIC_HEAT_CAPACITY>
    <IONIZATION_POTENTIAL>5.17</IONIZATION_POTENTIAL>

    <THERMAL_CONDUCTIVITY UNITS="Watts/meter/degree Kelvin">
    <!-- At 300K -->
      12
    </THERMAL_CONDUCTIVITY>
  </ATOM>

  <ATOM>
    <NAME>Aluminum</NAME>
        …
  …
    </ATOM>
</PERIODIC_TABLE>
```

Das folgende Dokument soll eingelesen werden und in eine WTScript-Datenstruktur umgewandelt werden, die für jedes Element ein Attribut mit dem Symbol des Elements enthält:



Die folgende Funktion wird aufgerufen, wenn ein öffnendes XML-Tag erkannt wird. Es speichert die Attribute des Tags in der globalen Variablen `CurrentAttributes`, damit sie im Endetag-Handler zur Verfügung stehen. Falls eine neue Beschreibung eines chemischen Elements beginnt, wird ein neues WTScript-Objekt erzeugt, das diese Beschreibung repräsentiert.

```
function StartChemie (name,attributes)
{
    CurrentAttributes = attributes;
    CurrentData = "";
    if (name == "ATOM")
        CurrentObject = new Object();
}
```

Die folgende Funktion wird aufgerufen, wenn Daten erkannt worden sind. Sie speichert die Daten für den Endetag-Handler:

```
function DataChemie (data)
{
    CurrentData = data;
}
```

Die folgende Funktion wird aufgerufen, falls ein schließendes XML-Tag erkannt wird. Sie führt je nach Namen des schließenden Tags verschiedene Aktionen aus:

- Wird SYMBOL erkannt, wird der Name aus den zugehörigen Daten zwischengespeichert, um als Attributname in der WTScript-Datenstruktur verwendet zu werden.
- Wird ATOM erkannt, so ist die Beschreibung eines chemischen Elements fertig und kann unter dem Symbolnamen eingefügt werden.
- Alle anderen Elemente werden unter dem Tagnamen in das Objekt eingefügt, das das chemische Element beschreibt.
- Beschreibt das aktuelle Tag numerische Daten, so kann in dem Attribut UNIT die Einheit der Daten angegeben sein. Diese Information wird aus dem in Starttag-Handler zwischengespeicherten Attributen ermittelt und in die Datenstruktur eingefügt.

```
function EndChemie (name)
{
    switch (name)
    {
        case "SYMBOL":
           CurrentName = CurrentData;
            break;
        case "ATOM":
            //End of element description: add whole element to WTScript
            // object
            periodicTable[CurrentName] = CurrentObject;
            delete CurrentObject;
            break;
        case "NAME":
        case "ELECTRON_CONFIGURATION":
           CurrentObject[name] = CurrentData;
           break;
        case "PERIODIC_TABLE":   //end of description: cleanup
            delete CurrentName;
            delete CurrentData;
            delete CurrentObject;
            break;
        default:
           //All the other information about a chemical element contain
           // numerical data and may contain the attribut UNIT, so create
           // as Number object and add the
           // attribute unit to this object if necessary
           CurrentObject[name] = new Number(CurrentData);
           if (CurrentAttributes.length == 1
              && CurrentAttributes[0].name == "UNITS")
              CurrentObject[name].unit = CurrentAttributes[0].value;
        }
```

```
}

// Erzeugen eines Document-Objekts für dir XML-Datei
doc = new Document(WT_SYSTEM.BASEDIR + "/allelements.xml" );

// Handlerobjekt vorbereiten
handlerObject = new Object();
handlerObject.StartElementHandler = StartChemie;
handlerObject.EndElementHandler = EndChemie;
handlerObject.CharacterDataHandler = DataChemie;

//Datenstrukur für die Informationen aus der XML-Datei, hier fügen die Element-
Handler die Information ein.
periodicTable = new Object();

//Filter aufrufen
try {
        WT_Filter.XML_SAXParse( doc, handlerObject,true );
}
catch (exc) {
    document.writeln(exc);
    exitTemplate();
}
```

7.14 WT_LdapConnection-Klasse

Für die Unterstützung des Internet-Kommunikationsprotokolls LDAP (**L**ightweight **D**irectory **A**ccess **P**rotocol) in WebTransactions ist in WTScript die Klasse `WT_LdapConnection` definiert. `WT_LdapConnection` unterstützt nur synchrone Operationen. Erweiterte Operationen werden nicht unterstützt.

Der vorliegende Abschnitt behandelt folgende Themen:

- LDAP-Directory-Service (Überblick)
- LDAP-Fehlermeldungen
- Eingebaute Methoden der Klasse `WT_LdapConnection`
- Beispiele

7.14.1 LDAP-Directory-Service - Überblick

Dieser Abschnitt basiert auf dem SLAPD und SLURPD Administratorhandbuch, einem Dokument der University of Michigan. Das Copyright für das Administratorhandbuch hält die University of Michigan. Die hier vorliegende Beschreibung beschränkt sich auf einen allgemeinen Überblick. Nähere Informationen zu LDAP im BS2000/OSD finden Sie im Administratorhandbuch zu „interNet Services“.

LDAP-Directory-Service für Verzeichnisse (Directories)

Ein Directory (Verzeichnis) ist mit einer Datenbank vergleichbar, die auf Attributen aufbauende, beschreibende Informationen enthält. Die Informationen eines Directory werden im allgemeinen häufiger gelesen als geschrieben. Für Verzeichnisse sind deshalb normalerweise die komplizierten Transaktions- oder Roll-Back-Schemata regulärer Datenbanken für komplexe umfangreiche Aktualisierungen nicht implementiert. Änderungen werden entweder komplett oder gar nicht ausgeführt, vorausgesetzt, sie sind überhaupt erlaubt. Verzeichnisse sind dafür ausgelegt, schnelle Antworten auf umfangreiche Nachschlage- und Suchvorgänge zu ermöglichen. Zur Erhöhung der Verfügbarkeit und Zuverlässigkeit bei einer gleichzeitigen Verringerung der Antwortzeit kann die Information eines Directory verteilt werden. Bei einer verteilten Datenhaltung können temporäre Inkonsistenzen zwischen den Datenbeständen akzeptiert werden, wenn sie am Ende wieder übereinstimmen.

Es gibt verschiedene Möglichkeiten, einen Verzeichnis-Service bereitzustellen. Die unterschiedlichen Methoden ermöglichen die Speicherung von verschiedenen Informationstypen, stellen unterschiedliche Anforderungen daran, wie auf diese Informationen verwiesen wird, wie sie abgefragt und aktualisiert werden, wie sie vor unbefugtem Zugriff geschützt werden usw. Einige Verzeichnis-Services sind lokal und decken mit ihrem Service nur einen eingeschränkten Bereich ab (z.B. der FINGER-Service einer einzelnen Maschine). Andere Services sind global und stellen ihre Dienste einem größerem Umfeld (z.B. dem gesamten Internet) zur Verfügung. Globale Services sind normalerweise verteilt, d.h. die Daten sind

auf viele Maschinen verteilt, und diese Maschinen realisieren den Verzeichnis-Service durch ihre Zusammenarbeit. Typischerweise wird durch einen globalen Service ein einheitlicher Namensbereich festgelegt, der immer dieselbe Sicht auf die Daten bietet, unabhängig, von welchem Standpunkt aus die Betrachtung erfolgt; d.h. ein globaler Directory Service liefert auf eine Anfrage immer das gleiche Ergebnis, unabhängig davon, an welche der beteiligten Maschinen die Anfrage gestellt wird.

LDAP-Funktionalität

LDAP ist ein Directory Service Protocol, das über TCP/IP abgewickelt wird. LDAP ist in RFC 1777 "The Lightweight Directory Access Protocol" definiert.

Der LDAP-Directory-Service arbeitet nach dem Client-Server-Modell. Ein oder mehrere LDAP-Server enthalten die Daten, die den LDAP-Directory Tree bilden. Ein LDAP-Client baut eine Verbindung zu einem LDAP-Server auf und stellt an diesen eine Anfrage. Der Server antwortet mit dem Ergebnis oder mit einem Verweis (typischerweise auf einen anderen LDAP-Server), der angibt, wo der Client weitere Informationen erhalten kann. Unabhängig davon, zu welchem LDAP-Server ein Client eine Verbindung herstellt, sieht er stets dieselbe Verzeichnisstruktur. Ein Name, der einem LDAP-Server angegeben wird, verweist auf denselben Eintrag, auf den er auch verweisen würde, wenn er einem anderen LDAP-Server angegeben würde. Dies ist eine wichtige Funktion eines globalen Verzeichnis-Services wie LDAP.

Das LDAP-Verzeichnis-Servicemodell basiert auf Einträgen (entries). Ein Eintrag besteht aus Attributen und hat einen ausgezeichneten Namen (distinguished name, DN), der den Eintrag eindeutig identifiziert. Jedes der Attribute eines Eintrags verfügt über eine Typbezeichnung und einen oder mehrere Werte. Die Typbezeichnungen sind üblicherweise Abkürzungen wie *cn* für common name oder *mail* für email address. Die Werte werden vom jeweiligen Typ des Attributs bestimmt. Beispielsweise kann ein *mail*-Attribut den Wert *babs@umich.edu* enthalten. Ein *jpegPhoto*-Attribut würde ein Foto im binären JPEG/JFIF-Format enthalten.

In LDAP werden die Einträge in einer hierarchischen Baumstruktur angeordnet, die politische, geografische und/oder organisatorische Grenzen reflektiert. Einträge, die Ländernamen repräsentieren, erscheinen in der Baumstruktur an erster Stelle (oben). Unter diesen stehen die Einträge für Bundesländer oder nationale Organisationen. Darunter können Einträge folgen, die sich auf Personen, organisatorische Einheiten, Drucker, Dokumente oder andere Sachverhalte bzw. Dinge beziehen.

7.14.2 LDAP-Fehlermeldungen

Meldungsnummer	Meldungstext
0300	0300:Fehler: LDAP(%d) - %s
0301	0301:Fehler: LDAP - ungültige Parameter
0302	0302:Fehler: LDAP Initialisierung - Die Initialisierung einer LDAP Sitzung kann nicht durchgeführt werden.
0303	0303:Fehler: LDAP Sitzungskennzeichen - Das Kennzeichen der LDAP Sitzung wurde nicht gefunden.

Die Fehlerbehandlung in WTScript ist beschrieben im Abschnitt „Fehlerbehandlung durch Ausnahmen (Exceptions)“ auf Seite 322. Ein Beispiel zur Behandlung der `ldapError`-Ausnahme finden Sie auf Seite 273.

7.14.3 Konstruktor

Der Konstruktor der Klasse `WT_LdapConnection` erzeugt ein LDAP-Objekt. Dabei wird eine Verbindung mit dem Rechner *hostname* und der Portnummer *port* eröffnet.

```
WT_LdapConnection()
WT_LdapConnection(hostname)
WT_LdapConnection(hostname, port)
```

Rückgabewert

Referenz auf das erzeugte LDAP-Objekt oder NULL

Parameter

hostname
: spezifiziert Rechner der Verbindung.
Defaultwert: "localhost"

port
: spezifiziert die Portnummer der Verbindung.
Defaultwert: 389

Beispiel

```
L=new WT_LdapConnection("localhost");
// LDAP Operationen ausführen
delete L;
```

7.14.4 Methode add

Diese Methode fügt Einträge zum LDAP-Verzeichnis hinzu. Als Parameter sind erforderlich der „Distinguished Name“ *dn* und ein Array mit den Attributen des neu hinzuzufügenden Eintrags. Die Werte der Attribute können Zeichenketten oder Arrays sein.

`add(`*dn*`, `*entry*`)`

Rückgabewert

kein Rückgabewert

Parameter

dn spezifiziert den Distinguished Name (DN) des neuen Eintrags.

entry enthält Informationen über den Eintrag.

Beispiel

```
dn="cn= Hans G. Müller, ou=board, o=FTS, c=DE";
entry= new object();
entry.sn="Hansl";
entry.mail=new Array("Hans@my_company.net",
                     "HGMüller@my_company.net",
                     "NNN@my_company.net");
entry.objectclass="Person";
L=new WT_LdapConnection("localhost");
L.bind();
L.add(dn, entry);
L.unbind();
delete L;
```

7.14.5 Methode bind

Diese Methode richtet eine LDAP-Sitzung ein. Die Parameter *bind_rdn* und *bind_password* regeln den Zugang zum LDAP-Verzeichnis. Falls beim Aufruf von `bind` für keinen der beiden optionalen Parameter ein Argument angegeben wird, wird ein anonymer Zugang eingerichtet. Es wird nur die LDAP-Version 3 unterstützt.

```
bind()
bind(bind_rdn)
bind(bind_rdn, bind_password)
```

Rückgabewert

kein Rückgabewert

Parameter

bind_rdn
: spezifiziert eine Benutzerkennung
Default: anonyme Benutzerkennung

bind_password
: Passwort

Beispiel

```
L=new WT_LdapConnection("localhost");
L.bind();
// LDAP Operationen ausführen
L.unbind();
delete L;
```

7.14.6 Methode bindSasl

Diese Methode richtet eine LDAP-Sitzung mit SASL ein.

`bindSasl(`*bind_rdn*`,` *mechanism*`,` *credentials*`,` *userid*`,` *realm*`)`

Parameter

bind_rdn
spezifiziert einen Eintrag im LDAP-Verzeichnis.

mechanism
spezifiziert den gewünschten SASL-Mechanismus.

credentials
spezifiziert die Berechtigung des gewünschten SASL-Mechanismus.

userid spezifiziert eine Kennung.

realm spezifiziert eine Arbeitsgruppe. Falls nicht gefordert, geben Sie einen Leerstring an.

Rückgabewert

kein Rückgabewert

Beispiel

```
L=new WT_LdapConnection("localhost");
L.setOption("LDAP_OPT_PROTOCOL_VERSION","LDAP_VERSION3");
L.bindSasl( "cn=admin,o=FTS,c=de", "DIGEST-MD5", "secret" , "admin", "");
// LDAP Operationen ausführen
L.unbind();
delete L;
```

7.14.7 Methode compare

Diese Methode vergleicht den Wert eines Attributs mit einem Eintrag.

`compare(`*dn*`, `*attribute*`, `*value*`)`

Rückgabewert

`boolean`-Wert: `true` bei Gleichheit, `false` bei Ungleichheit.

Falls ein Attribut mehrere Werte besitzt, wird bei der ersten Übereinstimmung der Wert `true` zurückgeliefert.

Parameter

dn enthält den Distinguished Name (DN) des Eintrags, der den Eintrag im Directory adressiert.

attribute
spezifiziert das Attribut mit dem durch *dn* adressierten Eintrag.

value spezifiziert den Wert des Attributs *attribute* mit dem durch *dn* adressierten Eintrag im Directory.

Beispiel

```
dn="cn= Hans G. Müller, ou=board, o=FTS, c=DE";
att="password";
val="secret"
L=new WT_LdapConnection("localhost");
L.bind();
r=L.compare(dn, att, val);
if (r == true)
{ // Übereinstimmung }
else
{ // keine Übereinstimmung }
L.unbind();
delete L;
```

7.14.8 Methode deleteEntry

Diese Methode löscht einen Eintrag im LDAP-Verzeichnis, der durch den Parameter *dn* spezifiziert wird.

```
deleteEntry(dn)
```

Rückgabewert

kein Rückgabewert

Parameter

dn enthält den Distinguished Name (DN), der den zu löschenden Eintrag adressiert.

Beispiel

```
dn="cn= Hans G. Müller, ou=board, o=FTS, c=DE";
L=new WT_LdapConnection("localhost");
L.bind();
L.deleteEntry(dn);
L.unbind();
delete L;
```

7.14.9 Methode equals

Die Methode `equals()` vergleicht das aufrufende LDAP-Objekt mit dem als Argument übergebenen Objekt auf Gleichheit hinsichtlich der Klasse, Attribute und Werte.

```
equals(object)
```

Rückgabewert

`boolean`-Wert: `true` bei Gleichheit, `false` bei Ungleichheit

Parameter

object spezifiziert das LDAP-Objekt, mit dem das aufrufende Objekt verglichen werden soll.

7.14.10 Methode explodeDn

Diese Methode zerlegt den im Parameter *dn* spezifizierten Distinguished Name (DN), der von der Methode `getDn` geliefert wurde, in seine Komponenten. Die einzelnen Teile entsprechen den RDNs.

`explodeDn(`*dn*`, `*with_attrib*`)`

Rückgabewert

Array, bestehend aus den Komponenten des DN

Parameter

dn spezifiziert den DN des Eintrags, der zerlegt werden soll.

with_attrib
spezifiziert, ob RDNs mit dem Namen der Attribute angezeigt werden sollen oder nicht. Um RDNs mit den Namen der Attribute zu erhalten, muss der Parameter *with_attrib* auf `true` gesetzt werden. Wenn der Parameter *with_attrib* auf `false` gesetzt wird, werden nur die Werte der Attribute geliefert.

Beispiel

```
basedn="o=FTS, c=DE";
filter="((ou=board)(cn=Hans G. Müller))";
scope="LDAP_SCOPE_SUBTREE";
justthese=new Array("ou","cn","mail");
L=new WT_LdapConnection("localhost");
L.bind();
Count=L.search(basedn, filter, scope, justthese);
Id=L.firstEntry();
if((dn=L.getDn(Id))!=NULL)
{
  document.writeln(dn);
  // Output:
  // cn=Hans G. Müller, ou=board, o=FTS, c=DE
  A=L.explodeDn(dn, true);
  document.writeln(A);
  // Output:
  // [cn=Hans G. Müller,ou=board,o=FTS,c=DE]
  B=L.explodeDn(dn, false);
  document.writeln(B);
  // Output:
  // [Hans G. Müller,board,FTS,DE]
}
L.unbind();
delete L;
```

7.14.11 Methode firstEntry

Diese Methode liefert das Ergebniskennzeichen (result_entry_id) des ersten Eintrags eines Ergebnisses.

Mit den Methoden `firstEntry` und `nextEntry` können Sie die Einträge eines Ergebnisses sequentiell lesen. Der Rückgabewert der Methode `firstEntry` wird beim ersten Aufruf der Methode `nextEntry` als Argument übergeben, um weitere Einträge zu erhalten.

```
firstEntry()
```

Rückgabewert

Ergebnis-ID (result_entry_id) des ersten Eintrags.

Parameter

keine

Beispiel

```
basedn="o=FTS, c=DE";
filter="((ou=marketing)(cn=Hans*))";
scope="LDAP_SCOPE_SUBTREE";
justthese=new Array("ou","cn","mail");
L=new WT_LdapConnection("localhost");
L.bind();
Count=L.search(basedn, filter, scope, justthese);
for(Id=L.firstEntry();Id!=0;Id=L.nextEntry(Id))
{
 if((dn=L.getDn(Id))!=NULL)
       document.writeln("DN: "+dn);
}
L.unbind();
delete L;
```

7.14.12 Methode getClassName

Die Methode liefert den Namen der Klasse, der das aufrufende Objekt angehört, als Zeichenkette zurück.

```
getClassName()
```

Rückgabewert

Zeichenkette, die die Klasse des aufrufenden Objekts angibt, hier "WT_LdapConnection".

Parameter

keine

7.14.13 Methode getDn

Diese Methode liefert den Distinguished Name (DN) eines Ergebnis-Eintrags zurück.

```
getDn(result_entry_id)
```

Rückgabewert

DN eines Eintrags

Parameter

result_entry_id
: spezifiziert die ID des Eintrags, dessen DN zurückgeliefert werden soll. *result_entry_id* muss ein gültiger Wert sein.

Beispiel

Siehe Methode `firstEntry`

7.14.14 Methode getEntries

Diese Methode vereinfacht das Lesen mehrerer Ergebnis-Einträge (mit Attributen und Werten). Die gesamte Information des Ergebnisses einer Anfrage an LDAP (siehe Abschnitt „Methode search“ auf Seite 264) wird mit Hilfe eines `getEntries()`-Aufrufs in einem WTScript-Objekt zur Verfügung gestellt.

```
getEntries()
```

Rückgabewert

Objekt der Klasse Array, das für alle gefundenen Einträge ein Objekt enthält.

Jeder Eintrag in dem Array ist ein Objekt mit folgenden Attributen:

– attr: Objekt mit den Attributen des gefundenen Eintrags
– dn: enthält den Distinguished Name (DN) für den gefundenen Eintrag.
– count: enthält die Anzahl der Attribute.

Binäre Werte werden als Base64-codierte Zeichenkette dargestellt.

Beispiel

```
basedn="o=FTS, c=DE";
filter="((ou=marketing)(cn=Hans*))";
scope="LDAP_SCOPE_SUBTREE";
justthese=new Array("ou","cn","mail");
L=new WT_LdapConnection("localhost");
L.bind();
Count=L.search(basedn, filter, scope, justthese);
ResultObject=L.getEntries();
L.unbind();
delete L;
```

7.14.15 Methode getOption

Diese Methode liefert den aktuellen Wert einer Option.

```
getOption(option)
```

Rückgabewert

Wert der Option *option* als Zeichenkette

Parameter

option spezifiziert die Option, deren Wert geliefert werden soll.

Für den Parameter *option* können Sie folgende Werte angeben:

```
"LDAP_OPT_DEREF"
"LDAP_OPT_SIZELIMIT"
"LDAP_OPT_TIMELIMIT"
"LDAP_OPT_PROTOCOL_VERSION"
"LDAP_OPT_ERROR_NUMBER"
"LDAP_OPT_REFERRALS"
"LDAP_OPT_RESTART"
"LDAP_OPT_ERROR_STRING"
"LDAP_OPT_MATCHED_DN"
"LDAP_OPT_HOST_NAME"
```

Beschrieben sind die genannten Werte in folgendem Dokument:

http://www.openldap.org/devel/cvsweb.cgi/~checkout~/doc/drafts/draft-ietf-ldapext-ldap-c-api-xx.txt

Beispiel

```
L=new WT_LdapConnection("localhost");
L.bind();
sizelimit=L.getOption("LDAP_OPT_SIZELIMIT");
L.unbind();
delete L;
```

7.14.16 Methode modify

Diese Methode modifiziert einen vorhandene Eintrag im LDAP-Verzeichnis.

`modify(`*dn*`, `*entry*`)`

Rückgabewert

kein Rückgabewert

Parameter

dn enthält den Distinguished Name (DN), der den zu ändernden Eintrag adressiert.

entry enthält Informationen über den Eintrag *dn*.

Beispiel

```
dn="cn=Hans G. Müller, ou=board, o=FTS, c=DE";
entry = new object();
entry.mail="Hans.G.Müller@my_company.net";
L=new WT_LdapConnection("localhost");
L.bind();
L.modify(dn, entry);
L.unbind();
delete L;
```

7.14.17 Methode nextEntry

Diese Methode liefert das Ergebniskennzeichen (result_entry_id) des nächsten Ergebnis-Eintrags.

Mit den Methoden `firstEntry` und `nextEntry` werden die Einträge eines Ergebnisses sequentiell gelesen. Der Rückgabewert der Methode `firstEntry` wird beim ersten Aufruf der Methode `nextEntry` als Argument übergeben, um den nächsten Eintrag zu erhalten. Bei weiteren `nextEntry`-Aufrufen wird das Ergebnis des jeweils vorhergehenden `nextEntry`-Aufrufs als Argument übergeben.

`nextEntry(`*result_entry_id*`)`

Rückgabewert

Ergebnis-ID (result_entry_id) des nächsten Eintrags

Parameter

result_entry_id
: spezifiziert das Ergebnis, dessen Einträge ausgegeben werden sollen.

Beispiel

Siehe Methode `firstEntry`.

7.14.18 Methode search

Diese Methode durchsucht den LDAP-Verzeichnisbaum mit Hilfe eines Filters. Zu einem Basis-Eintrag (*base_dn*) im LDAP-Verzeichnis werden die Informationen aus einem spezifizierten Suchbereich geliefert. Mit Hilfe optionaler Parameter lässt sich das Suchergebnis weiter eingrenzen.

```
search(base_dn, filter, scope)
search(base_dn, filter, scope, attributes)
search(base_dn, filter, scope, attributes, sizelimit)
```

Rückgabewert

Anzahl der Einträge

Parameter

base_dn
spezifiziert den Basis-Eintrag, ab dem gesucht werden soll.

filter spezifiziert den Filter für die Suchoperation. Mehrere Filter können mit boolschen Operatoren unter Verwendung von Klammern verknüpft werden.

Die allgemeine Syntax für einen Filter lautet:

filter ::= (*attribute comp_op value*) | (*boolean_op* (*filter1*) [(*filter2*)] ...)

Erläuterung der Syntax-Elemente von filter

attribute
spezifiziert ein Attribut.

comp_op
spezifiziert einen Vergleichsoperator.

Folgende Vergleichsoperatoren stehen zur Verfügung:

=
Gesucht werden alle Einträge, die das Attribut *attribute* enthalten, und bei denen der Wert des Attributs *attribute* mit dem Wert *value* identisch ist.

Beispiel

```
(cn=Hans)
```

Suchergebnis: Alle Einträge, für die `"cn=Hans"` gilt

>=

Gesucht werden alle Einträge, die das Attribut *attribute* enthalten, und bei denen der Wert des Attributs *attribute* größer oder gleich dem Wert *value* ist.

Beispiel

```
(cn>=Hans)
```

Suchergebnis: Alle Einträge im Bereich "cn=Hans" bis "cn=Z..."

<=

Gesucht werden alle Einträge, die das Attribut *attribute* enthalten, und bei denen der Wert des Attributs *attribute* kleiner oder gleich dem Wert *value* ist.

Beispiel

```
(cn<=Hans)
```

Suchergebnis: Alle Einträge im Bereich "cn=A..." bis "cn=Hans"

~=

Gesucht werden alle Einträge, die das Attribut *attribute* enthalten, und bei denen der Wert des Attributs *attribute* annähernd dem Wert *value* entspricht.

Beispiel

```
(cn~=Meier)
```

Suchergebnis: Einträge, wie z.B.: "cn=Meier" und "cn=Meyer"

=*

Gesucht werden alle Einträge, die das Attribut *attribute* enthalten.

Beispiel

```
(cn=*)
```

Suchergebnis: Alle Einträge, die das Attribut cn enthalten

boolean_op

spezifiziert einen boolschen Operator.

Folgende boolschen Operatoren stehen zur Verfügung:

& Gesucht werden alle Einträge, die den Kriterien sämtlicher spezifizierter Filter entsprechen.

| Gesucht werden alle Einträge, die dem Kriterium von mindestens einem der spezifizierten Filter entsprechen.

! Gesucht werden diejenigen Einträge, die dem spezifizierten Filterkriterium **nicht** entsprechen. Dieser Operator kann nur auf einen einzelnen Filter angewendet werden.

Erlaubt ist z.B. folgender Ausdruck: `(!(filter1))`
Unzulässig hingegen ist der Ausdruck: `(!(filter1)(filter2))`

Beispiel

```
(|(cn=Hans)(cn=Hias))
```

Suchergebnis: alle Einträge, die das Attribut cn enthalten, und bei denen der Wert von cn entweder "Hans" oder "Hias" lautet.

value

spezifiziert einen Wert für das Attribut *attribute*, der als Filterkriterium verwendet wird.
Bei der Angabe eines Wertes für *value* können Sie Wildcards verwenden und so gezielt nach Einträgen suchen, bei den das Attribut *attribute* einen Wert besitzt, der

– eine spezifizierte Zeichenfolge enthält bzw.
– mit einer spezifizierten Zeichenfolge beginnt bzw.
– mit einer spezifizierten Zeichenfolge endet.

Beispiel

```
(cn=H*)
```

Als Suchergebnis werden alle Einträge geliefert, bei denen der Wert des Attributs `cn` mit dem Zeichen „H“ beginnt.

scope spezifiziert den Suchbereich.

Mögliche Werte sind:

– `"LDAP_SCOPE_SUBTREE"`

Diese Option bewirkt, dass alle Informationen unterhalb des Eintrags *base_dn* geliefert werden.

– `"LDAP_SCOPE_BASE"`

Diese Option bewirkt, dass alle Informationen des durch den Parameter *base_dn* spezifizierten Eintrags geliefert werden.

– `"LDAP_SCOPE_ONELEVEL"`

Diese Option bewirkt, dass nur Informationen geliefert werden, die im Level direkt unterhalb des durch den Parameter *base_dn* spezifizierten Eintrags liegen.

attributes

Mit dem Parameter *attributes* können die vom Server gelieferten Attribute und Werte eingeschränkt werden. Dieser Parameter ist ein Array mit den Namen der Attribute (DN wird immer angezeigt):

z.B. `a=new Array("mail","sn","cn");`

sizelimit

enthält Informationen über den Eintrag.
Der Parameter *sizelimit* begrenzt die Anzahl der Einträge, die gefunden werden. Der Wert 0 für diesen Parameter bedeutet keine Grenze.

Beispiel

```
basedn="o=FTS, c=DE";
filter="(&(ou=marketing)(cn=Hans*))";
scope="LDAP_SCOPE_SUBTREE";
justthese=new Array("ou","cn","mail");
L=new WT_LdapConnection("localhost");
L.bind();
Count=L.search(basedn, filter, scope, justthese);
ResultObject=L.getEntries();
L.unbind();
delete L;
```

Weitere Beispiele zu Filtern finden Sie im Abschnitt „WebTransactions und LDAP: Beispiele“ auf Seite 271.

7.14.19 Methode setOption

Diese Methode setzt den Wert einer Option.

```
setOption(option, newval)
```

Rückgabewert

kein Rückgabewert

Parameter

option spezifiziert die Option, die gesetzt werden soll.

Für folgende Options können mit dem Parameter *newval* Werte spezifiziert werden:

```
"LDAP_OPT_DEREF"
"LDAP_OPT_SIZELIMIT"
"LDAP_OPTTIMELIMIT
"LDAP_OPT_PROTOCOL_VERSION"
"LDAP_OPT_REFERRALS"
"LDAP_OPT_RESTART"
"LDAP_OPT_HOSTNAME"
"LDAP_OPT_VERSION3"
```

newval spezifiziert den Wert, auf den die Option *option* gesetzt werden soll.

Beispiel

```
L=new WT_LdapConnection("localhost");
L.bind();
L.setOption("LDAP_OPT_SIZELIMIT", "100");
L.unbind();
delete L;
```

7.14.20 Methode toString

Die Methode `toString()` liefert eine Zeichenkette zurück, in der jedes Attribut mit seinem aktuellen Wert angeführt wird. Wenn ein Attribut ein Objekt vom Typ `function` ist, wird nicht der Wert zurückgeliefert, sondern die Definition der Funktion.

Diese Methode können Sie verwenden, um ein neues Objekt mit identischen Attributwerten zu erzeugen.

Um Endlosverkettungen zu vermeiden, beendet die Methode `toString` die Ausgabe bei Rekursion. Die Ausgabewird dort abgebrochen, wo dieselbe Objektreferenz ein zweites Mal ausgegeben würde.

```
toString()
```

Rückgabewert

Zeichenkette, in der jedes Attribut mit seinem Wert angeführt ist.

Parameter

keine

7.14.21 Methode unbind

Diese Methode beendet die LDAP-Sitzung.

```
unbind()
```

Rückgabewert

kein Rückgabewert

Parameter

keine

Beispiel

Siehe Beispiele zu `bind` und `bindSasl`

7.14.22 Methode valueOf

Die Methode `valueOf()` des LDAP-Objekts liefert eine Referenz auf das aufrufende Objekt zurück.

```
valueOf()
```

Rückgabewert

Referenz auf das aufrufende Objekt

Parameter

keine

7.14.23 WebTransactions und LDAP: Beispiele

Nachfolgend finden Sie drei Beispiele für den Einsatz der Klasse `WT_LdapConnection`:

– Suchoperation
– Vergleichsoperation
– LDAP-Ausnahmenbehandlung in WTScript

Beispiel einer Suchoperation

In diesem Beispiel werden die Attribute „Organisationseinheit“, „Nachname“, „Vorname“ und „email-Adresse“ für alle Personen der Organisation „`FTS`“ gesucht, wobei der Nachname mit „H“ beginnt und der Vorname „Sepp“ lauten soll.

```
<wtOnCreateScript>
dn = "o=FTS, c=DE";
filter = "(|(sn=H*)(givenname=Sepp))";
justthese = new Array( "ou", "sn", "givenname", "mail");
scope = "LDAP_SCOPE_SUBTREE";
l = new WT_LdapConnection("localhost");
l.bind();
// Suchen im Verzeichnis
count = l.search(dn, filter, scope, justthese);
info = l.getEntries();
document.writeln(info);
l.unbind();
delete l;
</wtOnCreateScript>
```

Beispiel einer Vergleichsoperation

Im diesem Beispiel wird überprüft, ob das angegebene Passwort mit dem Passworteintrag im Verzeichnis übereinstimmt, der durch den DN spezifiziert wird.

