Benutzerhandbuch - Deutsch

FUJITSU

WebTransactions V7.5

Client-APIs für WebTransactions

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 Architektur des Client-Zugriffs in WebTransactions

Folgende Abbildung zeigt die Architektur des Client-Zugriffs in WebTransactions. Die farbig hinterlegten Bereiche sind dabei Gegenstand dieses Handbuchs:

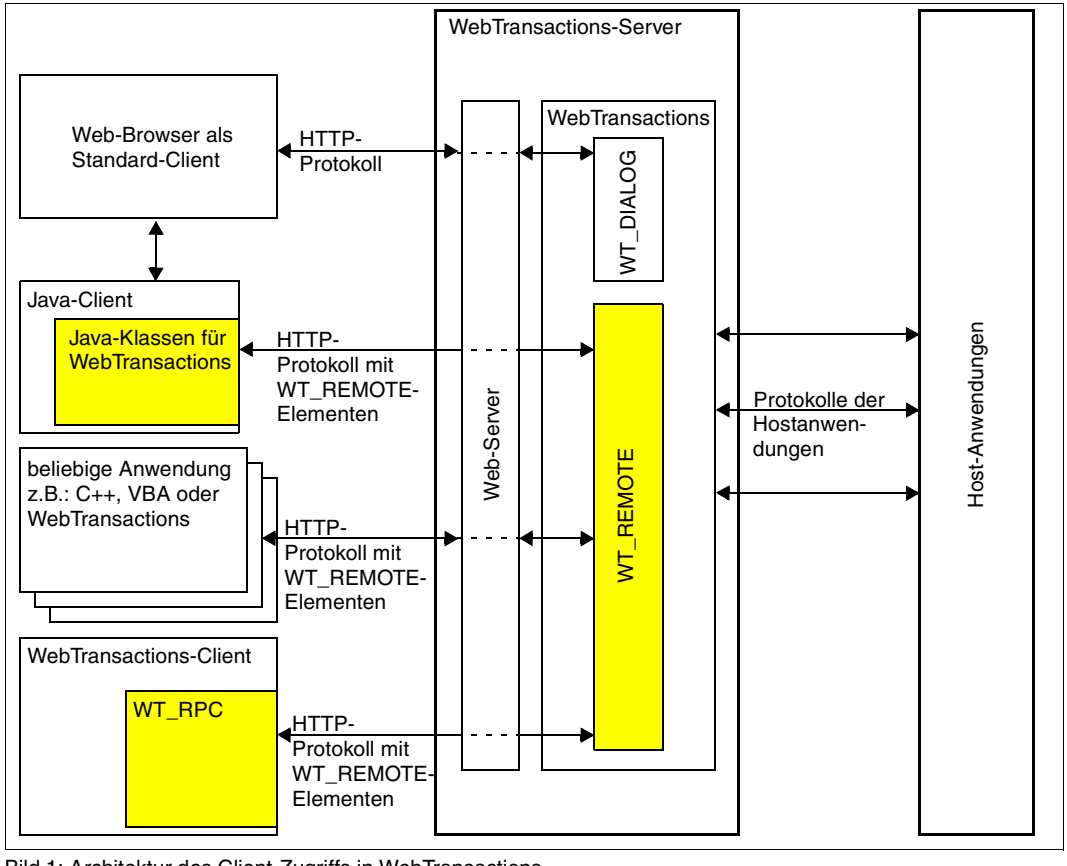

Bild 1: Architektur des Client-Zugriffs in WebTransactions

Web-Browser

Der Web-Browser diente bereits bisher als Standard-Client. Über den Web-Browser können Benutzer die bisher nur über Terminal-Emulationen zur Verfügung stehenden Host-Anwendungen über eine Web-Oberfläche bedienen. Der Client-Zugriff erfolgt damit ausschließlich auf die Elemente der von der WebTransactions-Anwendung erzeugten Webseiten.

WebTransactions-Server

Der WebTransactions-Server ist der Rechner, der die WebTransactions-Anwendung und den Web-Server für den Client-Zugriff auf diese Anwendung zur Verfügung stellt.

Web-Server

Alle Clients greifen grundsätzlich mit Hilfe des HTTP-Protokolls über den Web-Server auf die WebTransactions-Anwendung zu. Ist der Client ein Web-Browser, so werden die Anfragen an `WT_DIALOG` weitergeleitet, das die Steuerung der Endbenutzer-Sitzung übernimmt.

Enthalten die HTTP-Nachrichten `WT_REMOTE`-Elemente, so werden die Anfragen an die `WT_REMOTE`-Schnittstelle von WebTransactions weitergeleitet, wo sie abgearbeitet werden.

WT_REMOTE

`WT_REMOTE` ist eine offene Schnittstelle von WebTransactions für beliebige Clients. Damit wird es möglich, von beliebigen Programmen aus auf die Ressourcen (Objekte und Methoden) von WebTransactions-Anwendungen zuzugreifen und so deren Funktionalität in anderen Anwendungen nutzbar zu machen. Die einzige Voraussetzung dafür ist, dass der Client in der Lage ist, mehrteilige HTTP-Nachrichten (so genannte Multipart-Nachrichten) zu versenden.

WebTransactions-Client

Der WebTransactions-Client ist ein Rechner, auf dem eine WebTransactions-Anwendung läuft, die auf eine andere WebTransactions-Anwendung zugreift.

WT_RPC

Mit `WT_RPC` bietet WebTransactions auch eine eigene Programmier-Schnittstelle für verteilte WebTransactions-Anwendungen an. Dabei wird die Kommunikation mit dem Web-Server und damit die Verwendung der Schnittstelle `WT_REMOTE` intern von `WT_RPC` durchgeführt und damit vor dem Programmierer verborgen. Dadurch können WebTransactions-Anwendungen ohne großen Programmieraufwand auf andere WebTransactions-Anwendungen zugreifen.

Java-Client

Mit den `WTJavaClient`-Klassen können Sie Java-Applets und Java-Programme für den Zugriff auf WebTransactions schreiben. Die Methoden der Java-Klassen bilden dabei die Aufrufe der `WT_REMOTE`-Schnittstelle nach.

1.3 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.

- Ein Referenz-Handbuch, das für alle Liefereinheiten gilt und die WebTransactions Template-Sprache WTML beschreibt:

  Template-Sprache

  Nach einem Überblick über WTML finden Sie

  – die lexikalischen Elemente, die in WTML verwendet werden.

  – die klassenunabhängigen globalen Funktionen, wie z.B. `escape()` oder `eval()`.

  – die eingebauten Klassen und Methoden, wie z.B. die Klassen `Array` oder `Boolean`.

  – die WTML-Tags, die die WebTransactions-spezifischen Funktionen enthalten.

  – die WTScript-Anweisungen, die Sie in den WTScript-Bereichen angeben können.

  – die Klassen-Templates, mit denen Sie die Auswertung gleichartiger Objekte automatisieren können.

  – die Master-Templates, die von WebTransactions als Schablone verwendet werden und für ein einheitliches Layout sorgen.

  – eine Beschreibung der Java-Integration, mit der Sie eigene Java-Klassen in WebTransactions instanziieren und der Userexits, mit denen Sie eigene C/C++-Funktionen integrieren können.

  – die mit WebTransactions fertig ausgelieferten UserExits.

  - die XML-Konvertierung für die portable Darstellung von Daten für die Kommunikation mit externen Anwendungen über XML-Nachrichten und die Konvertierung von WTScript-Datenstrukturen in XML-Dokumente.

- 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 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.

1.4 Konzept und Zielgruppe dieses Handbuchs

Dieses Handbuch wendet sich an alle, die Clients für WebTransactions-Anwendungen erstellen oder Funktionsbausteine von WebTransactions-Anwendungen auf mehrere Server verteilen wollen.

Die einzelnen Kapitel beschreiben die hierfür notwendigen Protokolle und Schnittstellen.

Das Handbuch ergänzt das einführende WebTransactions-Handbuch „Konzepte und Funktionen“ und das WebTransactions-Referenzhandbuch „Template-Sprache“ um alle Client-spezifischen Informationen.

1.5 Neue Funktionen

Einen Überblick über alle Neuerungen in WebTransactions V7.5 finden Sie im WebTransactions-Handbuch „Konzepte und Funktionen“.

1.6 Darstellungsmittel

Diese Dokumentation verwendet die folgenden Darstellungsmittel:

Auszeichnung	Bedeutung
`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 selbst dürfen Sie nicht angeben
...	optionale ein oder mehrmalige Wiederholung des vorhergehenden Elements
	wichtige Hinweise und weiterführende Informationen
CR LF	Zeilenvorschub-Darstellung für HTTP-Beispiele
►	Aufforderungszeichen, wenn Sie etwas tun sollen.
	verweist auf weiterführende Informationen

2 Client-Konzept von WebTransactions

Das hier beschriebene Client-Konzept erlaubt den flexiblen Einsatz von WebTransactions als Datenlieferant für die verschiedensten Clients.

2.1 Der Standard-Client Web-Browser

Das Konzept des Client-Server-Zugriffs auf WebTransactions unterscheidet sich grundsätzlich von den üblichen dialogorientierten WebTransactions-Anwendungen mit einem Web-Browser als Standard-Client. Um den Unterschied zu verdeutlichen, wird im Folgenden zuerst noch einmal der Standardfall gezeigt, wie also WebTransactions Host-Anwendungen auf Dialog-Anwendungen im WWW umsetzt. Über einen Web-Browser, der auf den meisten Plattformen zur Verfügung steht, kann über das HTTP-Protokoll auf Dialog-Anwendungen zugegriffen werden, die bis dahin nur über Terminal-Emulationen zur Verfügung standen.

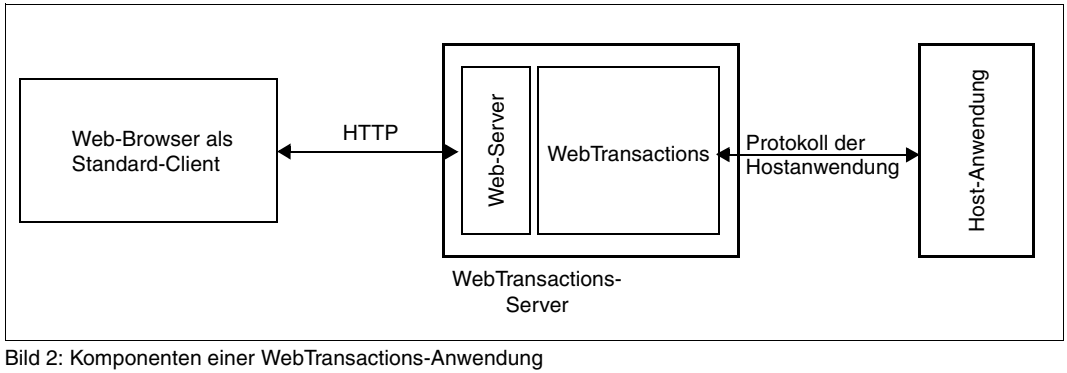

Bild 2: Komponenten einer WebTransactions-Anwendung

Der Browser sendet seine Anfragen über den Web-Server an WebTransactions. Der Browser kann die vom WebTransactions-Server erzeugten HTML-Seiten darstellen und der Benutzer am Browser kann entsprechend den Vorgaben in den Templates über die WebTransactions-Anwendung mit der Host-Anwendung kommunizieren. Einen direkten Einfluss auf den Ablauf der WebTransactions-Anwendung hat der Benutzer nur soweit, wie dies von der WebTransactions-Anwendung vorgesehen ist (z.B. ein Button zum Beenden der Sitzung).

2.2 Die Schnittstelle WT_REMOTE

Mit Hilfe der offenen Client-Schnittstelle `WT_REMOTE` können beliebige Clients unmittelbar auf Ressourcen einer WebTransactions-Anwendung zugreifen. Spezielle Einschalungen dieser Schnittstelle (`WT_RPC` und die `WTJavaClient`-Klassen) repräsentieren die WebTransactions-Ressourcen durch Stellvertreter und ermöglichen so einen Zugriff, als ob diese Teil der Client-Anwendung wären.

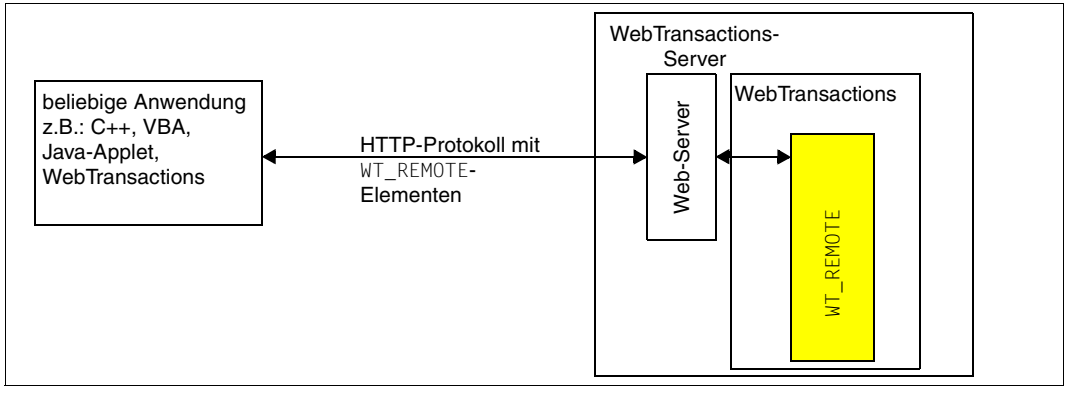

Bild 3: Die `WT_REMOTE`-Schnittstelle für den Client-Zugriff

Eine beliebige Anwendung sendet hierbei Anfragen in Form mehrteiliger HTTP-Nachrichten in einem speziellen Format an den Web-Server. Dieser leitet die Nachrichten an die `WT_REMOTE`-Schnittstelle von WebTransactions weiter, wo sie interpretiert und abgearbeitet werden.

Der wesentliche Unterschied zum Standard-Client ist der, dass nicht nur ein Zugriff auf die Webseiten der WebTransactions-Anwendung möglich ist, sondern auch ein direkter Zugriff auf die ferne WebTransactions-Anwendung nach Art eines Remote Procedure Call. Dabei sind die folgenden Zugriffe für die Client-Anwendungen möglich:

- Starten einer WebTransactions-Sitzung
- Ausführen eines Kommandos in der WebTransactions-Sitzung:
  - Daten zur WebTransactions-Sitzung senden
  - Daten von der WebTransactions-Sitzung empfangen
  - WebTransactions-Objekte erzeugen
  - WebTransactions-Methoden aufrufen
- Beenden einer WebTransactions-Sitzung

Damit wird es für Clients möglich, aktiv auf eine WebTransactions-Anwendung Einfluss zu nehmen und diese von außen zu steuern, bzw. die Funktionalität einer WebTransactions-Anwendung für eigene Zwecke zu verwenden.