```
<wtOnCreateScript>
dn = "cn=Susi Meier, ou=Entwicklung, o=FTS, c=DE";
value = "secretpassword";
attr = "password";
l = new WT_LdapConnection ();
l.bind();
// Vergleich der Werte des angegebenen Attributs mit dem Verzeichniseintrag
if (l.compare(dn, attr, value)) {
         document.writeln("Passwort korrekt.");
}
else {
         document.writeln("Fehlerhafter Versuch! Passwort nicht korrekt.");
}
l.unbind();
delete l;
</wtOnCreateScript>
```

Beispiel einer Ausnahmenbehandlung in WTScript für die Ausnahme ldapError

In diesem Beispiel sichert der try-Block die Anweisungsfolge zwischen (1) und (2) vor dem Auftreten der `ldapError`-Ausnahme. Der anschließenden catch-Block führt die Fehlerbehandlung durch. Näheres zur Ausnahmenbehandlung in WTScript finden Sie im Abschnitt „Ausnahmenbehandlung“ auf Seite 325.

```
<wtOnCreateScript>
dn = "o=FTS, c=DE";
filter = "(|(sn=H*)(givenname=Sepp))";
justthese = new Array( "ou", "sn", "givenname", "mail");
scope = "LDAP_SCOPE_SUBTREE";
try
{       ------------------------------------------------------- (1)
   l = new WT_LdapConnection("localhost");
   l.bind();
   // Suchen im Verzeichnis
   count = l.search(dn, filter, scope, justthese);
   info = l.getEntries();"LDAP_OPT_
   document.writeln(info);
   l.unbind();
   delete l;
}       ------------------------------------------------------- (2)
catch(ldapError)
{
   document.writeln(ldapError.text);
   switch(ldapError.msg#)
{
      case 303:
      l.unbind();
      break;
      default:
}
   delete l;
}
</wtOnCreateScript>
```

7.15 WT_Userexit-Klasse

Objekte der Klasse `WT_Userexit` ermöglichen Ihnen den Aufruf von Userexits. Dabei entspricht jedes Objekt der Klasse jeweils einer Userexit-Bibliothek. Die in dieser Bibliothek definierten Funktionen stehen Ihnen dann als Methoden dieses Objekts zur Verfügung.

7.15.1 Konstruktoren

```
WT_Userexit()
WT_Userexit(library)
```

Durch die Angabe von *library* kann der Name einer Bibliothek angegeben werden. Userexits, die über dieses Objekt aufgerufen werden, werden in der angegebenen Bibliothek gesucht.

Der Name wird ohne Suffix angegeben. WebTransactions ergänzt das Suffix `.so` bzw. `.dll` entsprechend dem Betriebssystem. Die angegebene Bibliothek wird von WebTransactions im Basisverzeichnis und - falls dort keine solche Bibliothek gefunden wird - im Installationsverzeichnis gesucht.

Ist *library* nicht angegeben, wird die Default-Bibliothek `WTUserexits.[so|dll]` verwendet.

7.15.2 Methoden

Alle Benutzerfunktionen, die in gemeinsam nutzbaren Bibliotheken liegen bzw. in WebTransactions eingebunden sind, stehen als Methoden von Objekten der Klasse `WT_Userexit` zur Verfügung. Ein Userexit *function* kann dann folgendermaßen im Scriptbereich und innerhalb von Auswertungsoperatoren aufgerufen werden:

```
mylib.function(...);
WT_Userexit.function(...);
```

mylib ist ein Objekt der Klasse `WT_Userexit`. Die Funktion *function* wird in der dem Objekt zu Grunde liegenden Bibliothek gesucht.

Durch die Notation `WT_Userexit.`*function* können Sie auf die Klassenmethoden der `WT_Userexit`-Klasse zugreifen: *function* wird dann in der Bibliothek `WTUserexits.[dll|so]` gesucht (zunächst im Basisverzeichnis und - falls dort nicht vorhanden - im Installationsverzeichnis).

In BS2000/OSD werden keine dynamischen Bibliotheken unterstützt. Hier müssen Sie die Funktionen fest zum WTHolder-Programm hinzubinden.

Beispiel

Im folgenden Beispiel wird die Benutzerfunktion `myFunction` in der Bibliothek `myLibrary.so` aufgerufen, die sich in einem vom Benutzer angelegten Unterverzeichnis `myDir` des Basisverzeichnisses befindet.

```
myUserExit = new WT_Userexit( 'myDir/myLibrary');
res = myUserExit.myFunction(1,2,3);
```

8 WTML-Tags

WTML-Tags enthalten die WebTransactions-spezifischen Funktionen. Sie haben die Form `<wt...>` oder `</wt...>` und entsprechen damit dem HTML-Standard.

WTML-Tags werden von WebTransactions beim Lesen des Templates interpretiert. Hierbei geschieht Folgendes:

– Auswerten der Kontrollstrukturen und Anweisungen, die den Aufbau der HTML-Seite steuern
– Ausführen von Aktionen und Anweisungen, die der Aufbereitung der Daten für den Aufbau der HTML-Seite dienen
– Zwischenspeichern von Aktionen und Anweisungen zur Nachbearbeitung der vom Browser zurückerhaltenen Daten

Alle Zeichen außerhalb der umschließenden spitzen Klammern gehören nicht mehr zum WTML-Tag sondern zu HTML- oder Script-Bereichen. Gehören sie zu HTML-Bereichen, so werden sie normalerweise vom Browser angezeigt - mit folgenden beiden Ausnahmen:

– Stehen zwischen einem Zeilenanfang und der öffnenden spitzen Klammer nur Leerzeichen und Tabulatoren, so gehören diese zum WTML-Tag und werden bei der Ausgabe ignoriert.

– Endet ein WTML-Tag am Ende einer Zeile, so gehört der folgende Zeilenumbruch zum WTML-Tag und wird bei der Ausgabe ignoriert.

Dieses Vorgehen erlaubt es, die WTML-Tags übersichtlich in eigene Zeilen zu schreiben und durch Einrückung zu strukturieren, ohne hierdurch das Erscheinungsbild des Templates zu beeinträchtigen.

Übersicht über die WTML-Tags

Es gibt die in den folgenden beiden Tabellen aufgelisteten WTML-Tags. Im Anschluss an die Tabellen werden diese WTML-Tags im Einzelnen beschrieben. Beispiele, die das Zusammenspiel verschiedener WTML-Tags illustrieren, finden Sie in Kapitel 11.

WTML-Tag	Funktion	Syntax
Rem-Tag	Inline-Dokumentation des Templates; wird bei der Generierung der HTML-Seite ignoriert	<wtRem ...> oder <wtRem> ... </wtRem>
Dataform-Tag	markiert den Bereich der zur WebTransactions-Anwendung zurückgeschickten Daten; entspricht dem HTML-Form-Tag	<wtDataform ...> ... </wtDataform>
Exit-Tag	beendet die Verarbeitung des gewünschten Bereichs	<wtExit ...>
Include-Tag	verweist auf andere Template-Dateien, die eingeschoben werden, z.B. Vorlagen für ein einheitliches, leicht änderbares Look & Feel der HTML-Seiten	<wtInclude ...>
OnCreateScript-Tag	öffnendes und schließendes OnCreateScript-Tag klammern ein WTScript, das beim Lesen des Templates direkt ausgeführt wird (Einsatz siehe OnCreateScript-Tag)	<wtOnCreateScript> ... </wtOnCreateScript>
OnReceiveScript-Tag	öffnendes und schließendes OnReceiveScript-Tag klammern ein WTScript, das beim Empfang der vom Browser geschickten Daten ausgeführt wird (Einsatz siehe OnReceiveScript-Tag)	<wtOnReceiveScript> ... </wtOnReceiveScript>

Die `OnCreateScript/OnReceiveScript`-Tags lassen sich nach ihrem Ausführungszeitpunkt unterscheiden:

– `OnCreateScript`-Tags werden direkt beim Lesen des Templates ausgeführt und die beschriebenen Wertzuweisungen sind sofort gültig, d.h. beim Generieren der aktuellen HTML-Seite.

– `OnReceiveScript`-Tags werden erst ausgeführt, nachdem diese HTML-Seite generiert, zum Browser gesendet und die vom Browser geschickten Daten empfangen wurden, also vor dem Lesen des folgenden Templates. Sie eignen sich für die Nachbearbeitung der Daten des aktuellen Dialogschritts.

Für Kontrollstrukturen stehen Ihnen die folgenden WTML-Tags zur Verfügung:

WTML-Tag	Funktion	Syntax
If-/Else-/Endif-Tag	Einfachauswahl gemäß Bedingung	<wtIf ...> ... <wtElse> ... <wtEndIf>
DoWhile-Tag	Schleife, die durchlaufen wird, bis die Laufbedingung ungültig wird	<wtDoWhile ...> ... </wtDoWhile>
Do-/Until-Tag	Schleife, die durchlaufen wird, bis die Abbruchbedingung gültig wird	<wtDo> ... <wtUntil(...)>

Schlüsselwörter in WTML-Tags

Die WTML-Tags und die in ihnen vorkommenden Schlüsselwörter können beliebig groß/kleingeschrieben werden. Die WTML-Tags enthalten wie HTML-Tags in beliebiger Reihenfolge eine Reihe von Eigenschaften, die durch bestimmte Schlüsselwörter bezeichnet sind. Nach dem Schlüsselwort folgt ein Gleichheitszeichen und ein Wert für diese Eigenschaft. Der Wert kann als String oder einfacher String-Ausdruck angegeben werden.

Weglassen von Hochkommas

Enthält die Zeichenkette, die einen Wert innerhalb eines WTML-Tags repräsentiert, weder Leerzeichen noch Tabulatoren noch Zeilenumbrüche (oder sind diese Zeichen jeweils entwertet), so können die einschließenden Hochkommas entfallen: Ist das erste Zeichen des Wertes kein Hochkomma, so werden alle Zeichen bis zum ersten WhiteSpace oder bis zur abschließenden Klammer („>“) als Wert berücksichtigt.

Beispiel

```
<wtInclude Name=hallo\  Welt>
```

8.1 Rem - Kommentare einfügen

Mit dem Rem-Tag können Sie Kommentare zur Inline-Dokumentation in das Template einfügen. Diese Kommentare werden im Gegensatz zu HTML-Kommentaren nicht an den Browser geschickt. Rem-Tags können im HTML-Bereich stehen sowie innerhalb von WTML-Tags; und zwar an den Stellen, an denen WhiteSpaces erlaubt sind. Innerhalb von String-Ausdrücken werden Rem-Tags nicht erkannt (d.h. als feste Zeichenkette ausgewertet).

```
<wtRem comments>
```

```
<wtRem>
   comments
</wtRem>
```

Für Rem-Kommentare sind zwei Formatvarianten möglich:

- Bei der ersten Variante steht der Kommentar direkt innerhalb der spitzen Tag-Klammern. In dieser Variante kann der Kommentar kein Zeichen „>“ enthalten, da dieses Zeichen als Ende des Kommentars interpretiert würde. Geschachtelte oder leere Kommentare sind in dieser Variante nicht zulässig.
- In der zweiten Variante steht der Kommentar zwischen öffnendem und schließendem Rem-Tag. Der Kommentar darf in dieser Variante auch das Zeichen „>“ enthalten. Damit haben Sie die Möglichkeit, Kommentare beliebig zu schachteln und auch WTML-Tags auszukommentieren. Leere Kommentare sind ebenfalls zulässig.

8.2 Dataform - Formularbereich definieren

Einleitendes und abschließendes Dataform-Tag klammern einen Formularbereich. In diesem Formularbereich definieren Sie mit den üblichen HTML-Mitteln Dialogelemente, wie z.B. Buttons, Eingabefelder, mehrzeilige Textfelder (textarea) oder auch Auswahllisten. Der Anwender am Browser kann dann Daten eingeben und Funktionen auswählen. Drückt er auf den Submit-Button, werden die Eingaben an WebTransactions geschickt.

Das Dataform-Tag von WebTransactions hat die Form:

```
<wtDataform [Name="name"] [OnSubmit="OnSubmitHandler"] [ASYNC_PAGE="asyncPage"]>
Bereich
</wtDataform>
```

name frei wählbarer Name.

OnSubmitHandler
JavaScript-Code, der vom Browser ausgeführt wird, nachdem der Anwender den Submit-Button gedrückt hat (noch bevor die Daten abgeschickt werden). Dieser kann z.B. die Benutzereingaben einer Plausibilitäts-Prüfung unterziehen.

asyncPage
Falls Sie dieses Attribut setzen, wird die Seite auch dann von WebTransactions verarbeitet, wenn sie nicht in die sequenzielle Folge der Dialogschritte passt (asynchrone Kommunikation). Sie können damit eine Seite an WebTransactions schicken, die nicht der zuletzt ausgegebenen Seite entspricht. *asyncPage* spezifiziert das Template, das diese asynchrone Anforderung bearbeitet (siehe hierzu auch WebTransactions-Handbuch „Konzepte und Funktionen“).

Bereich.
Hier definieren Sie mit den üblichen HTML-Mitteln Dialogelemente, wie z.B. Buttons oder Eingabefelder. Der Bereich kann daneben auch HTML-Text enthalten.

WebTransactions ersetzt zur Laufzeit die Dataform-Tags durch „normale“ HTML-Form-Tags:

```
<form method="post" name="name" onsubmit="OnSubmitHandler" action=
url_of_webtransactions>
<input type="hidden" ...> ...
Bereich
</Form>
```

Das Action-Attribut wird dabei automatisch mit der URL der WebTransactions-CGI-Komponente `WTPublish` versorgt, zu der das ausgefüllte Formular geschickt werden soll. Außerdem generiert WebTransactions eine Reihe versteckter Felder, die z.B. zum Wiederauffinden der Sitzung mitgeschickt werden.

Das Abschluss-Tag `</wtDataform>` wird von WebTransactions z.Zt. nur in ein `</Form>` umgesetzt. Damit Sie Erweiterungen des Abschluss-Tags in folgenden Versionen nutzen können, sollten Sie jetzt schon das WTML-Tag und nicht das HTML-Form-Tag zum Abschluss des zu sendenden Abschnitts benutzen.

Es können auch mehrere Formularbereiche der Form `<wtDataform> ... </wtDataform>` in einem Template stehen. Es wird dann jeweils nur derjenige Formularbereich mit seinen Daten an die WebTransactions-Anwendung geschickt, dessen Submit-Button gedrückt wurde.

Alternative Schreibweise

Einige Browser (z.B. Netscape) stellen keine Dialogelemente dar, wenn diese nicht in einer Klammer `<Form> ... </Form>` stehen. Wenn Sie für die Template-Programmierung einen solchen Browser verwenden und trotzdem das Template-Layout offline am Browser überprüfen wollen, können Sie für das Dataform-Tag folgende Schreibweise verwenden:

```
<Form WEBTRANSACTIONS [Name="name"] [OnSubmit="submitFunction"]>
Bereich
</Form>
```

Diese Schreibweise hat den Nachteil, dass das Tag nicht mit `<wt>` beginnt und damit nicht von den Standard-HTML-Tags abgehoben ist.

8.3 Exit - Verarbeitung abbrechen

Mit dem Exit-Tag können Sie die Verarbeitung im angegebenen Bereich des aktuellen Templates abbrechen. Das Exit-Tag kann überall innerhalb des HTML-Bereichs stehen. Die bis dahin bekannten Receive-Regeln werden noch ausgeführt.

```
<wtExit scope={"TEMPLATE"|"DIALOGSTEP"|"SESSION"}>
```

`scope` Mit dem Parameter `scope` bestimmen Sie, wo die Verarbeitung abgebrochen und wo die Weiterverarbeitung wieder aufsetzen soll:

`TEMPLATE`
: Die Verarbeitung des aktuellen Templates wird abgebrochen. Die Weiterverarbeitung setzt bei der nächsten Anweisung im rufenden Template fort. Bei Aufruf auf oberster Ebene entspricht `scope="TEMPLATE"` dem `scope="DIALOGSTEP"`

`DIALOGSTEP`
: Die Verarbeitung aller an diesem Dialogschritt beteiligten Templates wird abgebrochen.

`SESSION`
: Die aktuelle WebTransactions-Sitzung wird zum nächstmöglichen Zeitpunkt beendet. Das Ergebnis des aktuellen WTML-Dokuments ist die letzte an den Browser gesendete Seite.

  Das Verhalten bei diesem Parameter ist äquivalent zu `WT_SYSTEM.EXIT_SESSION="TRUE"`.

Siehe auch

„Funktion exitDialogStep()“ auf Seite 98, „Funktion exitSession()“ auf Seite 102 und „Funktion exitTemplate()“ auf Seite 103.

8.4 Include - Templates einbinden

Mit dem Include-Tag können Sie ein Template einbinden. Das Include-Tag kann überall innerhalb des HTML-Bereichs stehen.

Das includierte Template wird komplett eingeschoben. Innerhalb des includierten Templates können die gleichen Sprachmittel verwendet werden wie in jedem anderen Template. Die WTML-Tags müssen allerdings in jedem Template syntaktisch vollständig sein, z.B. ist es *nicht* zulässig, eine IF-Kontrollstruktur im includierenden Template beginnen zu lassen und im includierten Template abzuschließen.

```
<wtInclude Name="fileName">
```

fileName

Name des Templates, das eingeschoben werden soll. *fileName* ist ein relativer Dateiname. Das Suffix `.htm` des Dateinamens müssen Sie nicht angeben. WebTransactions sucht das entsprechende Template gemäß eingestellter Sprache und eingestelltem Stil.

Sie können *fileName* als feste Zeichenkette angeben oder auch als einfachen String-Ausdruck.

Beispiel

```
<wtInclude Name="header">
```

Sie können z.B. die Definition eines Seitenkopfes einschieben, um ein einheitliches, später leicht modifizierbares Look & Feel der HTML-Seiten zu gewährleisten. Die Vorlage `header` kann sowohl HTML-Bereiche, als auch WTML-Tags (zur Modifikation der in ihr dargestellten Daten) enthalten.

8.5 IF/ELSE/ENDIF - Kontrollstruktur

Die IF-Kontrollstruktur kann überall innerhalb des HTML-Bereichs stehen. Sie kann sowohl konstanten HTML-Text, Standard-HTML-Tags als auch WTML-Tags enthalten. IF-Kontrollstrukturen können beliebig geschachtelt werden.

```
<wtIf (Bedingung)>
   Block1
[<wtElse>
   Block2]
{<wtEndIf> | </wtIf>}
```

Bedingung

Beliebiger Ausdruck. In diesen Ausdrücken können aus Kompatibilitätsgründen auch die numerischen Vergleichsoperatoren verwendet werden (`# ==`, `#!=`, `#>`, `#<`, `#<=`, `#>=`), siehe auch Abschnitt „Vergleichsoperatoren, die numerischen Vergleich erzwingen (nur in WTML-Tags)“ auf Seite 73.
WebTransactions wertet *Bedingung* beim Lesen des Templates aus, konvertiert den Ausdruck ggf. in den Typ `boolean` und interpretiert in Abhängigkeit vom Ergebnis *Block1* oder *Block2*: Ist das Ergebnis der Wert true, so wird *Block1* ausgeführt. Andernfalls wird – falls vorhanden – *Block2* ausgeführt.

Block1
Block2

Entsprechend der Auswertung der Bedingung wird beim Lesen des Templates *Block1* oder *Block2* durchlaufen. Diese Blöcke können sowohl konstanten HTML-Text, Standard-HTML-Tags als auch WTML-Tags enthalten, die dann abhängig von der Bedingung berücksichtigt werden oder nicht. *Block2* (der ELSE-Zweig) ist optional.

Beispiel

Sie wollen die generierten Formate um einen Button ergänzen, der den aktiven Abbruch der Sitzung erlaubt.

```
<wtIf ( "##WT_POSTED.ExitButton#" != "abbrechen")>

<wtREM HTML-Generierung gemäß automatisch konvertierter Templates
       einschließlich aller OnReceive-Tags!>

   <wtDataform Name="exitForm">

   Wenn Sie wollen, k&ouml;nnen Sie die Sitzung
        <Input Action="SUBMIT" Name="ExitButton" Value="abbrechen">
   </wtDataform>

<wtElse>

     <wtoncreatescript>
     <!--
      WT_HOST.OSD_0.close();
      exitSession();
     //-->
    </wtoncreatescript>

Ende, Sie haben die Sitzung beendet, wir danken f&uuml;r Ihr Interesse.

<wtEndIf>
```

Die IF-Kontrollstruktur wird von WebTransactions beim Lesen des Templates interpretiert. Die Bedingung wird also immer bereits **während der Interpretation des Templates** geprüft. Auch wenn Sie OnReceiveScript-Tags in einen Zweig der IF-Kontrollstruktur schreiben, wird die Bedingung zum **Generierungszeitpunkt** der HTML-Seite ausgewertet. Je nach Gültigkeit der Bedingung werden die Aktionen aus dem OnReceiveScript-Tag zur verzögerten Ausführung zwischengespeichert oder ignoriert.

IF-Kontrollstrukturen dieser Art sind somit **nicht** geeignet, um die Ausführung von Bearbeitungsschritten von Bedingungen abhängig zu machen, die erst zum Receive-Zeitpunkt ausgewertet werden sollen. Eine Verzweigung z.B. in Abhängigkeit von den geposteten Daten ist nicht möglich. Verwenden Sie für solche Fälle die WTScript-if-Kontrollstruktur (innerhalb von OnReceiveScript-Tags). Die WTScript-if-Kontrollstruktur ist im Abschnitt „if-Verzweigung“ auf Seite 297 beschrieben.

8.6 DO WHILE-Schleife

DO WHILE-Schleifen können überall innerhalb des HTML-Bereichs stehen. Beim Lesen der HTML-Seite wiederholt WebTransactions die Auswertung des Template-Abschnitts *Block* solange wie *Bedingung* gültig ist. Danach wird die Abarbeitung des Templates hinter dem Schleifenkonstrukt fortgesetzt.

```
<wtDoWhile (Bedingung)>
Block
</wtDoWhile>
```

Bedingung
siehe IF-Kontrollstruktur (Abschnitt „IF/ELSE/ENDIF - Kontrollstruktur“ auf Seite 285).

Block Der Template-Abschnitt *Block* kann sowohl konstanten HTML-Text, Standard-HTML-Tags als auch WTML-Tags enthalten.

DO WHILE-Schleifen werden immer zum **Generierungszeitpunkt** der HTML-Seite durchlaufen. Die Bedingung wird also immer bereits bei der Interpretation des Templates geprüft. Enthält der Rumpf der Schleife OnReceiveScript-Tags, werden die entsprechenden Bearbeitungsschritte zur verzögerten Ausführung zwischengespeichert. DO WHILE-Schleifen dieser Art sind somit **nicht** geeignet, um die Anzahl der Iterationen von Bedingungen abhängig zu machen, die erst zum Receive-Zeitpunkt ausgewertet werden sollen (z.B. von geposteten Daten). Verwenden Sie für solche Fälle die WTScript-Schleifen `while` oder `for` (innerhalb von OnReceive-Script-Tags). Die WTScript-Schleifen sind ab Abschnitt „while-Schleife“ auf Seite 299 beschrieben.

8.7 DO UNTIL-Schleife

DO UNTIL-Schleifen können überall innerhalb des HTML-Bereichs stehen. Bei der Interpretation der HTML-Seite wiederholt WebTransactions die Auswertung des Template-Abschnitts *Block* solange, bis *Bedingung* gültig wird. Danach wird die Abarbeitung des Templates hinter dem Schleifenkonstrukt fortgesetzt.

`<wtDo>`
Block
`<wtUntil(`*Bedingung*`)>`

Bedingung
: siehe IF-Kontrollstruktur (Abschnitt „IF/ELSE/ENDIF - Kontrollstruktur“ auf Seite 285).

Block Der Template-Abschnitt *Block* kann sowohl konstanten HTML-Text, Standard-HTML-Tags als auch WTML-Tags enthalten.

Beispiel

Sie haben in Ihrer Host-Anwendung ein Format, auf dem seitenweise geblättert werden kann. Dazu steht ein Feld mit dem Namen `POSITION` zur Verfügung, in das die folgenden Eingaben geschrieben werden können: + (Seite weiter), - (Seite zurück), ++ (ans Ende) und -- (an den Anfang). Solange noch eine weitere Seite vorhanden ist, steht „+“ in dem Feld. Wenn das Ende der Liste erreicht ist, belegt die Host-Anwendung dieses Feld mit dem Wert „--“ vor. Um die gesamte Liste in einer HTML-Seite anzuzeigen, können Sie die folgende Schleife programmieren:

```
<wtRem An den Anfang positionieren>

<wtoncreatescript>
<!--
      WT_HOST.OSD_0.POSITION.Value="--";
//-->
</wtoncreatescript>

<wtDo >
   <wtRem Eine Seite von der Host-Anwendung lesen>
   <wtoncreatescript>
   <!--
        WT_HOST.OSD_0.send();
        WT_HOST.OSD_0.receive();
   //-->
   </wtoncreatescript>

   <wtRem Ausgabe der Daten auf HTML-Seite>
      ##Listenzeile1.Value# <br>
      ##Listenzeile2.Value# <br>
      ##Listenzeile3.Value# <br>

   <wtRem Nächste Seite lesen, falls Ende nicht erreicht>

   <wtIf ("##POSITION.Value#" !="--")>

      <wtoncreatescript>
      <!--
            WT_HOST.OSD_0.POSITION.Value="+";
      //-->
      </wtoncreatescript>

   <wtEndIf>

<wtUntil("##POSITION.Value#" =="--")>
```

DO UNTIL-Schleifen werden immer zum **Generierungszeitpunkt** der HTML-Seite durchlaufen. Die Bedingung wird also immer bereits bei der Interpretation des Templates geprüft. Enthält der Rumpf der Schleife OnReceiveScript-Tags, werden die entsprechenden Bearbeitungsschritte zur verzögerten Ausführung zwischengespeichert. DO UNTIL-Schleifen dieser Art sind somit **nicht** geeignet, um die Anzahl der Iterationen von Bedingungen abhängig zu machen, die erst zum Receive-Zeitpunkt ausgewertet werden sollen (z.B. abhängig von geposteten Daten). Verwenden Sie für solche Fälle die WTScript-Schleifen `while` oder `for` (innerhalb von OnReceiveScript-Tags). Die WTScript-Schleifen sind ab Abschnitt „while-Schleife“ auf Seite 299 beschrieben.

8.8 OnCreateScript - WTScript zum Generierungszeitpunkt

Zwischen öffnendem und schließendem OnCreateScript-Tag können Sie ein Programm in WTScript formulieren. Dieses wird von WebTransactions während der Interpretation des Templates zum Ablauf gebracht, **bevor** die Seite an den Browser geschickt wird. Das Script wird also auf dem WebTransactions-Server ausgeführt („server-seitige“ Scripts). WTScript-Programme innerhalb von OnCreateScript-Tags unterscheiden sich hierdurch von JavaScript-Programmen innerhalb von „gewöhnlichen“ HTML-<script>-Tags, die erst vom Browser interpretiert werden („client-seitiges“ JavaScript).

```
<wtOnCreateScript>
CreateScript
</wtOnCreateScript>
```

CreateScript

WTScript-Programm, das von WebTransactions bei der Interpretation des Templates ausgeführt wird. Die WTScript-Statements, die Sie hier verwenden können, sind im Kapitel „WTScript-Anweisungen (innerhalb von OnCreateScript/OnReceive-Script)“ auf Seite 293ff beschrieben.

Wollen Sie das Template vorab offline in einem Browser anschauen, so lässt sich die Anzeige verbessern, indem Sie das WTScript-Programm in einem HTML-Kommentar verstecken:

```
<wtOnCreateScript>
<!--
CreateScript
// -->
</wtOnCreateScript>
```

8.9 OnReceiveScript - WTScript nach Erhalt der Browser-Daten

Zwischen öffnendem und schließendem OnReceiveScript-Tag können Sie ein Programm in WTScript formulieren. Dieses wird von WebTransactions in die Reihe der übrigen OnReceive-Tags eingeordnet. Es wird also verzögert ausgeführt, nämlich erst nach Empfang der vom Browser geschickten Daten; d.h. erst nachdem die aktuelle HTML-Seite generiert, zum Browser gesendet und die vom Browser geschickten Daten empfangen wurden.

WTScript-Programme innerhalb von OnReceiveScript-Tags werden - wie WTScript-Programme innerhalb von OnCreateScript-Tags - von WebTransactions und nicht vom Browser interpretiert („server-seitige“ Scripts).

```
<wtOnReceiveScript>
ReceiveScript
</wtOnReceiveScript>
```

ReceiveScript

Programm in WTScript, das von WebTransactions nach Empfang der vom Browser geposteten Daten ausgeführt wird. Die WTScript-Statements, die Sie hier verwenden können, sind im Kapitel „WTScript-Anweisungen (innerhalb von OnCreateScript/OnReceiveScript)“ auf Seite 293ff beschrieben.

Wollen Sie das Template vorab offline in einem Browser anschauen, so lässt sich die Anzeige verbessern, indem Sie das WTScript -Programm in einem HTML-Kommentar verstecken:

```
<wtOnReceiveScript>
<!--
ReceiveScript
// -->
</wtOnReceiveScript>
```

9 WTScript-Anweisungen (innerhalb von OnCreateScript/OnReceiveScript)

In den WTScript-Bereichen, die Sie mit `<wtOnCreateScript>` oder `<wtOnReceiveScript>` einleiten, können Sie WTScript-Anweisungen angeben, die auf der Server-Seite ausgeführt werden. WTScript-Anweisungen sind an JavaScript V1.2 angelehnt, d.h., es werden im Wesentlichen die gleichen Sprachkonzepte unterstützt. Lediglich das Objektmodell von JavaScript wurde durch ein eigenes ersetzt, das im WebTransactions-Handbuch „Konzepte und Funktionen“ beschrieben ist. Dieser Abschnitt beschreibt die server-seitigen WTScript-Anweisungen, die WebTransactions unterstützt.

Browser geben den ihnen unbekannten WTScript-Code als Klartext aus. Sie verhindern dies, indem Sie die Scripts in Kommentare einschließen: `<!--` am Anfang und `//-->` am Ende.

Übersicht der Anweisungen

WTScript-Anweisungen können Ausdrücke und weitere Anweisungen enthalten und werden mit einem Strichpunkt ; abgeschlossen. Sie werden der Reihe nach ausgeführt. Die Anweisungen können Nebeneffekte haben wie z.B. die Auswertung von Ausdrücken oder die Wertzuweisung für Variablen. Im Gegensatz zu Ausdrücken haben aber die Anweisungen selbst keinen Wert oder Datentyp.

In diesem Kapitel wird das Verhalten der einzelnen Anweisungen bei der Abarbeitung des WTScripts beschrieben. WebTransactions unterstützt folgende Anweisungen:

- leere Anweisung
- Anweisungen zur Ablaufsteuerung (bedingte Verzweigungen und Schleifen): `if`, `while`, `do/while`, `for`, `for/in`, `switch`, `break`, `continue`, `return`
- Anweisungen zur Deklaration von Variablen und Funktionen: `var` und `function`
- Ausdrücke und Anweisung `with`, um Objektverweise bei Anweisungen verkürzt schreiben zu können.
- Anweisungen zur Fehlerbehandlung: `throw`, `try`, `catch`, `finally`

9.1 Leere Anweisung

Für eine leere Anweisung wird keine Aktion ausgeführt.

```
;
```

Beispiel

Die folgende Schleife sucht im Array `a` nach dem ersten nicht definierten Element

```
for(i=0; i<a.length && a[i]; i++);
```

9.2 Ausdruck als Anweisung

Aus einem Ausdruck wird eine Anweisung, wenn Sie einen Strichpunkt anschließen. Dazu gehören z.B. Zuweisungen, Funktionsaufrufe sowie die Inkrement-/Dekrement-Operatoren.

expression ;

expression
Beliebiger Ausdruck (siehe Kapitel „Ausdrücke und Operatoren“ auf Seite 69)

Beschreibung

Der Ausdruck *expression* wird ausgewertet.

Beispiele

```
output = "Hello" + name;
WT_SYSTEM.STYLE = WT_POSTED.STYLE;

          //Wertzuweisung

document.write("welcome, " + name);

          //Aufruf der write-Methode des document-Objekts

counter++; //Inkrement-Operator
6*7; // nutzlose Anweisung
```

9.3 Anweisungsblock als Anweisung

Ein Anweisungsblock besteht aus keiner, einer oder mehreren Anweisungen, die in geschweifte Klammern eingeschlossen werden.

```
{
  [anweisung] ...
}
```

anweisung
: Einzelne Anweisung oder Anweisungsblock. Dazu gehören z.B. Zuweisungen, Funktionsaufrufe, Inkrement-/Dekrement-Operatoren.

Beschreibung

Ein Anweisungsblock wird ausgeführt, indem die einzelnen Anweisungen der Reihe nach ausgeführt werden.

9.4 Anweisungen zur Ablaufsteuerung

Anweisungen zur Ablaufsteuerung, auch Kontrollstrukturen genannt, steuern den Ablauf eines Programms. Durch Bedingungen legen Sie fest, welche Anweisungen überhaupt ausgeführt werden (Verzweigung) und wie oft sie ausgeführt werden (Schleifen).

9.4.1 if-Verzweigung

Die `if`-Verzweigung führt Anweisungen abhängig von einer Bedingung aus. Ist die Bedingung nicht erfüllt (falsch), können (optional) andere Anweisungen ausgeführt werden. Sollen mehrere Anweisungen bedingt ausgeführt werden, müssen Sie diese in geschweiften Klammern zusammenfassen.

Die Bedingung kann ein beliebiger Ausdruck sein, der sich auf einen logischen Wert (`true`/`false`) abbilden lässt (siehe Kapitel „Ausdrücke und Operatoren“ auf Seite 69). Die Anweisung kann geschachtelt sein.

`if` *(bedingung)* *block1* `[ else` *block2* `]`

bedingung
: Ausdruck, der die Bedingung darstellt und ausgewertet wird

block1, block2
: eine einzelne Anweisung oder ein Anweisungsblock

Beschreibung

Der Ausdruck *bedingung* wird ausgewertet und in den Datentyp `boolean` konvertiert. Ist das Ergebnis der Wert `true`, so wird *block1* ausgeführt, ist der Wert `false`, wird *block2* ausgeführt.

Bei geschachtelten If-Strukturen gehört der `else`-Zweig immer zu der innersten `if`-Anweisung, der noch kein `else`-Zweig zugeordnet ist.

```
if ( expression1 )
   if ( expression2 )
      block1
   else
      block2
```

Die erste `if`-Anweisung enthält nur einen `true`-Zweig, der aus der zweiten `if`- Anweisung besteht.

Beispiel 1

Mit einer if-Verzweigung können Sie z.B. im Start-Template eine Variable als Referenz auf das globale oder verbindungsspezifische System-Objekt anlegen, falls ein solches existiert. Gibt es für die Verbindung ein verbindungsspezifisches System-Objekt (das nicht undefiniert ist), so wird die Variable `host_system` mit dem Wert des verbindungsspezifischen System-Objekts belegt, ansonsten mit dem des globalen.

```
if (WT_HOST.myComObj.WT_SYSTEM != null)
   host_system = WT_HOST.myComObj.WT_SYSTEM;
else
   host_system = WT_SYSTEM;
```

Beispiel 2

Eine Kommunikation mit der Host-Anwendung wird nur durchgeführt, wenn sich der Stil nicht geändert hat.

```
if (WT_SYSTEM.TravStyle == "NoChange")
{
   host.send();
   host.receive();
}
```

9.4.2 while-Schleife

Mit Hilfe von `while`-Schleifen können Sie Anweisungen ausführen und solange wiederholen, wie die Schleifenbedingung erfüllt (wahr) ist. Wenn die Bedingung nicht erfüllt (falsch) ist, wird die Schleifenbearbeitung abgebrochen und mit der Anweisung fortgesetzt, die direkt nach der `while`-Schleife steht.

[*label*:] while (*bedingung*) *block*

---

label spezifiziert ein Label.
Die Angabe eines Labels ermöglicht es, sich in einer `break`- oder `continue`-Anweisung auf dieses Label zu beziehen.

bedingung
Ausdruck, der vor Beginn der `while`-Schleife und vor jedem Durchlauf geprüft wird.

block eine Anweisung oder ein Anweisungsblock.

Beschreibung

Der Ausdruck *bedingung* wird wiederholt ausgewertet und in den Datentyp `boolean` konvertiert. Solange das Ergebnis der Wert `true` ist, wird *block* ausgeführt.

Trifft die Schleifenbedingung immer zu, dann ergibt sich eine Endlosschleife!

Enthält *block* eine `break`-Anweisung, so wird wie folgt verfahren:

– `break` ohne *label* angegeben:

  Die Schleife wird abgebrochen. Die Bearbeitung wird fortgesetzt mit der Anweisung, die direkt auf die `while`-Schleife folgt.

– `break` mit *label* angegeben:

  Die nächstliegende Anweisung, die mit dem Label *label* markiert ist, wird abgebrochen und die Bearbeitung mit der Anweisung fortgesetzt, die unmittelbar auf die abgebrochene Anweisung folgt.

Enthält *block* eine `continue`-Anweisung, so wird wie folgt verfahren:

– `continue` ohne *label* angegeben:

  Die Ausführung von *block* wird abgebrochen. *bedingung* wird erneut ausgewertet und die `while`-Schleife entsprechend fortgesetzt.

– `continue` mit *label* angegeben:

  Die Ausführung von *block* wird abgebrochen. *bedingung* der mit dem Label *label* markierten Schleife wird erneut ausgewertet und die mit *label* markierte Schleife entsprechend fortgesetzt.

Beispiel für Endlosschleife

```
while(true)
{
}
```

Beispiel

Die folgende `while`-Schleife gibt die definierten Werte des Array `a` in den Spalten einer Tabellenzeile aus

```
document.write("<tr>");
i=0;
while(i < a.length && a[i])
{
   document.write("<td>" + a[i++] + "</td>");
}
document.write("</tr>");
```

9.4.3 do/while-Schleife

Mit Hilfe der `do/while`-Schleifen können Sie Anweisungen ausführen und so oft wiederholen, wie die Bedingung erfüllt (wahr) ist. Wenn die Bedingung nicht erfüllt (falsch) ist, wird die Schleifenbearbeitung beendet und mit der Anweisung fortgesetzt, die direkt nach der `do/while`-Schleife steht.

Die Anweisungen werden auf jeden Fall mindestens einmal durchlaufen, bevor die Bedingung überprüft wird.

```
[label:] do block while ( bedingung );
```

label Spezifiziert ein Label, auf das man sich mit `break` oder `continue` beziehen kann.

block Eine Anweisung oder ein Anweisungsblock.

bedingung
Ausdruck, der nach jedem Schleifendurchlauf überprüft wird. Der Ausdruck wird in den Datentyp `boolean` konvertiert und solange dies `true` liefert, wird die Schleife fortgesetzt.

Beschreibung

Wenn *block* eine `break`- oder `continue`-Anweisung enthält, wird verfahren, wie in Abschnitt „while-Schleife“ auf Seite 299 beschrieben.

Beispiel

```
a = 0;
arr = new Array;
do {
    arr[a] = a;
}
while ( a++<100 );
```

9.4.4 for-Schleife

Mit der `for`-Schleife können Sie eine Anweisung oder einen Anweisungsblock solange ausführen lassen wie die Schleifenbedingung zutrifft.

```
[label:] for ( [ init ]; [ bedingung ]; [ update ] ) block
```

label spezifiziert ein Label.
Die Angabe eines Labels ermöglicht es, sich in einer `break`- oder `continue`-Anweisung auf dieses Label zu beziehen.

init *init* kann ein Ausdruck oder eine Variablendeklaration sein. Diese entspricht einer Deklaration vor der `for`-Schleife.
init wird zu Beginn einmal ausgeführt. Typischerweise wird hiermit ein Schleifenzähler initialisiert.

Die folgenden Schritte werden wiederholt ausgeführt:

bedingung
bedingung wird ausgewertet und in den Datentyp `boolean` konvertiert. Ist das Ergebnis der Wert `true` oder ist *bedingung* nicht vorhanden, wird *block* ausgeführt. Ist das Ergebnis `false`, so wird die Schleife abgebrochen.

update Ausdruck, der nach Abarbeiten des Schleifenkörpers ausgewertet wird. Hier wird typischerweise der Schleifenzähler verändert.

block Anweisung oder Anweisungsblock als Schleifenkörper.

Beschreibung

Enthält *block* eine `break`-Anweisung, so wird wie folgt verfahren:

– `break` ohne *label* angegeben:

  Die Schleife wird abgebrochen und die Bearbeitung wird fortgesetzt mit der Anweisung, die direkt auf die `for`-Schleife folgt.

– `break` mit *label* angegeben:

  Die nächstliegende Anweisung, die mit dem Label *label* markiert ist, wird abgebrochen und die Bearbeitung mit der Anweisung fortgesetzt, die unmittelbar auf die abgebrochene Anweisung folgt.

Enthält *block* eine `continue`-Anweisung, so wird wie folgt verfahren:

– `continue` ohne *label* angegeben:

  Die Ausführung von *block* wird abgebrochen und *update* ausgeführt. *bedingung* wird erneut ausgewertet und die `for`-Schleife entsprechend fortgesetzt.

– `continue` mit *label* angegeben:

  Die Ausführung von *block* wird abgebrochen. *bedingung* der mit dem Label *label* markierten Schleife wird, nachdem *update* ausgeführt wurde, erneut ausgewertet und die mit *label* markierte Schleife entsprechend fortgesetzt.

Beispiel 1

Im folgenden Beispiel wird ein Zähler `i` definiert und mit dem Wert 1 initialisiert. Bei jedem Schleifendurchgang wird der Zähler um 1 erhöht. Ist sein Wert größer als 100 und damit die Bedingung der zweiten Anweisung nicht mehr erfüllt, wird die Schleife beendet.

Mit Hilfe des Schleifenzählers wird bei jedem Schleifendurchgang das Quadrat des aktuellen Werts gebildet. Das Ergebnis wird im aktuellen HTML-Dokument ausgegeben (`document.write`-Methode, siehe Abschnitt „Methode write / writeln“ auf Seite 164).

```
for(i = 1; i <= 100; i++)
    {
     var x = i * i;
     document.write("<br>The square of " + i + " is " + x);
    }
```

Beispiel 2

Mit der folgenden `for`-Schleife können Sie alle Felder des aktuellen Bildschirms lesen (ist kein Feld mehr vorhanden, wird `$END` zurückgegeben). Mit der `if`-Verzweigung können Sie prüfen, ob der Endbenutzer Werte geändert hat und diese den entsprechenden Host-Objekten des OSD-/MVS-Kommunikationsobjekts zuweisen.

```
...
for (element = WT_HOST.con.$FIRST.Name;
     element != "$END";
     element = WT_HOST.con.$NEXT.Name)
   if ( WT_POSTED[element] != WT_HOST.con[element].Value )
     WT_HOST.con[element].value = WT_POSTED[element];
...
```

Beispiele für `for`-Anweisungen mit Label, die Sprungziel einer Anweisung `break` *label* bzw. einer Anweisung `continue` *label* sind, finden Sie in Abschnitt „break-Anweisung“ auf Seite 309 bzw. Abschnitt „continue-Anweisung“ auf Seite 312.

9.4.5 for/in-Schleife

Diese Anweisung ist eine Schleife über die Attribute eines Objekts (*object*). Das können z.B. die name/value-Paare sein, die als Attribute des Posted-Objekts vom Browser empfangen werden, oder die Elemente eines Arrays. Die Variable *name* speichert das aktuelle Attribut von *object*.

```
[label:] for ( [var] name in object ) block
```

label spezifiziert ein Label.
Die Angabe eines Labels ermöglicht es, sich in einer `break`- oder `continue`-Anweisung auf dieses Label zu beziehen.

name Variable, welcher der Name eines Objekt-Attributs oder der Name/Index eines Array-Elements zugewiesen wird, d.h. die Eigenschaft eines Objekts (siehe Abschnitt „Object-Klasse“ auf Seite 186). Die Variable *name* wird implizit deklariert. Rufen Sie die `for`-Schleife in einer Funktion auf und wollen die Variable lokal anlegen, müssen Sie `var` *name* verwenden siehe Abschnitt „var-Anweisung“ auf Seite 315).

object Ausdruck. Liefert das Objekt, dessen Attribute in der Schleife durchlaufen werden.

block spezifiziert die Anweisung(sfolge), die für jede Eigenschaft ausgeführt werden soll

Beschreibung

Der Ausdruck *object* wird ausgewertet und - falls nötig - in den Datentyp `object` konvertiert. Die Namen der Eigenschaften werden als String der Variablen *name* zugewiesen. Nach jeweils einer solchen Wertzuweisung wird *block* ausgeführt, z.B. werden die Namen der Objekteigenschaften ausgegeben.

Enthält *block* eine `break`-Anweisung, so wird wie folgt verfahren:

– `break` ohne *label* angegeben:

  Die Schleife wird abgebrochen. Die Bearbeitung wird fortgesetzt mit der Anweisung, die direkt auf die `for/in`-Schleife folgt.

– `break` mit *label* angegeben:

  Die nächstliegende Anweisung, die mit dem Label *label* markiert ist, wird abgebrochen. Die Bearbeitung wird mit der Anweisung fortgesetzt, die unmittelbar auf die abgebrochene Anweisung folgt.

Wird während der Ausführung von *block* eine `continue`-Anweisung durchlaufen, so wird die Ausführung von *block* abgebrochen und wie folgt verfahren:

– `continue` ohne *label* angegeben:

  Es wird der Variablen die nächste Eigenschaft des Objekts zugewiesen und die `for/in`-Schleife entsprechend fortgesetzt.

– `continue` mit *label* angegeben:

  Es wird auf den Anfang der mit *label* bezeichneten Schleife positioniert; dort wird der Schleifenvariablen der nächste Wert zugewiesen und diese Schleife damit fortgesetzt.

Beispiel 1

In diesem Beispiel sammelt die Funktion alle Umsatzzahlen (`turnover`) des 1.Quartals und gibt sie in einer HTML-formatierten Zeichenkette aus.

```
...
function properties(turnover) {
     for (var i in turnover) {
          document.write(turnover[i] + "<p>")
          }
}
...
```

Beispiel 2

Diese `for/in`-Schleife gibt die Namen aller Host-Objekte und die Eigenschaft HTMLValue aus. Das Kommunikationsobjekt für die Verbindung heißt „Verb“.

```
for (obj in WT_HOST.Verb)
{
   document.write (obj + ":" + obj.HTMLValue +"<br>");
}
```

Beispiel 3

Enthält eine HTML-Seite eine Liste mit Namen, aus der der Benutzer mehrere Einträge auswählen kann, so liefert WebTransactions die ausgewählten Einträge in einem Array zurück. Ist nur ein Eintrag ausgewählt, so wird ein String zurückgegeben. Die folgende `for`-Schleife gibt alle ausgewählten Namen aus. Wurde nur ein Eintrag ausgewählt, so ist `multi` kein Array, und die Schleife wird nicht durchlaufen.

```
b = "Namen:";
if (WT_POSTED.multi)
{
     for ( a in WT_POSTED.multi )
          b += " " + WT_POSTED.multi[a];
     if ( b == "Namen:" )
          b += " " + WT_POSTED.multi;
}
else
{
     b+ = "Nothing selected!";
}
```

9.4.6 switch-Anweisung

Diese Anweisung dient der Auswahl unter mehreren Fällen (`case`). Eine oder mehrere Anweisungen werden ausgeführt, falls der Wert eines Ausdrucks, der als Eingangsbedingung geprüft wird, mit dem Wert des Ausdrucks eines Falls (`label`) übereinstimmt. Die Abarbeitung der einzelnen Fälle kann mit der Anweisung `break` abgebrochen werden.

```
switch (ausdruck1)
{
      case ausdruck2: anweisung1 ... [break [label:];]
      case ausdruck2: anweisung1 ... [break [label:];]
      ...
      [default:] anweisung2 ...
}
```

ausdruck1
: Beliebiger Ausdruck (siehe Kapitel „Ausdrücke und Operatoren“ auf Seite 69)

ausdruck2
: Ausdruck, dessen Wert mit dem Wert von *ausdruck1* verglichen wird

anweisung1
: Eine oder mehrere Anweisungen, die bei Gleichheit der Ausdrücke *ausdruck1* und *ausdruck2* abgearbeitet werden.

anweisung2
: Eine oder mehrere Anweisungen, die ausgeführt werden, falls kein *ausdruck2* gleich *ausdruck1* ist.

break [*label*]
: Mit `break` wird die Abarbeitung der `switch`-Anweisung abgebrochen. Die Bearbeitung wird mit der Anweisung fortgesetzt, die unmittelbar auf die `switch`-Anweisung folgt.

  Mit `break` *label* wird die Abarbeitung der nächsten, die `switch`-Anweisung enthaltenden Anweisung abgebrochen, die mit dem Label *label* markiert ist. Die Bearbeitung wird mit der Anweisung fortgesetzt, die unmittelbar auf die abgebrochene Anweisung folgt.

Beschreibung

Der Ausdruck *ausdruck1* wird ausgewertet und mit den Ausdrücken *ausdruck2* hinter allen Schlüsselwörtern `case` verglichen - auf numerische, lexikalische Gleichheit usw. Die Ausführung des Scripts wird mit dem Fall fortgesetzt, bei dem der Wert des *ausdruck2*-Ausdrucks mit *ausdruck1* übereinstimmt. Es werden dieser Fall und alle darauffolgenden Fälle abgearbeitet.

Wird kein gleicher Ausdruck gefunden und sind hinter dem Schlüsselwert `default`-Anweisungen spezifiziert, dann werden diese abgearbeitet. Ist `default` nicht spezifiziert, so findet in der `switch`-Anweisung keine Aktion statt.

Beispiel 1

Mit der folgenden `switch`-Anweisung können Sie den Wert von Benutzereingaben z.B. in einer Drop-Down-Liste auf die Host-Objekte abbilden. `host` zeigt auf die Host-Datenobjekte der Verbindung.

```
switch (WT_POSTED.COUNTRY)
{
   case "Belgium": host.Country.Value= "1"; break;
   case "France": host.Country.Value= "2"; break;
   case "Germany": host.Country.Value= "3"; break;
   case "Greece": host.Country.Value= "4"; break;
   default: host.Country.Value= "0";
}
```

Beispiel 2

Abhängig vom geposteten Wert in `FORMSTYLE` wird `WT_SYSTEM.STYLE` mit einem Wert versorgt.

```
   WT_SYSTEM.TravStyle = "change";
   switch (WT_POSTED.FORMSTYLE)
   {
      case "Forms":  WT_SYSTEM.STYLE = "forms";
          break;
      case "Simple":  WT_SYSTEM.STYLE = "simple";
          break;
      case "Green":  WT_SYSTEM.STYLE = "green";
          break;
      case "Enhanced":  WT_SYSTEM.STYLE = "enhanced";
          break;
      default:  WT_SYSTEM.TravStyle = "NoChange";
      //WT_SYSTEM.STYLE bleibt unveraendert!
          break;
   }
```

9.4.7 break-Anweisung

Mit der `break`-Anweisung können Sie eine Schleife oder eine `switch`-Anweisung vorzeitig beenden. Der vorzeitige Abbruch einer Schleife bzw. `switch`-Anweisung kann z.B. als Reaktion auf Fehlerfälle sinnvoll sein.

```
break [label]
```

Beschreibung

Je nachdem ob Sie die `break`-Anweisung ohne oder mit einem Label *label* angeben, hat die `break`-Anweisung folgende Funktionalität:

– `break` beendet die Abarbeitung der innersten `for-` , `for/in-`, `while-`, `do/while-` oder `switch`-Anweisung und übergibt die Kontrolle an die Anweisung direkt nach der Schleife bzw. `switch`-Anweisung.

– `break` *label* bricht die nächstliegende Anweisung ab, die mit dem Label *label* markiert ist, und setzt die Bearbeitung mit der Anweisung fort, die unmittelbar auf die abgebrochene Anweisung folgt.

Wenn die `break`-Anweisung außerhalb von `for-`,`for/in-`, `while-`, `do/while-` oder `switch`-Anweisungen verwendet wird, wird ein Laufzeitfehler gemeldet.

Beispiel 1

Im folgenden Beispiel wird die `while`-Schleife beim vierten Durchlauf abgebrochen. Die Funktion `test` liefert als Ergebnis den Wert der Multiplikation `3 *x`.

```
function test(x) {
   var a = 0;
   while (a < 6) {
     if (a == 3)
        break;
     a ++;
   }
   return a * x;
}
```

Beispiel 2

Soll der Benutzer in einer HTML-Seite mehrere Eingabefelder mit Werten versorgen, so können Sie die Vollständigkeit der Eingabe mit der folgenden Schleife überprüfen.

```
bInputComplete = true;
for(field in WT_POSTED)
{
        if WT_POSTED[field] = ""  //Benutzer hat nichts eingegeben
        {
                bInputComplete=false;
                break;
        }
}
if (bInputComplete)
...
```

Beispiel 3

Falls `typeof a[i][j] == 'undefined'`, wird der Programmfluss mit der Anweisung *anweisung* fortgesetzt.

```
start: for ( i=0; i<=99; i++ )
{
      j=0;
      nextloop: for( ; j<=99; j++ )
      {
               // beende äussere Schleife, wenn Element nicht existiert
               if ( typeof a[i][j] == 'undefined' )
                    break start;
      }
}
anweisung;
```

9.4.8 continue-Anweisung

Mit dieser Anweisung können Sie die Ausführung der Anweisungen in einer `for`- , `for/in`-, `while`- oder `do/while`-Schleife beenden und mit dem nächsten Schleifendurchlauf fortfahren.

```
continue [label]
```

Beschreibung

Die `continue`-Anweisung unterbricht die Ausführung der innersten sie enthaltenden `for`-, `for/in`-, `while`- oder `do/while`-Schleife und setzt die Schleifenausführung wie folgt fort:

– Falls kein Label spezifiziert ist, setzt die `continue`-Anweisung die innerste sie enthaltene Schleife mit dem nächsten Iterationsschritt fort.

– Falls ein Label *label* spezifiziert ist, setzt `continue` *label* die mit dem Label *label* markierte Schleife mit dem nächsten Iterationsschritt fort.

Bei einer `while`-Schleife wird die Abarbeitung mit der Auswertung der Schleifenbedingung fortgesetzt, bei einer `for`-Schleife mit der Abarbeitung von *update*. Wenn die Schleifenbedingung wahr ergibt, wird die Schleife erneut durchlaufen. Bei einer `do/while`-Schleife wird die Endebedingung erneut geprüft. Bei einer `for/in`-Schleife wird der nächste Durchlauf gestartet.

Wenn die `continue`-Anweisung außerhalb von `for`-, `for/in`-, `while`- oder `do/while`-Schleifen verwendet wird, wird ein Laufzeitfehler gemeldet.

Beispiel 1

Sie können die Verarbeitung von Feldern, die der Endbenutzer nicht versorgt hat (Wert `null`) übergehen, indem Sie die Elemente des Posted-Objekts auf `null` abfragen:

```
for(i in WT_POSTED){
     if (WT_POSTED[i] == null)
        continue;
        ... // hier folgt die weitere Verarbeitung
}
```

Beispiel 2

In der folgenden Schleife werden die Benutzereingaben abgearbeitet. Alle leeren Eingaben werden ignoriert.

```
nCount=3;
var eingabe;
for (field in WT_POSTED)
{
   if (WT_POSTED[field] = "")  //Benutzer hat nichts eingegeben
      continue;
   eingabe[i++]=WT_POSTED[field];
      ...
}
```

Beispiel 3

In diesem Beispiel wird zu einem Städtenamen das Land ermittelt, in dem die genannte Stadt liegt.

```
cities = new Object;
cities.Germany = new Array ("Aachen", "Bonn",  "Essen", "Frankfurt",
"Muenchen, "Wuerzburg");
cities.GreatBritain = new Array
("Birmingham","Exeter","Glasgow","Hull","London","Warwick");
cities.France = new Array ("Bourges","Cannes","Nantes", "Orleans", "Paris",
"Rennes");
function where(name)
{
  outer: for (country in cities)
  {
      for (i=0; i< cities[country].length; i++)
      {
          if (cities[country][i] == name)
              return country;
          // Wenn der Wert des aktuellen Array-Elements lexikalisch größer
          // ist, als der gesuchte Städtename, kann die Bearbeitung mit
          // dem nächsten Land fortgesetzt werden.
          else if (cities[country][i] > name)
             continue outer;
      }

  }
  return "";
}
```

9.5 return-Anweisung

Mit `return` gibt eine Funktion das Ergebnis ihrer Bearbeitung zurück an den aufrufenden WebTransactions-Anwendungsteil.

```
return [ retValue ]
```

retValue
Ausdruck, der als Returnwert übergeben wird.

Beschreibung

Die `return`-Anweisung beendet die Ausführung einer Funktion. Ist *retValue* vorhanden und ein Ausdruck, so wird dieser ausgewertet und sein Wert zurückgeliefert. Steht die `return`-Anweisung außerhalb von Funktionen, wird ein Laufzeitfehler gemeldet.

Beispiel 1

Die folgende Funktion berechnet die Standard-Arbeitsstunden eines Mitarbeiters abhängig von der Zahl der Arbeitstage in einem Monat und von der vereinbarten täglichen Arbeitszeit. Das Ergebnis kann als Wert oder als Ausdruck spezifiziert werden.

```
...
function  arbeitszeit(tage,stunden)
{
     arbeitsstunden = tage * stunden;
     return (arbeitsstunden);
}
...
```

oder

```
function  arbeitszeit(tage,stunden)
{
     return (tage * stunden);
}
```

Beispiel 2

Der Variablen `result` wird der Aufruf der Funktion `square` mit dem Parameter `3` zugewiesen. Mit der `document.write`-Methode wird das Ergebnis ausgegeben.

```
...
function square(x) {
   var y=x * x;
   return y ;
}
result= square(3);
document.write("Das Quadrat von 3 ist " + result);
...
```

9.6 var-Anweisung

Sie können eine Variable durch einfache Wertzuweisung anlegen (implizit) oder mit dem Schlüsselwort `var` deklarieren (explizit). Diese beiden Variablendeklarationen unterscheiden sich nur innerhalb von Funktionen. Hier legt eine explizite Deklaration eine lokale Variable an, die nur in der Funktion zur Verfügung steht. Eine implizite Variablendeklaration dagegen legt sowohl innerhalb als auch außerhalb von Funktionen globale Variablen an, die für das gesamte Template (bis zum letzten OnReceive-Bearbeitungsschritt) gültig sind.

`var { identifier | identifier=value } [{ ,identifier | ,identifier=value }...]`

`var` Schlüsselwort für eine explizite Variablendeklaration

identifier
Name der Variablen. Für die Bildung von Variablennamen gelten die Regeln für Bezeichner (siehe Abschnitt „Bezeichner“ auf Seite 44).

value Zuweisung eines Ausgangswerts, d.h. Initialisierung der Variablen. Die Variable erhält dann den Wert und den entsprechenden Typ. Geben Sie keinen Wert an, wird automatisch der Wert `undefined` zugewiesen. Das bedeutet, dass nichts gespeichert ist.

Beschreibung

Eine explizite Variablendeklaration ist obligatorisch, wenn Sie eine lokale Variable innerhalb einer Funktion deklarieren wollen, da eine einfache Wertzuweisung eine globale Variable anlegt.

Sie können mehrere Variablen auf einmal deklarieren. Dazu trennen Sie die Variablennamen durch Kommata.

JavaScript kennt nur „loose typing“, d.h. keine Festlegung des Variablentyps bei der Deklaration. Datentypen werden automatisch konvertiert, wenn die Ausführung dies erfordert.

Beispiel 1

```
...
function set1()
{
     var x = 17;     // lokale Variablendeklaration
}
function set2()
{
     document.write("Der Wert von x ist " + x);
                    // sinnlos, da x in dieser Funktion nicht bekannt,
                    // Ausgabe: Der Wert von x ist undefined
}
```

Beispiel 2

```
function set1()
{
     x = 17;          //globale Variablendeklaration
}
function set2()
{
     document.write("Der Wert von x ist " + x);
                    // Ausgabe: Der Wert von x ist 17
}

set1();
set2();
...
```

Beispiel 3

```
function berechne(uebergabe1, uebergabe2)
{
     var ergebnis = uebergabe1 + uebergabe2;
     return ergebnis;
}
```

Das Ergebnis der Funktion `berechne` hängt also von den Datentypen der Übergabeparameter ab. Sind beide vom Typ `number`, so ist auch das Ergebnis vom Typ `number`. Ist aber einer der beiden Parameter vom Typ `string`, so wird ein String zurückgegeben.

`berechne(4,7);` liefert als Ergebnis 11

`berechne("4",7);` liefert als Ergebnis hingegen "47"

9.7 function-Anweisung

Mit Funktionen können Sie Anweisungsfolgen zusammenfassen, die mehrmals in einem Template verwendet werden. Durch die Definition einer Funktion verwenden Sie eine einfache Anweisung, um komplexe Aktionen auszulösen, ohne die gleichen Anweisungen mehrfach schreiben zu müssen.

Mit der `function`-Anweisung können Sie Funktionen selbst definieren im Gegensatz zu vordefinierten Funktionen, die Sie ohne vorherige Definition aufrufen können. Funktionen von eingebauten Klassen sind in Kapitel „Eingebaute Klassen und Methoden“ auf Seite 127 beschrieben.

```
function [ identifier ] ( [ parameter [, parameter ]... ] ) { [ statement... ] }
```

identifier

Name, mit dem die Funktion aufgerufen, d.h. ausgeführt wird. Für *identifier* gelten die Regeln für Bezeichner (siehe Abschnitt „Bezeichner“ auf Seite 44).

Sie können Funktionen auch anonym verwenden, indem Sie die Definition der Funktion direkt einer Variablen zuweisen.

Beispiel

```
erg = function(x)
{
     x += 42;
}
```

parameter

Werte, die vom aufrufenden WebTransactions-Anwendungsteil an die Funktion zur Verarbeitung übergeben werden. *parameter* ist ein einfacher Bezeichner (siehe Abschnitt „Bezeichner“ auf Seite 44). Übergeben wird der Wert der Variablen, nicht die Adresse der Variablen („call-by-value“). Wenn die Funktion den Wert des Parameters ändert, gilt diese Änderung nicht im aufrufenden WebTransactions-Anwendungsteil.

Beispiel 1

Wird eine Funktion mit weniger Parametern aufgerufen als bei der Definition angegeben sind, so sind die übrigen Parameter undefiniert.

```
function myFunc(a,b)
{
   document.write(a + " " + b);
}
myFunc("Hallo"); // liefert die Ausgabe: Hallo undefined
```

Beispiel 2

Geben Sie beim Aufruf einer Funktion mehr Parameter an als bei der Deklaration, so können Sie auf die zusätzlichen Parameter nicht über Namen zugreifen. Es besteht allerdings innerhalb der Funktion die Möglichkeit, auf solche Parameter über das `arguments`-Array zuzugreifen.

Sie definieren die Funktion `myfunc` folgendermaßen:

```
myfunc(wort1,wort2)
{
   ...
}
```

und rufen sie folgendermaßen auf

```
myfunc("hallo","schöne neue","Welt");
```

so können Sie innerhalb der Funktion auf den 1. Parameter mit `wort1` oder `arguments[0]` zugreifen, auf den dritten Parameter aber lediglich mit `arguments[2]`.

statement ...

Eine Folge von Anweisungen definiert die Ablauflogik der Funktion.

Beschreibung

Die Deklaration gibt der Funktion einen Namen und spezifiziert die Anweisungen, die ausgeführt, und die Parameter, die an die Funktion übergeben werden sollen. Sie rufen eine Funktion auf, indem Sie den Namen der Funktion angeben und dahinter - in runde Klammern eingeschlossen - die Parameter. Werden keine Parameter übergeben, sind die runden Klammern leer.

Sie dürfen `function`-Anweisungen innerhalb von Funktionen verwenden. Die `function`-Anweisung erzeugt dann eine lokale Variable vom Typ `function` in der umgebenden Funktion.

Beispiel

```
function f()
{
  function g(x)
  {
    return 6*x;
  }
  return g(7);
}
```

Die Funktion `g` ist außerhalb der Funktion `f` nicht mehr gültig. Die `function`-Anweisung ist damit gleichwertig mit der Zuweisung eines Funktionsliterals an eine lokale Variable (siehe folgender Abschnitt):

```
function f()
{
  var g = function (x){return 6*x;};
  return g(7);
}
```

Funktionen können nicht nur Werte entgegennehmen, sondern auch zurückliefern (siehe Abschnitt „return-Anweisung“ auf Seite 313).

Beispiele für die Deklaration und den Aufruf von Funktionen finden Sie im Abschnitt „return-Anweisung“ auf Seite 313.

9.8 Funktionsliteral

Neben der `function`-Anweisung (siehe Seite 317), wird die anonyme Verwendung von Funktionen ohne Identifier – das Funktionsliteral – unterstützt.

Ein Funktionsliteral kann als Teil eines Ausdrucks an sehr vielen Stellen im Skript und sogar im Auswertungsoperator verwendet werden. Als Wert liefert es eine Funktion zurück, die Sie beispielsweise einer Variablen zuweisen und diese dann ausführen können.

```
function ( [ parameter [, parameter  ]... ] ) { [ statement ... ] }
```

parameter …

Werte, die vom aufrufenden WebTransactions-Anwendungsteil an die Funktion zur Verarbeitung übergeben werden. *parameter* ist ein einfacher Bezeichner (siehe Abschnitt „Bezeichner“ auf Seite 44). Übergeben wird der Wert der Variablen, nicht die Adresse der Variablen („call-by-value“). Wenn die Funktion den Wert des Parameters ändert, gilt diese Änderung nicht im aufrufenden WebTransactions-Anwendungsteil.

statement …

Eine Folge von Anweisungen definiert die Ablauflogik der Funktion.

Beispiel

```
sq = function(x){return x*x;};
a=sq(5);
```

9.9 with-Anweisung

Diese Anweisung legt ein Objekt als Standard-Objekt für Anweisungen fest. So können Sie Anweisungen mit Objektverweisen verkürzt schreiben.