Für die wichtigsten Anwendungsfälle (Java-Applets und WebTransactions selbst als Client) existieren Standardbibliotheken, die in den beiden folgenden Kapiteln beschrieben sind. Für alle übrigen Clients finden Sie eine detaillierte Beschreibung der Schnittstelle WT_REMOTE im Kapitel „Anhang: Die Schnittstelle WT_REMOTE“ auf Seite 63.

2.3 Klassenbibliothek WT_RPC für WebTransactions-Clients

Mit dem Client/Server-Protokoll WT_REMOTE kann jede Anwendung, die HTTP-Multipart-Nachrichten versenden kann, Zugriff auf eine WebTransactions-Anwendung erhalten. Für WebTransactions-Anwendungen als Clients steht bereits eine komfortable Schnittstelle in Form einer Klassenbibliothek mit dem Namen WT_RPC zur Verfügung, die eine unkomplizierte Kommunikation über dieses Protokoll ermöglicht.

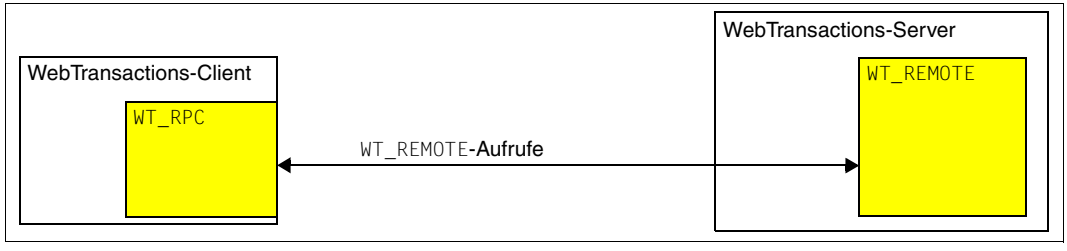

Bild 4: Die WT_RPC-Klasse von WebTransactions

Die Klassenbibliothek WT_RPC stellt eine Reihe von Methoden für die Kommunikation mit WebTransactions-Serveranwendungen über die WT_REMOTE-Schnittstelle zur Verfügung, ohne dass die technischen Details zur HTTP-Kommunikation berücksichtigt werden müssen. Die Kommunikation erfolgt zwar nach wie vor über den Web-Server, dies ist für den Programmierer auf der Client-Seite aber nicht sichtbar. WT_RPC stellt damit eine High-Level-Schnittstelle dar.

Diese Schnittstelle bietet die folgende Funktionalität:

- Starten und Beenden einer fernen WebTransactions-Anwendung
- Aufrufen ferner Methoden
- lokale Definition ferner Methoden, um diese wie lokale Methoden verwenden zu können

Diese Funktionalität bietet alles Nötige, um verteilte WebTransactions-Anwendungen auf einfache Weise zu realisieren.

2.4 Klassenbibliothek für Java-Clients

Mit der Klassenbibliothek `WTJavaClient.jar` können Sie Applets und Java-Programme schreiben, die mit den Methoden der vordefinierten Klassen auf Daten einer WebTransactions-Anwendung zugreifen. Die Klassen von `WTJavaClient.jar` basieren auf dem JDK V1.1. Die entsprechenden Methoden nutzen intern die Schnittstelle `WT_REMOTE`, verbergen diese aber hinter einer wesentlich einfacher zu nutzenden Objektschnittstelle.

Ein Java-Client für WebTransactions besteht aus verschiedenen Klassen:

– aus selbst- und vordefinierten Klassen auf Basis der JDK V1.1, die die grafische Darstellung des Programms am Bildschirm und die Anwendungslogik enthalten

– aus `WTJavaClient`-Klassen, die für die Verbindung zu einer WebTransactions-Sitzung und den Datenaustausch sorgen.

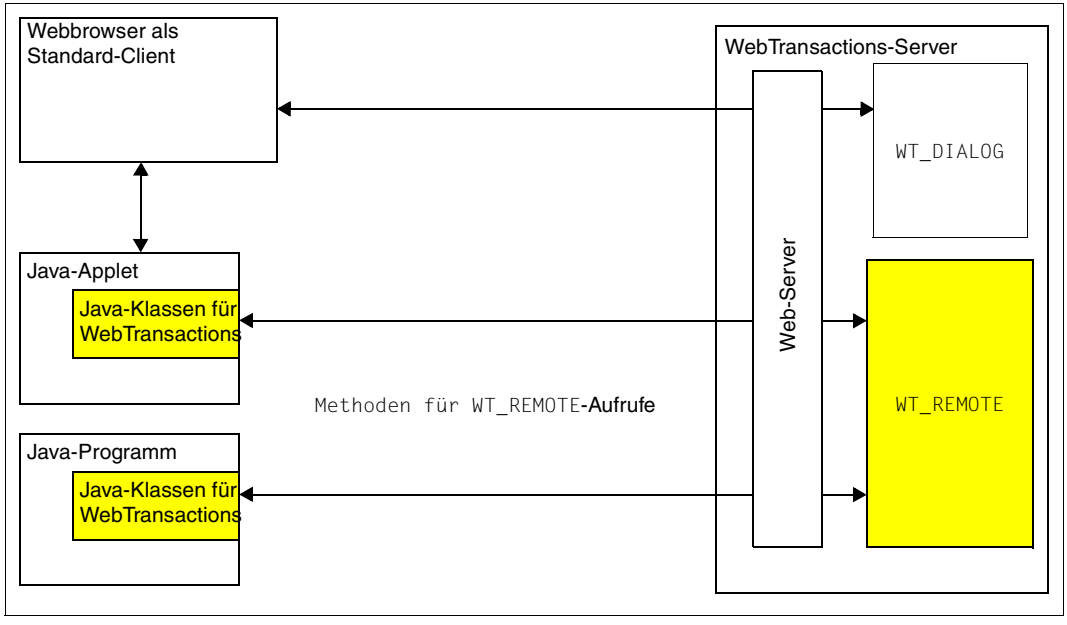

Bild 5: Java-Clients für WebTransactions

Ein Applet unterscheidet sich von einer Java-Anwendung dadurch, dass die Top-Klasse von der Klasse `java.applet.Applet` abgeleitet sein muss. Bei einer Java-Anwendung ist dies nicht der Fall, sie muss lediglich über eine `main`-Methode verfügen. Außerdem gelten unterschiedliche Sicherheitskonzepte.

In der Klassenbibliothek `WTJavaClient.jar` sind drei Klassen enthalten:

– `WTSession` zum Starten und Beenden von WebTransactions-Sitzungen und zum Wiederaufnehmen einer neuen oder bereits bestehenden Sitzung

– `WTObject` zur Repräsentation von Daten einer WebTransactions-Sitzung, zum Aufbau von Datenstrukturen und zum Setzen und Abfragen von Objekten

– `WTObjectRemoteAccess` zum eigentlichen Datenaustausch mit einer WebTransactions-Sitzung, zum Instanziieren von Objekten in der WebTransactions-Sitzung und zum Aufruf von Methoden in der WebTransactions-Sitzung

2.4.1 Applet für WebTransactions-Zugriff

Der Applet-Aufruf wird in einem Template definiert und als Teil der HTML-Seite an den Browser geschickt. Der Browser startet das Applet automatisch und lädt dazu die Applet-Klasse und die Java-Klassen für WebTransactions auf den Client-Rechner.

Die `WTJavaClient`-Klassen müssen auf dem Rechner liegen, auf dem auch der Web-Server und WebTransactions laufen, da ein Applet nur eine Verbindung zu dem Rechner aufbauen darf, von dem es geladen wurde. Deswegen wird das Archiv `WTJavaClient.jar` bei der Installation im Verzeichnis `webtav75` im Dokumentverzeichnis des Web-Servers abgelegt.

Wenn das Applet eine Verbindung zu einer WebTransactions-Sitzung aufbauen soll, müssen die Parameter der Sitzung mit dem PARAM-Tag übergeben werden.

Beispiel

```
<APPLET NAME="CLIPBOOK"
          CODE="clipBook.class"
          ARCHIVE="/webtav75/JavaDemo/java/clipBook.jar"
          WIDTH="516" HEIGHT="207">
    <PARAM NAME="SERVER"     VALUE="##WT_SYSTEM.CGI.SERVER_NAME#">
    <PARAM NAME="SERVERPORT" VALUE="##WT_SYSTEM.CGI.SERVER_PORT#">
    <PARAM NAME="HREF"       VALUE="##WT_SYSTEM.HREF#">
    <PARAM NAME="LANGUAGE"   VALUE="##WT_SYSTEM.LANGUAGE#">
    <PARAM NAME="TRACELEVEL" VALUE="1">
  </APPLET>
```

Mit diesen Werten kann sich das Applet an die bestehende Verbindung zur WebTransactions-Sitzung anhängen (Methode `attach`) und auf die Daten dieser WebTransactions-Anwendung zugreifen. Ebenso kann ein Applet selbst eine WebTransactions-Anwendung starten (Methode `open`).

Mit WebTransactions werden Java-Demo-Anwendungen ausgeliefert, die dieses Vorgehen verdeutlichen. Sie werden bei der Installation in das Unterverzeichnis `webtav75/JavaDemo` im Dokumentverzeichnis des Web-Servers installiert, wenn Sie die Option **WebTransactions Demo Applikationen** auswählen. Bei einer Demo-Anwendung handelt es sich um eine Notizbuch-Funktion, die nach einem nicht synchronen Aufruf während der gesamten Laufzeit der WebTransactions-Sitzung Notizen aufnehmen kann, die vorhandenen Notizen anzeigt und sie beim Beenden zurückschreibt.

2.4.2 Java-Programm für WebTransactions-Zugriff

Das Java-Programm kann eine WebTransactions-Anwendung mit der Methode `open` selbst starten. Java-Programm und die `WTJavaClient`-Klassen liegen auf dem Client-Rechner, der auf die WebTransactions-Anwendung zugreift.

2.4.3 Datenaustausch zwischen Java-Client und WebTransactions

Der Java-Client überträgt mit der Methode `upload` Daten an die WebTransactions-Sitzung. Abhängig vom Inhalt werden diese Daten als neue globale Variablen oder als Attribute von `WT_SYSTEM` oder `WT_HOST` in der WebTransactions-Sitzung angelegt.

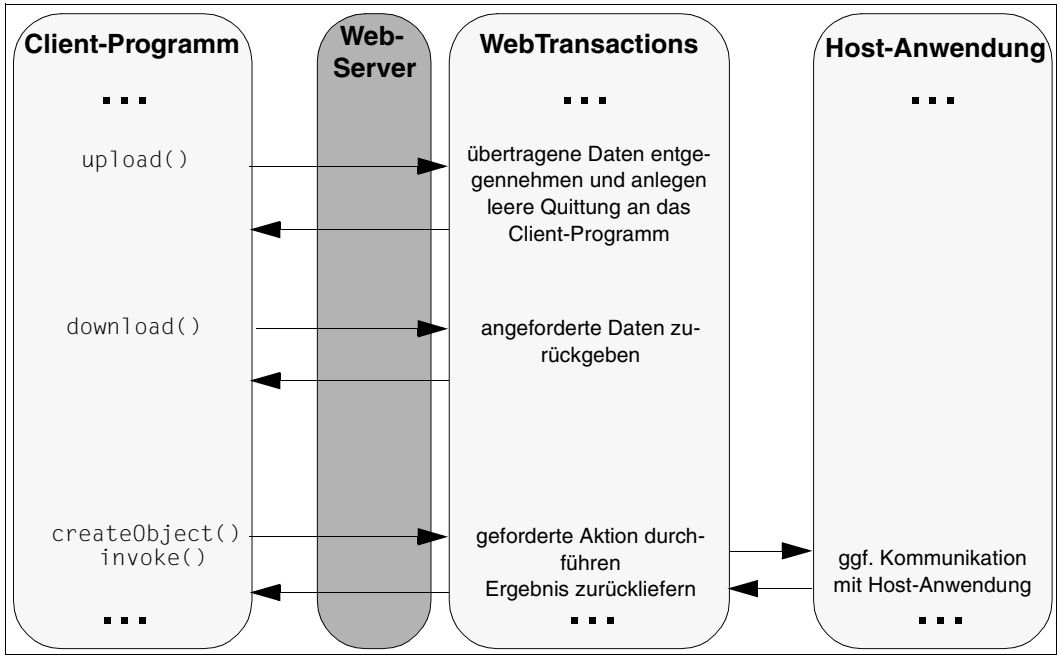

Bild 6: Datenaustausch zwischen Java-Client und WebTransactions

Umgekehrt kann der Java-Client mit der Methode `download` auch Daten der adressierten WebTransactions-Sitzung abfragen. Zudem können in der WebTransactions-Sitzung mit der Methode `CreateObject` eingebaute oder selbst definierte Konstruktoren und mit der Methode `invoke` Methoden und Funktionen ausgeführt werden. Das Ergebnis eines solchen Aufrufs wird zurück an das Client-Programm übertragen.

3 Die Klasse WT_RPC

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. Die Klasse WT_RPC wird in dem mitgelieferten Template wtRPC.htm definiert, das Sie unter config/forms in Ihrem Basisverzeichnis finden. Um Objekte dieser Klasse anlegen zu können, müssen Sie wtRPC.htm in ihrem Template inkludieren. Ein Beispiel für die Anwendung dieser Klasse finden Sie im Kapitel „Beispiel: Verteilte WebTransactions-Anwendung mit WT_RPC“ auf Seite 57.

3.1 Konstruktor

Der Konstruktor WT_RPC legt ein neues WT_RPC-Objekt an, über das die WT_REMOTE-Aufrufe zu einer fernen WebTransactions-Anwendung abgewickelt werden können.

```
WT_RPC()
WT_RPC(urlOfWebTA , basedir)
```

Wird der Konstruktor ohne Argumente aufgerufen, so wird nur das Kommunikationsobjekt angelegt. Soll zusätzlich die ferne WebTransactions-Anwendung gestartet werden, sind die folgenden Argumente notwendig:

urlOfWebTA
: URL des Programms WTPublish auf dem Rechner, dessen WebTransactions-Anwendung gestartet werden soll. *urlOfWebTA* muss auf das Programm WTPublish.exe oder WTPublishISAPI.dll auf der fernen Maschine verweisen (z.B. http://remoteMachine/cgi-bin/WTPublish.exe).

basedir
: String mit dem Basisverzeichnis der fernen WebTransactions-Anwendung.

War der Aufruf erfolgreich und konnte die ferne WebTransactions-Anwendung gestartet werden, so erhält das Attribut WT_CONNECTED des neuen WT_RPC-Objekts den Wert true. Andernfalls erhält es den Wert false.

3.2 Attribute

Ein Objekt der Klasse `WT_RPC` besitzt drei Attribute, die angesprochen werden können. Diese Attribute werden in der Regel von den Methoden gesetzt und verwendet, können aber zu Kontrollzwecken auch von der Client-Anwendung verwendet werden.

```
WT_URL
WT_BASEDIR
WT_CONNECTED
```

`WT_URL`
: enthält die URL zum Aufruf der WebTransactions-Anwendung. Diese URL wird durch den Aufruf von `open` oder vom Konstruktor gesetzt, sofern dort eine URL als Argument angegeben wurde.