```
with ( object ) block
```

object Ausdruck, der ausgewertet und in ein Objekt konvertiert wird.

block Einzelne Anweisung oder Anweisungsblock

Beschreibung

Innerhalb von *block* werden Namen relativ zu diesem Objekt interpretiert. Geschachtelte `with`-Anweisungen werden von innen nach außen ausgewertet.

Beispiel

Wenn Sie mehrere Attribute des System-Objekts mit geposteten Werten versorgen wollen, ermöglicht die `with`-Anweisung eine kompakte Schreibweise:

```
...
with(WT_SYSTEM)
{
COMMUNICATION_ERROR_FORMAT = "wtstart";
STYLE                      = WT_POSTED.STYLE;
LANGUAGE                   = WT_POSTED.LANGUAGE;
TIMEOUT_APPLICATION        = WT_POSTED.TIMEOUT_APPLICATION;
TIMEOUT_USER               = WT_POSTED.TIMEOUT_USER;
...
}
```

9.10 Fehlerbehandlung durch Ausnahmen (Exceptions)

Die in WebTransactions auftretenden Fehler lassen sich folgenden Fehlerklassen zuordnen:

- Syntaxfehler (z.B. falsches Schlüsselwort)
- Laufzeitfehler (z.B. Objekt ist nicht vorhanden, Array-Index nicht im Bereich, etc.)
- Kommunikationsfehler (z.B. Verbindungsaufbau nicht erfolgreich)
- globale Fehler (z.B. Template nicht vorhanden)

Die Fehlerbehandlung durch Ausnahmen (Exceptions) vereinfacht wesentlich die Behandlung von Fehlern, die während der Ausführung eines WTScripts auftreten (Laufzeitfehler). Damit lassen sich schwerwiegende Programmfehler, aber auch andere außergewöhnliche Situationen, durch eine entsprechende Ausnahmenbehandlung abfangen.

9.10.1 Error-Objekt

Grundsätzlich wird bei jedem Laufzeitfehler eine Ausnahme (Exception) geworfen. Die Ausnahme ist ein Objekt vom Typ `Error`.

Das `Error`-Objekt ist wie folgt aufgebaut:

Attributname	**Inhalt**
type	String, der den Fehlertyp kennzeichnet. Es gibt folgende Fehlertypen: – "CommunicationError" – "GlobalError" – "RTSError" – "JavaError"
msgNo	Nummer der Fehlermeldung aus der Message-Datei
text	vollständiger Text der Fehlermeldung aus der Message-Datei
position	Position, an der der Fehler aufgetreten ist (optional)
position.line	Zeile, in der der Fehler aufgetreten ist
position.col	Spalte, in der der Fehler aufgetreten ist
position.path	Name des Templates, in dem der Fehler aufgetreten ist

Somit können Sie als Programmierer diese Exceptions per `try`/`catch`-Block auffangen und eine eigene Fehlerbehandlung (Exception-Handling) implementieren. Wenn Ausnahmen nicht aufgefangen werden, tritt die Default-Funktionalität in Kraft: Die Fehlermeldungen werden dann mit Hilfe des Error-Templates ausgegeben.

Wenn innerhalb eines Skript-Teils eine Ausnahme geworfen wird, wird die Abarbeitung des Skript-Teils abgebrochen und kein Ergebnis zurückgeliefert. Die Ausführung aller aufrufenden Skript-Teile wird ebenfalls abgebrochen, bis der Programmfluss einen Skript-Abschnitt erreicht, der die Ausnahme abfängt und verarbeitet. Nach der Abarbeitung eines solchen Skript-Teils, d.h. nach der Behandlung der Ausnahme, wird die Ausnahme zurückgesetzt und hat keinen Einfluss auf den weiteren Programmfluss.

Es sind zwei Gruppen von Ausnahmen zu unterscheiden:

– implizite Ausnahmen
– explizite Ausnahmen

Implizite Ausnahmen werden vom WebTransactions-Laufzeitsystem bei Laufzeitfehlern als Instanzen der `Error`-Klasse geworfen. Explizite Ausnahmen sind im nachfolgenden Abschnitt beschrieben.

Error-Objekt in einem dynamisch erzeugten Skript

Ein Syntax-Fehler kann nicht nur im „normalen“ Template auftreten, sondern auch in einem Skript, das mit den Funktionen `eval()` oder `setTimeout()` erzeugt wurde. In diesem Fall werden zusätzliche Informationen erzeugt, um die Fehlersuche zu erleichtern:

- Das Error-Objekt erhält am Unterobjekt `Position` die Attribute `strLine`, `strColumn` und `strText`.

  `strText` enthält den String, der dynamisch geparsed wurde, `strLine` und `strColumn` beziehen sich auf diesen String.

- Im WebTransactions-Trace werden die Werte für `strLine`, `strColumn` und `strText` ebenfalls ausgegeben.

- Falls im Error-Template die Platzhalter `%(strLine)` , `%(strColumn)` und/oder `%(strText)` enthalten sind, werden diese bei lokalisierbaren Fehlern mit den aktuellen Werten versorgt.

9.10.2 Explizite Ausnahmen

Explizite Ausnahmen können Sie in Ihrem WTScript mit der `throw`-Anweisung werfen:

`throw` *expression*;

Mit *expression* spezifizieren Sie den Wert der Ausnahme: Wahlweise können Sie ein Literal oder die Instanz einer Klasse angeben.

Beispiel 1: Ausnahmen sind Literale

```
throw 42;               // generiert eine Ausnahme mit dem number Wert 42
throw "zweiundvierzig"; // generiert eine Ausnahme mit dem Stringwert "zweiundvierzig"
throw true;             // generiert eine Ausnahme mit dem boolean Wert true
```

Beispiel 2: Ausnahme ist ein Objekt

```
function UserException (message)   // Konstruktor für Objekt vom Typ UserException
{
   this.message=message;
   this.name="UserException";
}
myUserException=new UserException("Wert ist ungültig!");
throw myUserException;          // generiert eine Ausnahme vom Objekttyp UserException
```

9.10.3 Ausnahmenbehandlung

Bis zur Version 6 von WebTransactions war ein `try`-Block ohne folgenden `catch`-Block möglich. Ab der Version 7 erscheint in einem solchen Fall eine Fehlermeldung.

Mit der Ausnahmenbehandlung können Sie vordefinierte implizite oder explizite Ausnahmen abfangen und eine entsprechende Fehlerbehandlung einleiten. Hierfür stellt Ihnen WTScript die folgenden Sprachmittel zur Verfügung:

– `try`-Block
– `catch`-Block
– `finally`-Block

Der `try` ... `finally`-Block ist wie folgt aufgebaut:

```
try { zu sichernder Code-Abschnitt } ---------------------------------------- (1)
[catch ( identifier if condition ){}]  // mehrfacher catch Block  ----------- (2)
[catch ( identifier if condition ){}]
...
[catch ( identifier ) {}]          // einzelner catch Block
[finally{}] ------------------------------------------------------------ (3)
```

(1) Code-Abschnitt sichern mit dem try-Block

Den Code-Abschnitt in Ihrem Skript, für den Sie bestimmte Ausnahmen abfangen wollen, fassen Sie in einem `try`-Block zusammen.

(2) Fehlerbehandlung durchführen mit dem catch-Block

Unmittelbar auf den `try`-Block müssen ein oder mehrere `catch`-Blöcke folgen. In den `catch`-Blöcken führen Sie die eigentliche Fehlerbehandlung für Ausnahmen durch, die bei der Abarbeitung des `try`-Blocks geworfen werden. *identifier* bezeichnet innerhalb eines `catch`-Blocks die zu behandelnde Ausnahme und existiert nur für die Zeitspanne der Abarbeitung dieses `catch`-Blocks.

Man unterscheidet zwischen einem einzelnen `catch`-Block und einem mehrfachen `catch`-Block. Da der `try`-Block verschiedene Arten von Ausnahmen werfen kann, können Sie darauf mit mehrfachen `catch`-Blöcken reagieren, die die einzelnen, verschiedenen Ausnahmen abfangen. An den mehrfachen `catch`-Block können Sie einen `catch`-Block ohne Bedingung (bedingungslosen `catch`-Block) anhängen, der alle unerwarteten Ausnahmen abfängt.

(3) finally-Block

Im `finally`-Block fassen Sie alle Anweisungen zusammen, die in jedem Fall ausgeführt werden sollen, unabhängig davon, ob eine Ausnahme geworfen und/oder abgefangen wurde oder nicht.

Geworfene Ausnahmen beenden immer den Ausführungskontext einer Funktion/Methode oder eines Konstruktors. Eine `with`-Anweisung restauriert immer das Standardobjekt, gleichgültig ob eine Ausnahme geworfen wurde oder nicht.

Beispiel

```
function throwException(i, j)
{
  if ( i == j )
    throw true;
  if ( Math.round(100 * ( i + j ) ) == 42 )
    throw 42;
  if ( Math.round(100 * ( i + j ) ) == 21 )
    throw "einundzwanzig";
  else {
    excObj = new Object();
    excObj.type = "any sum";
    excObj.sum = Math.round(100 * ( i + j );
    throw excObj;
  }
}

try {
   for ( i=0; i<10; i++ )
   {
     throwException(Math.random(), Math.random());
   }
}

catch ( exc if exc == 42 )
{
  document.write("<BR>Die Summe aus " + i + " + " + j + " = " + exc );
}
catch ( exc if exc == "einundzwanzig" )
{
  document.write("<BR>Die Summe aus " + i + " + " + j + " = " + exc );
}
catch ( exc if exc.type == "any sum" )
{
```

```
  document.write("<BR>Die Summe aus " +exc.i+ " + " +exc.j+ " = " +
  exc.sum );
}
catch ( exc )
{
  document.write("<BR>"+ i + " und " + j + " sind gleich. That's " + exc );
}
finally
{
  document.write("<BR>nach allen Exceptions");
}
```

10 Klassen-Templates (*.clt)

Mit Klassen-Templates können Sie die Auswertung gleichartiger Objekte, z.B. von gleichartigen Host-Datenobjekten, automatisieren. Statt für jedes Host-Datenobjekt dieselben Anweisungen immer wieder zu schreiben, definieren Sie entsprechende Klassen-Templates. Ein Klassen-Template wird in einer Datei mit dem Suffix `.clt` (class template) abgelegt.

Klassen-Templates werden ausgewertet. Das Ergebnis der Auswertung wird dann anstatt des Klassen-Template-Aufrufs in den Ausgabestrom des aufrufenden Templates eingefügt. In Klassen-Templates können Sie die gleichen Sprachmittel einsetzen wie in allen anderen Templates. Das rufende Objekt wird im Klassen-Template mit `WT_THIS` referenziert, auf seinen Namen wird mit `##WT_THIS#` oder `WT_THIS.toString()` zugegriffen (siehe Abschnitt „WT_THIS - Zugriff auf das rufende Objekt“ auf Seite 330). Beim Interpretieren wird jedes Vorkommen von `WT_THIS` durch das rufende Objekt ersetzt.

Klassen-Templates sind wie die normalen Templates im Standardfall im Verzeichnis *basedir*`/config/forms` abgelegt. Auch für Klassen-Templates können Sie unterschiedliche Stile und unterschiedliche Sprachen realisieren, wobei dieselbe Suchstrategie wie für normale Templates angewandt wird.

Es gibt zwei verschiedene Arten Klassen-Templates aufzurufen:

– implizit
– explizit

Impliziter Aufruf

Implizit wird ein Klassen-Template aufgerufen, wenn auf ein Host-Datenobjekt der Auswertungsoperator angewendet wird oder wenn für ein Host-Datenobjekt die `toString`–Methode ohne Argument aufgerufen wird:

```
## hostobject #
hostobject.toString()
```

Beim impliziten Aufruf wird der Typ des Host-Datenobjekts ermittelt und daraus der Name des Klassen-Templates gebildet: *type*`.clt`.

Der Typ wird durch das Kommunikationsmodul bestimmt. Bei openUTM wird das Attribut `IOTYPE` ausgewertet, bei OSD/MVS das Attribut `Type`.

Expliziter Aufruf

Beim expliziten Aufruf wird der Name des Klassen-Templates bei der Methode `toString` angegeben (siehe Abschnitt „Methode toString“ auf Seite 167). Sie haben so die Möglichkeit, Klassen-Templates unabhängig vom jeweiligen Typ des Host-Datenobjekts zu definieren und diese gezielt aufzurufen

---

##hostobjekt`.toString(`*name*`)#`

Das Suffix `.clt` wird von WebTransactions automatisch ergänzt.

10.1 WT_THIS - Zugriff auf das rufende Objekt

Innerhalb von Klassen-Templates können Sie über das Schlüsselwort `WT_THIS` auf das rufende Objekt zugreifen:

`WT_THIS` liefert eine Referenz auf das rufende Objekt. Sie können so beispielsweise Attribute dieses Objekts abfragen und verändern (siehe Abschnitt „Beispiel - Klassen-Templates und WT_THIS“ auf Seite 331).

Allerdings wirken der Auswertungsoperator und die Methoden `toString` / `valueOf` bei `WT_THIS` anders als beim rufenden Objekt: Sie liefern den Namen des rufenden Objekts. Dieser Unterschied macht auch Sinn, da sonst ein Ausdruck wie `##WT_THIS#` dazu führen würde, dass sich das Klassen-Template bis in alle Ewigkeit immer wieder selbst aufruft. Außerdem ist es von Vorteil, im Klassen-Template nicht nur auf das Objekt, sondern auch auf dessen Namen zugreifen zu können: Beispielsweise können Sie so aus dem Namen des rufenden Objekts Namen für HTML-Tags an der Oberfläche ableiten:

```
<INPUT TYPE="TEXT" NAME="##WT_THIS#" VALUE="##WT_THIS.Value#">
```

Der vom Browser gepostete Wert kann dann innerhalb des Klassen-Templates folgendermaßen abgefragt werden:

```
WT_POSTED[WT_THIS]
```

10.2 Beispiel - Klassen-Templates und WT_THIS

Das folgende Beispiel zeigt ein Klassen-Template `INPUT.clt` für Eingabefelder bei openUTM.

Der Abschnitt behandelt geschützte Felder, die als Text in die Seite ausgegeben werden. Für jedes Ausgabefeld werden die Darstellungsattribute überprüft und die Darstellung in der generierten HTML-Seite angepasst. Soll das Feld z.B. farbig dargestellt werden, so wird um den Text ein Font-Tag mit der entsprechenden Farbe generiert, oder soll es hervorgehoben werden (`wt_this.Intensity == 'H'`), so wird es in der generierten HTML-Seite auch in <b> eingeschlossen.

```
if ( wt_this.Visible == 'Y' )
{
  output = wt_this.HTMLValue;
  if ( wt_this.Inverse == 'Y' )
  {
    if (wt_this.Color && wt_this.Color.toUpperCase() != 'N' && wt_this.Color
!= ' ' )         output = '<font COLOR=#000000 STYLE=\"background-color:'
   +colors[wt_this.Color-1] + '\">'+output+'</font>';
  }
  else if (wt_this.Color&&wt_this.Color.toUpperCase()!= 'N' && wt_this.Color
!= ' ' )      output = '<font color=' + colors[wt_this.Color-1] + '>' +
   output + '</font>';
  if (wt_this.Intensity == 'H')
    output = '<b>' + output + '</b>';
  if (wt_this.Blinking == 'Y')
    output = '<i>' + output + '</i>';
  if (wt_this.Underlined == 'Y')
    output = '<u>' + output + '</u>';
  document.write( output );
}
```

11 Master-Templates (.wmt)

Master-Templates werden von WebTransactions bei der Generierung der Automask und der format-spezifischen Templates als Schablone verwendet und sorgen für ein einheitliches Layout.

Master-Templates können wie jedes andere Template feste HTML-Bereiche sowie beliebige WTML-Tags und WTScripts enthalten. Zusätzlich stehen in Master-Templates spezielle Master-Template-Tags zur Verfügung - kurz MT-Tags genannt. Die einzelnen MT-Tags werden in den Abschnitten „Lines-Tag“ auf Seite 335 bis „GlobalSettings-Tag“ auf Seite 352 beschrieben.

Die Dateinamen von Master-Templates sind durch das Suffix `.wmt` gekennzeichnet.

Stärke des Master-Template-Konzepts

Das Master-Template-Konzept zeigt seine Stärke v.a. bei Host-Anwendungen, bei denen viele Formate einen ähnlichen Aufbau haben: z.B. eine feste Einteilung in Kopfzeile, Arbeitsbereich und Fußzeile. In solchen Fällen genügt es, den Aufbau einmal im Master-Template festzulegen und dieses Master-Template bei der Generierung der format-spezifischen Templates als Schablone zuzuweisen. Alle generierten Templates erhalten dann automatisch den gewünschten Aufbau.

Dies erhöht die Konsistenz (corporate look and feel) und reduziert die Entwicklungskosten für Ihre WebTransactions-Anwendungen. Änderungen und Anpassungen in den Formaten der Host-Anwendungen können Sie an zentraler Stelle leicht und schnell nachziehen.

Standard-Master-Templates

Für die Liefereinheiten WebTransactions for OSD, WebTransactions for MVS und WebTransactions for openUTM werden jeweils spezielle Standard-Master-Templates mit ausgeliefert, die Sie individuell anpassen aber auch unverändert einsetzen können. Die Standard-Master-Templates enthalten bereits alle WTML-Tags und WTScripts, die für alle Templates der jeweiligen Liefereinheit gleich sind, z.B. eine Prüfung, ob ein privates System-Objekt existiert.

Die Standard-Templates der einzelnen Liefereinheiten werden in den jeweiligen Benutzerhandbüchern detailliert beschrieben.

Einsatz des Master-Templates

Über WebLab geben Sie an, welches Master-Template für die Generierung herangezogen werden soll. Voreingestellt ist dabei das Standard-Master-Template der jeweiligen Liefereinheit.

Einige Generierungsoptionen (z.B. die Generierungsmethode oder die Darstellungsattribute) können Sie auch unmittelbar mit WebLab festlegen. Falls im Master-Template die Generierungsoption bereits gesetzt ist, wird dieser Wert im WebLab als Voreinstellung angezeigt. Für diese Optionen gilt immer der mit WebLab gesetzte Wert - und zwar für das gesamte Template. Wenn Sie einen Wert mit WebLab ändern, wird damit der im Master-Template gesetzte Wert übersteuert.

Wenn Sie eine oder mehrere Optionen mehrfach im Master-Template gesetzt haben, gilt immer der zuletzt angegebene Wert.

Syntax der MT-Tags

MT-Tags beginnen jeweils mit zwei Prozent-Zeichen und enden mit einem Prozent-Zeichen. Die Zeichenfolge %% ist für MT-Tags reserviert.

Auf den Tag-Namen kann bei einigen MT-Tags optional eine Parameter-Liste folgen. Sowohl Tag-Namen als auch Parameternamen können Sie in beliebiger Groß-/Kleinschreibung angeben. Wird ein Parameter mehrmals gesetzt, wird der zuletzt gesetzte Wert berücksichtigt.

Zwischen den einzelnen Syntaxelementen können Leerzeichen stehen.

MT-Tags werden auch dann berücksichtigt, wenn sie innerhalb von HTML- oder WTML-Kommentaren stehen.

11.1 Lines-Tag

Das Lines-Tag legt fest, an welcher Stelle und auf welche Art die format-spezifischen Teile generiert werden.

```
%%Lines parameters%
```

Die einzelnen Parameter sind in folgender Liste beschrieben:

Parameter	Bedeutung	Mögliche Werte	Voreinstellung
`LayoutOnly`	Ermöglicht die Implementierung eigener Funktionen/Methoden zur Übernahme der Eingabeparameter.	**Yes** Es wird nur die Darstellung der Hostobjekte in das Template generiert. Jegliche weitere Aktionen, wie z.B. das Erzeugen des Objekts `wtInputFields` wird unterdrückt. **No** Voller Funktionsumfang wie bisher: Variablen und Anweisungen zur Übernahme der Eingaben werden generiert.	No
`Breaks`	Steuert bei statischen Darstellungsattributen und statischem Text die Generierung des Bildschirm-Layouts.	**Yes** Es werden Zeilenvorschubzeichen nach jedem Feld einer Zeile generiert. Zusätzlich wird in einer Kommentarzeile der zugehörige Feldinhalt angezeigt. **No** Alle Felder einer Bildschirmzeile werden im Template in einer Zeile dargestellt. Kommentare mit dem Feldinhalt zum Capture-Zeitpunkt werden nicht erstellt.	Yes
`PartialFormat`	Spezifiziert Master-Template für partielle oder vollständige Format-Generierung.	**Yes** Master-Template für partielle Format-Generierung **No** Master-Template für vollständige Format-Generierung	No
`StartLine`	Erste Formatzeile, die generiert werden soll	Integer	`1`

Parameter	Bedeutung	Mögliche Werte	Voreinstellung
EndLine	Letzte Formatzeile, die generiert werden soll. WebTransactions prüft, ob EndLine kleiner als StartLine ist. Wenn dies der Fall ist, wird das MT-Tag Lines ignoriert.	Integer	Last Line
StartPattern	Die erste Formatzeile, die generiert werden soll, folgt auf die Zeile mit dem angegebenen String.	String	None
EndPattern	Die letzte Formatzeile, die generiert werden soll, geht der Zeile mit dem angegebenen String voran.	String	None
CellsDelimiter	Begrenzung von Zellen	String (einzelnes Zeichen) Wenn alle Zeichen einer Formatzeile dem in CellsDelimiter angegeben Zeichen entsprechen, wird der String durch die Zeichenkette ersetzt, die im Parameter CellsDelimiterReplace steht. **i** Dieser Parameter hat nur Auswirkungen auf Felder vom Typ FIXTEXT.	
CellsDelimiterReplace	Zeichenkette für Ersetzung	Zeichenkette, die für eine Zeile eingesetzt wird, in der alle Zeichen dem Wert von CellsDelimiter entsprechen.	HTML-Tag für eine neue Tabellenzelle
CursorInProtectedField	Setzen des Cursors in geschützte Felder	**Yes** Bei den generierten Templates kann der Cursor in geschützte Felder gesetzt werden. **No** Bei den generierten Templates kann der Cursor nicht in geschützte Felder gesetzt werden.	NO

Parameter	Bedeutung	Mögliche Werte	Voreinstellung
`Generate`	Generierungsmethode	**Class** Die Ein-/Ausgabebearbeitung der Host-Datenobjekte basiert auf Klassen-Templates, d.h. jedes Host-Datenobjekt wird über ein Klassen-Template angesprochen (`INPUT.clt` bzw. `OUTPUT.clt`). Diese Methode ist zu empfehlen, wenn für Ein- bzw. Ausgabefelder globale Einstellungen vorgenommen werden sollen. Falls entsprechende Klassen-Templates noch nicht existieren, werden sie automatisch angelegt. **Inline** Alle notwendigen Schritte für die HTML-Darstellung und die Verarbeitungslogik sind über HTML-Tags, WTML-Tags und WTScripts unmittelbar im generierten Template enthalten. Es werden also keine Verarbeitungsschritte in Userexits oder Klassen-Templates ausgelagert. Wenn die Templates später editiert werden sollen, so ist dies die flexibelste Form.	`Inline`

Parameter	Bedeutung	Mögliche Werte	Voreinstellung
`DisplayAttributes`	Unterstützung von Feld-Attributen für die Anzeige	**Dynamic** Wird diese Option verwendet, dann werden ggf. die Funktionen `taggedInput()` zur Generierung der Eingabefelder und `taggedOutput()` für die Generierung der Ausgabefelder verwendet. Siehe auch die Parameter `taggedInput` und `taggedOutput`. **Static** Die Werte der Anzeige-Attribute werden fest in den Code der generierten Templates übernommen - so wie sie in der FLD-Datei definiert oder mit dem Capture-Verfahren aufgezeichnet wurden. **No** Anzeige-Attribute werden nicht berücksichtigt.	`No`
`StaticText` (nur OSD und MVS)	Umsetzung geschützter Felder in statischen HTML-Text	**Yes** Ist diese Option aktiviert, wird der Text, der bei den Formaten in geschützten Feldern (Ausgabefeldern vom Typ „Protected“) steht, direkt als Text in den HTML-Datenstrom generiert. Dabei wird davon ausgegangen, dass geschützte Felder in der Regel statisch sind, also von der Host-Anwendung nicht mit verschiedenen Werten versorgt werden. Falls einzelne geschützte Felder variabel sind, so müssen Sie diese Felder manuell mit Werten versorgen, d.h. die Templates entsprechend programmieren. Diese Option sollten Sie aktivieren, wenn nur wenige geschützte Felder eines Formats variablen Text enthalten. **No** Geschützte Felder werden von WebTransactions dynamisch ausgewertet.	`No`

Parameter	Bedeutung	Mögliche Werte	Voreinstellung
`DisplayEuro`	Anzeige des Euro-Symbols	**Yes** Enthalten feste Texte eines Formats das Euro-Symbol (ehemals Currency-Symbol), so wird dieses in die HTML-Escape-Sequenz `&#8364` (Euro) konvertiert. **No** Hiermit wird in festen Texten das ehemalige Currency-Symbol nicht in das Euro-Symbol umgesetzt, sondern in X'A4' (¤).	`No`
`MenuBar` (nur für MVS)	Umsetzung der ersten Formatzeile als Menü	**Yes** Markierbare Felder in der ersten Zeile des Formats werden als Menübefehle erkannt und zu Pulldown-Menüs umgesetzt. Diese Option wird für ISPF-Anwendungen empfohlen. **No** Die erste Zeile der Formate wird wie üblich umgesetzt.	`No`
`TaggedInput`	Verwendung der Funktion `taggedInput()`	**Disabled** Die Funktion `taggedInput()` wird in den generierten Templates nicht verwendet und ist im Master-Template nicht definiert. **Enabled** Die Funktion `taggedInput()` wird entsprechend der Darstellungsattribute verwendet, die Sie im Master-Template oder WebLab angegeben haben. Beachten Sie, dass durch das `Lines`-MT-Tag einige zusätzliche HTML-Tags erzeugt werden können. **Enforced** Bei der Generierung der Templates wird nur die `taggedInput`-Funktion genutzt. Das MT-Tag `Lines` erzeugt keine zusätzlichen HTML-Tags.	Disabled

Parameter	Bedeutung	Mögliche Werte	Voreinstellung
`TaggedOutput`	Verwendung der Funktion `taggedOutput()`	**Disabled** Die Funktion `taggedOutput()` wird in den generierten Templates nicht verwendet und ist im Master-Template nicht definiert. **Enabled** Die Funktion `taggedOutput()` wird entsprechend der Darstellungsattribute verwendet, die Sie im Master-Template oder WebLab angegeben haben. Beachten Sie, dass durch das `Lines`-MT-Tag einige zusätzliche HTML-Tags erzeugt werden können. **Enforced** Bei der Generierung der Templates wird nur die `taggedOutput`-Funktion genutzt. Das MT-Tag `Lines` erzeugt keine zusätzlichen HTML-Tags.	Enabled

Zusammenspiel der Start- und End-Bedingungen

Das Format wird beginnend von der ersten Zeile, die eine Start-Bedingung erfüllt (`StartLine` oder `StartPattern`), zeilenweise umgesetzt. Die Umsetzung endet mit der ersten Zeile, die eine End-Bedingung erfüllt (`EndLine` oder `EndPattern`) - unabhängig davon, welcher Parameter für die Start-Bedingung angegeben wurde.

Beispiel

```
%%Lines
StartLine = 4
StartPattern = "Options"
EndPattern = "Ende"
EndLine=15
%
```

Angenommen, im Format wäre der String „Options“ erstmals in Zeile 2 enthalten und der String „Ende“ erstmals in Zeile 18. Der umgesetzte Bereich würde in diesem Fall die Zeilen 3 bis 15 umfassen.

WebTransactions prüft, ob `EndLine` kleiner als `StartLine` ist. Wenn dies der Fall ist, wird das MT-Tag `Lines` ignoriert.

11.2 Options-Tag

Mit dem Options-Tag können Sie sowohl Werte für standardmäßige Master-Template-Tags als auch individuelle Master-Template-Tags definieren. Zu diesem Zweck stehen eine standardmäßige und eine erweiterte Syntax zur Verfügung.

11.2.1 Options-Tag (standardmäßige Syntax)

Bei der standardmäßigen Syntax können Sie mit den Parametern des Options-Tags globale Werte spezifizieren, die dann an allen Stellen eingesetzt werden, an denen sich ein dem Parameter entsprechendes Tag befindet. Für das Options-Tag selbst wird in den Templates kein Code generiert.

Wenn ein Master-Template mehrere Options-Tags enthält, die den gleichen Parameter setzen, wird für das entsprechende Tag der jeweils zuletzt explizit gesetzte Wert berücksichtigt.

```
%%Options parameters%
```

Die einzelnen Parameter sind in folgender Liste beschrieben:

Parameter	Bedeutung	Mögliche Werte	Voreinstellung
`JavaUtil`	Enthält den Namen des Java-Pfades für das Java-Utility-Package *	String	`java.util`
`CommObj`	Referenz auf das Kommunikationsobjekt	String	je nach verwendetem Protokoll: `OSD_0`, `MVS_0` oder `UTM_0`

`NationalVariant`	Sprachvariante der zwischen WebTransactions und Host-Anwendung ausgetauschten Nachrichten	`"International"`
		`"English (UK)"`
		`"English (USA)"`
		`"Swedish"`
		`"German"`
		`"French"`
		`"Italian"`
		`"Spanish"`
		`"Swiss"`
		`"Norwegian"`
		`"Danish"`
		`"French-Belgian"`
			`"International"`

Damit ein im Options-Tag gesetzter Wert berücksichtigt wird, muss der Parameter **vor** Verwendung des dem Parameter entsprechenden Tags gesetzt werden.

* *Hinweise*

1. Die Angaben für `JavaUtil` sind nur sinnvoll im Master-Template `JAVA_BO.wmt`.
2. Die Einstellungen werden implizit verwendet bei der Ersetzung des `ObjectCreate`-Tags zur Referenzierung der verschiedenen Java-Klassen, aus denen die Objekte rekrutiert werden.

Beispiel

- Master-Template-Code:

```
%%NationalVariant%
%%Options NationalVariant = "Italian"%
%%NationalVariant%
%%Options NationalVariant = "Spanish"%
%%NationalVariant%
```

- generierter Code:

```
International //(Voreinstellung)
Italian
Spanish
```

11.2.2 Options-Tag (erweiterte Syntax)

Mit der erweiterten Syntax des Options-Tags können Sie eigene Master-Template-Tags definieren. Dies kann z.B. sinnvoll sein, wenn Sie ein Stück Text mehrfach im Master-Template verwenden wollen oder wenn Sie unterschiedliche Texte erzeugen wollen, je nachdem, ob das Master-Template zur Generierung einer Automask oder eines formatspezifischen (individuellen) Templates verwendet wird.

```
%%Options
      destination="Automask | Individual | Both"
      tagName="tagname"
block
%
```

`destination`
: Hier wird der Templatetyp angegeben, bei dessen Generierung das selbstdefinierte Tag gültig ist.

tagname
: Name des selbstdefinierten Tags. Groß- und Kleinschreibung wird beachtet.

block

Textbereich, der als Ersetzungstext für das Tag gemerkt wird.

Hinweis

Das Tag-Endezeichen **%** muss als einziges Zeichen in einer eigenen Zeile angegeben werden. Somit ist gesichert, dass innerhalb des Blockes auch weitere Master-Template-Tags verwendet werden können.

Bei der Formulierung des Textes für *block* müssen Sie auf folgende Dinge achten:

1. Der Text muss in die Umgebung passen, in der sie das selbstdefinierte Tag verwenden. Das bedeutet zum Beispiel: wurde in einem selbstdefinierten Tag nur WTScript verwendet, darf dieses Tag nur in einem WTScript-Bereich verwendet werden.
2. In dem Text für *block* dürfen Sie nur die Tags `%%CommObj%`, `%%NationalVariant%`, `%%Format%`, `%%Source%`, `%%ObjectName%`, `%%PackageName%`, `%%JavaUtil%`, und die von Ihnen selbstdefinierten Tags verwenden. Diese geschachtelten Tags werden durch den Inhalt zum Zeitpunkt der Ersetzung des äußeren Tags ersetzt.

Sie verwenden die selbstdefinierten Master-Template-Tags in der Form

%%tagname%

wobei Sie für *tagname* genau den Namen angeben müssen, den Sie beim Parameter `tagName` definiert haben (Groß-/Kleinschreibung beachten!).

Beispiele

Beispiel 1:

Darstellung eines unterschiedlichen Bildes, je nachdem, ob das verwendete Template die Automask ist oder ein formatspezifisches Template.

Master-Template:

```
….
%%GenerationInfo%
%%Rem Verwende Bild1 für individuelle Templates%
%%Options tagName="Image" destination="Individual"
<img src="##WT_SYSTEM.WWWDOCS_VIRTUAL#/image/Bild1.gif">
%
%%Rem Verwende Bild2 für Automask%
%%Options tagName="Image" destination="Automask"
<img src="##WT_SYSTEM.WWWDOCS_VIRTUAL#/image/Bild2.gif">
%
…
<body bgcolor="#C0C0C0">
%%Image%
<form WebTransactions name="%%Format%">
```

In der Automask generierter Code:

```
<body bgcolor="#C0C0C0">
<img src="##WT_SYSTEM.WWWDOCS_VIRTUAL#/image/Bild2.gif">
<form WebTransactions name="Automask">
```

Im formatspezifischen Template generierter Code:

```
<body bgcolor="#C0C0C0">
<img src="##WT_SYSTEM.WWWDOCS_VIRTUAL#/image/Bild1.gif">
<form WebTransactions  name="Trav0">
```

Beispiel 2:

Unterschiedliche Generierung der Funktion `taggedInput`, um in der Automask redundante Abfragen zu vermeiden und sie dadurch performanter zu machen.

Ausschnitt aus dem Master-Template:

```
%%Rem Für individuelle Templates. Es muss noch einmal überprüft werden, ob ein
  Feld, das zum Capture-Zeitpunkt ein Eingabefeld war,auch zum jetzigen
  Zeitpunkt noch ein Eingabefeld ist%
%%Options destination="Individual" tagName="taggedInputCrossCall"
    if ( hostObject.Type == 'Protected'  && hostObject.Markable == 'No' )
              return taggedOutput( hostObject );
%
%%Rem In der Automask ist diese Abfrage nicht nötig%
%%Options destination="Automask" tagName="taggedInputCrossCall"
%

function taggedInput( hostObject )
{ %%taggedInputCrossCall%

currentLength = hostObject.Length;
input = '<input type=' + (hostObject.Visible == 'No' ? '"password"' : '"text"' );
if ( WT_BROWSER.is_ie || WT_BROWSER.is_ns61up)
{
input += ' class="box" style="width:' + (currentLength * WT_BROWSER.charWidth + 1) +
             'px';
    input += (hostObject.Blinking  == 'Yes' ? '; background-color:#FFC0C0' : '');
    input += (hostObject.Underline == 'Yes' ? ( hostObject.Intensity == 'Reduced' ? ';
              color:#A0A0FF' : '; color:#0000A0' ) :( hostObject.Intensity == 'Reduced' ?
             '; color:#A0A0A0' : '' )) + '"';
}

input += ' name="' + hostObject.Name + '" size="' + currentLength
                   + '" maxlength="' + currentLength
                   + '" value="' + hostObject.Value
                   + (hostObject.Input == 'Numeric'?'" numeric="1':'')
                   + (hostObject.Markable == 'Yes'?'" markable="1':'')
                               + '"/>';
document.write( input );
}
```

Ausschnitt aus der generierten Automask (Abfrage nicht generiert)**:**

```
function taggedInput( hostObject )
{
currentLength = hostObject.Length;
input = '<input type=' + (hostObject.Visible == 'No' ? '"password"' : '"text"' );
if ( WT_BROWSER.is_ie || WT_BROWSER.is_ns61up)
{
input += ' class="box" style="width:' + (currentLength * WT_BROWSER.charWidth + 1) +
            'px';
     input += (hostObject.Blinking  == 'Yes' ? '; background-color:#FFCOCO' : '');
     input += (hostObject.Underline == 'Yes' ? ( hostObject.Intensity == 'Reduced' ? ';
            color:#AOAOFF' : '; color:#OOOOAO' ) :( hostObject.Intensity == 'Reduced' ?
            '; color:#AOAOAO' : '' )) + '"';
}
input += ' name="' + hostObject.Name + '" size="' + currentLength
                  + '" maxlength="' + currentLength
                  + '" value="' + hostObject.Value
                  + (hostObject.Input == 'Numeric'?'" numeric="1':'')
                  + (hostObject.Markable == 'Yes'?'" markable="1':'')
                              + '"/>';
document.write( input );
}
```

Ausschnitt aus einem individuellen Template (Abfrage generiert)**:**

```
function taggedInput( hostObject )
{
if ( hostObject.Type == 'Protected' && hostObject.Markable == 'No' )
            return taggedOutput( hostObject );
currentLength = hostObject.Length;
input = '<input type=' + (hostObject.Visible == 'No' ? '"password"' : '"text"' );
if ( WT_BROWSER.is_ie || WT_BROWSER.is_ns61up)
{
input += ' class="box" style="width:' + (currentLength * WT_BROWSER.charWidth + 1) +
            'px';
     input += (hostObject.Blinking  == 'Yes' ? '; background-color:#FFCOCO' : '');
     input += (hostObject.Underline == 'Yes' ? ( hostObject.Intensity == 'Reduced' ? ';
            color:#AOAOFF' : '; color:#OOOOAO' ) :( hostObject.Intensity == 'Reduced' ?
            '; color:#AOAOAO' : '' )) + '"';
}
input += ' name="' + hostObject.Name + '" size="' + currentLength
                  + '" maxlength="' + currentLength
                  + '" value="' + hostObject.Value
                  + (hostObject.Input == 'Numeric'?'" numeric="1':'')
                  + (hostObject.Markable == 'Yes'?'" markable="1':'')
                              + '"/>';
document.write( input );
}
```

11.3 Rem-Tag

Dieses Tag erscheint ausschließlich im Master-Template und erlaubt es dem Master-Template-Entwickler, das Master-Template zu kommentieren, ohne dass die Kommentarzeilen in das generierte Template übernommen werden.

```
%%Rem comment%
```

Die einzelnen Parameter sind in folgender Liste beschrieben:

Parameter	Bedeutung	Mögliche Werte	Voreinstellung
comment	Kommentar einfügen	Beliebiger String	Keine

11.4 OnReceiveCopies-Tag

OnReceiveCopies-Tags dürfen Sie nur in WTScript-Bereichen verwenden (sinnvoll nur in OnReceiveScript-Bereichen). Dieses Tag wird in den generierten Templates durch Anweisungen ersetzt, die zum OnReceive-Zeitpunkt die vom Browser empfangenen Werte in die entsprechenden Host-Datenobjekte übertragen.

```
%%OnReceiveCopies parameters%
```

Die einzelnen Parameter sind in folgender Liste beschrieben:

Parameter	Bedeutung	Mögliche Werte	Voreinstellung
`StartLine`	erste Zeile des Formatbereichs, für dessen Felder gepostete Werte auf Host-Datenbereiche übertragen werden sollen.	Integer	`1`
`EndLine`	letzte Zeile des Formatbereichs, für dessen Felder gepostete Werte auf Host-Datenbereiche übertragen werden sollen.	Integer	`Last Line`
`StartPattern`	Die erste Zeile des Formatbereichs, für dessen Felder gepostete Werte auf Host-Datenbereiche übertragen werden sollen, folgt auf die Zeile mit dem angegebenen String.	String	`None`
`EndPattern`	Die letzte Zeile des Formatbereichs, für dessen Felder gepostete Werte auf Host-Datenbereiche übertragen werden sollen, geht der Zeile mit dem angegebenen String voran.	String	`None`

Beispiel

- Master-Template-Code:

```
<wtOnReceiveScript>
<!--
host.WT_FOCUS.Field = WT_CURSOR;
%%OnReceiveCopies%
//-->
</wtOnReceiveScript>
```

- Der generierte Code könnte wie folgt aussehen:

```
<wtOnReceiveScript>
<!--
host.WT_FOCUS.Field = WT_CURSOR;
// ****** copy posted values to host objects ****
host1.E_05_025_001.Value = WT_POSTED.E_05_025_001;
//-->
</wtOnReceiveScript>
```

11.5 GenerationInfo-Tag

Das GenerationInfo-Tag erzeugt einen WTML-Kommentar, der Informationen über die Generierungsoptionen und die Umstände der Generierung enthält.

Das Tag können Sie im festen HTML-Bereich, innerhalb von WTML-Tags oder innerhalb von WTScript-Bereichen angeben.

```
%%GenerationInfo%
```

Das Tag erzeugt beispielsweise die folgenden Zeilen:

```
<wtrem>*****************************************************************************</wtrem>
<wtrem>** WTML document: AutomaskOSD                                                **</wtrem>
<wtrem>*****************************************************************************</wtrem>
<wtrem>**                                                                           **</wtrem>
<wtrem>** Document generation based on Master Template :                            **</wtrem>
<wtrem>**  C:\Program Files\webtransactionsv75\weblab\OSD.wmt                        **</wtrem>
<wtrem>**                                                                           **</wtrem>
<wtrem>** Generated at Tue Jun 08 12:44:20 2010                                     **</wtrem>
<wtrem>**                                                                           **</wtrem>
<wtrem>** Options used by the generator :                                           **</wtrem>
<wtrem>**   -  %OPTIONS:                                                           **</wtrem>
<wtrem>**      CommObj = OSD_0                                                      **</wtrem>
<wtrem>**      NationalVariant = International - PartialFormatMode = No             **</wtrem>
<wtrem>**   -  %OPTIONS:                                                           **</wtrem>
<wtrem>**      self defined Tag = taggedInputCrossCall                              **</wtrem>
<wtrem>**   -  %OPTIONS:                                                           **</wtrem>
<wtrem>**      self defined Tag = taggedInputCrossCall                              **</wtrem>
<wtrem>**   -  %OPTIONS:                                                           **</wtrem>
<wtrem>**      self defined Tag = taggedOutputCrossCall                             **</wtrem>
<wtrem>**   -  %OPTIONS:                                                           **</wtrem>
<wtrem>**      self defined Tag = taggedOutputCrossCall                             **</wtrem>
<wtrem>**   -  %LINES:                                                             **</wtrem>
<wtrem>**      TaggedInput = Enabled - TaggedOutput = Enabled                       **</wtrem>
<wtrem>**      DisplayAttributes = Dynamic - CursorInProtectedField = Yes           **</wtrem>
<wtrem>**   -  %RECEIVES:                                                          **</wtrem>
<wtrem>**      Parameters not specified                                             **</wtrem>
<wtrem>*****************************************************************************</wtrem>
<wtrem>** WebTransactions V7.5             Fujitsu Technology Solutions 2010 **</wtrem>
<wtrem>*****************************************************************************</wtrem>
```

11.6 Format-Tag

Das Format-Tag wird in den generierten Templates durch den Format-Namen des aktuell umgesetzten Formats ersetzt.

```
%%Format%
```

Beispiel

```
<wtDataform=%%Format%>
```

11.7 CommObj-Tag

Das CommObj-Tag wird in den generierten Templates durch den String ersetzt, der im Parameter `CommObj` des MT-Tags `Options` angegeben ist (Referenz auf das Kommunikationsobjekt). Voreingestellt ist - je nach verwendetem Protokoll: `OSD_0`, `MVS_0` oder `UTM_0`.

```
%%CommObj%
```

Beispiel

Master-Template-Code:

```
%%Options CommObj="host1"%
if (%%CommObj%.WT_SYSTEM!= null)
host_system = %%CommObj%.WT_SYSTEM;
```

11.8 NationalVariant-Tag

Das NationalVariant-Tag wird in den generierten Templates durch den String ersetzt, der im Parameter `NationalVariant` des MT-Tags `Options` angegeben ist (Sprachvariante der zwischen WebTransactions und Host-Anwendung ausgetauschten Nachrichten).

```
%%NationalVariant%
```

11.9 GlobalSettings-Tag

Mit dem GlobalSettings-Tag können Sie generelle Einstellungen für das Master-Template festlegen. Das GlobalSettings-Tag kann überall im Master-Template stehen, darf aber nur einmal vorkommen. Wenn Sie das GlobalSettings-Tag mehrmals im Master-Template verwenden, erhalten Sie eine Fehlermeldung in WebLab und die Generierung wird nicht begonnen.

```
%%GlobalSettings parameters%
```

Die einzelnen Parameter sind in folgender Liste beschrieben:

Parameter	Bedeutung	Mögliche Werte	Voreinstellung
`Protocol`	Protokoll der Host-Anwendung	"OSD" "MVS" "UTM" "JAVA" "WSDL"	keine

Beispiel

```
%%GlobalSetting protocol= "OSD"%
```

11.10 Source-Tag

Das Source-Tag ist nur gültig für das Master-Template `Java_BO.wmt` und enthält den Namen der kopierten Beschreibungsdatei im Basisverzeichnis (z.B. `WSDL/BO1.wsdl`).

```
%%Source%
```

11.11 ObjectName-Tag

Das ObjectName-Tag ist nur gültig für die Master-Templates `WSDL.wmt` und `Java_BO.wmt` und enthält den Namen des Java-Objekts (entweder den BO- oder den EJB-Namen) oder des Services (im Falle von Web-Services).

```
%%ObjectName%
```

11.12 PackageName-Tag

Das PackageName-Tag ist nur gültig für das Master-Template `Java_BO.wmt` und enthält den Namen des Java-Package, in dem das Objekt enthalten ist.

```
%%PackageName%
```

11.13 BinaryFile-Tag

Das BinaryFile-Tag ist nur gültig für das Master-Template `Java_BO.wmt` und liefert den Namen der n-ten kopierten Datei.

```
%%BinaryFile parameters%
```

Die einzelnen Parameter sind in folgender Liste beschrieben:

Parameter	Bedeutung	Mögliche Werte	Voreinstellung
`Number`	Name der n-ten kopierten binären Datei	Dateiname, z.B. JAVA/BO1	1

Beispiel

```
%%BinaryFile Number=2%
```

11.14 ArchiveName-Tag

Das ArchiveName-Tag ist nur gültig für das Master-Template `Java_BO.wmt` und enthält den Namen des Java-Archivs.

```
%%ArchiveName%
```

11.15 MethodInterface-Tag

Das MethodInterface-Tag ist nur gültig für die Master-Templates `WSDL.wmt` und `Java_BO.wmt` und wird für die Generierung der Darstellung der einzelnen Methoden verwendet.

```
%%MethodInterface%
```

12 Server-seitige Schnittstellen - Java-Integration und Userexits

Den Funktionsumfang Ihrer WebTransactions-Anwendung können Sie wie folgt erweitern:

- Die Integration von Java in WebTransactions ermöglicht es Ihnen, eigene Java-Klassen in WebTransactions zu instanziieren. Danach können Sie die Methoden der so erzeugten Objekte ausführen und auf die Objekt-Attribute zugreifen.
- Mit Userexits können Sie eigene C/C++-Funktionen in WebTransactions integrieren.

Beispielsweise können Sie einen Zugriff auf strukturierte Daten (Tabellen, Listen) in einem Userexit realisieren oder die Koordinaten einer Grafik ermitteln, die in eine HTML-Seite eingebettet ist.

Ferner können Sie mit Userexits auch dynamische HTML-Seiten aufbauen, ohne eine Host-Anwendung anzuschließen. Die zu präsentierenden Daten können z.B. aus einer Datei eingelesen werden. In diesem Einsatzfall kann man WebTransactions als komfortable Einschalung der CGI-Schnittstelle verstehen.

Beispiele für einen solchen Einsatzfall sind:

– Gästebuchfunktionen

– Verwaltung und Abfrage von Telefonlisten

– Administrationsoberfläche für den HTTP-Server. In diesem Fall müssen Userexits die Konfigurationsdatei des http-Dämons lesen und schreiben sowie den Dämon neu starten.

WebTransactions stellt Ihnen einen Satz von bereits fertig definierten C/C++-Userexits zur Verfügung. Diese werden z.T. von WebTransactions intern genutzt - z.B. in den Start-Templates. Sie können die fertig ausgelieferten Userexits jedoch auch für die von Ihnen erstellten oder angepassten Templates nutzen. Die Schnittstellen dieser Userexits sind in Abschnitt „Mit WebTransactions fertig ausgelieferte C/C++-Userexits“ auf Seite 376ff. beschrieben.

12.1 Java-Integration in WebTransactions

Die Java-Integration ermöglicht es Ihnen, in Ihrer WebTransactions-Anwendung beliebige Java-Klassen zu instanziieren, die Methoden der so erzeugten Objekte auszuführen und auf die Attribute zuzugreifen.

Damit Java-Programme oder -Anwendungen auf dem WebTransactions-Rechner ablaufen können, muss auf diesem Rechner eine Java-Laufzeitumgebung (Java Virtual Machine, JVM) verfügbar sein.

Im Zusammenhang mit der Java-Integration ist zu beachten:

- Dezimalzahlen, die zwischen WTScript und Java ausgetauscht werden, verändern ihre Darstellung. Beispielsweise wird der Wert 2.1 nach dem Übergang zu Java und wieder zurück zu 2.099999904632568.
- WTScript unterstützt keine Unicode-Zeichen. Standardmäßig wird bei der Umwandlung von Strings an der Java-Schnittstelle immer von ISO-8859-1 nach UTF-16 und umgekehrt konvertiert. Ab WebTransactions V6 kann diese Konvertierung über die Variable `WT_SYSTEM.JAVA_CHARSET` gesteuert werden. Sie muss vor dem ersten WT_JAVA-Zugriff gesetzt werden. Ihr Wert bezeichnet eine Laufzeitbibliothek (Name ohne .so oder .dll), die passende Konvertierungsfunktionen enthält. Diese Laufzeitbibliothek wird im Basisverzeichnis und im Installationsverzeichnis unter `lib` gesucht.

  In der Standardauslieferung ist eine Laufzeitbibliothek für den Zeichensatz ISO-8859-2 (Mitteleuropa) enthalten. Weitere Bibliotheken können bei Bedarf geliefert werden.

  Beispiel

```
<html>
<head>
<title>Zeichensatz-Test</title>
</head>
<body>
<H1>Character Set Test</H2>
<pre style="font-size: 16pt">
ISO-8859-2

<wtoncreatescript>
<!--
// Browser und Java auf ISO-8859-2 umstellen
WT_SYSTEM.CHARSET = WT_SYSTEM.JAVA_CHARSET = 'ISO-8859-2';

// Zeichenmatrix ausgeben
hex = '0123456789ABCDEF';
document.writeln ('  <b>0 1 2 3 4 5 6 7 8 9 A B C D E F</b>');
for (i=0; i<16; i++)
```

```
{
  document.write ('<b>',hex[i],'</b>');
  if (i<2 || i==8 || i==9)
  {
    document.writeln();
    continue;
  }
  for (j=0;j<16; j++)
    document.write (' ', eval ('"\\x'+hex[i]+hex[j]+'"'));
  document.writeln();
}

// Java string erzeugen
a = new WT_JAVA.java.lang.String (WT_POSTED.s||'');
//-->
</wtoncreatescript>

<form webtransactions>
Um die richtige Verarbeitung in Java zu testen, können im folgenden
Eingabefeld Zeichen eingegeben und an WebTA geposted werden
(z.B. durch Copy/Paste aus der Zeichenmatrix oben).
Diese werden dann in einen Java-String konvertiert, mit den
Java-Methoden toUpperCase und toLowerCase umgewandelt und wieder in
WebTA-Strings konvertiert und ausgegeben.

<input type="text" name="s"> <input type="submit" value="Konvertieren">
</form>
Eingegeben wurde: ##WT_POSTED.s#
Java toUpperCase: ##a.toUpperCase()#
Java toLowerCase: ##a.toLowerCase()#
</pre>
</body>
</html>
```

12.1.1 Java-Laufzeitumgebung installieren

Wenn Sie in WebTransactions die Java-Integration oder Java-Userexits verwenden wollen, muss die Java-Laufzeitumgebung **vor** der Installation von WebTransactions installiert werden.

Wenn auf Ihrem WebTransactions-Rechner bereits eine Java-Laufzeitumgebung installiert sind keine weiteren Schritte notwendig.

Falls auf Ihrem Rechner die Java Laufzeitumgebung noch nicht installiert ist, finden Sie in der folgenden Tabelle Hinweise, von woher Sie die Software beziehen können:

Windows **Solaris** **Linux**	Für 32-Bit-Windows-Plattformen, Solaris und Linux stellt Sun Microsystems die Java-Laufzeitumgebung zur Verfügung (freeware). Sie können diese unmittelbar von der Sun Java Software-Webseite downloaden (*http://java.sun.com*).
BS2000/OSD Posix	Für BS2000/OSD ab V3.0 gehört die Java-Umgebung JENV (BS2000/OSD) V1.3 zum Grundausbau, muss aber gegebenenfalls noch installiert und für Posix aktiviert werden.

12.1.2 Java-Unterstützung aktivieren

WebTransactions aktiviert eine Java Virtual Machine (VM). Deshalb müssen die Java- Bibliotheken mit den nativen Java-Funktionen zugänglich sein. Bei der WebTransactions-Installation werden Sie aufgefordert, den entsprechenden Pfad anzugeben. (Das Installationsverzeichnis der JRE ist für Unix- oder Windows-Plattform das Verzeichnis, in dem sich die Bibliothek `jvm.dll` befindet). WebTransactions setzt dann automatisch die notwendigen Dateiverweise.

Wenn WebTransactions die Java-Bibliotheken nicht laden kann (z.B. weil die Dateien nach der Installation von WebTransactions in ein anderes Verzeichnis verschoben wurden oder Sie Java erst nachträglich installieren), wird eine Fehlermeldung ausgegeben. In einem solchen Fall müssen Sie die Dateiverweise selber anlegen.

In den folgenden Abschnitten sind die nötigen Dateiverweise am Beispiel von JDK 1.3.1 ausgeführt (abhängig von der WebTransactions-Plattform).

Windows

Erzeugen Sie im WebTransactions-Installationsverzeichnis im Unterverzeichnis `lib` einen Dateiverweis `javasys2` auf das Verzeichnis, das die Bibliothek `jvm.dll` enthält. Also z.B. den Verweis

`C:\`*`installdir`*`\lib\javasys2` auf das Verzeichnis `C:\jdk1.3.1\jre\bin\classic`.

Solaris/Linux

Erzeugen Sie im WebTransactions-Installationsverzeichnis im Unterverzeichnis `lib` die folgenden symbolischen Verweise:

`javalib -> .../jdk1.3.1/jre/lib/i386` (Verzeichnis mit zusätzlichen shared objects)
`javasys -> .../jdk1.3.1/jre/lib`
`javathreads -> .../jdk1.3.1/jre/lib/i386/native_threads`
`javavm -> .../jdk1.3.1/jre/lib/i386/server` (Verzeichnis mit `libjvm.so`)

Dabei steht „ ...“ für das Installationsverzeichnis des JDK.

BS2000 OSD ab V2.0

Bei der öffentlichen Installation in der Posix-Umgebung werden die Dateiverweise automatisch gesetzt. Bei einer Privat-Installation müssen die Dateiverweise für `javasys` und `javathreads` unter Umständen explizit gesetzt werden. Erzeugen Sie dazu im WebTransactions-Installationsverzeichnis `/opt/WebTrans/7.5/lib` die folgenden symbolischen Verweise:

`javalib -> .../jdk1.3.1/jre/lib/i386` (Verzeichnis mit zusätzlichen shared objects)
`javasys -> .../jdk1.3.1/jre/lib`

```
javathreads -> .../jdk1.3.1/jre/lib/i386/native_threads
javavm -> .../jdk1.3.1/jre/lib/i386/server (Verzeichnis mit libjvm.so)
```

Dabei steht „ ...“ für das Installationsverzeichnis des JDK.

12.1.3 Java Virtual Machine (JVM) parametrisieren

Zur Parametrisierung der in WebTransactions verwendeten Java Virtual Machine (JVM) stehen Ihnen zwei Möglichkeiten zur Verfügung:

- JVM mit Hilfe von System-Attributen parametrisieren
- JVM mit Hilfe des Arrays `WT_SYSTEM.JAVA_OPTIONS` parametrisieren

Java Virtual Machine (JVM) mit Hilfe von System-Attributen parametrisieren

Mit folgenden System-Attributen steuern Sie die Java Virtual Machine (JVM):

System-Attribut	Compiler-Option für Java	Bedeutung
`JAVA_CHECK_SOURCE='1'`		Prüft, ob die Quelldateien neueren Datums sind, als die Klassen-Dateien.
`JAVA_CLASSPATH`	-classpath	Setzt den Suchpfad für die Klassen-Dateien. Mehrere Verzeichnisse werden mit Strichpunkt getrennt. Voreinstellung: *basedir*`/java`
`JAVA_DEBUG='1'`	-debug	Aktiviert entfernte Fehlersuche (nur für die Debug-Version der Java Systembibliothek).
`JAVA_DEBUG_PORT`		Port für die entfernte Fehlersuche
`JAVA_DISABLE_ASYNC_GC='1'`	-noasyncgc	Schaltet die asynchrone Speicherbereinigung aus.
`JAVA_DISABLE_CLASS_GC='1'`	-noclassgc	Schaltet die Speicherbereinigung aus.
`JAVA_ENABLE_VERBOSE_GC='1'`	-verbosegc	Druckt eine Nachricht bei jeder Speicherbereinigung.
`JAVA_INITIAL_HEAP_SIZE`	-ms *number*	Initialisiert den Java-Speicher.
`JAVA_MAX_HEAP_SIZE`	-mx *number*	Setzt den maximalen Java-Speicher.
`JAVA_NATIVE_STACK_SIZE`	-ss *number*	Setzt die maximale Speichergröße für Thread-Verarbeitung.
`JAVA_STACK_SIZE`	-oss *number*	Setzt die minimale Speichergröße für Thread-Verarbeitung.
`JAVA_VERBOSE='1'`	-verbose	Schaltet den Verbose-Modus aus.

System-Attribut	Compiler-Option für Java	Bedeutung
JAVA_VERIFY_MODE='0'	-noverify	Prüft keine Klassen.
JAVA_VERIFY_MODE='1'	-verifyremote	Prüft nur Klassen, die über das Netz eingelesen werden.
JAVA_VERIFY_MODE='2'	-verify	Prüft alle Klassen.

Java Virtual Machine (JVM) mit Hilfe des Arrays WT_SYSTEM.JAVA_OPTIONS parametrisieren

Sie können über ein Array WT_SYSTEM.JAVA_OPTION alle Standard-Argumente an die JVM übergeben. Alle Parameter für die JVM müssen vor dem ersten Zugriff auf WT_JAVA definiert sein. Spätere Änderungen innerhalb der Session sind nicht möglich.

Beispiel

```
WT_SYSTEM.JAVA_OPTIONS = new Array('-verbose', '-Xms6m');
a = new WT_JAVA.java.lang.String("Hello World!")
```

12.1.4 Java-Objekte in WTScript erzeugen

Java-Objekte erzeugen Sie in WTScript mit dem `new`-Operator.

```
var foo = new WT_JAVA.classname();
```

classname

Name der Java-Klasse, die instanziiert werden soll. *classname* muss immer der vollqualifizierte Klassennamen sein. Bei Konstruktoren ohne Parameter können die Klammern auch weggelassen werden.

Beispiel

```
var myString = new WT_JAVA.java.lang.String("Hello world");
```

Java-Objekte, die auf diese Weise erzeugt wurden, werden von WTScript als Objekte behandelt, für die folgende Besonderheiten zu beachten sind:

– Im Objektbaum von WebLab wird nur das Objekt angezeigt. Attribute und Methoden sind nicht sichtbar. Als Wert des Objekts wird das Ergebnis der implizit von dem Objekt ausgeführten *toString()*-Methode angezeigt.

– Trotzdem stehen alle Methoden und Attribute des Objektes zur Verfügung. Dies betrifft sowohl Attribute, die in der zugehörigen Klasse definiert sind, als auch geerbte Attribute.

Variablen primitiver Java-Typen (char, int, double, etc.) können nicht auf die oben beschriebene Weise erzeugt werden. Diese Variablen erzeugen Sie in WTScript wie folgt über die zugehörige Wrapper-Klasse (hier: Double):

```
var foo = (new WT_JAVA.java.lang.Double(wert)).doubleValue();
```

wert spezifiziert den Wert, den die Variable erhalten soll.

Analog verfahren Sie bei char/Char, int/Int, long/Long, etc.

12.1.5 Java-Objekte in WTScript verwenden

Bis auf folgende Einschränkungen können Sie Java-Objekte in WTScript genauso wie in Java verwenden:

– WTScript-Objekte können nicht von Java-Objekten erben.

– Für Java-Objekte können in WTScript keine zusätzlichen Attribute und Methoden definiert werden.

– In WTScript gibt es keinen cast-Operator. Dieser wird in der Regel auch nicht benötigt, da automatisch geprüft wird, in welche Typen der Wert ohne Datenverluste umgewandelt werden kann. Auch Casts auf andere Objekttypen, die evtl. beim Aufrufen von Methoden notwendig sind, werden automatisch vorgenommen, wenn sie nach den Regeln von Java zulässig sind. Dabei werden auch primitive Java-Datentypen konvertiert. Entgegen dem Verhalten von Java werden allerdings die Typen auch auf „kleinere“ Typen (z.B. int → short) konvertiert, wenn dies ohne Datenverlust möglich ist.

In WTScript greifen Sie wie folgt auf Elemente von Java-Objekten, d.h. Attribute und Methoden, zu:

`var foo =` *objectname*`.`$\begin{cases} method() \\ attribute \end{cases}$`;`

---

objectname
: Name des Java-Objekts, dessen Methode *method()* ausgeführt bzw. auf dessen Attribut *attribute* zugegriffen werden soll.

method()
: Name der Methode, die ausgeführt werden soll.

attribute
: Name des Attributs, auf das zugegriffen werden soll.

Beispiel

```
<wtOnCreateScript>
<!--
 // Java-String erzeugen
  myString = new WT_JAVA.java.lang.String("Hello world at " + new Date() +
                                                                    "\r\n");
// Aufruf der Java-Methode die den String in ein Byte-Array umwandelt
  byteArray = myString.getBytes();
// Java FileOutputStream (Datei zum Anhängen öffnen) erzeugen
  fileout = new WT_JAVA.java.io.FileOutputStream("D:\\temp\\jtest.txt",
                                                                     true);
// Java-Methode aufrufen und als Argument Java-Byte-Array übergeben
  fileout.write (byteArray);
// Datei schließen
  fileout.close();
//-->
</wtOnCreateScript>
```

12.1.6 Zugriff auf Klassenelemente

In WTScript greifen Sie wie folgt auf Klassenelemente, d.h. Attribute und Methoden, zu.

`var foo = WT_JAVA.`*classname*`.`{*method*`()`, *attribute*}`;`

---

classname
: Name der Java-Klasse, deren Methode *method()* ausgeführt bzw. auf deren Attribut *attribute* zugegriffen werden soll.

method()
: Name der Methode, die ausgeführt werden soll.

attribute
: Name des Attributs, auf das zugegriffen werden soll.

12.1.7 Java-Methoden in WTScript aufrufen

Beim Aufruf von Java-Methoden in WTScript sind einige Besonderheiten bei Rückgabewert, Parameterübergabe und Ausnahmenbehandlung zu beachten.

Rückgabewert

In WTScript werden Rückgabewerte von Java-Methoden wie folgt behandelt:

- Falls der Rückgabewert gemäß Signatur der Java-Methode von primitivem Datentyp ist, kann er in WTScript wie ein Wert von primitivem Datentyp verwendet werden.
- Falls der Rückgabewert ein Java-Objekt ist, wird dieses nach Möglichkeit genauso konvertiert wie ein WTScript-Objekt.
- Liefert eine Java-Methode `null` als Ergebnis, so ist das WTScript-Ergebnis `undefined`.

Parameterübergabe

Als Argumente beim Aufruf von Java-Methoden sind zulässig:

– Werte primitiven Datentyps
– Objekte, die durch Java-Methodenaufrufe erzeugt wurden

Andere Objekte sind als Argumente bei Aufrufen von Java-Methoden nicht zulässig.

Java-Objekte der Typen `Byte`, `Short`, etc. werden von Java als Objekte behandelt, eine Konvertierung in primitive Objekte muss explizit vorgenommen werden.

Im Einzelnen sind die zulässigen Parametertypen sowie die bei der Parameterübergabe ggf. vorgenommenen Typ-Konvertierungen in der Tabelle auf Seite 366 dargestellt

WTScript-Typ	Konvertierung in Java-Typ
boolean- / Boolean-Objekte	Boolean
string- / String-Objekte	Default: `java.lang.String` Wenn der String nur ein Zeichen enthält, wird auch eine Konvertierung in `char` versucht.
number- / Number-Objekte	Default: `double` Falls eine Konvertierung in den Default-Typ nicht möglich ist, wird auf der Suche nach einer passenden Konvertierung eine Konvertierung in alle anderen numerischen Typen versucht, solange eine Konvertierung ohne Verlust der Genauigkeit möglich ist.
Java-Objekte	Default: Typ, der bei der Erzeugung des Objekts verwendet wurde. Falls eine Konvertierung in den Default-Typ nicht möglich ist, wird auf der Suche nach einer passenden Konvertierung eine Konvertierung in alle Superklassen und implementierten Interfaces versucht.
Java-Primitives	Default: Korrespondierender primitiver Java-Typ Falls eine Konvertierung in den Default-Typ nicht möglich ist, wird auf der Suche nach einer passenden Konvertierung eine Konvertierung in alle Typen versucht, für die der Wert gültig ist. `char`-Werte werden nicht als numerischen Werte behandelt.
andere WTScript-Objekte	als Argument nicht zulässig

Typ-Konvertierungen bei Java-Methodenaufrufen und Zuweisungen

Abarbeitung eines Java-Methodenaufrufs durch WTScript

Bei der Abarbeitung eines Java-Methodenaufrufs *m*(*arg-1*, *arg-2*, ... , *arg-n*) verfährt WTScript wie folgt:

1. WTScript sucht im aktuellen Objekt nach einer Java-Methode *m*, deren Signatur zu den Default-Java-Typen passt:
   - Bei erfolgreicher Suche wird die Methode *m* ausgeführt.
   - Bei erfolgloser Suche führt WTScript Schritt 2 der Reihe nach für die einzelnen Argumente *arg-1*, *arg-2*, ... durch.
2. WTScript sucht eine Methode *m* mit einer Signatur, die kompatibel ist zum überprüften Argument, d.h. deren korrespondierender formaler Parameter einen Datentyp aufweist, in den das Argument konvertiert werden kann:
   - Bei erfolgloser Suche wird eine entsprechende Fehlermeldung ausgegeben und die Methode *m* nicht ausgeführt.
   - Bei erfolgreicher Suche wird Schritt 2 mit dem nächsten Parameter wiederholt usw., bis ggf. alle Argumente überprüft sind.

   Kann auf diese Weise eine Java-Methode *m* mit einer zu *arg-1* bis *arg-n* kompatiblen Signatur gefunden werden, dann wird diese Methode ausgeführt.

Ausnahmenbehandlung

Ausnahmen (Exceptions), die bei Methodenaufrufen auftreten können, werden in WTScript-Ausnahmen „umgewandelt“ und lassen sich mit dem WTScript-`try`/`catch`-Mechanismus abfangen (siehe Abschnitt „Fehlerbehandlung durch Ausnahmen (Exceptions)“ auf Seite 322).

12.1.8 Attribute lesen und ändern

Bei schreibendem Zugriff auf Attribute gelten im Wesentlichen die gleichen Regeln wie bei Methodenaufrufen. WTScript-Objekte der Typen `Number`, `Boolean`, `String` werden in primitive Typen `number`, `boolean`, `string` umgewandelt. Danach wird geprüft, ob sie zum Typ des Attributes kompatibel sind.

Wird der Name eines Attributs nicht gefunden, wird dieser Fehler verschwiegen und kein Attribut angelegt. Bei lesendem Zugriff auf ein nicht existierendes Attribut ist das Ergebnis `undefined`.

12.1.9 Java-Arrays in WTScript erzeugen und verwenden

Java-Objekte erzeugen Sie in WTScript mit dem `new`-Operator. Die Elemente des Arrays sind nicht initialisiert.