`WT_BASEDIR`
: enthält das Basisverzeichnis der fernen WebTransactions-Anwendung. Dieses Basisverzeichnis wird durch den Aufruf von `open` oder dem Aufruf des Konstruktors mit Parametern gesetzt, sofern dort ein Basisverzeichnis als Argument angegeben wurde.

`WT_CONNECTED`
: enthält den Status der Verbindung zur fernen WebTransactions-Anwendung. Dieser Status wird durch den Aufruf von `open` auf `true` gesetzt, sofern die Verbindung erfolgreich aufgebaut wurde. Andernfalls wird er auf `false` gesetzt.

3.3 Methoden

Im Folgenden sind die Methoden der Klasse `WT_RPC` beschrieben.

3.3.1 Methode open

Die Methode `open` startet eine ferne WebTransactions-Anwendung oder stellt die Verbindung mit einer laufenden WebTransactions-Anwendung her.

`open()`
`open(`*urlOfWebTA*`)`
`open(`*urlOfWebTA*`, `*basedir*`)`
`open(`*urlOfWebTA*`, `*basedir*`, `*session*`, `*signature*`)`

---

urlOfWebTA

URL des Programms `WTPublish` auf dem Rechner, dessen WebTransactions-Anwendung gestartet werden soll. *urlOfWebTA* muss auf das Programm `WTPublish.exe` oder `WTPublishISAPI.dll` auf der fernen Maschine verweisen (z.B. `http://remoteMachine/cgi-bin/WTPublish.exe`).

Wird die Methode ganz ohne Parameter aufgerufen, so werden zum Aufbau der Verbindung die Werte der Attribute `WT_URL` und `WT_BASEDIR` des `WT_RPC`-Objekts verwendet; diese müssen zuvor dem Konstruktor oder einem vorangehenden Aufruf von `open` übergeben worden sein.

basedir

Basisverzeichnis der fernen WebTransactions-Anwendung. Wird die Methode ohne Parameter *basedir* aufgerufen, so wird zum Aufbau der Verbindung der Wert des Attributs `WT_BASEDIR` des `WT_RPC`-Objekts verwendet; dieses muss zuvor dem Konstruktor der Klasse `WT_RPC` oder einem vorangehenden Aufruf von `open` übergeben worden sein.

session, *signature*

Werden bei der Methode `open` zusätzlich für die Parameter *session* und *signature* die Werte der Attribute `WT_SYSTEM.SESSION` und `WT_SYSTEM.SIGNATURE` einer entfernten bereits laufenden WebTransactions-Anwendung übergeben, so wird eine Verbindung mit dieser WebTransactions-Anwendung hergestellt. Es wird in diesem Fall keine neue Sitzung gestartet.

Die Werte der übergebenen Parameter werden in den Attributen `WT_URL` und `WT_BASEDIR` des `WT_RPC`-Objekts abgelegt. Ist das Starten der entfernten WebTransactions-Anwendung erfolgreich, so wird das Attribut `WT_CONNECTED` auf `true`, andernfalls auf `false` gesetzt. Der Wert dieses Attributs wird auch von der Methode als Ergebnis zurückgegeben. Werden die Parameter *session* und *signature* nicht angegeben, so wird in jedem Fall versucht, eine neue

entfernte Sitzung zu starten. Besteht in diesem Fall bereits eine Verbindung für das aktuelle Objekt, so wird die zuvor verbundene entfernte WebTransactions-Anwendung beendet (siehe Abschnitt „Methode close“ auf Seite 26).

3.3.2 Methode close

Die Methode `close` beendet die ferne WebTransactions-Anwendung, die mit diesem `WT_RPC`-Objekt verbunden ist. Das Attribut `WT_CONNECTED` des Objekts wird auf `false` gesetzt.

```
close()
```

3.3.3 Methode invoke

Die Methode `invoke` ruft eine Funktion in der fernen WebTransactions-Anwendung auf.

```
invoke(name, codeBase, argArray)
```

name gibt den Namen der Funktion in der fernen WebTransactions-Anwendung an

codeBase
: gibt das WTML-Dokument an, das die Funktionsdefinition enthält

argArray
: ist ein Array, dessen Elemente der entfernten Funktion als Argumente übergeben werden

Die Methode liefert das Ergebnis der fernen Funktion zurück. Besteht keine Verbindung zu einer fernen WebTransactions-Anwendung, so wird `null` zurückgeliefert.

Beispiel

Eine ferne Funktion `add`, die im WTML-Dokument `calc.htm` definiert ist, berechnet die Summe all ihrer Parameter und liefert sie als Ergebnis. Durch einen vorangehenden Konstruktoraufruf wurde bereits ein `WT_RPC`-Objekt `rwt` für die ferne WebTransactions-Anwendung angelegt. Der folgende Aufruf liefert dann das Ergebnis `42` in die Variable `answer` zurück.

```
answer = rwt.invoke( 'add', 'calc', new Array( 1, 2, 3, 4, 32 ));
```

3.3.4 Methode addMethod

Der Aufruf einer fernen Funktion durch die Methode `invoke` ist etwas umständlich, da das definierende Dokument immer wieder angegeben werden muss und da die Parameter in einem Array übergeben werden müssen.

Der Aufruf der Methode `addMethod` definiert eine neue Methode des zu Grunde liegenden `WT_RPC`-Objekts als Stellvertreter für die ferne Funktion. Die ferne Funktion kann nun als Methode des `WT_RPC`-Objekts aufgerufen werden.

`addMethod(`*name*`,` *codeBase*`)`

---

name gibt den Namen der Funktion in der fernen WebTransactions-Anwendung an

codeBase
gibt das WTML-Dokument an, das die Funktionsdefinition enthält

Beispiel

Für die im Abschnitt „Methode invoke“ auf Seite 26 beschriebene Funktion `add` kann durch den folgenden Aufruf eine lokale Stellvertretermethode definiert werden. Die ferne Funktion kann danach über diese Stellvertretermethode aufgerufen werden.

```
rwt.addMethod( 'add', 'calc.htm' );
answer = rwt.add( 1, 2, 3, 4, 32 );
```

3.4 Verteilte Anwendungen mit WT_RPC entwickeln

Wenn Sie verteilte WebTransactions-Anwendungen entwickeln wollen, empfiehlt es sich, in der folgenden Reihenfolge vorzugehen.

1. Funktionalität definieren

   Zunächst legen Sie dabei die Funktionalität der WebTransactions-Anwendung fest. Diese Funktionalität sollten Sie in Form von Funktionen zur Verfügung stellen, d.h. Sie definieren ein API für die gewünschte Funktionalität. Um später leichter auf die Funktionen zugreifen zu können, empfiehlt es sich, das API in einem eigenen Template zur Verfügung zu stellen:

   ```
   //myAPI
   function umsatz(company) {...}
   ```

   Um den Test und die spätere Verteilung zu erleichtern, stellen Sie die gesamte Funktionalität sinnvollerweise als Methoden eines Objekts zur Verfügung:

   ```
   myAPI = new Object();
   myAPI.umsatz = umsatz;
   ```

2. Funktionalität lokal testen

   Testen Sie dann die Funktionen zunächst lokal in der gleichen WebTransactions-Anwendung. So können Sie während des Tests auf die volle Funktionalität von WebLab zurückgreifen. Sie müssen nur ein Test-Template schreiben, dass auf die Funktionen zugreift. Dieses Test-Template inkludiert die Funktionen, die Sie definiert haben und implementiert eine Oberfläche für den Test:

   Umsatz:

   ```
   ##myAPI.umsatz(WT_POSTED.c)#
   ```

   Testen Sie jetzt so lange, bis Ihre Funktionen zufrieden stellend arbeiten.

3. WebTransactions-Anwendung verteilen

   In diesem Schritt müssen Sie nun zwei Dinge tun:

   - Verteilen Sie Ihre Dokumente auf zwei WebTransactions-Anwendungen. Die eine enthält die Templates, die ihre Funktionalität bereitstellen (Server). Die Zweite enthält die Templates, die Sie für den Test verwendet haben (Client).
   - In Ihrem Testtemplate auf dem Client ersetzen Sie das API-Objekt durch ein Objekt der Klasse `WT_RPC` und definieren die Methoden auf dem Server als Methoden dieses Objektes:

     ```
     myAPI = new WT_RPC(...);
     myAPI.addMethod('umsatz',...);
     ```

Wenn Sie beim Konstruktor des `WT_RPC`-Objekts in Ihrer Client-Anwendung die URL und das Basisverzeichnis Ihrer WebTransactions-Anwendung angegeben haben, greift Ihre Client-Anwendung remote auf Ihre WebTransactions-Anwendung zu und Sie haben Ihre Anwendung verteilt.

4 Das Java-Package com.siemens.webta

Java-Klassen werden in so genannten Packages zusammengefasst. Für die Kommunikation mit WebTransactions wird das Package `com.siemens.webta` ausgeliefert. Es wird bei der Installation als `WTJavaClient.jar`-Archiv sowohl im Unterverzeichnis `lib` im Installationsverzeichnis von WebTransactions abgelegt als auch im Verzeichnis `webtav75` im Dokumentverzeichnis des Web-Servers. Dieses Package enthält die Klassen

- `WTSession`, die Methoden zum Aufbau einer Verbindung zur WebTransactions-Anwendung bereitstellt
- `WTObject`, die Methoden für die Objekt-Repräsentation von entfernten `WTObject`-Daten einer laufenden WebTransactions-Sitzung bereitstellt
- `WTObjectRemoteAccess`, die Methoden zum Austausch von Daten mit der WebTransactions-Anwendung bereitstellt

Um mit den `WTJavaClient`-Klassen von WebTransactions zu arbeiten, erweitern Sie die Java-Umgebungsvariable `CLASSPATH` um den Pfad, unter dem die Klassen zu erreichen sind. Im Quellprogramm selbst geben Sie das Package mit der Anweisung `import` Ihrem Java-Programm bekannt.

Beispiel

```
import com.siemens.webta.*;
```

4.1 Klasse WTSession

Die Klasse `WTSession` enthält die Basis-Methoden für die Kommunikation mit WebTransactions. Ein Objekt der Klasse `WTSession` referenziert eine entfernte WebTransactions-Sitzung. Mit einem Objekt dieser Klasse kann eine neue WebTransactions-Sitzung gestartet werden oder das Objekt kann mit einer bestehenden Sitzung verbunden werden. Im Konstruktor geben Sie dazu die Adressierung an. Durch Verwendung der Methoden `open` oder `attach` legen Sie fest, ob eine neue oder eine bestehende WebTransactions-Sitzung verwendet werden soll.

Die Methoden nutzen intern die Schnittstelle `WT_REMOTE`, die in Kapitel „Anhang: Die Schnittstelle WT_REMOTE“ auf Seite 63 beschrieben ist.

```
public class WTSession
```

4.1.1 Konstruktoren

Mit einem Konstruktor legen Sie ein neues Objekt für den Zugriff auf eine entfernte WebTransactions-Sitzung an. Sie übergeben hier die Information, die zur Adressierung einer Sitzung dient.

Es lassen sich zwei Fälle unterscheiden:

- Eine neue WebTransactions-Sitzung wird angelegt und verwendet. In diesem Fall ist es ausreichend, die WebTransactions-Anwendung durch Angabe von Server und Basisverzeichnis angeben.
- Eine bereits existierende WebTransactions-Sitzung soll verwendet werden. Hier werden zur Adressierung zusätzlich die Sitzung-Id und die Signatur dieser Sitzung benötigt, die durch den Parameter `href` übergeben werden können.

Diese Unterscheidung ist nicht endgültig, da die Entscheidung, ob eine neue oder bestehende Sitzung angesprochen wird, erst durch Verwendung der Methoden `open` bzw. `attach` festgelegt wird. Es ist möglich in der Methode `attach` Sitzung-Id und die Signatur nachträglich anzugeben, um eine laufende Sitzung anzusprechen. Sie können aber auch die Methode `open` verwenden und damit ggf. die zuvor mit `href` übergebenen Sitzungsparameter ignorieren.

Verwendung in Applets

Für die Verwendung in Applets gibt es einen speziellen Konstruktor, der nur das Applet-Objekt erhält und sich alle notwendigen Angaben aus entsprechend benannten Parametern des Applets ermittelt. Hier werden ebenfalls beide Fälle, also neue und bereits bestehende WebTransactions-Sitzungen, unterstützt.

4.1.1.1 WTSession für eine neue WebTransactions-Sitzung

`WTSession` erzeugt ein neues Objekt `WTSession`, das für eine neue WebTransactions-Sitzung verwendet werden soll.

```
WTSession(String protocol, String server, int serverPort,
          String WTScriptName, String basedir)
WTSession(String server, int serverPort, String WTScriptName, String basedir)
WTSession(String protocol, String server, String WTScriptName, String basedir)
```

protocol

Protokoll für die Verbindung; dieser Parameter kann entfallen, wenn das HTTP-Protokoll verwendet wird.
Mögliche Werte: http, https
Voreinstellung: http

server

Internetadresse oder symbolischer Name des Rechners, auf dem die WebTransactions-Anwendung liegt

serverPort

Port für die HTTP-Verbindung; Voreinstellung:

80 falls für *protocol* http angegeben ist

443 falls für *protocol* https angegeben ist

WTScriptName

Pfad für den Aufruf von WTPublish (z.B. `/cgi-bin/WTPublish.exe` oder `/scripts/WTPublishISAPI.dll`)

basedir

Basisverzeichnis der WebTransactions-Anwendung

Beachten Sie, dass Sie einen der Parameter *protocol* oder *serverPort* angeben müssen. Die Parameter dürfen nicht gemeinsam entfallen.

Beispiel

```
WTSession osd1=new WTSession("http", "111.222.111.222", 8080,
                             "/cgi-bin/WTPublish.exe",
                             "c:\\WebTABase\\osd_test");
```

4.1.1.2 WTSession für eine bereits bestehende WebTransactions-Sitzung

`WTSession` erzeugt ein neues Objekt `WTSession`, das an eine bereits bestehende WebTransactions-Sitzung angehängt werden soll.

```
WTSession(String protocol, String server, int serverPort, String href)
WTSession(String server, int serverPort, String href)
WTSession(String protocol, String server, String href)
WTSession(String server, String href)
```

protocol

Protokoll für die Verbindung; dieser Parameter kann entfallen, wenn das HTTP-Protokoll verwendet wird.
Mögliche Werte: http, https
Voreinstellung: http

server

Internetadresse oder symbolischer Name des Rechners, auf dem die WebTransactions-Anwendung liegt

serverPort

Port für die HTTP-Verbindung; Voreinstellung:

80 falls für *protocol* http angegeben ist

443 falls für *protocol* https angegeben ist

href

relativer URL für die WebTransactions-Sitzung (entspricht den Werten der Attribute `HREF` oder `HREF_ASYNC` des globalen Systemobjekts)

Beispiel

```
string href="cgi-bin/WTPublish.exe?WT_SYSTEM_BASEDIR=c:/myBase&
             WT_SYSTEM_FORMAT=myStart&WT_SYSTEM_SESSION=E-43585543569&
             WT_SYSTEM_SIGNATURE=1242545991206130607";

WTSession osd1=new WTSession("http", "rechner1", 8080, href);
```

4.1.1.3 WTSession für ein Applet

`WTSession` erzeugt ein neues Objekt `WTSession` für eine neue oder bereits bestehende WebTransactions-Sitzung in einem Applet.