```
var = new WT_JAVA.classname [size];
```

classname
: Name der Klasse, der die Array-Elemente angehören. *classname* muss immer der vollqualifizierte Klassenname sein.

size
: spezifiziert die Anzahl der zu erzeugenden Elemente. *size* kann wahlweise als Zahl oder als String (z.B. "12") angegeben werden. Auch Variablen sind möglich.

  Strings der Form [0-9]*[0-9a-zA-Z]* werden akzeptiert, wobei die Ziffern am Anfang in eine Zahl umgewandelt werden und der Rest ignoriert wird. Beginnen diese Strings nicht mit einer Ziffer, dann ist die Angabe ungültig und führt zu einem Fehler.

Arrays mit Elementen primitiven Java-Datentyps

Arrays, deren Elemente von primitivem Java-Datentyp sind, können Sie in WTScript nicht direkt mit dem `new`-Operator erzeugen. Wenn Sie dennoch solche Arrays benötigen, können Sie die folgende Klasse verwenden:

```
public class ArrayFactory {
  private ArrayFactory() {
  }
  public static byte[] newByteArray(int i) {return new byte[i];}
  public static short[] newShortArray(int i) {return new short[i];}
  public static int[] newIntArray(int i) {return new int[i];}
  public static long[] newLongArray(int i) {return new long[i];}
  public static float[] newFloatArray(int i) {return new float[i];}
  public static double[] newDoubleArray(int i) {return new double[i];}
  public static char[] newCharArray(int i) {return new char[i];}
  public static boolean[] newBooleanArray(int i) {return new boolean[i];}
}
```

Mehrdimensionale Arrays

Mehrdimensionale Arrays müssen Sie über die Schnittstellen von `java.lang.reflect.Array` erzeugen.

Beispiel

```
m = new WT_JAVA.MyObject; ------------------------------------------------ (1)
c = m.getClass();
MyObjectArray = WT_JAVA.java.lang.reflect.Array.newInstance(c,10); (2)
                                        //Array von MyObjects, 10 Elemente
c = MyObjectArray.getClass(); ------------------------------------------- (3)
MultiArray = WT_JAVA.java.lang.reflect.Array.newInstance(c, 20); -------  (4)
```

Diese Anweisungen (1) - (4) entsprechen der folgenden Java Anweisung:

```
MyObject[][] MultiArray = new MyObject[20][10];
```

Die Anweisungen (1) - (2) können Sie durch folgende Anweisung ersetzen:

```
MyObjectArray = new WT_JAVA.MyObject[10];
```

Auf Elemente von Arrays kann mit der von WTScript bekannten Syntax zugegriffen werden. Für Zuweisungen an Array-Elemente gelten die gleichen Regeln wie bei Feldern. Allerdings wird der übergebene Index auf Gültigkeit geprüft, bevor der Zugriff erfolgt.

Bei Zuweisungen zwischen Array-Variablen gelten die in Java festgelegten Regeln zur Typ-Überprüfung. Da die Java-Implementierung von WTScript die Typen vor der Zuweisung prüft, erledigt WTScript die Fehlerbehandlung und es wird keine Java-Ausnahme geworfen, wenn Anweisungen auftreten, die die Typregeln verletzen

WTScript-Arrays werden nie in Java-Arrays konvertiert, sondern wie andere WTScript-Objekte behandelt und bei Interaktion mit Java ignoriert.

12.1.10 WTScript-Operatoren mit Java-Objekten verwenden

WTScript-Operatoren können auch auf Java-Objekte angewendet werden. Bei der Verwendung des „+“- Operators werden Java-Objekte immer in Strings umgewandelt.

Wenn Sie numerisch rechnen wollen, müssen Sie die Objekte zuerst in Zahlen umwandeln.

Beispiel

Statt

```
x = o1 + o2;          // Stringverknüpfung
```

schreiben Sie

```
x = 1*o1 + 1*o2;      // Addition
```

12.1.11 Beispiel

Im folgenden Beispiel wird ein Java-String erzeugt und in eine Datei ausgegeben. Für die Dateiausgabe wird der String zuvor in ein Byte-Array umgewandelt.

```
<wtOnCreateScript>
<!--

// Java-String erzeugen
myString = new WT_JAVA.java.lang.String("Hello world at " + new Date() + "\r\n");

// Aufruf der Java-Methode, die den String in ein Byte-Array umwandelt
byteArray = myString.getBytes();

// Java FileOutputStream (Datei zum Anhängen öffnen) erzeugen
fileout = new WT_JAVA.java.io.FileOutputStream("D:\\temp\\jtest.txt", true);

// Java-Methode aufrufen und als Argument Java-Byte-Array übergeben
fileout.write (byteArray);

// Datei schließen
fileout.close();

//-->
</wtOnCreateScript>
```

12.2 C/C++-Userexits nutzen

C/C++-Userexits werden durch Methoden der Klasse WT_Userexit (siehe Abschnitt „WT_Userexit-Klasse“ auf Seite 274) aufgerufen.

Sie können auch mit mehreren Userexit-Bibliotheken arbeiten.

12.2.1 Ausgelieferte Dateien zur Unterstützung von C/C++-Userexits

Zur Unterstützung von Userexits stellt WebTransactions im Verzeichnis *install_dir*`/lib` folgende Dateien zur Verfügung:

`WTUserexit.c`
: Sourcen von Userexits. Einige Beispiele sind hier bereits enthalten. Eigene Userexits können Sie hinzufügen.

`WTPublic.h`
: Header-Datei mit den Funktionen, die innerhalb eines Userexit verwendbar sind und die Anweisungen und den Auswertungsoperator von WebTransactions abbilden.

`WTUserexits.so` (für Unix-Plattformen) / `WTUserexits.dll` (für Windows)
: Gemeinsam nutzbare Bibliothek, in der die ausgelieferten Beispiel-Userexits bereits eingebunden sind. Diese Bibliothek verwendet WebTransactions als Default-Bibliothek, falls keine Bibliothek angegeben ist. Eigene Userexits können Sie hinzubinden (siehe unten). Sie sollten aber dann die Bibliothek aus dem Installationsverzeichnis in Ihr Basisverzeichnis kopieren. Sie können aber auch eigene Userexit-Bibliotheken erzeugen.

`WTSystemExits.so` (für Unix-Plattformen) / `WTSystemExits.dll` (für Windows)
: Gemeinsam nutzbare Bibliothek, die die von den Start-Templates verwendeten Userexits enthält, z.B. `Getfile` (siehe „Beispiel 1“ auf Seite 374).

  Diese fertig ausgelieferten Userexits sind in Abschnitt „Mit WebTransactions fertig ausgelieferte C/C++-Userexits“ auf Seite 376 beschrieben. Für BS2000/OSD sind die fertig ausgelieferten Userexits im WTHolder-Programm enthalten.

12.2.2 C/C++-Userexits definieren

Definieren Sie die Funktion für den Userexit als:

```
char *NewUserExit (void *holder, int ac, char *av[]);
```

holder Steuerstruktur der Sitzung

ac enthält die Anzahl der Argumente

av enthält die Argumente, die beim Aufruf der Methode `WT_Userexit.`*function*`(...)` angegeben sind.

Die Funktion muss eine Zeichenkette zurückliefern, welche das Template liest.

Das Working-Directory von WebTransactions ist das Session-Directory im *tmp*-Verzeichnis. Startet der Userexit einen Prozess, der länger lebt als WebTransactions, dann sollten Sie für diesen Prozess das Working-Directory wechseln. Andernfalls kann WebTransactions bei Sitzungsende das temporäre Session-Directory nicht entfernen.

12.2.3 C/C++-Userexits einbinden

Das Einbinden der Userexits unterscheidet sich je nach WebTransactions-Plattform.

Auf Unix- und Windows-Plattformen integrieren Sie die Userexits entweder in die gemeinsam nutzbare Default-Bibliothek `WTUserexits.{so|dll}` oder stellen sie in eigenen gemeinsam nutzbaren Bibliotheken zur Verfügung.

Unter OSD werden die Userexits fest in das WTHolder-Programm eingebunden.

Windows

Um die von Ihnen definierten Userexits zu übersetzen und in die gemeinsam nutzbare Bibliothek `WTUserexits.dll` einzubinden oder um eigene Userexit-Bibliotheken zu erstellen, verwenden Sie eine Entwicklungsumgebung, die die Erzeugung von dll-Bibliotheken erlaubt, z.B. Visual C++.

Richten Sie in dieser Entwicklungsumgebung ein Projekt ein, das eine dll-Bibliothek erzeugt. Diesem Projekt können Sie anschließend den Beispiel-Source-Code WTUserexit.c hinzufügen.

Da Userexits gezielt über den Namen der dll-Bibliothek aufgerufen werden, ist es möglich, verschiedene dll-Bibliotheken mit jeweils unterschiedlicher Funktionalität einzurichten.

Falls Sie Funktionen von WebTransactions aus `WTPublic.h` in Userexits verwenden, müssen Sie die `WTKernel.lib` dazubinden.

Unix-Plattformen (nur WebTransactions-Liefereinheiten openUTM/OSD/MVS)

Um die von Ihnen definierten Userexits zu übersetzen und in die gemeinsam nutzbare Bibliothek *library* aufzunehmen, geben Sie folgenden Aufruf ein:

```
cc -G -share -o library.so userexit1.c ... userexitn.c
```

library`.so`
: Mit *library* spezifizieren Sie die Bibliothek. Existiert noch keine Bibliothek *library*`.so`, so wird eine solche erzeugt.

userexit1`.c ... `*userexitn*`.c`
: Sourcen der C/C++-Funktionen, die übersetzt und eingebunden werden sollen.

OSD (nur WebTransactions-Liefereinheiten openUTM/OSD)

Unter POSIX können keine gemeinsam nutzbaren Bibliotheken (shared libraries) eingesetzt werden: Alle Userexits müssen fest ins WTHolder-Programm eingebunden sein. Hierzu binden Sie mit Hilfe der mitgelieferten Makefile das WTHolder-Programm neu - zusammen mit den Userexits. Sämtliche Module des WTHolder-Programms stehen Ihnen in folgender Bibliothek zur Verfügung:

install_dir`/lib/libWTHolderUTMV4.a (für WTHolderUTMV4)`

Gehen Sie folgendermaßen vor:

– Erweitern Sie die mit ausgelieferte Datei `WTuserexits.c` um Ihre neuen Userexits.

– Prüfen Sie, ob der Name Ihres Userexits eventuell mit dem Namen eines mitausgelieferten Userexits übereinstimmt. Da kein Userexit den gleichen Namen benutzen darf, wie die mit WebTransactions ausgelieferten, müssen Sie Ihren Userexit in diesem Fall umbenennen. Dies gilt vor allem dann, wenn Sie im Quellcode gelieferte Exits wie „Getfile“ als eigene Exits weiterentwickeln.

– Erzeugen Sie mit Hilfe des mit ausgelieferten Makefile (*install_dir*`/lib/Makefile`) ein neues Programm `WTHolderUTMV4`.

– Kopieren Sie dieses neue Programm ins Basisverzeichnis unter dem Namen `WTHolder`.

Sie müssen die im Makefile verwendeten Namen für die UPIC- und CMX-Bibliotheken ggf. Ihrem aktuellen System anpassen.

12.2.4 Beispiele für C/C++-Userexits

Beispiel 1

Der folgende Userexit liefert den Inhalt einer Textdatei zurück. Er ist in der ausgelieferten Userexit-Bibliothek `WTSystemExits.{dll|so}` enthalten.

```
/*****************************************************************************/
/*                                                                           */
/* Getfile( file )                                                           */
/*                                                                           */
/*****************************************************************************/
/*                                                                           */
/* The contents of the textfile file is returned.                            */
/*                                                                           */
/* This exit is used by predefined start pages (wtstart.htm, wtstartUTM.htm,..)*/
/* and should neither be modified nor removed !!!                            */
/*                                                                           */
/*****************************************************************************/
char* Getfile(void *wtholder, int ac, char *av[])
{
    FILE*       p_file;
    int         bytesRead;
    char*       ct;

    if ( ac == 0 )
        return( "" );

    p_file = fopen (av[0], "r" );    /* open file for reading */
    if ( p_file == NULL )
        return( "" );

    /* Since th caller does not free the returned Data reuse the buffer */
    returnStringLen = 0;
    if ( returnStringSize - returnStringLen < 2 )
        returnString = realloc( returnString, returnStringSize += 1024 );

    /* read the Lines in the file and append them to the return string */
    while( ( bytesRead = fread( returnString + returnStringLen, 1,
                                  returnStringSize -
 returnStringLen, p_file ) ) != 0 )
    {
        returnStringLen += bytesRead;
        if ( returnStringSize - returnStringLen < 2 )
            returnString = realloc( returnString, returnStringSize += 1024 );
    }
```

```
    returnString[returnStringLen] = ´\0´;
    return returnString;
}
```

Beispiel 2

Das Beispiel zeigt den Userexit `UXEurope` zur Auswertung der Bildkoordinaten eines „clickable image“:

```
char *UXEurope (wt_holderCommId *wtholder, int ac, char *av[])
{
    int x,y;


    if (ac == 2)
    {
        x = (int) atoi (av[0]);
        y = (int) atoi (av[1]);

        if (x > 180 && y > 90 && x < 270 && y < 125)
        return ("1"); /* Belgium */
        if (x > 160 && y > 184 && x < 228 && y < 213)
        return ("2"); /* France */
        if (x > 232 && y > 60 && x < 325 && y < 90)
        return ("3"); /* Germany */
        if (x > 400 && y > 260 && x < 500 && y < 310)
        return ("4"); /* Greece */
        if (x > 280 && y > 210 && x < 350 && y < 260)
        return ("5"); /* Italy */
        if (x > 0 && y > 240 && x < 100 && y < 280)
        return ("6"); /* Portugal */
        if (x > 40 && y > 290 && x < 120 && y < 330)
        return ("7"); /* Spain */
        if (x > 200 && y > 140 && x < 330 && y < 180)
        return ("8"); /* Switzerland */
        if (x > 70 && y > 20 && x < 250 && y < 60)
        return ("9"); /* United Kingdom */
    }
    return ("0");
}
/wtOnCreateScript>
```

12.3 Mit WebTransactions fertig ausgelieferte C/C++-Userexits

WebTransactions stellt Ihnen einen Satz von bereits fertig definierten C/C++-Userexits zur Verfügung. Diese werden z.T. von WebTransactions intern genutzt - z.B. in den Start-Templates.

Die Userexits, die im Folgenden beschrieben sind, können Sie jedoch auch für die von Ihnen erstellten oder angepassten Templates nutzen. Der Quellcode dieser Userexits wird in der Datei `WTUserExits.c` für eigene Weiterentwicklungen zur Verfügung gestellt (einzige Ausnahme: der Quellcode des Userexits `GetInstallDir` wird **nicht** mit ausgeliefert).

Die fertig definierten Userexits befinden sich je nach WebTransactions-Plattform:

OSD im WTHolder und müssen so angesprochen werden, als würden sie sich in einer Bibliothek mit dem Namen `WTSystemExits` befinden

Unix-Plattform
in der Bibliothek `WTSystemExits.so`

Windows
in der Bibliothek `WTSystemExits.dll`

Die folgende Tabelle gibt einen Überblick über diese Userexits:

Name	Funktion	Beschreibung
`CheckLogin`	prüft, ob das angegebene Kennwort der Benutzerkennung zugeordnet ist.	Seite 378
`CheckProcess`	prüft, ob die angegebene Prozessnummer im System existiert und ob der Prozess ein WTHolder-Prozess ist.	Seite 378
`Creationtime`	liefert den Erstellungszeitpunkt der angegebenen Datei im cfile-Format	Seite 379
`Delfile`	löscht eine Datei. Diese Funktion benötigt zur Ausführung einen Administrationsprozess als Umgebung.	Seite 380
`FreeBuffer`	gibt den Puffer frei, der von `GetDir` und `GetFile` (gemeinsam) benutzt wird.	Seite 380
`FreeNameInPool`	gibt standardmäßig den ersten durch den Prozess reservierten Eintrag aus einer Liste frei - falls gewünscht auch einen bestimmten Namen aus der Liste oder alle reservierten Einträge.	Seite 380
`Getdate`	liefert die angegebene Zeit im cfile-Format.	Seite 381
`Getdir`	liefert die Namen aller Dateien, die sich im angegebenen Verzeichnis befinden und (optional) einem angegebenen Muster entsprechen.	Seite 381
`Getfile`	liefert den Inhalt der angegebenen Datei als Zeichenkette.	Seite 381

Name	Funktion	Beschreibung
GetInstallDir	liefert das Installationsverzeichnis von WebTransactions. Bei diesem Userexit wird der Quellcode **nicht** mit ausgeliefert.	Seite 382
Gettime	liefert die aktuelle Zeit im cfile-Format.	Seite 382
LockNameInPool	reserviert den ersten freien Eintrag aus einer Liste. Reserviert optional einen bestimmten Namen aus der Liste.	Seite 383
Modificationtime	liefert die Zeit der letzten Veränderung der angegebenen Datei.	Seite 383
Putfile	erzeugt eine Datei mit dem Inhalt der übergebenen Zeichenkette. Diese Funktion benötigt zur Ausführung einen Administrationsprozess als Umgebung.	Seite 384
ReleaseStationName	gibt einen mit ReserveStationName reservierten Namen frei.	Seite 384
ReplaceByConfigFile	ersetzt eine Zeichenfolge durch eine andere - gesteuert durch eine Ersetzungstabelle.	Seite 385
ReserveStationName	stellt sicher, dass Namen zeitgleich jeweils nur einmal verwendet werden. Prüft beispielsweise, ob der gegebene Stationsname für eine Verbindung zu einer OSD-Anwendung bereits verwendet wird.	Seite 386
SendMail	verschickt eine Nachricht mit dem SMTP-Protokoll (nur Text, keine MIME-Unterstützung).	Seite 387
WTSleep	versetzt den Holder für die angegebene Anzahl von Millisekunden in einen Wartezustand.	Seite 388

12.3.1 CheckLogin

```
CheckLogin( config_file, user_name, password )
```

Aufgabe:

Prüft, ob das angegebene Kennwort der Benutzerkennung zugeordnet ist. *config_file* wird relativ zum Basisverzeichnis gesucht. *config_file* muss in jeder Zeile zwei Spalten enthalten, die durch WhiteSpace-Zeichen (siehe Seite 32) getrennt sind.

Ergebnis:

Falls das Kennwort dem Benutzernamen zugeordnet ist, wird der Benutzername zurückgegeben, andernfalls ein Leerstring.

Beispiel

Inhalt von *basedir/config_file*:

```
user      password
smith     key
...
```

12.3.2 CheckProcess

```
CheckProcess( process_id )
```

Aufgabe:

Prüft, ob die angegebene Prozessnummer im System existiert und ob der Prozess ein WTHolder-Prozess ist.

Ergebnis:

`"alive"`
Die angegebene Prozessnummer existiert im System und der Prozess ist ein WTHolder-Prozess.

`"dead"` oder `""` (Leerstring)
Die angegebene Prozessnummer im System existiert nicht oder der Prozess ist kein WTHolder-Prozess.

12.3.3 Creationtime

`Creationtime( file_name )`

Aufgabe:

Liefert den Erstellungszeitpunkt der angegebenen Datei im ctime-Format. Der Pfad kann absolut oder relativ angegeben werden. Relative Pfadangaben beziehen sich auf das temporäre Sitzungsverzeichnis.

Ergebnis:

Erstellungszeitpunkt als Zeichenkette. Falls die Datei nicht existiert, liefert dieser Userexit ein Fragezeichen.

Beispiel

```
<wtoncreatescript>
<!--
ex = new WT_Userexit('WTSystemExits');
document.writeln('<br> Creationtime("session info")=
',ex.Creationtime('../'+WT_SYSTEM.SESSION+'.info'));
document.writeln('<br> Creationtime(WT_SYSTEM.BASEDIR+"/basedir_file")=
',ex.Creationtime(WT_SYSTEM.BASEDIR+'/config'));
document.writeln('<br> Creationtime("c:/abs_file")=
',ex.Creationtime('c:/windows'));
document.writeln('<br> Creationtime("unknown_file")=
',ex.Creationtime('unknown_file'));
//-->
</wtoncreatescript>
```

Das Beispiel liefert folgendes Ergebnis:

```
Creationtime("session info")=Tue Jun 08 16:10:28 2010
Creationtime(WT_SYSTEM.BASEDIR+"/basedir_file")=Tue Jun 08 12:37:23 2010
Creationtime("c:/abs_file")=Thu Nov 02 13:18:34 2006
Creationtime("unknown_file")=?
```

12.3.4 Delfile

```
Delfile( file_name )
```

Aufgabe:
Löscht die angegebene Datei. Der Pfad kann absolut oder relativ angegeben werden. Relative Pfadangaben beziehen sich auf das temporäre Sitzungsverzeichnis.

Ergebnis:
Leerstring bei Erfolg oder Fehlermeldung.

12.3.5 FreeBuffer

```
FreeBuffer()
```

Aufgabe:
Freigabe des Puffers, der von `GetDir` und `GetFile` gemeinsam benutzt wird.

Ergebnis:
kein Rückgabewert.

12.3.6 FreeNameInPool

```
FreeNameInPool ( config_file [,  {name_if_not_first_reserved_by_this_process | "ALL"}  ] )
```

Aufgabe:
Gibt standardmäßig den ersten durch diesen Prozess reservierten Eintrag aus einer Liste frei. Gibt optional einen bestimmten Namen aus der Liste oder alle reservierten Einträge frei.

Ergebnis:
Name des freigegebenen Eintrags oder Leerstring. Leerstring wird geliefert, wenn alle Einträge freigegeben wurden oder wenn kein Eintrag freigegeben werden konnte.

Verweis:
LockNameInPool

Beispiel

siehe `LockNameInPool`

12.3.7 Getdate

`Getdate(` *numeric_time_value* `)`

Aufgabe:
Liefert die angegebene Zeit im ctime-Format.

Ergebnis:
Zeit als Zeichenkette oder Leerstring, wenn der Zeitwert ungültig war.

12.3.8 Getdir

`Getdir(` *dir_name_relative_to_basedir* `[ ,` *pattern* `]` `)`

Aufgabe:
Liefert die Namen aller Dateien, die sich im angegebenen Verzeichnis befinden und (optional) dem Muster *pattern* entsprechen.

Ergebnis:
Zeichenkette, die durch Neue-Zeile-Zeichen getrennt die Dateinamen enthält.

Nach dem letzten Aufruf von `Getdir/Getfile` sollte `FreeBuffer()` benutzt werden, um den angeforderten Puffer freizugeben.

12.3.9 Getfile

`Getfile(` *file_name* `)`

Aufgabe:
Liefert den Inhalt der angegebenen Datei, relative Pfadangaben beziehen sich auf das temporäre Sitzungsverzeichnis.

Ergebnis:
Datei-Inhalt als Zeichenkette.

Nach dem letzten Aufruf von `Getdir/Getfile` sollte `FreeBuffer()` benutzt werden, um den angeforderten Puffer freizugeben.

12.3.10 GetInstallDir

```
GetInstallDir()
```

Aufgabe:
Liefert das Installationsverzeichnis von WebTransactions.

Ergebnis:
Installationsverzeichnis.

Bei diesem Userexit wird der Quellcode **nicht** mit ausgeliefert.

12.3.11 Gettime

```
Gettime()
```

Aufgabe:
Liefert die aktuelle Zeit im ctime-Format.

Ergebnis:
Zeit als Zeichenkette.

12.3.12 LockNameInPool

```
LockNameInPool( config_file [,name] )
```

Aufgabe:

Reserviert den ersten freien Eintrag aus einer Liste. Reserviert optional einen bestimmten Namen aus der Liste. Ist ein Name als reserviert gekennzeichnet, aber der reservierende Prozess existiert nicht mehr, wird er wie frei behandelt und neu vergeben.

Ergebnis:

Name des reservierten Eintrags oder Leerstring.

Verweis:

FreeNameInPool

Beispiel:

Inhalt von *basedir/config_file*:

```
FREE            STATION1   comment
FREE            STATION2   comment
...
```

Die ersten 16 Bytes der *config_file* enthalten `"FREE "` oder die Prozessnummer des reservierenden Prozesses. Danach folgt der Name und ggf. nach WhiteSpace-Zeichen (siehe Seite 32) ein Kommentar.

12.3.13 Modificationtime

```
Modificationtime( file_name [, 'N' ])
```

Aufgabe:

Liefert die Zeit der letzten Veränderung der angegebenen Datei.

Ergebnis:

Zeichenkette im ctime-Format oder - falls beim Aufruf das optionale zweite Argument `'N'` angegeben wurde - als Zahl (internes Zeitformat).

12.3.14 Putfile

`Putfile( file_name , content , length )`

Aufgabe:

Erzeugt eine Datei mit dem Inhalt der übergebenen Zeichenkette, relative Pfadangaben beziehen sich auf das temporäre Sitzungsverzeichnis.

Ergebnis:

Leerstring bei Erfolg oder Fehlermeldung.

12.3.15 ReleaseStationName

`ReleaseStationName ( station_name )`

Aufgabe:

Freigeben eines mit `ReserveStationName` reservierten Namens.

Ergebnis:

`OK`

Der Stationsname wurde gefunden und der entsprechende Eintrag wurde aus der Datei mit den verwendeten Stationsnamen entfernt.

`NOTOK`

Der in *station_name* angegebene Wert entspricht keinem Stationsnamen.

`ERROR`

Bei der Ausführung des Userexits trat ein Fehler auf. In diesem Fall können Sie davon ausgehen, dass der angegebene Stationsname nicht freigegeben wurde.

Verweis:

ReserveStationName

12.3.16 ReplaceByConfigFile

`ReplaceByConfigFile( replace_file , key_to_be_replaced )`

Aufgabe:

Ersetzt eine Zeichenfolge durch eine andere gesteuert durch eine Ersetzungstabelle in *replace_file*. *replace_file* muss in jeder Zeile zwei Spalten enthalten, die durch WhiteSpace-Zeichen (siehe Seite 32) getrennt sind.

Ergebnis:

Ersatzzeichenfolge, falls der Eintrag gefunden wurde, andernfalls Leerstring.

Beispiel

Inhalt von *basedir/replace_file*:

```
key1           replacement1      COMMENT!
123.45.6.7     STATION2          relation ip-adress to station name
123.45.6.8     USERx             relation ip-adress to user name
user1          station1          relation user name to station
...
```

12.3.17 ReserveStationName

`ReserveStationName( station_name )`

Aufgabe:

Stellt sicher, dass Namen zeitgleich jeweils nur einmal verwendet werden. Prüft beispielsweise, ob der gegebene Stationsname für eine Verbindung zu einer OSD-Anwendung bereits verwendet wird.

Ergebnis:

`OK`

Der Stationsname wird derzeit nicht verwendet. Der Stationsname wird reserviert und ein entsprechender Eintrag wird in der Datei erzeugt, in der die verwendeten Stationsnamen verzeichnet sind.

`NOTOK`

Der Stationsname wird derzeit verwendet oder es ist kein solcher Stationsname definiert.

`ERROR`

Bei der Ausführung des Userexits trat ein Fehler auf. In diesem Fall können Sie davon ausgehen, dass der angegebene Stationsname nicht verwendet werden kann.

Verweis:

ReleaseStationName

12.3.18 SendMail

`SendMail(` *Server*, *From*, *To*, *CC*, *BCC*, *Subject*, *Body*, *Header*)

Aufgabe:

Verschickt eine Nachricht mit SMTP-Protokoll an einen oder an mehrere Empfänger. Dabei müssen Sie folgende Parameter versorgen:

Server Internetadresse oder Symbolischer Name des Mail-Servers.

From Mail-Adresse des Absenders.

To Mail-Adresse des Empfängers, Sie können auch mehrere Mail-Adressen angeben, die Sie mit einem Semikolon „;“ trennen.

CC Abkürzung für CarbonCopy; Mail-Adresse des Kopie-Empfängers, Sie können auch mehrere Mail-Adressen angeben, die Sie mit einem Semikolon „;“ trennen.

BCC Abkürzung für Blind Carbon Copy; Mail-Adresse des Kopie-Empfängers, Sie können auch mehrere Mail-Adressen angeben, die Sie mit einem Semikolon „;“ trennen.

Subject ⁎Betreff-Zeile der Mail.

Body Mail-Text.

Header Header-Zeile für die Mail, die Zeichenkette muss ggf. die notwendigen Zeilenvorschübe enthalten.

Server, *From*, *To*, *Subject* und *Body* sind Pflichtparameter. Die Parameter *CC* und *BCC* können auch als leere Zeichenkette übergeben werden.

Ergebnis:

Die Methode `SendMail` liefert die Zeichenkette `OK` oder eine Fehlermeldung zurück, falls der Verbindungsaufbau nicht durchgeführt werden konnte. Wenn die Kommunikation mit dem SMTP Server fehlerhaft war, wird dies in der WebTransactions-Trace-Datei protokolliert.

Folgende Rückgabewerte sind möglich:

„OK“

„Incomplete function call.“

„Memory allocation failed.“

„Function call: WSAStartup() failed.“

„Creation of a socket failed.“

„Function call: gethostbyname() failed.“

„Connection to socket failed.“

„Receive from socket failed.“

„Send to socket failed.“

Beispiel

```
<wtoncreatescript>
<!-
var MailServer = "smtpmail.server.de";
var MailFrom = "Bundestrainer@dfb.de";
var MailTo1 = "Terrier <Berti.Vogts@unknown.de>";
var MailTo2 = "Franzl <Franz.Beckenbauer@fcb.com>";
var MailCc = "Papst@vatican.va";
var MailBcc = "";
var MailSubject = "WM2010";
var MailBody = "Der Ball ist rund und das Spiel dauert 90 Minuten.";
var MailHeader = "Content-Type: text/plain; charset=ISO-8859-1";
SMTPExit = new WT_Userexit();
SMTPExit.SendMail(MailServer,MailFrom,MailTo1+';'+MailTo2,MailCc,
MailBcc,MailSubject,MailBody);
delete SMTPExit;
//-->
</wtoncreatescript>
```

12.3.19 WTSleep