`WTSession(Applet` *app*`)`

---

app aktuelles Applet

Dieser Konstruktor kann nur in einem Java-Applet verwendet werden, wenn die erforderlichen Parameter zum Aufbau einer Verbindung zu einer WebTransactions-Anwendung über Parameter in der HTML-Seite übergeben werden. Beachten Sie, dass HTML Groß-/Kleinschreibung nicht unterscheidet.

Folgende Parameter können Sie einem Applet mit dem PARAM-Tag in einem WTML-Template mitgeben:

- Verbindungsparameter, die Sie immer angeben müssen:

  protocol Protokoll für die Verbindung; dieser Parameter kann entfallen, wenn das HTTP-Protokoll verwendet wird.
  Mögliche Werte: http, https
  Voreinstellung: http

  server Internetadresse oder symbolischer Name des Rechners, auf dem die WebTransactions-Anwendung liegt

  serverPort Port für die HTTP-Verbindung; Voreinstellung:

  80 falls für *protocol* http angegeben ist

  443 falls für *protocol* https angegeben ist

- Verbindungsparameter für eine **neue** WebTransactions-Sitzung:

  WTScriptName
  Pfad für den Aufruf von WTPublish (z.B. `/cgi-bin/WTPublish.exe` oder `/scripts/WTPublishISAPI.dll`)

  basedir Basisverzeichnis der WebTransactions-Anwendung

- Verbindungsparameter für eine **bestehende** WebTransactions-Sitzung:

  href relativer URL für die WebTransactions-Sitzung (entspricht den Werten der Attribute `HREF` oder `HREF_ASYNC` des globalen Systemobjekts)

Statt des Parameters *href* können Sie auch folgende Parameter angeben:

WTScriptName
: Pfad für den Aufruf von `WTPublish` (z.B. `/cgi-bin/WTPublish.exe` oder `/scripts/WTPublishISAPI.dll`)

basedir
: Basisverzeichnis der WebTransactions-Anwendung

session
: Sitzungs-Id der laufenden WebTransactions-Sitzung, entspricht dem Attribut `WT_SYSTEM.SESSION` des globalen Systemobjekts

signature
: Signatur der laufenden WebTransactions-Sitzung, entspricht dem Attribut `WT_SYSTEM.SIGNATURE` des globalen Systemobjekts

Beispiel

Im Folgenden sehen Sie einen Ausschnitt aus einem WTML-Template, in dem ein Applet mit den erforderlichen Parametern für eine bereits bestehende Verbindung zu einer WebTransactions-Sitzung aufgerufen wird.

```
...
<APPLET   code="Applet1.class"
          archive="/webtav75/WTJavaClient.jar"
          codebase="/applets">

   <PARAM name="protocol" value="http">
   <PARAM name="server" value="111.222.111.222">
   <PARAM name="serverPort" value="8080">
   <PARAM name="WTScriptName" value="/cgi-bin/WTPublish.exe">
   <PARAM name="basedir" value="c:/WebTAbase">
   <PARAM name="session" value="##WT_SYSTEM.SESSION#">
   <PARAM name="signature" value="##WT_SYSTEM.SIGNATURE#">
</APPLET>
...
```

Das Applet-Objekt kann dann mit folgendem Konstruktor-Aufruf erzeugt werden:

```
WTSession myApp = new WTSession(this);
```

4.1.2 Methoden

Im Folgenden sind die Methoden der `WTSession`-Klasse in alphabetischer Reihenfolge beschrieben.

4.1.2.1 Methode attach

Die Methode `attach` stellt eine Verbindung zu einer bestehenden WebTransactions-Sitzung her. Falls im Konstruktor bereits eine Sitzung angegeben wurde (durch die Parameter *href* oder *session* und *signature*), so kann `attach` ohne Parameter verwendet werden. Andernfalls ist die Ausprägung mit den Parametern *session* und *signature* zu verwenden. Die Methode attach liefert das aktuelle `WTSession`-Objekt zurück. Dabei ruft die Methode `attach` eine WebTransactions-Sitzung nicht direkt auf. Dies geschieht mit den Methoden der Klasse `WTObjectRemoteAccess`, siehe hierzu auch Abschnitt „Klasse WTObjectRemoteAccess“ auf Seite 51.

Mögliche Ausnahmen sind im Abschnitt „Ausnahmen“ auf Seite 43 beschrieben.

```
WTSession attach(String session, String signature) throws
          WTSessionConnectionException, WTSessionParameterException
WTSession attach() throws
          WTSessionConnectionException, WTSessionParameterException
```

session
: Sitzungs-Id der laufenden WebTransactions-Sitzung, entspricht dem Attribut `SESSION` des globalen Systemobjekts

signature
: Signatur der laufenden WebTransactions-Sitzung, entspricht dem Attribut `SIGNATURE` des globalen Systemobjekts

Beispiel

```
myApp.attach();
```

4.1.2.2 Methode close

Die Methode `close` beendet eine WebTransactions-Sitzung, die mit den Methoden `open` oder `attach` gestartet wurde. Sie setzt im zugrunde liegenden Objekt die Adressierung einer WebTransactions-Anwendung zurück, da die bislang referenzierte WebTransactions-Sitzung nicht mehr existiert. Wenn noch keine Sitzung gestartet wurde, wird nur auf die Adressierung einer WebTransactions-Anwendung zurückgesetzt. Mögliche Ausnahmen sind im Abschnitt „Ausnahmen“ auf Seite 43 beschrieben.

```
void close() throws
          WTSessionConnectionException,
          WTCloseSessionException
```

Beispiel

```
myApp.close();
```

4.1.2.3 Methode open

Die Methode `open` startet eine neue WebTransactions-Sitzung und liefert das aktuelle `WTSession`-Objekt zurück. Die Adressierung der WebTransactions-Anwendung wurde bereits mit dem Konstruktor festgelegt. Die Methode `open` ruft vor dem Start einer neuen WebTransactions-Sitzung implizit die Methode `close` auf und nach dem erfolgreichen Start die Methode `attach`.

Wenn die neue Sitzung mit speziellen Timeout-, Sprach- oder Stil-Einstellungen gestartet werden soll, können Sie die entsprechenden Methoden vor der Methode `open` aufrufen. Mögliche Ausnahmen sind im Abschnitt „Ausnahmen“ auf Seite 43 beschrieben.

```
WTSession open() throws
          WTSessionConnectionException,
          WTSessionParameterException,
          WTCloseSessionException
```

Beispiel

```
myApp.open();
```

4.1.2.4 Methode setApplTimeout

Die Methode `setApplTimeout` setzt den Wert des Systemobjekt-Attributs `TIMEOUT_APPLICATION` für den nächsten Aufruf der WebTransactions-Anwendung. `setApplTimeout` liefert das aktuelle `WTSession`-Objekt zurück.

`WTSession setUserTimeout(int` *applTimeout*`)`

---

applTimeout
: Zeitspanne in Sekunden für Anworten von der Host-Anwendung innerhalb der WebTransactions-Sitzung

Beachten Sie, dass bei dieser Methode nicht mit der WebTransactions-Anwendung kommuniziert wird. Der gesetzte Wert für `TIMEOUT_APPLICATION` wird zwischengespeichert und erst mit der nächsten Kommunikationsmethode (z.B. `open`) gesendet.

Beispiel

```
myApp.setApplTimeout(60);
```

4.1.2.5 Methode setLanguage

Die Methode `setLanguage` setzt den Wert des Systemobjekt-Attributs `LANGUAGE` für den nächsten Aufruf der WebTransactions-Anwendung. `setLanguage` liefert das aktuelle `WTSession`-Objekt zurück.

`WTSession setLanguage(String` *language*`)`

---

language
: Verzeichnis `config` der WebTransactions-Anwendung, in dem die Templates für die entsprechende Sprache der Oberfläche abgelegt sind.

Beachten Sie, dass bei dieser Methode nicht mit der WebTransactions-Anwendung kommuniziert wird. Der gesetzte Wert für `LANGUAGE` wird zwischengespeichert und erst mit der nächsten Kommunikationsmethode (z.B. `open`) gesendet.

Beispiel

```
myApp.setLanguage("engl");
```

4.1.2.6 Methode setStyle

Die Methode `setStyle` setzt den Wert des Systemobjekt-Attributs `STYLE` für den nächsten Aufruf der WebTransactions-Anwendung. `setStyle` liefert das aktuelle `WTSession`-Objekt zurück.

`WTSession setStyle (String` ***style***`)`

---

style Unterverzeichnis im Verzeichnis `config` der WebTransactions-Anwendung, in dem die Templates für den entsprechenden Stil der Oberfläche abgelegt sind.

Beachten Sie, dass bei dieser Methode nicht mit der WebTransactions-Anwendung kommuniziert wird. Der gesetzte Wert für `STYLE` wird zwischengespeichert und erst mit der nächsten Kommunikationsmethode (z.B. `open`) gesendet.

Beispiel

```
myApp.setStyle("lay1");
```

4.1.2.7 Methode setTraceLevel

Die Methode `setTraceLevel` aktiviert die Trace-Funktion für das `WTJavaClient`-Objekt.

```
void setTraceLevel(int traceLevel)
```

traceLevel

numerischer Wert, der die Protokollierungstiefe bestimmt. Die einzelnen Werte sind als Variablen definiert. Die Tabelle gibt einen Überblick, welche Werte Sie angeben können und welche Variablendefinition diesem Wert entspricht:

Wert	Variablendefinition	Bedeutung
0	`public static final int traceLevel_Off`	Die Trace-Funktion ist ausgeschaltet
1	`public static final int traceLevel_WTSession`	Trace der Klasse `WTSession`
2	`public static final int traceLevel_WTObjectRemoteAccess`	Trace der Klasse `WTObjectRemoteAccess`
4	`public static final int traceLevel_WTObject`	Trace der Klasse `WTObject`
8	`public static final int traceLevel_WTXMLHandler`	Trace des XML-Parsers

Tabelle 1: Ebenen der Protokollierung

Die verschiedenen Protokoll-Möglichkeiten können Sie kombinieren, indem Sie die numerischen Werte addieren oder die Konstanten logisch mit ODER verknüpfen.

Beispiel

```
myApp.setTraceLevel(WTSession.traceLevel_WTSession +
                    WTSession.traceLevel_WTObject);
```

Wenn Sie für die Konstanten die Summe der numerischen Werte einsetzen (`WTSession` =1 und `WTObject` =4), ist die folgende Zeile synonym zur ersten:

```
myApp.setTraceLevel(5);
```

4.1.2.8 Methode setUserTimeout

Die Methode `setUserTimeout` setzt den Wert des Systemobjekt-Attributs `TIMEOUT_USER` für den nächsten Aufruf der WebTransactions-Anwendung. `setUserTimeout` liefert das aktuelle `WTSession`-Objekt zurück.

`WTSession setUserTimeout(int` ***userTimeout***`)`

---

userTimeout

Zeitspanne in Sekunden für Anworten vom Benutzer innerhalb der WebTransactions-Sitzung

Beachten Sie, dass bei dieser Methode nicht mit der WebTransactions-Anwendung kommuniziert wird. Der gesetzte Wert für `TIMEOUT_USER` wird zwischengespeichert und erst mit der nächsten Kommunikationsmethode (z.B. `upload`) gesendet. Da der Wert erst beim nächsten Aufruf der WebTransactions-Anwendung wirksam wird, hat er bis dahin keine Auswirkungen auf die aktuell laufende Sitzung.

Beispiel

```
myApp.setUserTimeout(360);
```

4.1.3 Ausnahmen

Die Methoden `open`, `attach` und `close` des `WTSession`-Objekts können folgende Ausnahmen auslösen:

Ausnahme	Bedeutung
`WTSessionConnectionException`	Diese Ausnahme wird ausgelöst, wenn die Verbindungsparameter für eine WebTransactions-Sitzung nicht korrekt sind. Prüfen Sie die Parameter *protocol*, *server*, *serverPort* oder *WTScriptName*, die Sie beim Konstruktor `WTSession` angegeben haben.
`WTSessionParameterException`	Diese Ausnahme wird ausgelöst, wenn einer der Parameter *session* oder *signature* für eine WebTransactions-Sitzung nicht angegeben ist. Prüfen Sie diese Parameter, die Sie beim Konstruktor `WTSession` angegeben haben.
`WTCloseSessionException`	Diese Ausnahme wird ausgelöst, wenn die WebTransactions-Sitzung nicht geschlossen werden kann. Die Parameter *session* und *signature* werden aber in jedem Fall zurückgesetzt, sodass Sie diese Ausnahme in der Regel ignorieren können.

Tabelle 2: Ausnahmen für `WTSession`

4.1.4 Beispiel

Im Folgenden sehen Sie einen Ausschnitt aus einem Java-Programm.

```
// construct a WTSession object for a remote WebTransactions session
WTSession wtSession = new WTSession("PGTD1234",
                                     80,
                                    "/scripts/WTPublishISAPI.dll",
                                    "c:/basedirs/myAppl");
// set the application and user timeout object for the new
// WebTransactions session
wtSession.setApplTimeout("60").setUserTimeout("3600");
// open a new WebTransactions session for the WTSession object
wtSession.open();
...
```

4.2 Klasse WTObject

Die Klasse `WTObject` spiegelt den wesentlichen Teil des WebTransactions-Objektmodells wider. Sie stellt Methoden zur Bearbeitung der Objekte zur Verfügung und liefert der Klasse `WTObjectRemoteAccess` somit die Objekte und Methoden für den Datenaustausch mit der fernen WebTransactions-Anwendung.

```
public class WTObject
```

4.2.1 Konstruktor

`WTObject` erzeugt ein neues Objekt `WTObject`, das einem WebTransactions-Objekt entspricht. Datentypen und Klassen sind als Java-Variablen vordefiniert. Der Wert des Objekts wird als Zeichenkette verwaltet. Wenn Sie mit dem realen Wert arbeiten wollen, müssen Sie ihn in den entsprechenden Datentyp konvertieren.

```
WTObject(int objectType )
WTObject(int objectType, String objectValue)
WTObject(int objectType, int objectClass)
WTObject(int objectType, int objectClass, String objectValue)
```

objectType

Datentyp des neuen Objekts, Sie können folgende Werte angeben:

Java-Variablendefinition	**WTML-Datentyp**
`public static final int TYPE_UNDEFINED`	`undefined`
`public static final int TYPE_STRING`	`string`
`public static final int TYPE_NUMBER`	`number`
`public static final int TYPE_BOOLEAN`	`boolean`
`public static final int TYPE_OBJECT`	`object`
`public static final int TYPE_FUNCTION`	`function`

Tabelle 3: Java-Definitionen für WTML-Datentypen

objectClass

Klasse des neuen Objekts, Sie können folgende Werte angeben:

Variablendefinition in Java	**WTML-Klasse**
`public static final int CLASS_UNDEFINED`	`Undefined`
`public static final int CLASS_STRING`	`String`
`public static final int CLASS_NUMBER`	`Number`
`public static final int CLASS_BOOLEAN`	`Boolean`
`public static final int CLASS_OBJECT`	`Object`
`public static final int CLASS_ARRAY`	`Array`
`public static final int CLASS_REGEXP`	`Regexp`
`public static final int CLASS_FUNCTION`	`Function`
`public static final int CLASS_WTHOSTOBJECT`	`WT_Hostobject`
`public static final int CLASS_WTCOMMUNICATION`	`WT_Communication`
`public static final int CLASS_DOCUMENT`	`Document`
`public static final int CLASS_WTUSEREXIT`	`WT_Userexit`
`public static final int CLASS_DATE`	`Date`

Tabelle 4: Java-Definitionen für WTML-Klassen

Wenn Sie keine Klasse angeben, wird ein Objekt der Klasse `Undefined` angelegt.

objectValue

Wert des neuen Objekts; Wenn Sie einem Objekt keinen Wert zuweisen, wird das Objekt mit dem Wert `null` initialisiert.

Beispiel

```
WTObject local_obj= new WTObject(WTObject.TYPE_OBJECT,
                                 WTObject.CLASS_OBJECT);
```

4.2.2 Methoden

Im Folgenden sind die Methoden der Klasse `WTObject` in alphabetischer Reihenfolge beschrieben.

4.2.2.1 Methode getAttribute

Die Methode `getAttribute` liefert das angegebene Attribut des aktuellen Objekts als `WTObject`-Objekt zurück. Ist das gesuchte Attribut nicht vorhanden, wird der Wert `null` zurückgeliefert.

```
WTObject getAttribute(String attributeName)
```

attributeName
: Name des gesuchten Attributs, das auch in der Objekthierarchie angegeben werden kann

Beispiel

```
WTObject local_obj=remote_obj.download("WT_SYSTEM");
WTObject style=local_obj.getAttribute("STYLE");
WTObject script=local_obj.getAttribute("CGI.SCRIPT_NAME");
```

4.2.2.2 Methode getAttributeNames

Die Methode `getAttributeNames` liefert die Namen aller Attribute des aktuellen Objekts in einem Zeichenketten-Array zurück.

```
String [] getAttributeNames()
```

Beispiel

```
String[] Attribute=local_obj.getAttributeNames();
```

4.2.2.3 Methode getValueAsString

Die Methode `getValueAsString` liefert den Wert des aktuellen Objekts als Zeichenkette zurück. Beachten Sie, dass Sie den Wert in den entsprechenden Datentyp konvertieren müssen, wenn Sie mit dem realen Wert arbeiten wollen.

```
String getValueAsString()
```

Beispiel

```
String wert=local_obj.getValueAsString();
```

4.2.2.4 Methode getWTClass

Die Methode `getWTClass` liefert die Klasse des aktuellen Objekts als Konstante zurück. Die Bedeutung der Konstanten ist in der Tabelle „Java-Definitionen für WTML-Klassen“ auf Seite 45 beschrieben.

```
int getWTClass()
```

Beispiel

```
int objektKlasse=local_obj.getWTClass();
```

4.2.2.5 Methode getWTType

Die Methode `getWTType` liefert den Datentyp des aktuellen Objekts als Konstante zurück. Die Bedeutung der Konstanten ist in der Tabelle „Java-Definitionen für WTML-Datentypen“ auf Seite 44 beschrieben.

```
int getWTType()
```

Beispiel

```
int objektTyp=local_obj.getWTType();
```

4.2.2.6 Methode removeAttribute

Die Methode `removeAttribute` löscht das angegebene Attribut aus dem aktuellen Objekt und liefert das aktuelle Objekt zurück. Wenn das angegebene Attribut nicht existiert, hat die Methode keine Auswirkungen.

`WTObject removeAttribute(String` *attributeName*`)`

---

attributeName
: Name des Attributs, das gelöscht werden soll.

Beispiel

```
local_obj.removeAttribute("myObj");
```

4.2.2.7 Methode setAttribute

Die Methode `setAttribute` setzt das angegebene Attribut im aktuellen Objekt und liefert das gesetzte Objekt zurück. Wenn das gesetzte Attribut im aktuellen Objekt noch nicht existiert, wird es angelegt. Wenn bei der angegebenen Objekthierarchie eine Ebene nicht existiert, liefert `setAttribute` den Wert `null` zurück und das Attribut wird nicht gesetzt.

`WTObject setAttribute(String` *attributeName*`, WTObject` *object*`)`

---

attributeName
: Name des Attributs, das gesetzt oder angelegt werden soll. Das Attribut kann auch in der Objekthierarchie angegeben werden.

object Beschreibung des Attributs als `WTObject`-Objekt

Beispiel

```
local_obj.setAttribute("STYLE",new WTObject(WTObject.TYPE_STRING,"N"));
```

4.2.2.8 Methode setValue

Die Methode `setValue` setzt den Wert des aktuellen Objekts. Beachten Sie, dass der Wert immer eine Zeichenkette sein muss, auch wenn der Datentyp des Objekts keine Zeichenkette ist.

```
void setValue(String value)
```

value
: Wert des aktuellen Objekts als Zeichenkette

Beispiel

```
local_obj.setValue("42");
```

4.2.3 Ausnahmen

Für diese Klasse sind keine Ausnahmen definiert.

4.2.4 Beispiel

```
...
// create a new WebTransactions remote access object and open a
// WebTransactions session,
// it is supposed that the WTSession parameters are passed as parameters
// to the applet via the <param> tag.
WTObjectRemoteAccess wtSession = new WTObjectRemoteAccess (
                                 new WTSession(this).open());

// create and download WT_SYSTEM object from WebTransactions session
WTObject wt_system = wtSession.download("WT_SYSTEM");
// determine if connection runs via proxy HOST1 and store it in new system
// object
if (wt_system.getAttribute("CGI.REMOTE_HOST").getValueAsString().
equals("HOST1"))
   wt_system.setAttribute("PROXY", new WTObject(WTObject.TYPE_STRING,
"YES"));
else
   wt_system.setAttribute("PROXY", new WTObject(
                          WTObject.TYPE_STRING, "NO"));

// change attribute style of system object to value APPLREMOTE
wt_system.getAttribute("STYLE").setValue("APPLREMOTE");

// upload current wt_system object into WT_SYSTEM object of remote
// WebTransactions session
wtSession.upload(wt_system ,"WT_SYSTEM");

// download all value attributes of host objects of WT_HOST.MYCOM from
// WebTransactions
WTObject wt_host = wtSession.download("WT_HOST.MYCOM..Value");
WTObject my_comm = wt_host.getAttribute("MYCOM");
// shortcut to WT_HOST.MYCOM

// invoke remote function wtmlMeth of template func.htm with parameters
//of two host objects and store the return value in local variable
WTObject[] params = new WTObject[2];
params[0] = my_comm.getAttribute("E_01_001_80.Value");
params[1] = my_comm.getAttribute("E_02_001_80.Value");
String result = (wtSession.invoke(
                 "wtmlMeth", params, "func.htm")).getValueAsString();
...
```

4.3 Klasse WTObjectRemoteAccess

Die Klasse `WTObjectRemoteAccess` enthält Methoden für den entfernten Zugriff auf eine WebTransactions-Sitzung. Sie ermöglicht den Austausch von Daten und die entfernte Ausführung von Methoden und Funktionen. Zur Darstellung der Daten auf der Java-Seite werden Objekte der Klasse `WTObject` verwendet. Die entfernte Sitzung wird durch ein Objekt der Klasse `WTSession` angegeben, das alle Adressierungs-Informationen zusammenfasst.

```
public class WTObjectRemoteAccess
```

4.3.1 Konstruktor

`WTObjectRemoteAccess` erzeugt ein neues Objekt `WTObjectRemoteAccess`. Die Ausnahme ist im Abschnitt „Ausnahmen“ auf Seite 56 beschrieben.

```
WTObjectRemoteAccess(WTSession wtSession) throws
          WTSessionNotAttachedException
```

wtSession
: aktuelles `WTSession`-Objekt, das auf eine WebTransactions-Sitzung zeigt.

Beispiel

```
WTObjectRemoteAccess remote_obj=new WTObjectRemoteAccess(myApp);
```

4.3.2 Methoden

Beachten Sie, dass die folgenden Methoden nur korrekt ausgeführt werden können, wenn eine Verbindung zu einer WebTransactions-Sitzung geöffnet ist. Eine Verbindung öffnen Sie mit den Methoden `open` oder `attach` eines `WTSession`-Objekts, siehe hierzu auch Abschnitt „Klasse WTSession“ auf Seite 32.

4.3.2.1 Methode createObject

Die Methode `createObject` erzeugt in der fernen WebTransactions-Sitzung ein neues Objekt und liefert dieses Objekt als `WTObject`-Objekt zurück. Wenn der Methodenaufruf scheitert, wird die entsprechende Ausnahme ausgelöst, siehe hierzu Abschnitt „Ausnahmen“ auf Seite 56.

```
WTObject createObject(String name, String constructor [,WTObject[] parameters]
          [,String codebase]) throws WTSessionNotAttachedException,
          WTSessionConnectionException, WTXMLParserException,
          WTNoXMLException
```

name Name des neuen Objekts

constructor
Name des entfernten Konstruktors, mit dem das Objekt erzeugt werden soll

parameters
Ein Array von `WTObjects`-Objekten mit den Parametern des Konstruktors. Wenn der Konstruktor keine Parameter erwartet, entfällt dieser Parameter.

codebase
Gibt in der entfernten WebTransactions-Anwendung das Template an, in dem der Konstruktor definiert ist. Dieser Parameter kann entfallen, wenn der Konstruktor in der WebTransactions-Sitzung bereits bekannt ist, wenn es sich z.B. um den Konstruktor einer eingebauten Klasse handelt, oder wenn der Konstruktor dem globalen Systemobjekt `WT_SYSTEM` zugewiesen ist.

Beispiel

Das Folgende Beispiel erzeugt eine Funktion, die die Methode `receive` aufruft, ohne dass das gesamte Kommunikationsobjekt als Rückgabewert übertragen wird.

```
...
WTObject[] params=new WTObject[1];
params[0]=new WTObject(WTObject.TYPE_STRING,"{this.receive();}");
remote_obj.createObject("WT_HOST.osd.silentReceive",
                        "Function", params);
```

4.3.2.2 Methode download

Die Methode `download` überträgt die angegebene Objektstruktur aus einer WebTransactions-Sitzung in ein `WTObject`-Objekt und liefert dieses Objekt zurück. Wenn der Methodenaufruf scheitert, wird die entsprechende Ausnahme ausgelöst, siehe hierzu Abschnitt „Ausnahmen“ auf Seite 56.

```
WTObject download(String remotePattern) throws
                      WTSessionNotAttachedException,
                      WTSessionConnectionException,
                      WTXMLParserException,
                      WTNoXMLException
```

remotePattern

Name oder Suchbegriff des entfernten WebTransactions-Objekts. *remotePattern* können Sie folgendermaßen verwenden:

remotePattern	**Bedeutung**
object	Ein beliebiges Objekt mit allen Attributen. Sind Attribute selbst wieder Objekte, so wird die Konvertierung für diese Objekte rekursiv fortgeführt.
object`.`	Ein beliebiges Objekt ohne Attribute. Auf Grund des abschließenden Punkts werden keine Attribute dieses Objekts konvertiert.
object`..`	Ein beliebiges Objekt mit Attributen, aber ohne Unterobjekte, da 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.
object1`|`*object2* *object*`..`*wert1*`|`*wert2*	Mehrere Objekte oder mehrere Attribute unterhalb eines Objekts. Die Objekte *object1* und *object2* bzw. alle Attribute mit den Namen *wert1* oder *wert2*, die in Objekten direkt unterhalb des Datenobjekts *object* enthalten sind. Das Zeichen `|` bindet stärker als des Zeichen `.`, so dass folgendes Beispiel gilt: `WT_HOST|WT_SYSTEM.xyz` liefert `WT_HOST.xyz` **und** `WT_SYSTEM.xyz`

Tabelle 5: Mögliche Angaben für die zu ladende Objektstruktur

Auf oberster Objekt-Ebene ist eine ODER-Verknüpfung wie beispielsweise `"WT_SYSTEM | WT_HOST"` nicht erlaubt.

Beispiel

```
WTObject wt_system=remote_obj.download("WT_SYSTEM.CGI");
```

4.3.2.3 Methode invoke

Die Methode `invoke` ruft eine entfernte Methode oder Funktion in einer WebTransactions-Sitzung auf. `invoke` liefert den Rückgabewert der ausgeführten Methode oder Funktion als `WTObject` oder den Wert `null` zurück, wenn die Methode oder Funktion keinen Wert zurückgibt. Wenn der Methodenaufruf scheitert, wird die entsprechende Ausnahme ausgelöst, siehe hierzu Abschnitt „Ausnahmen“ auf Seite 56.

```
WTObject invoke(String methodName [, WTObjects[] parameters]
                      [,String codebase]) throws
                      WTSessionNotAttachedException,
                      WTSessionConnectionException,
                      WTXMLParserException,
                      WTNoXMLException
```

methodName
: Name der Methode oder Funktion, die auch in der Objekthierarchie angegeben werden kann, z.B. `host.myMeth`.

parameters
: Ein Array von `WTObject`-Objekten mit den Parametern der Methode oder Funktion. Dieser Parameter ist optional, d.h. die Methode oder Funktion muss keine Parameter haben.

codebase
: Gibt in der entfernten WebTransactions-Anwendung das Template an, in dem die auszuführende Methode oder Funktion definiert ist. Das Template wird entsprechend der Suchreihenfolge von WebTransactions gesucht, siehe hierzu auch das WebTransactions-Handbuch „Konzepte und Funktionen“.
Dieser Parameter kann entfallen, wenn die Methode oder Funktion in der WebTransactions-Sitzung bereits bekannt ist, wenn es sich z.B. um eine Methode der eingebauten Klassen oder um eine globale WTML-Funktion handelt, oder wenn die Methode dem globalen Systemobjekt `WT_SYSTEM` zugewiesen ist.