```
WTSleep( [waittime] )
```

Aufgabe:

Dieser Systemexit versetzt den Holder für die angegebene Anzahl von Millisekunden in einen Wartezustand.

waittime

Ausdruck, der in den Typ `number` konvertiert wird und die Anzahl der abzuwartenden Millisekunden angibt. Wird kein oder ein ungültiger Parameter angegeben, wird der Defaultwert 1000 ms verwendet.

Hinweis: Auf der OSD-Plattform wird immer mindestens 1 Sekunde gewartet.

Ergebnis:

Der String `OK` wird zurückgegeben.

13 XML-Konvertierung

Dieses Kapitel beschreibt die grundlegenden Konzepte der XML-Konvertierung für folgende Aufgaben:

- die portable Darstellung von Daten für die Kommunikation mit externen Anwendungen über XML-Nachrichten (XML=eXtended Markup Language) in Abschnitt „XML-Dokumente importieren und exportieren“ auf Seite 389
- die Konvertierung von WTScript-Datenstrukturen in XML-Dokumente und umgekehrt in Abschnitt „Datenstrukturen exportieren“ auf Seite 394

Eine Beschreibung, wie die Klasse `WT_Filter` für die Kommunikation zwischen WebTransactions-Anwendungen verwendet werden kann (Konvertierung von WTML-Funktionsaufrufen), finden Sie im WebTransactions-Handbuch „Client-APIs für WebTransactions“.

13.1 XML-Dokumente importieren und exportieren

Dieser Abschnitt beschreibt, wie beliebige XML-Dokumente in WTScript-Datenstrukturen überführt, und umgekehrt, wie beliebige WTScript-Datenstrukturen als XML-Dokumente dargestellt werden können. Damit wird die Kommunikation mit beliebigen externen Anwendungen möglich, die XML-Dokumente erzeugen oder verarbeiten können.

Werden XML-Dokumente mit Hilfe der Klasse `WT_Filter` importiert, dann müssen sie in eine interne Darstellung als WTScript-Objektbaum transformiert werden. Dazu wird die Struktur des XML-Dokuments auf einen Objektbaum abgebildet, dessen Blätter jeweils ein XML-Element repräsentieren. Die folgenden Abschnitte beschreiben, wie diese Datenobjekte aufgebaut sind.

Sollen WTScript-Datenstrukturen mit der Methode `objectTreeToXML` exportiert werden, dann müssen diese Datenstrukturen ebenfalls zunächst in das nachfolgend beschriebene Format eines XML-Objektbaums überführt werden, bevor sie exportiert werden können.

13.1.1 Aufbau der Struktur eines importierten XML-Objekts

Die XML-Elemente werden in WebTransactions folgendermaßen repräsentiert:

```
name
attribute
child
0
1
...
```

`name`
: Ein Attribut mit Datentyp `string` mit dem Namen des XML-Elements

`attribute`
: Dieses Attribut ist ein Objekt vom Typ `object` aus der Klasse `Object`, das für jedes Attribut des XML-Elements ein Attribut besitzt.

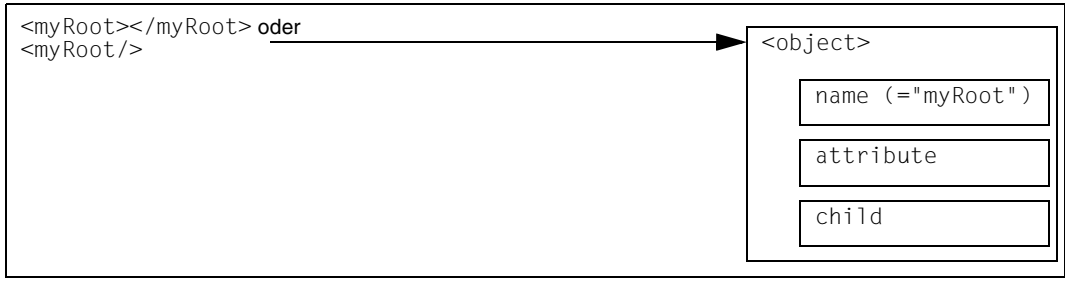
`child`
: Dieses Attribut ist ein Objekt vom Typ `object` aus der Klasse `Object`. Enthält das aktuelle XML-Element untergeordnete Elemente, so besitzt dieses Objekt für jeden Unterelementtyp ein Attribut mit dem Namen dieses XML-Elementtyps. Als Attribute dieses Unter-Objekts werden Verweise auf die Objekte `0`, `1`, ... des jeweiligen Typs eingefügt (siehe folgende Beschreibung).

`0`, `1`, ...
: Für jedes XML-Element innerhalb des aktuellen XML-Elements wird ein Attribut mit dem Index des XML-Unterelements als Name und dem Datentyp `object` und der Klasse `Object` angelegt, das wiederum als XML-Objekt aufgebaut ist.

13.1.2 Darstellung von XML-Elementen

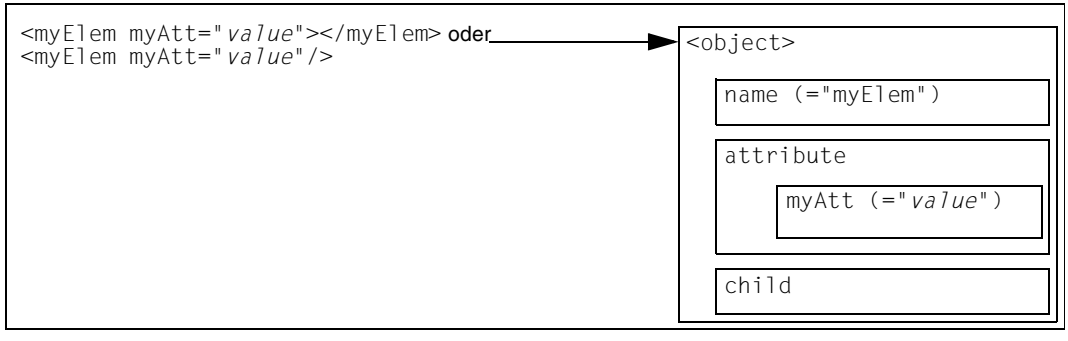
Die folgenden Abbildungen sollen dabei helfen, die oben beschriebenen Elemente der WTScript-Datenstruktur zu erläutern.

Darstellung eines einfachen XML-Elements



Das XML-Element `myRoot` wird auf ein Objekt abgebildet, bei dem das Attribut `name` mit dem Namen des XML-Elements (`myRoot`) besetzt ist.

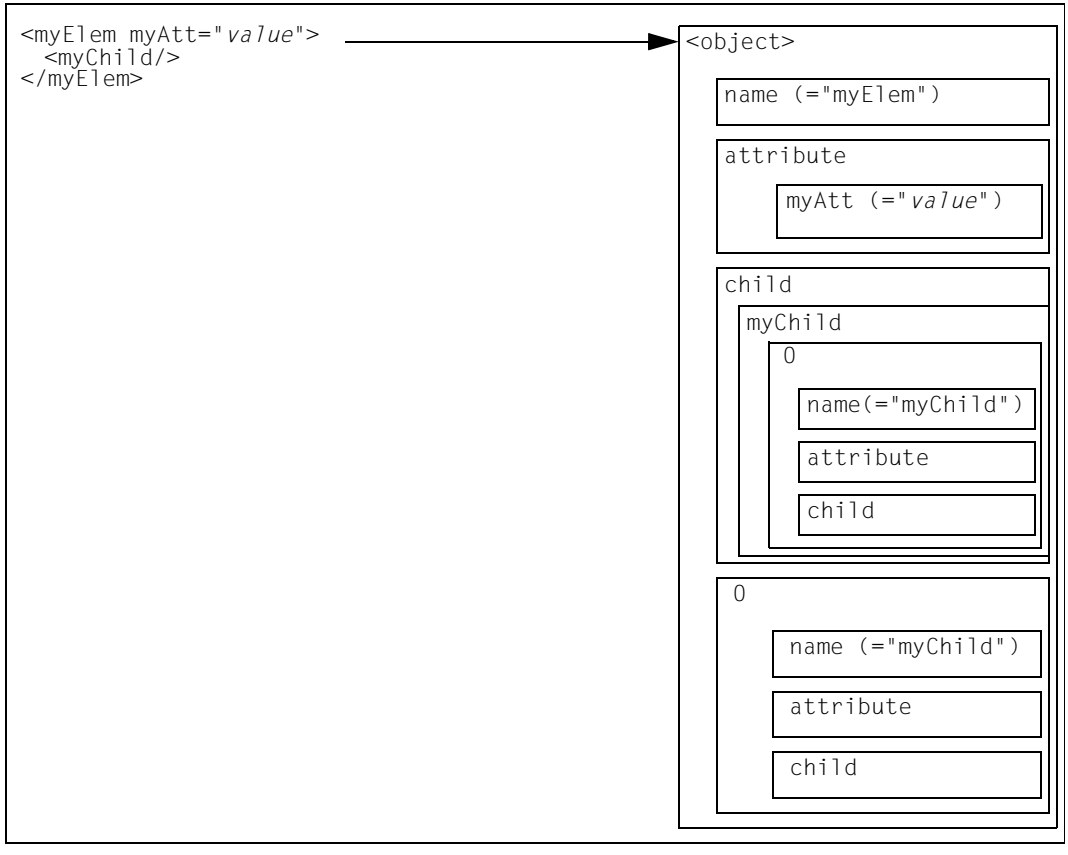
Darstellung eines XML-Elements mit Attributen



Das XML-Element `myElem` wird auf ein Objekt abgebildet, bei dem das Attribut `name` mit dem Namen des XML-Elements (`myElem`) besetzt ist.

Das Attribut `attribute` des Objekts erhält für jedes Attribut von `myElem` ein eigenes Attribut mit dem Namen des XML-Attributs (im Beispiel: `myAtt`) und dem Wert des XML-Attributs (im Beispiel: *value*).

Darstellung von Unter-Elementen eines XML-Elements



Das XML-Element `myElem` wird auf ein Objekt abgebildet, bei dem das Attribut `name` mit dem Namen des XML-Elements (`myElem`) besetzt ist.

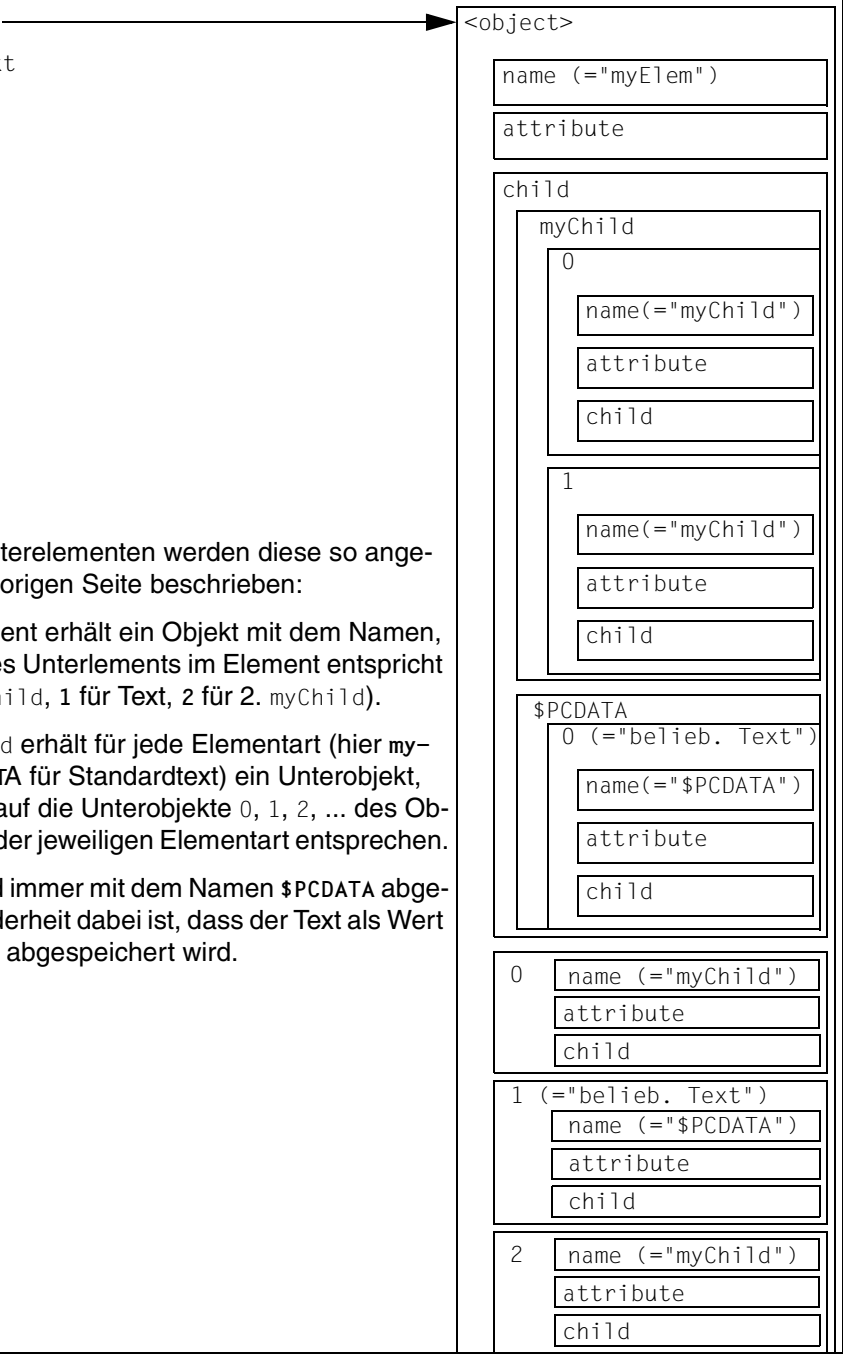
Das Attribut `attribute` des Objekts erhält für jedes Attribut von `myElem` ein eigenes Attribut mit dem Namen des XML-Attributs (im Beispiel: `myAtt`) und dem Wert des XML-Attributs (im Beispiel: *value*).

Zusätzlich wird für jedes Unterelement von `myElem` ein eigenes Unterobjekt in `<object>` angelegt, das entsprechend dem Index des Unterelements (0, 1, 2, 3, ...) benannt ist und rekursiv wieder so aufgebaut ist (im Beispiel: Objekt `0`).

Zusätzlich erhält das Attribut `child` für jeden XML-Elementtyp (hier `myChild`) ein Objekt, das wiederum aus Referenzen auf die zu diesem Objekttyp gehörenden Unterobjekte (0, 1, ...) besteht (siehe auch folgende Abbildung).

Darstellung mehrerer Unterelemente

```
<myElem>
    <myChild>
    belieb. Text
    <myChild/>
</myElem>
```



Bei mehreren Unterelementen werden diese so angelegt wie auf der vorigen Seite beschrieben:

Jedes Unterelement erhält ein Objekt mit dem Namen, der dem Index des Unterlements im Element entspricht (hier `0` für 1. `myChild`, `1` für Text, `2` für 2. `myChild`).

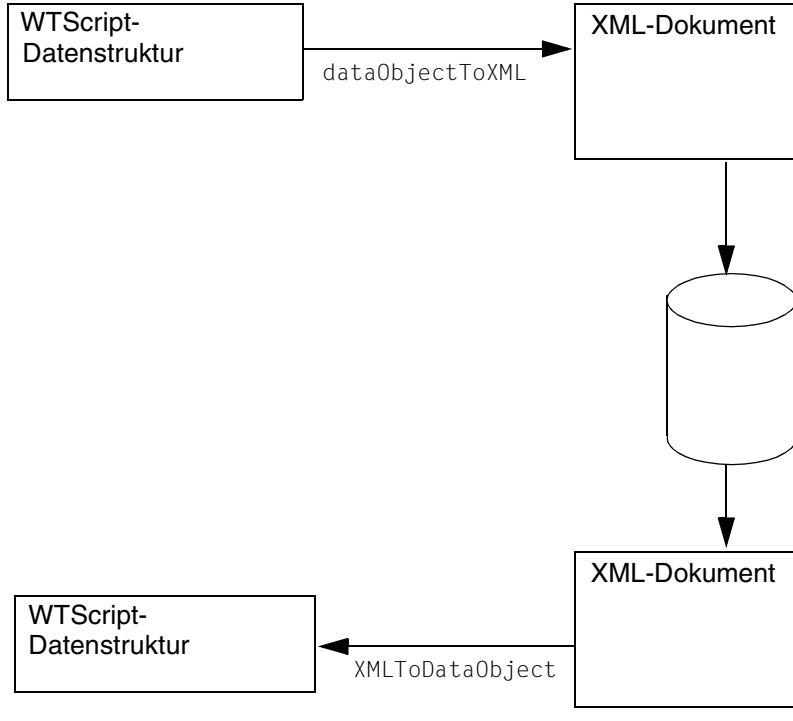
Das Attribut `child` erhält für jede Elementart (hier **`myChild`** und **`$PCDATA`**) ein Unterobjekt, das Referenzen auf die Unterobjekte `0`, `1`, `2`, ... des Objekts enthält, die der jeweiligen Elementart entsprechen.

Standardtext wird immer mit dem Namen **`$PCDATA`** abgelegt. Eine Besonderheit dabei ist, dass der Text als Wert des Unterobjekts abgespeichert wird.

13.2 Datenstrukturen exportieren

Dieser Abschnitt beschreibt Export und Import von WTScript-Datenstrukturen (Methoden `XMLToDataObject` und `dataObjectToXML`). Damit wird es möglich, WTScript-Datenstrukturen in ein speicherbares Format zur überführen, abzuspeichern und wieder einzulesen (mit den Methoden der Klasse `document`) und wieder in die entsprechenden WTScript-Datenstrukturen umzuwandeln.

Die Umwandlung erfolgt über die beiden Methoden `XMLToDataObject` und `dataObjectToXML`, wobei keine besonderen Konventionen zu beachten sind. Für die Umwandlung verwendet WebTransactions die unten beschriebene DTD (Document Type Definition, eine Beschreibung, wie ein XML-Dokument aufgebaut ist), die hier zu Informationszwecken aufgeführt ist.



DTD für die Darstellung von WTScript-Datenstrukturen in XML

Für die Repräsentation von WTScript-Datenstrukturen als XML-Dokumente gilt die folgende DTD:

```
<!ELEMENT data        ((undefined|number|boolean|
                        string|object)*)>
<!ELEMENT undefined   EMPTY>
<!ELEMENT number      (#PCDATA)>
<!ELEMENT boolean     (#PCDATA)>
<!ELEMENT string      (#PCDATA)>
<!ELEMENT object      (#PCDATA?
                       (undefined|number|boolean|
                        string|object|function)*)>
<!ELEMENT function    EMPTY>
<!ATTLIST undefined   name                      CDATA    #REQUIRED>
<!ATTLIST number      name                      CDATA    #REQUIRED>
<!ATTLIST boolean     name                      CDATA    #REQUIRED>
<!ATTLIST string      name                      CDATA    #REQUIRED>
<!ATTLIST object      name                      CDATA    #REQUIRED
                      class                     CDATA    #IMPLIED
                      reference                 CDATA    #IMPLIED>
<!ATTLIST function    name                      CDATA    #REQUIRED>
```

Neben der Verwendung innerhalb von WebTransactions erlaubt es der Export der Datenstrukturen gemäß dieser DTD auch, externe Anwendungen auf die solcherart gespeicherten WTScript-Datenstrukturen zugreifen zu lassen.

14 Beispiele

Die Beispiele in diesem Kapitel illustrieren das Zusammenspiel verschiedener WTML-Tags.

14.1 Stil umschalten

Dieses Beispiel zeigt, wie Sie einen Knopf zum Umschalten des Stils ergänzen und die Rückgabewerte in WebTransactions behandeln. Dazu wird in der if-Bedingung der `STYLE`-Knopf abgefragt:

– Wurde er gedrückt, wird das Systemattribut `STYLE` neu gesetzt. Da in diesem Fall keine Kommunikation mit der Host-Anwendung stattfindet, bleibt das Systemattribut `FORMAT` unverändert. Es wird also als Nächstes ein Template eingelesen, das dem aktuellen entspricht, jedoch einen anderen Oberflächenstil aufweist. WebTransactions sucht nun nicht im Standard-Template-Verzeichnis `config/forms` sondern in `config/graphic`.

– Ist der STYLE-Knopf nicht gedrückt, wird die Kommunikation mit dem Host durchgeführt. Als Folge wird das Template des nächsten Dialogschritts angezeigt, der Oberflächenstil bleibt unverändert.

HTML-Template:

```
...
<Input Type="SUBMIT" Name="STYLE" Value="Grafik">
...
<wtOnReceiveScript>
if ( WT_POSTED.STYLE == "Grafik")
   WT_SYSTEM.STYLE ="graphic";
else
   {
   WT_HOST.std.send();
   WT_HOST.std.receive();
   }
</wtOnReceiveScript>
```

14.2 Exit-Knopf abfragen

Um einen gezielten Ausstieg aus der Applikation anzubieten, wird in der if-Bedingung der Exit-Knopf abgefragt. Wurde er gedrückt, wird eine letzte Meldung ausgegeben und die Sitzung mit der globalen Funktion `exitSession()` beendet. Anderenfalls wird die Kommunikation mit dem Host durchgeführt.

HTML-Template:

```
....
<Input Type="SUBMIT" Name="Exit" Value="Ende">

<wtRem Ende oder Kommunikation mit Host>
<wtOnReceiveScript>
if ( WT_POSTED.Exit == "Ende")
   {
   document.write("Das war's ...");
   WT_HOST.std.close();
   exitSession();
   }
else
   {
   WT_HOST.std.send();
   WT_HOST.std.receive();
   }
</wtOnReceiveScript>
```

14.3 Daten speichern mit XML-Konvertierung

Dieses Beispiel geht von der Annahme aus, dass eine WebTransactions-Anwendung anwender-spezifische Daten (Anzeigestil, der vom Anwender ausgewählt werden kann, sowie verschiedene Anwenderdaten, z.B. Anwender-Nummer, letzte Verwendung der Anwendung etc.) verwalten soll.

Der Anwender identifiziert sich über eine Nummer, die auch als Basis für die Abspeicherung der Daten dient.

Die anwender-spezifischen Daten sind in der folgenden Datenstruktur abgelegt:

```
function UserData() { // Konstruktor
   // Klassen-Attribute:
   this.objectName = "";                // Name des Datenobjekts
   this.userNumber = 0;
   this.style = "";
   this.lastUsage = new Date;
   // Methoden (Speichern und Laden der Daten)
   this.save = saveUserData;
   this.load = loadUserData;
}
```

Die beiden Methoden `saveUserData` und `loadUserData` realisieren die Operationen zur XML-Konvertierung und Datenspeicherung:

```
function saveUserData() {
   XMLString = new String;

   // Sichere aktuell eingestellten Stil:
   this.style = WT_SYSTEM.STYLE
   // Konvertiere Daten nach XML:
   XMLString = WT_Filter.dataObjectToXML(this.objectName);
   // Schreibe Daten in Datei <WT_SYSTEM.BASEDIR>/<userNumber>.wtd:
   WT_Userexit.Putfile(WT_SYSTEM.BASEDIR + "/" + this.userNumber + ".wtd",
                       XMLString, XMLString.length);
}

function loadUserData() {
   XMLString = new String;