Beispiel

```
...
WTObject[] params=new WTObject[2];
params[0]=myApp.getAttribute("E_02_001_80");
params[1]=myApp.getAttribute("E_06_005_85");
remote_obj.invoke("myMeth",params, "\temp1.htm");
```

4.3.2.4 Methode upload

Die Methode `upload` überträgt ein `WTObject`-Objekt mit dem angegebene Namen in eine entfernte WebTransactions-Sitzung. `upload` arbeitet additiv. Das bedeutet: Objekte bzw. Attribute, die noch nicht existieren, werden angelegt. Bereits bestehende Objekte bzw. Attribute werden überschrieben.

```
Void upload(WTObject object, String remoteObjectName) throws
                     WTSessionNotAttachedException,
                     WTSessionConnectionException,
                     WTXMLParserException,
                     WTNoXMLException
```

object
: Objekt, das in die WebTransactions-Sitzung übertragen werden soll

remoteObjectName
: Name des übertragenen Objekts in der WebTransactions-Sitzung

Beispiel

```
remote_obj.upload(wt_system, "WT_SYSTEM");
```

4.3.3 Ausnahmen

Die Methoden `invoke`, `createObject`, `download`, `upload` und der Konstruktor des `WTObjectRemoteAccess`-Objekts können folgende Ausnahmen auslösen:

Ausnahme	Bedeutung
`WTSessionNotAttachedException`	Diese Ausnahme wird ausgelöst, wenn das aktuelle `WTSession`-Objekt keine Verbindung zu einer WebTransactions-Sitzung hat. Sie müssen erst mit den Methoden `open` oder `attach` des `WTSession`-Objekts eine Verbindung zu einer WebTransactions-Anwendung öffnen, bevor Sie Methoden des `WTObjectRemoteAccess`-Objekts nutzen können.
`WTSessionConnectionException`	Diese Ausnahme wird ausgelöst, wenn die Verbindungsparameter für eine neue WebTransactions-Sitzung nicht korrekt sind. Prüfen Sie die Parameter *protocol*, *server*, *serverPort* oder *WTScriptName*, die Sie beim Konstruktor `WTSession` angegeben haben.
`WTXMLParserException`	Diese Ausnahme wird ausgelöst, wenn ein WebTransactions-Fehler auftritt oder der Parser einen XML-Fehler im aktuellen Dokument gefunden hat. Bei einem WebTransactions-Fehler ist der Meldungstext Teil der Ausnahme-Meldung.
`WTNoXMLException`	Diese Ausnahme wird ausgelöst, wenn WebTransactions kein XML-Dokument als Antwort schickt. Das gesendete Dokument wird als Teil der Ausnahme-Meldung ausgegeben.

Tabelle 6: Ausnahmen von `WTObjectRemoteAccess`

5 Beispiel: Verteilte WebTransactions-Anwendung mit WT_RPC

Dieses Kapitel zeigt anhand eines Beispiels, wie die beschriebenen Schnittstellen verwendet werden können.

Das Beispiel beschreibt eine Filter-Anwendung für den HTTP-Host-Adapter am Beispiel des mitgelieferten WebTransactions-Client `WT_RPC` für `WTRemote`.

5.1 Einsatzszenario

Dieses Beispiel geht von zwei räumlich getrennten Host-Anwendungen `AppA` und `AppB` aus, die unter einer gemeinsamen Web-Oberfläche integriert werden sollen.

Die Host-Anwendungen enthalten unterschiedliche Formate `mA1` und `mA2` bzw. `mB1` und `mB2`, über die logisch äquivalente Daten ermittelt werden können. Zur Abwicklung eines Geschäftsvorfalls sind jeweils die beiden Formate folgendermaßen zu durchlaufen.

Der Terminalbenutzer meldet sich mit dem Kommando `logon user,password` bzw. `login u=user,p=password` an. Danach gelangt er mit dem Kommando `mA1` bzw. `mB1` auf das Format zur Eingabe der Recherche. Auf `mA1` bzw. `mB1` wird eine Personalnummer eingegeben, auf dem Format `mA2` bzw. `mB2` wird das Eintrittsdatum ausgegeben. Ziel der gemeinsamen Web-Oberfläche ist es, für beide Formate einen einheitlichen Zugriff auf das Eintrittsdatum zu gewährleisten.

Von den in den Formaten präsentierten Daten ist jeweils nur ein kleiner Teil für die jeweilige Recherche relevant:

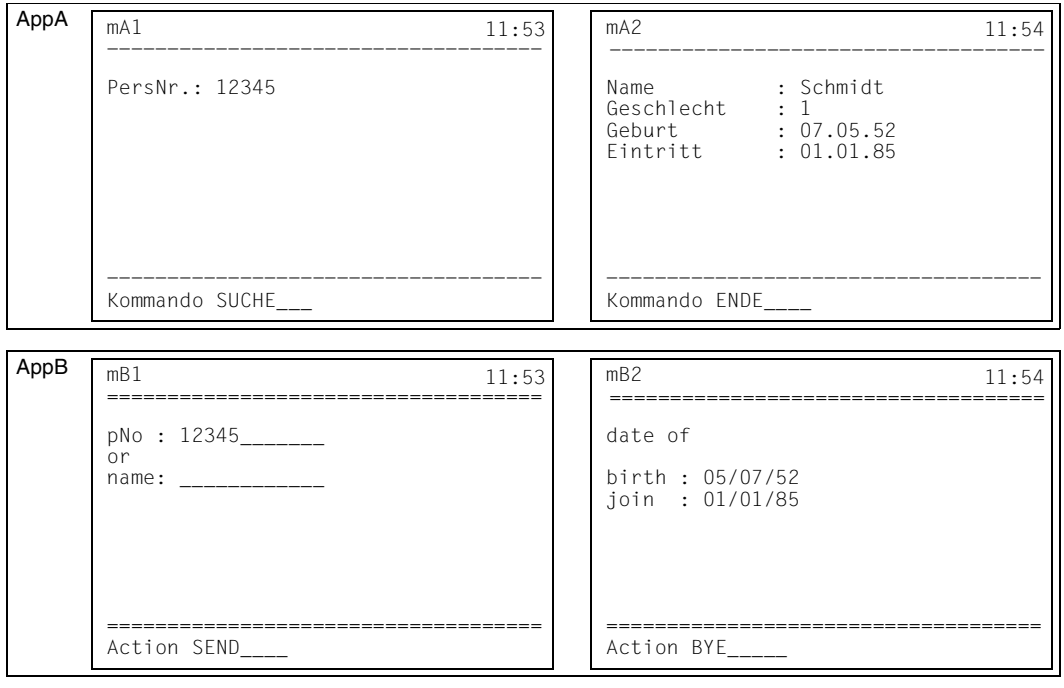

Bild 7: Die unterschiedlichen Host-Anwendungen und ihre Formate

5.2 Technisches Konzept

Um einen einheitlichen Zugriff auf die beiden Host-Anwendungen zu ermöglichen, wird ein API zur Abwicklung der Geschäftsvorfälle definiert. Es werden WTScript-Funktionen realisiert, die den Dialog mit der jeweiligen Host-Anwendung durchführen. Da die Hosts an unterschiedlichen Orten stehen, die Daten sehr umfangreich und die Formate sehr unterschiedlich sind, ist es sinnvoll, diese Anwendung zu verteilen und so eine Vorverdichtung der Daten zu erlauben. Für beide Hosts sollen auf unterschiedlichen WebTransactions-Rechnern, die jeweils räumlich nahe bei den jeweiligen Host-Rechnern stehen, die Methoden `logon(`*user*`,`*password*`)` und `eintritt(`*persNr*`)` als WebTransactions-Anwendungen zur Verfügung stehen, um die entsprechenden Aufgaben durchzuführen. Diese WebTransactions-Anwendungen könnte man auch als *Mainframe-Treiber* bezeichnen. Die Methoden dieser WebTransactions-Anwendungen werden von einer dritten WebTransactions-Integrations-Anwendung aufgerufen und im Web präsentiert.

Eine weitere Nutzungsmöglichkeit dieser *Mainframe-Treiber* ist, dass auf diese Weise eine reduzierte Sicht auf die Anwendung (Recherche nur nach Eintrittsdatum) für die Verwendung in anderen Anwendungen zur Verfügung gestellt werden kann.

5.3 Implementierung der Integrations-Anwendung

Die Recherche in den beiden Applikationen wird als verteilte WebTransactions-Anwendung realisiert. Je eine WebTransactions-Anwendung `WTSrvA` bzw. `WTSrvB` greift direkt auf die Applikation `AppA` bzw. `AppB` zu und stellt die Funktionalität als Funktionen `logon(`*user*`,`*password*`)` und `eintritt(`*persNr*`)` zur Verfügung. Eine WebTransactions-Integrations-Anwendung `WTInt` greift über die Klasse `WT_RPC` auf diese Funktionen zu:

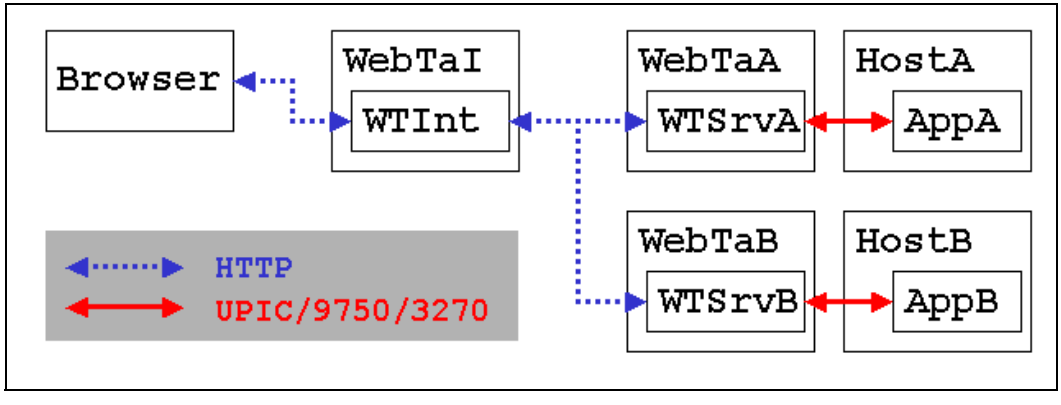

Bild 8: Die Integrations-Anwendung `WTInt`

WebTransactions-Anwendungen

Die WebTransactions-Anwendungen `WTSrvA` und `WTSrvB` enthalten je ein WTML-Dokument `accHost.htm`, in dem die Zugriffsfunktionen `logon` und `eintritt` realisiert werden. Die Dokumente werden sich leicht unterscheiden, da die Host-Anwendungen nicht identisch sind. Das Template für die Host-Anwendung `AppA` könnte z.B. folgendermaßen aussehen:

```
function logon(user, password)
{
  host = new WT_Communication("appA");
  host_system = host.WT_SYSTEM;
  host_system.HOST_NAME = "HostA";
  host_system.SYM_DEST = "AppA";
  host.open("OSD");
  host.receive();
  host.E_1_1.Value = 'logon ' + user + ',' + password;
  host.send();
  host.receive();
  return (E_1_1.Value == 'logged in');
}
function eintritt(persNr)
{
  WT_SYSTEM.ERROR = '';
  host = WT_HOST.appA;
  host.E_8_10.Value = 'GOTO mA1';
  host.send();
  host.receive();
  host.E_3_10.Value = persNr;
  host.send();
  host.receive();
  return (WT_SYSTEM.ERROR ? false : E_6_12.Value);
}
```

Die Funktion `logon` legt ein neues Kommunikationsobjekt an und öffnet über den OSD-Host-Adapter eine Verbindung mit der Host-Anwendung `AppA` auf dem Rechner `HostA`. Der erste Schirm wird empfangen und mit den Login-Daten abgeschickt. Bei positiver Quittung gibt die Funktion `true`, sonst `false` zurück.

Die Funktion `eintritt` gibt als Kommando `"GOTO mA1"` ein, um auf das Suchformat zu navigieren. Die als Parameter erhaltene Nummer wird in das Feld für Personalnummer eingetragen und die Suche durchgeführt. Falls kein Kommunikationsfehler aufgetreten ist, wird der Wert des Datumsfeldes zurückgegeben.

Die Funktionen realisieren eine bestimmte Fachlogik. Sie können, neben der Verwendung durch die WebTransactions-Integrations-Anwendung, auch lokal zum Abarbeiten bestimmter Aufgaben herangezogen werden.

WebTransactions-Integrations-Anwendung

Die WebTransactions-Integrations-Anwendung erzeugt die HTML-Oberfläche für den Browser. Die Daten ermittelt sie durch Zugriff auf die entfernten WebTransactions-Anwendungen. Der Zugriff wird über den HTTP-Host-Adapter mit Hilfe der Klasse `WT_RPC` abgewickelt.

Es werden zunächst zwei `WT_RPC`-Objekte `appA` und `appB` für die Verbindung zu den entfernten Host-Anwendungen angelegt. Für spätere Dialogschritte werden Referenzen im Systemobjekt hinterlegt. Die Methoden `logon` und `eintritt` werden mit Hilfe der Methode `addMethod` eingehängt. Nun kann z.B. über den Methodenaufruf `appA.logon` die entfernte Funktion ausgeführt werden.

Das abschließende HTML-Formular ermöglicht die Recherche in einer der beiden entfernten Applikationen:

```
<wtInclude name="wtRPC">
<wtOnCreateScript>
  if(! WT_SYSTEM._appA)
  {
    WT_SYSTEM._appA = appA =
      new WT_RPC('WebTaA/cgi-bin/WTPublish.exe', '/home/WTSrvA');
    appA.addMethod('logon', 'accHost');
    appA.addMethod('eintritt', 'accHost');
    WT_SYSTEM._appB = appB =
      new WT_RPC('WebTaB/scripts/WTPublish.exe', 'D:/WTSrvB');
    appB.addMethod('logon', 'accHost');
    appB.addMethod('eintritt', 'accHost');
    appA.logon('testuser', '123');
    appB.logon('testuser', '123');
  }
  else
  {
    appA = WT_SYSTEM._appA;
    appB = WT_SYSTEM._appB;
  }
</wtOnCreateScript>
Das Eintrittsdatum ist
##WT_POSTED.SEARCH == "A" ? appA.eintritt(WT_POSTED.PNUM)
                         : appB.eintritt(WT_POSTED.PNUM) #
<wtDataForm>
  Suche des Eintrittsdatums für die Personalnummer
  <INPUT TYPE="TEXT" NAME="PNUM"> in der Anwendung
  <INPUT TYPE="SUBMIT" NAME="SEARCH" VALUE="A"> oder
  <INPUT TYPE="SUBMIT" NAME="SEARCH" VALUE="B">
</wtDataForm>
```

6 Anhang: Die Schnittstelle WT_REMOTE

Dieser Anhang beschreibt die Schnittstelle WT_REMOTE. Mit WT_REMOTE steht eine Schnittstelle zwischen WebTransactions-Anwendungen und beliebigen Clients zur Verfügung, über die steuernd auf WebTransactions-Anwendungen zugegriffen werden kann.

Die Informationen in diesem Anhang sind speziell für den Zugriff von Fremd-Clients gedacht. Für die häufigsten Client-Anwendungen, Java-Applets und andere WebTransactions-Anwendungen, stehen vordefinierte Klassenbibliotheken zur Verfügung, wie etwa die WT-Klassen für Java-Clients und WT_RPC für WebTransactions-Clients. Auf diese Klassenbibliotheken gehen die entsprechenden Kapitel weiter vorne in diesem Handbuch näher ein.

6.1 Einführung

Bisher gab es nur zwei Wege zum Zugriff auf WebTransactions über einen Web-Browser: den synchronisierten und den nicht synchronisierten Dialog. Beide Methoden erzeugen HTML-Ausgaben, die vom Browser direkt angezeigt werden können.

Die Schnittstelle WT_REMOTE stellt eine neue Methode bereit, um über das Web Dienste einer WebTransactions-Sitzung anzufordern. Im Gegensatz zu den bisherigen Methoden besteht der Zweck der neuen Methode darin

- Dienste innerhalb einer WebTransactions-Sitzung aufzurufen
- Daten zu oder von einer WebTransactions-Sitzung zu übermitteln.

Die übermittelten Daten sind nicht auf HTML-Daten beschränkt, die vom Browser dargestellt werden können, sondern es können beliebige strukturierte Daten übermittelt werden, die in Form von XML-Dokumenten ausgetauscht werden.

Die folgenden Abschnitte beschreiben die Methoden der Schnittstelle WT_REMOTE, das Format der HTTP-Nachrichten, in denen diese codierten Methoden übermittelt werden müssen und schließlich der verwendeten XML-Dokumenttypen zur Codierung der Methoden innerhalb der HTTP-Nachrichten. Damit kann diese Schnittstelle nicht nur von den mitgelieferten Klassen für Java-Applets und WebTransactions-Clients, sondern auch von beliebigen anderen Anwendungen verwendet werden.

6.2 Methoden von WT_REMOTE

Dieser Abschnitt liefert einen Überblick über die verschiedenen Methoden, die WT_REMOTE anbietet. Die folgende Tabelle führt diese Methoden und deren Zweck auf:

Methode	Bedeutung
START_SESSION	eine neue Sitzung starten und die Sitzungsparameter für nachfolgende Anforderungen zurückliefern
EXIT_SESSION	Sitzung beenden
PROCESS_COMMANDS	Daten zu oder von einer WebTransactions-Sitzung übermitteln, Objekte erzeugen oder Methode aufrufen.

Tabelle 7: WT_REMOTE-Methoden

Beachten Sie, dass es keine WT_REMOTE-Methode für die Ausführung eines WTML-Dokuments gibt. WTML-Dokumente können über einen nicht synchronisierten Dialogschritt ausgeführt werden, indem das gewünschte Dokument als WT_ASYNC_PAGE angegeben wird. Solche asynchronen Aufrufe können beliebig mit WT_REMOTE-Aufrufen gemischt werden. Eine ausführliche Beschreibung, wie die HTTP-Nachrichten für die einzelnen Methoden aufgebaut sein müssen, finden Sie in den Abschnitten „Anforderungs-Nachrichten ohne Datenteil“ auf Seite 70“ und „Anforderungs-Nachrichten mit Steuer- und Datenteil“ auf Seite 72.

6.2.1 Methode START_SESSION

Diese Methode dient ausschließlich dazu, eine WebTransactions-Sitzung zu starten.

Diese Methode liefert als Antwort-Nachricht ein XML-Dokument, das die Sitzungs-Parameter für den nachfolgenden Zugriff auf diese Sitzung enthält (siehe auch Abschnitt „Antwort-Nachricht für START_SESSION“ auf Seite 90).

6.2.2 Methode EXIT_SESSION

Die Methode beendet die angegebene Sitzung und liefert als Antwort-Nachricht ein XML-Dokument mit einem leeren Element response als Bestätigung zurück (siehe auch Abschnitt „Antwort-Nachricht für EXIT_SESSION“ auf Seite 90). EXIT_SESSION kann nur auf eine laufende Sitzung angewendet werden, da der Steuerteil der Anforderungs-Nachricht Name/Value-Paare für die Sitzungs-ID und die Signatur enthalten muss. Ein Datenteil für diese Methode ist unnötig und wird daher ggf. ignoriert.

6.2.3 Methode PROCESS_COMMANDS

Diese Methode übermittelt Daten zu oder von einer WebTransactions-Sitzung, erzeugt durch einen Konstruktor-Aufruf ein Objekt in der WebTransactions-Sitzung oder ruft eine WTScript-Methode der WebTransactions-Sitzung auf. Dazu stehen verschiedene Aktionen zur Verfügung, die durch eine Anforderung ausgeführt werden können:

- `data`, `uploadData`
  Daten zur WebTransactions-Sitzung übermitteln
- `downloadData`
  Daten von der WebTransactions-Sitzung laden
- `createObject`
  Objekte in der WebTransactions-Sitzung erzeugen
- `callMethod`
  WTScript-Methoden der WebTransactions-Sitzung aufrufen

In einer Anforderungs-Nachricht mit der Methode `PROCESS_COMMANDS` können mehrere dieser Aktionen durchgeführt werden. Die Anforderungs-Nachricht kann sowohl an eine laufende Sitzung gerichtet sein, als auch in einer eigenen Sitzung ausgeführt werden, siehe hierzu auch folgenden Abschnitt.

Für jedes Element `downloadData`, `createObject` und `callMethod` in der Anforderungs-Nachricht wird im XML-Dokument der Antwort-Nachricht je ein Element `data` zurückgeliefert. Elemente `data` und `uploadData` in der Anforderungs-Nachricht werden in der Antwort-Nachricht nicht berücksichtigt. Außerdem kann die Antwort-Nachricht für jeden aufgetretenen Fehler ein Element `error` enthalten (siehe auch Abschnitt „Antwort-Nachrichten für PROCESS_COMMANDS“ auf Seite 91).

6.3 Einschritt- und Mehrschritt-Transaktionen

Über WT_REMOTE kann ein Client auf eine laufende WebTransactions-Sitzung zugreifen, indem er die Sitzungs-ID beim Client-Zugriff angibt. Ebenso ist es möglich, eine WebTransactions-Sitzung eigens durch den Client-Zugriff zu starten. Im ersten Fall handelt es sich immer um Mehrschritt-Transaktionen, im zweiten Fall sind sowohl Einschritt- als auch Mehrschritt-Transaktionen möglich.

6.3.1 Einschritt-Transaktionen

Bei einer Einschritt-Transaktion wird eine WebTransactions-Sitzung für die Ausführung einer Anforderungs-Nachricht gestartet, die Aktionen der Nachricht werden ausgeführt und die Sitzung danach wieder beendet. Dies alles geschieht über einen einzigen Client-Zugriff mit der Methode PROCESS_COMMANDS auf WT_REMOTE (siehe Abschnitt „Methode PROCESS_COMMANDS“ auf Seite 65).

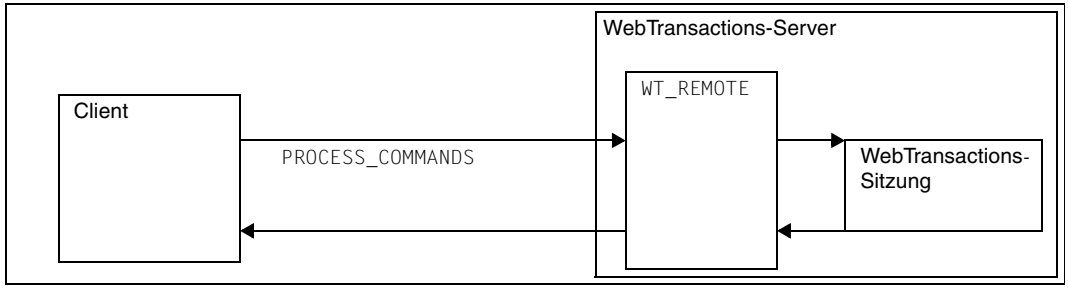

Bild 9: Einschritt-Transaktion

6.3.2 Mehrschritt-Transaktionen

Bei einer Mehrschritt-Transaktion gibt es zwei Möglichkeiten. Bei der ersten Möglichkeit wird eine WebTransactions-Sitzung durch den `WT_REMOTE`-Zugriff `START_SESSION` explizit gestartet, danach werden mehrere Client-Zugriffe mit `PROCESS_COMMANDS` durchgeführt und die Sitzung wird mit `EXIT_SESSION` schließlich beendet.

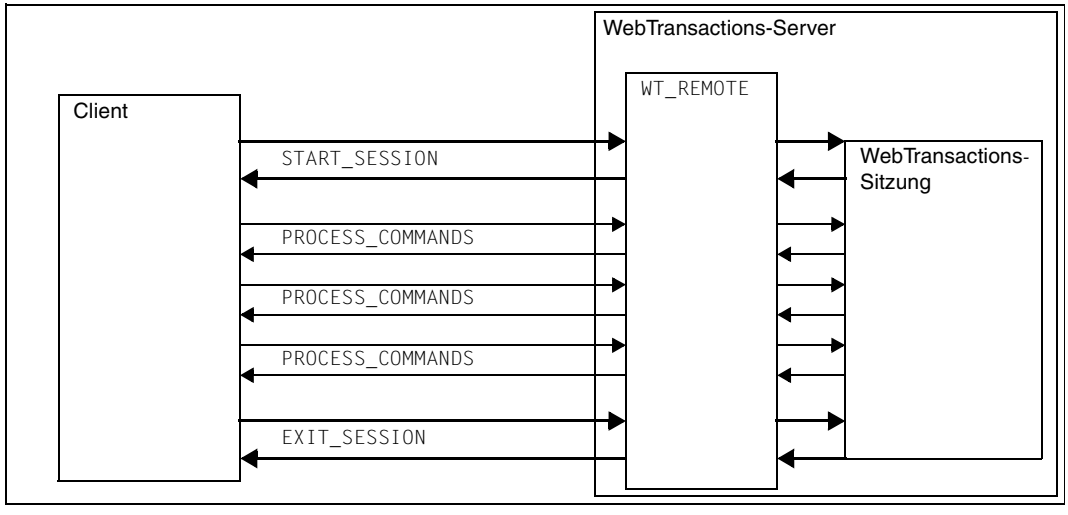

Bild 10: Mehrschritt-Transaktion (von WebTransactions gestartet)

Als zweite Möglichkeit kann ein Client eine bereits laufende WebTransactions-Sitzung ansprechen (z.B. ein Applet wird mit einer dynamischen Seite gestartet und klinkt sich in dieselbe Sitzung ein). Dies geschieht durch Angabe von `PROCESS_COMMANDS` mit den entsprechenden Sitzungs-Parametern.

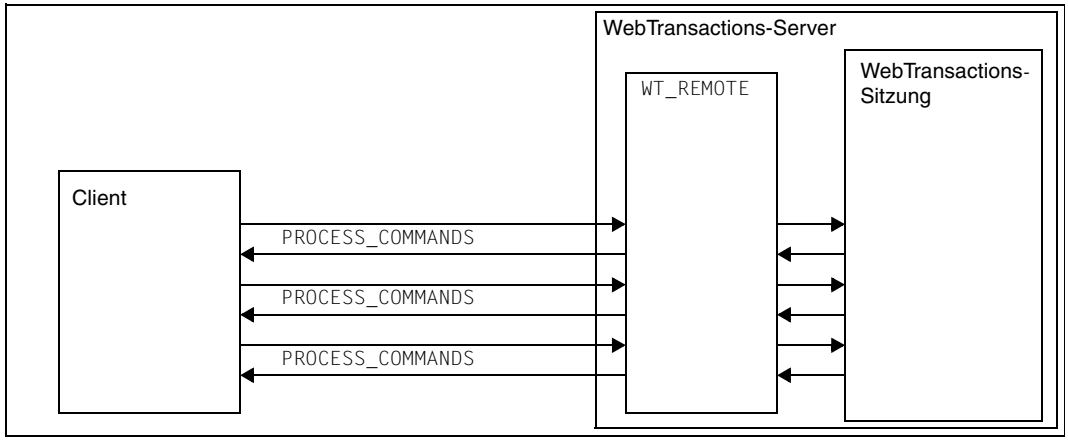

Bild 11: Mehrschritt-Transaktion (von anderem Client gestartet)

6.4 Aufbau der Anforderungs-Nachrichten für WT_REMOTE

Die Kommunikation zwischen dem Client und der WT_REMOTE-Schnittstelle besteht aus einer Folge von Anforderungs-Nachrichten des Client und Antwort-Nachrichten der WT_REMOTE-Schnittstelle. Die folgenden Abschnitte beschreiben den Aufbau dieser Nachrichten.

Die hier beschriebenen Anforderungen werden im WT_REMOTE-Kontext ausgeführt. Dies bedeutet, dass ein Satz globaler Variablen existiert, die ausschließlich für WT_REMOTE-Anforderungen verfügbar sind. Diese Variablen werden nicht automatisch gelöscht und stehen deshalb in mehreren Anforderungen zur Verfügung. Die Objekte WT_SYSTEM und WT_HOST werden gemeinsam mit den synchronen und asynchronen Zugriffen auf die entsprechende WebTransactions-Sitzung verwendet.

Die HTTP-Anforderungen an WebTransactions bestehen prinzipiell aus einem Steuerteil (control part) und einem optionalen Datenteil (data part). Sind beide Teile vorhanden, verwenden Sie eine HTTP-POST-Nachricht vom Mime-Typ multipart/mixed. Für die Methode PROCESS_COMMANDS sind die Nachrichten immer so aufgebaut. Fehlt der Datenteil (bei den Methoden START_SESSION und EXIT_SESSION), so kann eine einzelne POST-Nachricht mit dem Mime-Typ application/x-www-form-urlencoded oder eine GET-Nachricht verwendet werden.

Im Folgenden sind einige HTTP-Nachrichten dargestellt. Um die logische Struktur dieser Nachrichten aus HTTP-Header und Body deutlicher zu zeigen, wurde auf die Exaktheit einer hexadezimalen Darstellung verzichtet. Für das HTTP-Protokoll sind die Zeilenumbrüche wichtig:

– jede Zeile des Headers wird mit den Zeichen Carriage Return und Linefeed abgeschlossen (zwei Zeichen).

– den Abschluss des HTTP-Headers bildet eine zusätzliche Leerzeile (Carriage Return und Linefeed).

Deshalb werden die Zeilenumbruch-Zeichen explizit dargestellt als CR LF.

6.4.1 Anforderungs-Nachrichten ohne Datenteil

Der prinzipielle Aufbau einer POST-Nachricht bei fehlendem Datenteil sieht so aus:

```
POST /cgi-bin/WTPublish.exe HTTP/1.0 CR LF
Content-type: application/x-www-form-urlencoded CR LF
Content-length: Länge CR LF
CR LF
```

. . . Steuerteil der Nachricht . . .

Eine GET-Nachricht ist dagegen so aufgebaut:

```
http://server/cgi-bin/WTPublish.exe?<Steuerteil der Nachricht>
```

Dieser Aufbau soll jetzt noch anhand einiger Beispiele für die Methoden `START_SESSION` und `EXIT_SESSION` demonstriert werden.

Anforderungs-Nachrichten für START_SESSION

Der Pfad-Teil der URL für WebTransactions nach dem Programm `WTPublish` muss das Wort `startup` enthalten. Ein Aufruf dieser Methode könnte dann z.B. folgendermaßen aussehen:

- `WT_REMOTE`-Methode `START_SESSION` und HTTP-Methode `POST`:

  ```
  POST /cgi-bin/WTPublish.exe/startup HTTP/1.0 CR LF
  Content-type:application/x-www-form-urlencoded CR LF
  Content-length:60 CR LF
  CR LF
  WT_REMOTE=START_SESSION&WT_SYSTEM_BASEDIR=basisverzeichnis
  ```

  Die Länge 60 in `Content-length` ist hier ein Beispiel. Sie hängt von der konkreten Länge der Nachricht ab und wird durch die Angaben für Basisverzeichnis, Sitzungs ID etc. beeinflusst. Dies gilt auch für die weiteren Beispiele.