   // Daten aus Datei <WT_SYSTEM.BASEDIR>/<userNumber>.wtd laden:
   XMLString = WT_Userexit.Getfile(WT_SYSTEM.BASEDIR + "/"
                                   + this.userNumber + ".wtd")
   // XML in Daten (in Objekt this.objectName) konvertieren:
   WT_Filter.XMLToDataObject(XMLString);
   // gespeicherten Stil wiederherstellen:
   WT_SYSTEM.STYLE = this.style;
}
```

15 Kurzreferenzen

15.1 WTML-Tags

Funktion	Syntax
Kommentar-Tag	<wtRem *comments*> oder: <wtRem> *comments* </wtRem>
Dataform-Tag	<wtDataform [Name="*name*"] [OnSubmit="*OnSubmitHandler*"] [ASYNC_PAGE="*asyncPage*"]> *Bereich* </wtDataform>
Exit-Tag	<wtExit scope={"TEMPLATE"|"DIALOGSTEP"|"SESSION"}>
Include-Tag	<wtInclude Name="*fileName*">
IF-Tag	<wtIf (*Bedingung*)> *Block1* [<wtElse> *Block2*] {<wtEndIf> | </wtIf>}
DO WHILE-Tag	<wtDoWhile (*Bedingung*)> *Block* </wtDoWhile>
DO UNTIL-Tag	<wtDo> *Block* <wtUntil(*Bedingung*)>
OnCreateScript-Tag	<wtOnCreateScript> *CreateScript* </wtOnCreateScript>
OnReceiveScript-Tag	<wtOnReceiveScript> *ReceiveScript* </wtOnReceiveScript>

15.2 WTScript-Anweisungen

Funktion	Syntax
Leere Anweisung	;
Ausdruck als Anweisung	*expression*;
Anweisungs-Block als Anweisung	{*block* ...}
if-Verzweigung	if (*bedingung*) *block1* [else *block2*]
while-Schleife	[*label*:]while (*bedingung*) *block*
do/while-Schleife	[*label*:]do *block* while (*bedingung*);
for-Schleife	[*label*:]for ([*init*]; [*condition*]; [*update*]) *block*
for/in-Schleife	[*label*:]for (*name* in *object*) *block*

switch-Anweisung	switch (*expression*)
	{
	{case *label*: *block1*... [break;] }...
	[default: *block2*...]
	}

break-Anweisung	break [*label*];
continue-Anweisung	continue [*label*];
return-Anweisung	return [*retValue*] ;
Variablen-Deklaration und -Zuweisung	var { *identifier* | *identifier=value* } [{ ,*identifier* | ,*identifier=value* }...] ;

function-Anweisung	function *identifier* ([*parameter* {, *parameter* }...])
	{
	[*block*
	...]
	}

with-Anweisung	with (*object*) *block*

Fachwörter

Fachwörter, die an anderer Stelle erklärt werden, sind mit ->*kursiver* Schrift ausgezeichnet.

aktiver Dialog

Beim aktiven Dialog greift WebTransactions aktiv in die Steuerung des Dialogablaufs ein, d.h., das nächste zu verarbeitende ->*Template* wird von der Template-Programmierung bestimmt. Mit den ->*WTML*-Sprachmitteln können Sie z.B. mehrere ->*Host-Formate* in einer ->*HTML*-Seite zusammenfassen. Dabei wird am Ende eines Host- ->*Dialogschritts* keine Ausgabe an den ->*Browser* geschickt, sondern unmittelbar der Folgeschritt gestartet. Ebenso sind innerhalb **eines** Host-Dialogschritts mehrere Interaktionen zwischen Web- ->*Browser* und WebTransactions möglich.

Array

->*Datentyp*, der eine endliche Menge von Werten eines Datentyps enthalten kann. Der Datentyp kann sein

- ->*skalar*
- eine ->*Klasse*
- ein Array

Die Werte im Array werden durch einen numerischen Index angesprochen, der mit 0 beginnt.

Asynchrone Nachricht

Versteht WebTransactions als Nachricht, die ans Terminal geschickt wird, ohne dass sie vom Anwender ausdrücklich angefordert worden wäre - d.h. ohne dass der Anwender auf irgendeine Taste gedrückt oder auf ein Oberflächenelement geklickt hätte.

Attribut

Definiert eine Eigenschaft eines ->*Objekts*.
Ein Attribut kann z.B. Farbe, Größe oder Position eines Objekts oder selbst wieder ein Objekt sein. Attribute werden auch als ->*Variablen* verstanden und können abgefragt und verändert werden.

Aufrufseite

Eine ->*HTML*-Seite, die Sie benötigen, um eine ->*WebTransactions-Anwendung* zu starten. Auf dieser Seite steht der Aufruf, der WebTransactions mit dem ersten ->*Template* startet, dem Start-Template.

Ausdruck

Kombination aus ->*Literalen*, ->*Variablen*, Operatoren und Ausdrücken, deren Auswertung jeweils ein bestimmtes Ergebnis liefert.

Auswertungsoperator

WebTransactions versteht den Auswertungsoperator als Operator, der die angesprochenen ->*Ausdrücke* durch ihr Ergebnis ersetzt (Objekt-Attribut-Auswertung). Der Auswertungsoperator wird in der Form ##ausdruck# angegeben.

Automask-Template

Ein WebTransactions- ->*Template*, das von WebLab implizit beim Erzeugen eines Basisverzeichnisses oder explizit mit dem Befehl **Automask erzeugen** erstellt wird. Es wird verwendet, wenn kein formatspezifisches Template identifiziert werden kann. Ein Automask-Template enthält die Anweisungen, die für die dynamischen Formatabbildungen und zur Kommunikation notwendig sind. Es können verschiedene Varianten von Automask-Templates erstellt und über das System-Objekt-Attribut AUTOMASK ausgewählt werden.

Basisverzeichnis

Das Basisverzeichnis liegt auf dem WebTransactions-Server und ist die Grundlage einer ->*WebTransactions-Anwendung*. Im Basisverzeichnis liegen die ->*Templates* und alle Dateien oder Verweise auf Programme (Links), die für den Ablauf einer WebTransactions-Anwendung benötigt werden.

BCAM-Applikationsname

Entspricht dem openUTM-Generierungsparameter BCAMAPPL und ist der Name der ->*openUTM-Anwendung*, über den ->*UPIC* die Verbindung aufnehmen kann.

Benutzerkennung

Bezeichner für einen Benutzer. Einer Benutzerkennung können ein ->*Passwort* (zur ->*Zugangskontrolle*) und Zugriffsrechte (->*Zugriffskontrolle*) zugeordnet werden.

Berechtigungsprüfung

siehe ->*Zugangskontrolle*.

Browser

Programm, das zum Abrufen und Darstellen von ->*HTML*-Seiten erforderlich ist. Browser sind z.B. Microsoft Internet Explorer oder Mozilla Firefox.

Browser-Plattform

Betriebssystem des Rechners, auf dem ein ->*Browser* als Client für WebTransactions läuft.

Browserdarstellungs-Druck

Beim Browserdarstellungs-Druck von WebTransactions wird die im ->*Browser* dargestellte Information ausgedruckt.

Capture-Verfahren

Damit WebTransactions in der Ablaufphase die empfangenen ->*Formate* identifizieren kann, können Sie während einer ->*Sitzung* in WebLab für jedes Format einen bestimmten Bereich markieren und das Format benennen. Der Formatname und das ->*Erkennungskriterium* werden in der ->*Capture-Datenbank* gespeichert. Für das Format wird ein ->*Template* unter gleichem Namen generiert. Das Capture-Verfahren ist die Grundlage für die Bearbeitung formatspezifischer Templates für die Liefereinheiten WebTransactions for OSD und MVS.

Capture-Datenbank

Die Capture-Datenbank von WebTransactions enthält alle Formatnamen und die zugehörigen ->*Erkennungskriterien*, die mit dem ->*Capture-Verfahren* erzeugt wurden. Reihenfolge und Erkennungskriterien der Formate können mit WebLab bearbeitet werden.

CGI

(**C**ommon **G**ateway **I**nterface)
Normierte Schnittstelle für den Programmaufruf auf ->*Web-Servern*. Im Gegensatz zur statischen Ausgabe einer zuvor festgelegten ->*HTML*-Seite ermöglicht diese Schnittstelle den dynamischen Aufbau von HTML-Seiten.

Client

Anforderer und Nutzer von Diensten.

Cluster

Menge von identischen ->*WebTransactions-Anwendungen* auf verschiedenen Servern, die zu einem Lastverbund zusammengeschlossen sind.

Dämon

Bezeichnung für einen Prozesstyp in Unix-/POSIX-Systemen, der keine Ein-/Ausgaben auf Terminals durchführt und im Hintergrund abläuft.

Datentyp

Festlegung, wie der Inhalt eines Speicherplatzes zu interpretieren ist. Ein Datentyp hat einen Namen, eine Menge zulässiger Werte (Wertebereich) und eine bestimmte Anzahl von Operationen, die die Werte dieses Datentyps interpretieren und manipulieren.

Dialog

Beschreibt die gesamte Kommunikation zwischen Browser, WebTransactions und *->Host-Anwendung*. Er umfasst in der Regel mehrere *->Dialogzyklen*. Bei WebTransactions werden mehrere Dialogarten unterschieden:

- *->passiver Dialog*
- *->aktiver Dialog*
- *->synchronisierter Dialog*
- *->nicht-synchronisierter Dialog*

Dialogzyklus

Zyklus, der beim Ablauf einer *->WebTransactions-Anwendung* folgende Schritte umfasst:

- eine *->HTML*-Seite aufbauen und an den *->Browser* schicken
- auf Antwort vom Browser warten
- Antwortfelder auswerten und evtl. zur Weiterverarbeitung an die *->Host-Anwendung* schicken

Während des Ablaufs einer *->WebTransactions-Anwendung* werden mehrere Dialogzyklen durchlaufen.

Distinguished Name

Der Distinguished Name (DN) in *->LDAP* setzt sich hierarchisch aus mehreren Teilen zusammen (z.B. „Land, unterhalb von Land: Organisation, unterhalb von Organisation: Organisationseinheit, darunter: Gebräuchlicher Name“). Die Summe dieser Teile identifiziert ein Objekt innerhalb des Directory-Baums eindeutig.

Durch diese Hierarchie wird die eindeutige Benennung von Objekten selbst in einem weltweiten Directory-Baum sehr einfach:

- Der DN "Land=DE/Name=Emil Mustermann" reduziert das Eindeutigkeits-Problem auf das Land DE.
- Der DN "Organisation=FTS/Name=Emil Mustermann" reduziert es auf die Organisation FTS.
- Der DN "Land=DE/Organisation=FTS/Name=Emil Mustermann" reduziert es auf die Organisation FTS innerhalb des Landes DE.

Dokumentenverzeichnis

Verzeichnis des ->*Web-Servers*, in dem Dokumente liegen, auf die über das Netz zugegriffen werden kann. WebTransactions legt in diesem Verzeichnis Dateien zum Herunterladen ab, wie z.B. den WebLab-Client oder allgemeine Start-Seiten.

Domain Name Service (DNS)

Verfahren zur symbolischen Adressierung von Rechnern in Netzen. Bestimmte Rechner im Netz, die DNS- oder Name-Server, führen eine Datenbank mit allen bekannten Rechnernamen und IP-Nummern in ihrer Umgebung.

Dynamische Daten

Werden in WebTransactions durch das WebTransactions-Objektmodell abgebildet, z.B. als ->*Systemobjekt*, Host-Objekt oder Nutzereingaben am Browser.

Eigenschaft

Definiert die Beschaffenheit von ->*Objekten*, z.B. könnten Kundenname und Kundennummer Eigenschaften eines Objekts „Kunde“ sein. Diese Eigenschaften können innerhalb des Programms gesetzt, abgefragt und verändert werden.

EJB

(**E**nterprise **J**ava**B**ean)
Industriestandard auf Basis von Java, mit dem innerhalb einer verteilten, objektorientierten Umgebung selbstentwickelte oder auf dem Markt gekaufte Server-Komponenten zur Erstellung von verteilten Programmsystemen genutzt werden können.

EHLLAPI

(**E**nhanced **H**igh **L**evel **L**angugage **API**)
Programmschnittstelle z.B. von Terminal-Emulationen für die Kommunikation mit der SNA-Welt. Auf dieser Schnittstelle basiert die Kommunikation zwischen Transit-Client und dem SNA-Rechner, die über das Produkt TRANSIT abgewickelt wird.

Erkennungskriterium

Über Erkennungskriterien werden ->*Formate* einer ->*Terminal-Anwendung* identifiziert und Sie können auf die Daten des Formats zugreifen. Als Erkennungskriterium wählen Sie jeweils einen oder auch mehrere Bereiche des Formats, deren Inhalt das Format eindeutig identifiziert.

Felddatei (*.fld-Datei)

Enthält in WebTransactions die Struktur des Datensatzes eines ->*Formats* (Metadaten).

FHS

(**F**ormat **H**andling **S**ystem)
Formatierungssystem für BS2000/OSD-Anwendungen.

Field

Kleinster Teil eines ->*Service* und Element eines ->*Records* oder ->*Puffers*.

Filter

Programm oder Programmteil (z.B. eine Bibliothek) zur Umsetzung eines Formats in ein anderes (z.B. XML-Dokumente in ->*WTScript*-Datenstrukturen).

Format

Optische Darstellung auf alphanumerischen Bildschirmen, wird auch Maske oder Schirm genannt.

In WebTransactions wird ein Format jeweils durch eine ->*Felddatei* und ein Template repräsentiert.

Formatbeschreibungsquellen

Beschreibung mehrerer ->*Formate* in einer oder mehreren Dateien, die aus einer Format-Bibliothek (FHS/IFG) erzeugt wurden oder direkt am ->*Host* vorliegen für die Nutzung „sprechender“ Namen in Formaten.

Formattyp

(nur relevant bei FHS-Anwendungen und Kommunikation über UPIC)
Spezifiziert den Typ des Formats: #Format, +Format, -Format oder *Format.

Funktion

Benutzerdefinierte Code-Teile mit einem Namen und ->*Parametern*. Durch eine Beschreibung der Funktionsschnittstelle (oder Signatur) können Funktionen in Methoden aufgerufen werden.

Holder Task

Prozess, Task oder Thread in WebTransactions, je nach Betriebssystem-Plattform. Die Anzahl der Tasks entspricht der Anzahl der Benutzer. Die Task wird beendet, wenn sich der Benutzer abmeldet oder durch Timeout. Ein Holder Task entspricht genau einer ->*WebTransactions-Sitzung*.

Host

Rechner, auf dem die ->*Host-Anwendung* läuft.

Host-Adapter

Dienen dazu, bestehende *->Host-Anwendungen* an WebTransactions anzuschließen. Sie sorgen zur Laufzeit z.B. für den Auf- und Abbau von Verbindungen und für die Umsetzung der ausgetauschten Daten.

Host-Anwendung

Anwendung, die mit WebTransactions integriert ist.

Host-Plattform

Betriebssystem des Rechners, auf dem die *->Host-Anwendung* läuft.

Host-Daten-Druck

Beim Host-Daten-Druck von WebTransactions werden Informationen ausgedruckt, die von der *->Host-Anwendung* aufbereitet und gesendet wurden, z.B. Ausdruck von Host-Dateien.

Host-Datenobjekt

Bezeichnet in WebTransactions ein *->Objekt* der Datenschnittstelle zur *->Host-Anwendung*, das ein Feld mit all seinen Feldattributen repräsentiert. Es wird von WebTransactions nach dem Empfang von Daten der Host-Anwendung angelegt und existiert bis zum nächsten Datenempfang oder bis zum Beenden der *->Sitzung*.

Host-Steuerobjekt

In WebTransactions enthalten Host-Steuerobjekte Informationen, die nicht nur ein einzelnes Feld betreffen, sondern das gesamte *->Format*. Dazu gehören z.B. das Feld, in dem sich der Cursor befindet, die aktuelle Funktionstaste oder globale Formatattribute.

HTML

(**H**yper**t**ext **M**arkup **L**anguage)
Siehe *->Hypertext Markup Language*

HTTP

(**H**ypertext **T**ransfer **P**rotocol)
Protokoll zur Übertragung von *->HTML*-Seiten und Daten.

HTTPS

(**H**ypertext **T**ransfer **P**rotocol **S**ecure)
Protokoll zur gesicherten Übertragung von *->HTML*-Seiten und Daten.

Hypertext

Dokument mit Verweisen auf andere Stellen im gleichen oder in anderen Dokumenten, in die z.B. durch Anklicken mit der Maus gesprungen werden kann.

Hypertext Markup Language
Standardisierte Auszeichnungssprache für Dokumente im WWW.

JavaBean
Java-Programm (oder *->Klasse*) mit genau festgelegten Konventionen für die Schnittstellen, die eine Wiederverwendung in mehreren Anwendungen ermöglichen.

KDCDEF
openUTM-Werkzeug für die Generierung von *->openUTM-Anwendungen*.

Klasse
Enthält die Definition der *->Eigenschaften* und *->Methoden* eines *->Objekts*. Sie ist das Modell für die Instanziierung von Objekten und definiert deren Schnittstellen.

Klassen-Template
Ein Klassen-Template in WebTransactions enthält für die gesamte Objektklasse (z. B. Eingabe- oder Ausgabefeld) gültige, immer wiederkehrende Anweisungen. Klassen-Templates werden durchlaufen, wenn auf ein *->Host-Datenobjekt* der *->Auswertungsoperator* oder die `toString`-Methode angewendet wird.

Kommunikationsobjekt
Steuert eine Verbindung zu einer *->Host-Anwendung* und enthält Information über den aktuellen Zustand der Verbindung, über die zuletzt empfangenen Daten etc.

Konvertierungswerkzeuge
Dienstprogramme, die mit WebTransactions ausgeliefert werden. Mit den Konvertierungswerkzeugen werden die Datenstrukturen von *->openUTM-Anwendungen* analysiert und in Dateien abgelegt. Diese Dateien können Sie dann in WebLab als *->Formatbeschreibungsquellen* verwenden, um WTML-Templates und *->FLD-Dateien* zu generieren.
Die Basis für die Konvertierung können Cobol-Datenstrukturen oder IFG-Formatbibliotheken sein. Für Drive-Programme wird das Konvertierungswerkzeug mit dem Produkt Drive ausgeliefert.

LDAP

(**L**ightweight **D**irectory **A**ccess **P**rotocol)
Der X.500-Standard definiert als Zugriffsprotokoll DAP (Directory Access Protocol). Speziell für den Zugang zu X.500-Verzeichnisdiensten vom PC aus hat sich jedoch der Internet-Standard LDAP durchgesetzt.
Bei LDAP handelt es sich um ein vereinfachtes DAP-Protokoll, das nicht alle Möglichkeiten von DAP zulässt und mit DAP nicht kompatibel ist. Praktisch alle X.500-Verzeichnisdienste unterstützen neben DAP auch LDAP. In der Praxis kann es zu Verständigungsproblemen kommen, da es diverse Dialekte von LDAP gibt. Die Unterschiede der Dialekte sind in der Regel gering.

Literal

Zeichenfolge, die einen festen Wert repräsentiert. Literale dienen dazu, in Source-Programmen konstante Werte unmittelbar anzugeben („wörtliche“ Wertangabe).

Master-Template

WebTransactions-Template, das als Schablone für die Generierung der Automask und der formatspezifischen-Templates verwendet wird.

Message Queuing

Message Queuing (MQ) ist eine Form der Kommunikation, bei der die Nachrichten (Messages) nicht unmittelbar, sondern über zwischengeschaltete Warteschlangen (Queues) ausgetauscht werden. Sender und Empfänger können zeitlich und räumlich entkoppelt ablaufen, die Übermittlung der Nachricht wird garantiert, unabhängig davon, ob gerade eine Netzverbindung besteht oder nicht.

Methode

Objektorientierter Begriff für ->*Funktion*. Eine Methode wirkt auf das ->*Objekt*, in dem sie definiert ist

Modul-Template

Dient in WebTransactions dazu, ->*Klassen*, ->*Funktionen* und Konstanten global für eine komplette ->*Sitzung* zu definieren. Ein Modul-Template wird mit Hilfe der Funktion `import()` geladen.

MT-Tag

(**M**aster-**T**emplate-Tag)
Spezielle Tags in ->*Master-Templates* für die dynamischen Teile eines Master-Templates.

Multi-Tier-Architektur

Allen Client-/Server-Architekturen liegt eine Gliederung in einzelne Software-Komponenten, auch Schichten oder Tiers genannt, zugrunde: Man spricht von 1-Tier, 2-Tier-, 3-Tier und auch von Multi-Tier-Modellen. Man kann die Aufgliederung auf der physischen oder der logischen Ebene betrachten:

– Logische Software-Tiers liegen vor, wenn die Software in modulare Komponenten mit klaren Schnittstellen gegliedert ist.
– Physische Tiers liegen dann vor, wenn die (logischen) Softwarekomponenten im Netz auf verschiedene Rechner verteilt sind.

Mit WebTransactions sind Multi-Tier-Modelle sowohl auf physischer als auch logischer Tiers-Ebene möglich.

Name/Value-Paar

In den vom ->*Browser* geschickten Daten die Kombination z.B. von einem ->*HTML*-Eingabefeldnamen mit seinem Wert.

nicht-synchronisierter Dialog

Der nicht-synchronisierte Dialog von WebTransactions erlaubt es, den Prüfmechanismus des ->*synchronisierten Dialogs* zeitweise auszuschalten. So lassen sich ->*Dialoge* zwischenschieben, die außerhalb des synchronisierten Dialogs liegen und keinen Einfluss auf den logischen Zustand der ->*Host-Anwendung* haben. Dadurch können Sie z.B. in einer ->*HTML*-Seite eine Schaltfläche anbieten, um Hilfeinformationen aus der laufenden Host-Anwendung anzufordern und in einem separaten Fenster anzuzeigen.

Objekt

Elementare Einheit innerhalb eines objektorientierten Softwaresystems. Jedes Objekt hat einen Namen, über den es angesprochen werden kann, ->*Attribute*, die seinen Zustand definieren und ->*Methoden*, die auf das Objekt angewandt werden können.

openUTM

(**U**niversal **T**ransaction **M**onitor)
Transaktionsmonitor von Fujitsu Technlogy Solutions, verfügbar für BS2000/OSD, verschiedenste Unix- und Windows-Plattformen.

openUTM-Anwendung

->*Host-Anwendung*, die Dienstleistungen zur Verfügung stellt, die Aufträge von Terminals, ->*Client-Programmen* oder anderen Host-Anwendungen bearbeiten. openUTM übernimmt dabei u.a. die Transaktionssicherung und das Management der Kommunikations- und Systemressourcen. Technisch gesehen ist eine openUTM-Anwendung eine Prozessgruppe, die zur Laufzeit eine logische Einheit bildet.
Mit openUTM-Anwendungen kann sowohl über das Client/Server-Protokoll ->*UPIC* als auch über die Terminal-Schnittstelle (9750) kommuniziert werden.

openUTM-Client (UPIC)

Mit dem Produkt openUTM-Client (UPIC) können Sie Client-Programme für openUTM erstellen. openUTM-Client (UPIC) steht z.B. für Unix-, BS2000/OSD- und Windows-Plattformen zur Verfügung.

openUTM-Teilprogramm

Die Dienste einer ->*openUTM-Anwendung* werden durch ein oder mehrere openUTM-Teilprogramme realisiert. Sie sind über ->*Transaktionscodes* ansprechbar und enthalten spezielle openUTM-Funktionsaufrufe (z.B. KDCS-Aufrufe).

Parameter

Daten, die an eine ->*Funktion* oder ->*Methode* zur Verarbeitung übergeben werden (Eingabeparameter) oder Daten, die als Ergebnis von einer Funktion oder Methode zurückgeliefert werden (Ausgabeparameter).

passiver Dialog

Beim passiven Dialog von WebTransactions wird der Dialogablauf von der ->*Host-Anwendung* gesteuert, d.h., die Host-Anwendung bestimmt das nächste zu verarbeitende ->*Template*. Ein Anwender, der über WebTransactions auf die Host-Anwendung zugreift, durchläuft die gleichen Schritte wie beim Zugriff über ein Terminal. Passive Dialogsteuerung verwendet WebTransactions bei einer automatischen Umsetzung der Host-Anwendung oder wenn jedes Format der Host-Anwendung genau einem individuellen Template entspricht.

Passwort

In einer Anwendung für eine ->*Benutzerkennung* eingetragene Zeichenkette zur Authentisierung (->*Zugangsschutz*).

polling

Zyklische Abfrage auf Zustandsänderungen.

Pool

WebTransactions bezeichnet hiermit ein freigegebenes Verzeichnis, in dem WebLab ->*Basisverzeichnisse* anlegen und pflegen kann. Den Zugriff auf dieses Verzeichnis steuern Sie mit der Administration.

Posted-Objekt (`wt_Posted`**)**

Enthält in WebTransactions eine Liste der vom ->*Browser* zurückgeschickten Daten. Dieses ->*Objekt* wird von WebTransactions angelegt und lebt nur für die Dauer eines ->*Dialogzyklus*.

posten

Daten versenden

Projekt

Enthält in der WebTransactions-Entwicklungsumgebung verschiedene Einstellungen einer ->*WebTransactions-Anwendung*, die in einer Projektdatei (Endung `.wtp`) gespeichert werden. Sie sollten für jede WebTransactions-Anwendung, die Sie entwickeln, ein Projekt anlegen und zum Bearbeiten immer dieses Projekt öffnen.

Protokoll

Vereinbarungen über Verhaltensregeln und Formate bei der Kommunikation unter entfernten Partnern gleichen logischen Niveaus.

Protokolldatei

- openUTM-Client: Datei, in die bei abnormalem Beenden einer Conversation openUTM-Fehlermeldungen geschrieben werden.
- In WebTransactions werden Protokolldateien als Trace-Dateien bezeichnet.

Prozess

Der Begriff „Prozess“ wird als Oberbegriff für Prozess (Solaris, Linux und Windows) und Task (BS2000/OSD) verwendet.

Puffer

Definition eines Datensatzes, der von einem ->*Service* übertragen wird. Der Puffer dient zum Senden und zum Empfangen von Nachrichten. Zusätzlich gibt es einen speziellen Puffer für die Ablage der ->*Erkennungskriterien* und für Daten zur Darstellung am Bildschirm.

Roaming Session

->*WebTransactions-Sitzung*, die nacheinander oder gleichzeitig von verschiedenen ->*Clients* aus angesprochen werden kann.

Record

Definition eines Datensatzes, der in einem ->*Puffer* übertragen wird. Er beschreibt einen Teil des Puffers, der ein- oder mehrfach vorkommen kann.

Service-Anwendung

->*WebTransactions-Sitzung*, die abwechselnd von verschiedenen Benutzern aufgerufen werden kann.

Service-Knoten

Instanz eines ->*Service*. Beim Entwickeln und beim Ablauf einer ->*Methode* kann ein Service mehrfach instanziiert werden. Beim Modellieren und Code bearbeiten werden diese Instanzen als Service-Knoten bezeichnet.

Sichtbarkeit von Variablen

->*Objekte* und ->*Variablen* unterschiedlicher Dialogarten werden von WebTransactions in unterschiedlichen Adressräumen verwaltet. Das bedeutet, dass Variablen eines ->*synchronen Dialogs* im ->*asynchronen Dialog* oder im Dialog mit einer entfernten Anwendung nicht sichtbar und damit auch nicht zugreifbar sind.

Sitzung

Beginnt ein Endanwender mit einer ->*WebTransactions-Anwendung* zu arbeiten, so wird für ihn auf dem WebTransactions-Server eine WebTransactions-Sitzung eingerichtet. Diese Sitzung enthält alle für diesen Benutzer geöffneten Verbindungen zum ->*Browser*, zu speziellen ->*Clients* und ->*Hosts*.
Eine Sitzung kann gestartet werden

- durch Eingabe eines URL von WebTransactions im Browser.
- durch die Methode `START_SESSION` der Client/Server-Schnittstelle `WT_REMOTE`.

Eine Sitzung endet

- mit einer entsprechenden Eingabe des Benutzers im Ausgabebereich dieser ->*WebTransactions-Anwendung* (nicht über Standard-Buttons des Browsers).
- durch Überschreiten der konfigurierten Zeit, die WebTransactions auf eine Anwort von der ->*Host-Anwendung* oder vom ->*Browser* wartet.
- durch Terminierung mit HIlfe der WebTransactions-Administration.
- durch die Methode `EXIT_SESSION` der Client/Server-Schnittstelle `WT_REMOTE`.

Eine WebTransactions-Sitzung ist eindeutig durch eine ->*WebTransactions-Anwendung* und eine Session Id bestimmt. Während ihrer Lebensdauer existiert zu jeder WebTransactions-Sitzung auf dem WebTransactions-Server genau ein ->*Holder Task*.

Skalar

->*Variable*, die nur aus einem einzelnen Wert besteht - im Gegensatz zu einer ->*Klasse*, einem ->*Array* oder einer anderen komplexen Datenstruktur.

SOAP

(ursprünglich **S**imple **O**bject **A**ccess **P**rotocol)
Das ->*XML*-basierte SOAP-Protocol realisiert einen einfachen und transparenten Mechanismus, mit dem strukturierte und typisierte Informationen zwischen Rechnern in einer dezentralisierten, verteilten Umgebung ausgetauscht werden können.
SOAP stellt ein modulares Paketmodell sowie Mechanismen zum Verschlüsseln von Daten innerhalb von Modulen zur Verfügung. Dies ermöglicht die unkomplizierte Beschreibung der externen Schnittstellen eines ->*Web-Service*.

Stil

Realisiert in WebTransactions ein anderes Layout für ein ->*Template*, z.B. mit mehr oder weniger Grafikelementen für unterschiedliche ->*Browser*. Der Stil kann während einer ->*Sitzung* jederzeit geändert werden.

synchronisierter Dialog

Beim synchronisierten Dialog (Standardfall) überprüft WebTransactions automatisch, ob die Daten, die vom Web-Browser eingehen, auch wirklich die Antwort auf die letzte an den ->*Browser* geschickte ->*HTML*-Seite sind. Wenn z.B. der Anwender am Web-Browser über die Schaltfläche **Zurück** oder die History-Funktion zu einer „alten“ HTML-Seite der aktuellen ->*Sitzung* wechselt und diese zurückschickt, erkennt WebTransactions, dass die Daten nicht zum aktuellen ->*Dialogzyklus* passen und reagiert mit einer Fehlermeldung. Die zuletzt an den Browser gesendete Seite wird automatisch erneut an den Browser geschickt.

Systemobjekt (`wt_System`**)**

Das Systemobjekt von WebTransactions enthält ->*Variablen*, die während einer gesamten ->*Sitzung* existieren und erst am Ende einer Sitzung oder durch explizites Löschen wieder entfernt werden. Es ist immer sichtbar und identisch für alle Namensräume.

TAC

Siehe ->*Transaktionscode*

Tag

->*HTML*-, ->*XML*- und ->*WTML*-Dokumente bestehen aus Tags und dem eigentlichen Inhalt. Mit den Tags werden Auszeichnungen im Dokument durchgeführt z.B. Überschriften, Texthervorhebungen (fett, kursiv) oder Quellangaben für Grafikdateien.

TCP/IP

(**T**ransport **C**ontrol **P**rotocol/**I**nternet **P**rotocol)
Sammelname für eine Protokollfamilie in Rechnernetzen, die unter anderem im Internet verwendet wird.

Template

Vorlage für die Generierung von spezifischem Code. Ein Template enthält feste Teile, die bei der Generierung unverändert übernommen werden und variable Teile, die bei der Generierung durch die jeweils aktuellen Werte ersetzt werden. Ein Template ist eine *->WTML*-Datei mit speziellen Tags zur Steuerung der dynamischen Generierung einer *->HTML*-Seite und zur Verarbeitung der am *->Browser* eingegeben Werte. Es können mehrere Sätze von Templates parallel gehalten werden. Diese repräsentieren unterschiedliche Stile (z.B. viel/wenig Grafik, Java-Benutzung etc.).
WebTransactions nutzt verschiedene Arten von Templates:

- ->*Automask-Templates* für die automatische Umsetzung der ->*Formate* von MVS- und OSD-Anwendungen
- eigene Templates, die vom Programmierer selbst geschrieben werden, z.B. zur Steuerung eines *->aktiven Dialogs*
- formatspezifische Templates, die für eine spätere Nachbearbeitung generiert werden
- Include-Templates, die in andere Templates eingefügt werden
- *->Klassen-Templates*
- ->*Master-Templates* für ein einheitliches Layout fester Bereiche bei der Generierung der Automask und formatspezifischer Templates
- Start-Template, das als erstes Template einer WebTransactions-Anwendung durchlaufen wird

Template-Objekte

->Variablen zur Zwischenspeicherung von Werten für einen *->Dialogzyklus* in WebTransactions.

Terminal-Anwendung

Anwendung auf einem ->*Host*-Rechner, auf die über die 9750- oder 3270-Schnittstelle zugegriffen wird.

Terminal-Hardcopy-Druck

Beim Terminal-Hardcopy-Druck von WebTransactions wird die alphanumerische Darstellung des ->*Formats* gedruckt, wie es von einem Terminal oder einer Terminal-Emulation dargestellt würde.

Transaktion

Verarbeitungsschritt zwischen zwei Sicherungspunkten (innerhalb eines Vorgangs), der durch die ACID-Bedingungen gekennzeichnet ist (**A**tomicity, **C**onsistency, **I**solation und **D**urability). Die in einer Transaktion beabsichtigten Änderungen an der Anwenderinformation werden entweder alle oder gar nicht durchgeführt (Alles-oder-Nichts-Regel).

Transaktionscode/TAC

Name, über den ein openUTM-Vorgang oder ein *->openUTM-Teilprogramm* aufgerufen werden kann. Der Transaktionscode wird dem openUTM-Teilprogramm bei der openUTM-Konfigurierung zugeordnet. Einem Teilprogramm können auch mehrere TACs zugeordnet sein.

UDDI

(**U**niversal **D**escription, **D**iscovery and **I**ntegration)
Umfasst Verzeichnisse, die Beschreibungen von *->Web-Services* enthalten. Diese Informationen stehen Web-Usern allgemein zur Verfügung.

Unicode

Von der International Standardisation Organisation (ISO) und dem Unicode-Konsortium genormter alphanumerischer Zeichensatz zur Codierung von Zeichen – Buchstaben, Ziffern, Satzzeichen, Silbenzeichen, Sonderzeichen sowie Ideogrammen. Unicode fasst alle weltweit bekannten Textzeichen in einem einzigen Zeichensatz zusammen.
Unicode ist hersteller- und systemunabhängig. Es verwendet Zeichensätze der Länge zwei oder vier Bytes für die Codierung jedes Textzeichens. Diese Zeichensätze werden bei ISO als UCS-2 (Universal Character Set 2) beziehungsweise UCS-4 bezeichnet. Statt der durch ISO definierten Bezeichnung UCS-2 wird häufig die Bezeichnung UTF-16 (Unicode Transformation Format 16 Bit) verwendet, ein vom Unicode-Konsortium definierter Standard.
Neben der Nutzung von UTF-16 ist auch der Einsatz von UTF-8 (Unicode Transformation Format 8 Bit) weit verbreitet. UTF-8 ist inzwischen die globale Zeichen-Codierung im Internet.

UPIC

(**U**niversal **P**rogramming **I**nterface for **C**ommunication)
Trägersystem für openUTM-Clients, das über die X/Open-Schnittstelle CPI-C die Client-Server-Kommunikation zwischen CPI-C-Client-Anwendung und der openUTM-Anwendung ermöglicht.

URI

(**U**niform **R**esource **I**dentifier)
Oberbegriff für alle Namen und Adressen die im Internet Objekte referenzieren. Die allgemein gebräuchlichen URIs sind *->URLs*.

URL

(**U**niform **R**esource **L**ocator)
Beschreibung von Ort und Zugriffsart einer Ressource im Internet.

Userexit

In C/C++ implementierte Funktion, die der Programmierer aus einem *->Template* aufruft.

Variable

Speicherplatz für variable Werte, der einen Namen und einen ->*Datentyp* benötigt.

Vorgang

In ->*openUTM* Bearbeitung eines Auftrags durch eine *->openUTM-Anwendung*. Es gibt Dialog-Vorgänge und Asynchronvorgänge. Dem Vorgang werden von openUTM eigene Speicherbereiche zugeordnet. Ein Vorgang setzt sich aus einer oder mehreren ->*Transaktionen* zusammen.

Web-Server

Rechner und Software zum Bereitstellen von *->HTML*-Seiten und dynamischen Daten über *->HTTP.*

Web-Service

Dienst, der im Internet bereitgestellt wird, z.B. ein Währungsumrechnungs-Programm, und über das SOAP-Protokoll angesprochen werden kann. Die Schnittstelle eines Web-Service ist in ->*WSDL* beschrieben.

WebTransactions-Anwendung

Anwendung, die *->Host-Anwendungen* für den Internet-/Intranet-Zugriff integriert. Eine WebTransactions-Anwendung besteht aus

- einem *->Basisverzeichnis*
- einem Start-Template
- den *->Templates*, die die Umsetzung zwischen ->*Host* und ->*Browser* steuern
- protokollspezifischen Konfigurationsdateien

WebTransactions-Plattform

Betriebssystem des Rechners, auf dem WebTransactions läuft.

WebTransactions-Server

Rechner, auf dem WebTransactions läuft.

WebTransactions-Sitzung

Siehe *->Sitzung*.

WSDL

(**W**eb **S**ervices **D**escription **L**anguage)
Bietet ->*XML*-Sprachregeln für die Beschreibung von ->*Web-Services*. Ein Web-Service wird dabei durch eine Auswahl von Ports definiert.

WTBean

In WebTransactions werden ->*WTML*-Komponenten mit selbstbeschreibender Schnittstelle als WTBeans bezeichnet. Es wird zwischen inline und standalone WTBeans unterschieden:

– ein inline WTBean entspricht einem Teil eines WTML-Dokuments

– ein standalone WTBean ist ein eigenständiges WTML-Dokument

Verschiedene WTBeans gehören zum Produktumfang von WebTransactions, weitere WTBeans stehen Ihnen auf der WebTransactions-Homepage zum Download zur Verfügung:
ts.fujitsu.com/products/software/openseas/webtransactions.html

WTML

(**W**eb**T**ransactions **M**arkup **L**anguage)
Auszeichnungs- und Programmiersprache für WebTransactions ->*Templates*. WTML besteht aus ->*WTML-Tags*, die ->*HTML* erweitern, und der server-seitigen Programmiersprache ->*WTScript*, die z.B. den Datenaustauch mit ->*Host-Anwendungen* ermöglicht. WTML wird von WebTransactions und nicht vom ->*Browser* ausgeführt (serverside scripting).

WTML-Tag

(**W**eb**T**ransactions **M**arkup **L**anguage-Tag)
Spezielle Tags von WebTransactions zur Generierung der dynamischen Teile einer ->*HTML*-Seite mit Daten aus der ->*Host-Anwendung*.

WTScript

Server-seitige Programmiersprache von WebTransactions. WTScripts stehen ähnlich wie client-seitige JavaScripts in Bereichen, die mit speziellen Tags eingeleitet und beendet werden. Statt ->*HTML*-`SCRIPT`-Tags verwenden Sie hierfür jedoch ->*WTML-Tags*: `wtOnCreateScript` und `wtOnReceiveScript`. Damit zeigen Sie an, dass diese Scripts von WebTransactions und nicht vom ->*Browser* ausgeführt werden sollen und signalisieren zusätzlich den gewünschten Ausführungszeitpunkt. OnCreate-Scripts werden ausgeführt, bevor die Seite an den Browser geschickt wird. OnReceive-Scripts werden erst ausgeführt, nachdem die Antwort vom Browser empfangen wurde.

XML

(e**X**tensible **M**arkup **L**anguage)
Definiert eine Sprache zur logischen Strukturierung von Dokumenten mit dem Ziel, diese einfach zwischen verschiedenen Anwendungen auszutauschen.

XML-Schema

Ein XML-Schema im allgemeinen Sinn definiert die zulässigen Elemente und Attribute einer XML-Beschreibung. XML-Schemata können verschiedene Formate haben, z.B. DTD (**D**ocument **T**ype **D**efinition), XML Schema (W3C-Standard) oder XDR (**X**ML **D**ata **R**educed).

Zugangskontrolle

Prüfung, ob ein Benutzer berechtigt ist, unter einer bestimmten Benutzerkennung mit der Anwendung zu arbeiten.

Zugriffskontrolle

Überwachung der Zugriffe auf die Daten und ->*Objekte* einer Anwendung.

Abkürzungen

BO	**B**usiness **O**bject
CGI	**C**ommon **G**ateway **I**nterface
DN	**D**istinguished **N**ame
DNS	**D**omain **N**ame **S**ervice
EJB	**E**nterprise **J**ava**B**ean
FHS	**F**ormat **H**andling **S**ystem
HTML	**H**yper**t**ext **M**arkup **L**anguage
HTTP	**H**yper**t**ext **T**ransfer **P**rotocol
HTTPS	**H**yper**t**ext **T**ransfer **P**rotocol **S**ecure
IFG	**I**nteraktiver **F**ormat **G**enerator
ISAPI	**I**nternet **S**erver **A**pplication **P**rogramming **I**nterface
LDAP	**L**ightweight **D**irectory **A**ccess **P**rotocol
LPD	**L**ine **P**rinter **D**aemon
MT-Tag	**M**aster-**T**emplate-Tag
MVS	**M**ultiple **V**irtual **S**torage
OSD	**O**pen **S**ystems **D**irection
SGML	**S**tandard **G**eneralized **M**arkup **L**anguage
SOAP	**S**imple **O**bject **A**ccess **P**rotocol

SSL	**S**ecure **S**ocket **L**ayer
TCP/IP	**T**ransport **C**ontrol **P**rotocol/**I**nternet **P**rotocol
Upic	**U**niversal **P**rogramming **I**nterface for **C**ommunication
URL	**U**niform **R**esource **L**ocator
WSDL	**W**eb **S**ervices **D**escription **L**anguage
wtc	**W**eb**T**ransactions **C**omponent
WTML	**W**eb**T**ransactions **M**arkup **L**anguage
XML	e**X**tensible **M**arkup **L**anguage

Literatur

Unter der Web-Adresse *http://manuals.ts.fujitsu.com* stehen Ihnen sämtliche Handbücher zum Download zur Verfügung.

WebTransactions
Konzepte und Funktionen
Einführung

WebTransactions
Client-APIs für WebTransactions
Benutzerhandbuch

WebTransactions
Anschluss an openUTM-Anwendungen über UPIC
Benutzerhandbuch

WebTransactions
Anschluss an OSD-Anwendungen
Benutzerhandbuch

WebTransactions
Anschluss an MVS-Anwendungen
Benutzerhandbuch

WebTransactions
Zugriff auf dynamische Web-Inhalte
Benutzerhandbuch

WebTransactions
Web-Frontend für Web-Services
Benutzerhandbuch

Sonstige Literatur

Die Handbücher sind online unter *http://manuals.ts.fujitsu.com* zu finden oder in gedruckter Form gegen gesondertes Entgelt unter *http://manualshop.ts.fujitsu.com* zu bestellen.

***inter*Net Services**
Administratorhandbuch

Stichwörter

T

W