- `WT_REMOTE`-Methode `START_SESSION` und HTTP-Methode `GET`:

  ```
  http://server/cgi-bin/WTPublish.exe/startup?\
  WT_REMOTE=START_SESSION&WT_SYSTEM_BASEDIR=basisverzeichnis
  ```

Anforderungs-Nachrichten für EXIT_SESSION

- WT_REMOTE-Methode EXIT_SESSION und HTTP-Methode POST:

```
POST /cgi-bin/WTPublish.exe HTTP/1.0 CR LF
Content-type:application/x-www-form-urlencoded CR LF
Content-length:130 CR LF
CR LF
WT_REMOTE=EXIT_SESSION&WT_SYSTEM_BASEDIR=basisverzeichnis&\
WT_SYSTEM_SESSION=sitzungsID&WT_SYSTEM_SIGNATURE=signatur
```

- WT_REMOTE-Methode EXIT_SESSION und HTTP-Methode GET:

```
http://server/cgi-bin/WTPublish.exe?\
WT_REMOTE=EXIT_SESSION&WT_SYSTEM_BASEDIR=basisverzeichnis&\
WT_SYSTEM_SESSION=sitzungsID&WT_SYSTEM_SIGNATURE=signatur
```

Eine ausführliche Beschreibung des Steuerteils der Nachricht finden Sie im Abschnitt „Steuerteil der HTTP-Nachricht“ auf Seite 73.

6.4.2 Anforderungs-Nachrichten mit Steuer- und Datenteil

Diese Form von Anforderungs-Nachrichten wird für die `WT_REMOTE`-Methode `PROCESS_COMMANDS` verwendet, da diese Methode immer Parameter erfordert, die im Datenteil angegeben werden müssen.

Der Steuerteil hat den Mime-Typ `application/x-www-form-urlencoded` und gibt die ausgewählte WebTransactions-Sitzung an. Der Datenteil ist vom Mime-Typ `text/xml` und enthält die Daten, die in der aktuellen Anforderung bearbeitet werden sollen.

Da sowohl der Steuer- als auch der Datenteil keine beliebigen Werte annehmen können, wird die Nachrichten-Begrenzung (boundary) mit der Zeichenkette `<<'<<"<<42>>">>'>>` fest definiert. Der prinzipielle Aufbau einer solchen HTTP-Nachricht sieht also so aus:

```
POST /cgi-bin/WTPublish exe HTTP/1.0 CR LF
Content-type: multipart/mixed; boundary=<<'<<"<<42>>">>'>> CR LF
Content-length: 270 CR LF
CR LF
--<<'<<"<<42>>">>'>> CR LF
Content-type: application/x-www-form-urlencoded CR LF
CR LF
. . . Steuerteil der Nachricht . . . CR LF

--<<'<<"<<42>>">>'>> CR LF
Content-type: text/xml CR LF
CR LF
. . . Datenteil der Nachricht. . .
```

Dieser Aufbau einer Nachricht beschreibt eine Mehrschritt-Transaktion. Bei einer Einschritt-Transaktion, bei der die Methode `PROCESS_COMMANDS` eine eigene WebTransactions-Sitzung nur für die Ausführung der `WT_REMOTE`-Methode erzeugt, muss nach dem Programm `WTPublish` das Schlüsselwort `startup` mit angegeben werden:

```
POST /cgi-bin/WTPublish exe/startup HTTP/1.0 CR LF
Content-type: multipart/mixed; boundary=<<'<<"<<42>>">>'>> CR LF
Content-length: 270 CR LF
CR LF
--<<'<<"<<42>>">>'>> CR LF
Content-type: application/x-www-form-urlencoded CR LF
CR LF
. . . Steuerteil der Nachricht . . . CR LF

--<<'<<"<<42>>">>'>> CR LF
Content-type: text/xml CR LF
CR LF
. . . Datenteil der Nachricht . . .
```

6.4.3 Steuerteil der HTTP-Nachricht

Der Steuerteil der HTTP-Nachricht (Mime-Typ `application/x-www-form-urlencoded`) enthält Informationen darüber, welche WebTransactions-Anwendung angesprochen (Basisverzeichnis) und welche `WT_REMOTE`-Methode ausgeführt werden soll (`START_SESSION`, `EXIT_SESSION` oder `PROCESS_COMMANDS`, siehe auch Abschnitt „Methoden von WT_REMOTE“ auf Seite 64). Zusätzlich kann er weitere Informationen enthalten, die denen einer herkömmlichen, lokalen WebTransactions-Sitzung entsprechen:

– Anwendungs-Timeout
– Benutzer-Timeout
– Sprache
– Stil

Wird eine bereits laufende Sitzung angesprochen, d.h. bei einer Mehrschritt-Transaktion mit `PROCESS_COMMANDS` ober bei `EXIT_SESSION`, braucht der Steuerteil zusätzlich:

– Sitzungs-ID
– Signatur

Diese Informationen sind als Name/Value-Paare der Form *Name*=*Value* codiert und durch das Zeichen `&` verbunden. Die Syntax ist folgendermaßen:

```
  WT_SYSTEM_BASEDIR=basisverzeichnis
 &WT_REMOTE={START_SESSION|EXIT_SESSION|PROCESS_COMMANDS}
[&WT_SYSTEM_TIMEOUT_APPLICATION=timeoutAnwendung]
[&WT_SYSTEM_TIMEOUT_USER=timeoutBenutzer]
[&WT_SYSTEM_LANGUAGE=sprache]
[&WT_SYSTEM_STYLE=stil]
[&WT_SYSTEM_SESSION=sitzungsID
 &WT_SYSTEM_SIGNATURE=signatur]
```

Die Reihenfolge der Name/Value-Paare innerhalb des Nachrichten-Körpers ist beliebig.

Beispiel

```
POST /cgi-bin/WTPublish exe HTTP/1.0 CR LF
Content-type: multipart/mixed; boundary=<<'<<"<<42>>">>'>> CR LF
Content-length: 270 CR LF
CR LF
--<<'<<"<<42>>">>'>> CR LF
Content-type: application/x-www-form-urlencoded CR LF
CR LF
WT_SYSTEM_BASEDIR=basisverzeichnis&WT_REMOTE=PROCESS_COMMANDS\
&WT_SYSTEM_SESSION=sitzungsID&WT_SYSTEM_SIGNATURE=signatur CR LF
--<<'<<"<<42>>">>'>> CR LF
Content-type: text/xml CR LF
CR LF
. . . Datenteil der Nachricht . . .
```

6.4.4 Datenteil der HTTP-Nachricht

Der Datenteil der HTTP-Nachricht ist, falls vorhanden, vom Mime-Typ `text/xml` und enthält nähere Informationen darüber, wie die angegebene `WT_REMOTE`-Methode auszuführen ist. Er enthält sozusagen die Parameter der `WT_REMOTE`-Methode. Diese sind als XML-Dokument codiert (zur Syntax dieser XML-Dokumente siehe folgenden Abschnitt „XML-Dokumente für Anforderungs-Nachrichten“ auf Seite 76).

Beispiel 1

Im folgenden Beispiel wird eine vollständige HTTP-Nachricht angegeben, die in einer WebTransactions-Sitzung die Funktion `eval` aufruft. In diesem Fall richtet sich die Nachricht an eine bereits laufende Sitzung, das heißt, die Parameter `SESSION` und `SIGNATURE` müssen mit angegeben werden:

```
POST /cgi-bin/WTPublish exe HTTP/1.0 CR LF
Content-type: multipart/mixed; boundary=<<'<<"<<42>>">>'>> CR LF
Content-length: 270 CR LF
CR LF
--<<'<<"<<42>>">>'>> CR LF
Content-type: application/x-www-form-urlencoded CR LF
CR LF
WT_SYSTEM_BASEDIR=basisverzeichnis&WT_REMOTE=PROCESS_COMMANDS\
&WT_SYSTEM_SESSION=sitzungsID&WT_SYSTEM_SIGNATURE=signatur CR LF
--<<'<<"<<42>>">>'>> CR LF
Content-type: text/xml CR LF
CR LF
<request>
<callMethod name="eval">
<string name="0">2*21</string>
</callMethod>
</request>
```

Beispiel 2

In diesem Beispiel wird die Anforderung aus Beispiel 1 als Einschritt-Transaktion formuliert. Das heißt, für die Ausführung der Funktion wird eine eigene WebTransactions-Sitzung gestartet:

```
POST /cgi-bin/WTPublish exe/startup HTTP/1.0 CR LF
Content-type: multipart/mixed; boundary=<<'<<"<<42>>">>'>> CR LF
Content-length: 270 CR LF
CR LF
--<<'<<"<<42>>">>'>> CR LF
Content-type: application/x-www-form-urlencoded CR LF
CR LF
WT_SYSTEM_BASEDIR=basisverzeichnis&WT_REMOTE=PROCESS_COMMANDS CR LF
--<<'<<"<<42>>">>'>> CR LF
Content-type: text/xml CR LF
CR LF
<request>
<callMethod name="eval">
<string name="0">2*21</string>
</callMethod>
</request>
```

6.5 XML-Dokumente für Anforderungs-Nachrichten

Dieser Abschnitt beschreibt den Aufbau der XML-Dokumente in Anforderungs-Nachrichten, d.h. den Datenteil von mehrteiligen Anforderungs-Nachrichten.

Die folgenden Abschnitte verwenden DTDs (Document Type Definitions) zur Beschreibung der erlaubten Syntax dieser XML-Dokumente. Daher soll zunächst der Aufbau solcher DTDs an einem Beispiel kurz erläutert werden:

```
<!ELEMENT callMethod                    ((undefined | number | boolean |
                                        string | object | function)*)>
<!ELEMENT number                        (#PCDATA)>
<!ATTLIST callMethod    name        CDATA  #REQUIRED
                        codeBase    CDATA  #IMPLIED>
<!ATTLIST number        name        CDATA  #REQUIRED>
```

Diese vier Einträge einer DTD beschreiben den Aufbau zweier Elemente, nämlich der Elemente `callMethod` und `number`. Die Zeile `<!ELEMENT callMethod ...>` legt fest, dass das Element `callMethod` beliebig viele (`*`) Unterelemente enthalten kann, nämlich die Unterelemente `undefined`, `number`, `boolean`, `string`, `object` oder `function`. Die Zeile `<!ATTLIST callMethod ...>` legt dagegen fest, welche Attribute ein Element `callMethod` haben muss (`#REQUIRED`) oder haben kann (`#IMPLIED`). Im Fall von `callMethod` muss das Attribut `name` angegeben werden, das Attribut `codeBase` dagegen ist optional. Das Element `number` muss ein Attribut `name` und freien Text (`#PCDATA`) enthalten, der den Wert des Elements angibt.

Beispiel

```
<callMethod   name="eval">
   <string   name="0">
      2*21
   </string>
</callMethod>
```

Dieses XML-Dokument genügt der Syntax, die mit obiger DTD beschrieben wird. Es beschreibt den Aufruf der Methode `eval` mit einem Parameter. Der Name des Parameters ist `"0"`, d.h. es ist der erste Parameter. Der Parameter hat den Wert `"2*21"`.

6.5.1 Der Aufbau des XML-Dokuments (DTDrequest)

Der Datenteil einer Anforderungs-Nachricht besteht aus einem XML-Dokument, das gemäß der folgenden DTD aufgebaut ist, die im weiteren Text unter der Bezeichnung `DTDrequest` angesprochen wird:

```
<!ELEMENT request                          ((data | uploadData | downloadData
                                           |createObject | callMethod)*)>
<!ELEMENT data                             ((undefined | number | boolean |
                                           string | object | function)*)>
<!ELEMENT uploadData                       ((undefined | number | boolean |
                                           string | object | function)*)>
<!ELEMENT downloadData                     EMPTY>
<!ELEMENT createObject                     ((undefined | number | boolean |
                                           string | object | function)*)>
<!ELEMENT callMethod                       ((undefined | number | boolean |
                                           string | object | function)*)>
<!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 downloadData  name        CDATA  #REQUIRED>
<!ATTLIST createObject  name        CDATA  #REQUIRED
                        constructor CDATA  #REQUIRED
                        codeBase    CDATA  #IMPLIED>
<!ATTLIST callMethod    name        CDATA  #REQUIRED
                        codeBase    CDATA  #IMPLIED>
<!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>
```

Das Wurzelelement `request` des XML-Dokuments umschließt die Elemente, die die Aktionen definieren:

`data`	Daten zu einer WebTransactions-Anwendung übertragen
`uploadData`	Daten zu einer WebTransactions-Anwendung übertragen
`downloadData`	Daten von einer WebTransactions-Anwendung laden
`createObject`	einen Konstruktor in einer WebTransactions-Anwendung aufrufen
`callMethod`	eine Methode in einer WebTransactions-Anwendung aufrufen

Soll in einer HTTP-Nachricht nur eine dieser Aktionen ausgeführt werden, kann das betreffende Element auch selbst als Wurzelelement des XML-Dokuments verwendet werden ohne dass das Element `request` angegeben wird.

Beispiel

In diesem Beispiel enthält das Element `request` drei Unterelemente `data`, `downloadData` und `callMethod`.

```
<request>
   <data>
      <number name="answer">42</number>
   </data>
   <downloadData    name="WT_HOST"/>
   <callMethod   name="myMethod" codeBase="myTemplate.htm">
      <number   name="0">
         42
      </number>
      <number   name="1">
         24
      </number>
   </callMethod>
</request>
```

Der Aufbau der möglichen Unterelemente von `request` ist in den folgenden Abschnitten näher beschrieben.

6.5.2 Aufbau der Elemente data und uploadData (DTDdata)

Die Elemente `data` und `uploadData` und ihre Unterstrukturen sind äquivalente Elemente für das Übertragen von Daten an den Datenbereich der fernen WebTransactions-Sitzung. Die Methode arbeitet additiv, d.h. falls ein Objekt noch nicht existiert, wird es neu erzeugt, existiert es bereits, so werden zusätzliche Attribute erzeugt, bzw. bestehende Attribute verändert. Die Syntax beider Elemente ist identisch und entspricht der folgenden DTD (nachfolgende `DTDdata` bezeichnet):

```
<!ELEMENT data                          ((undefined | number | boolean |
                                        string | object | function)*)>
<!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>
```

Das Wurzelelement `data` (oder `uploadData`) dieser DTD enthält eine beliebige Folge der Elemente `undefined`, `number`, `boolean`, `string`, `object` und `function`. Diese Element-Typen entsprechen den gleichnamigen Datentypen von WTScript. Jedes Element verfügt über ein Pflicht-Attribut `name`, das den Namen der Variablen oder des Variablenteils festlegt.

Die Elemente `undefined` und `function` sind immer leer. Die Elemente `number`, `boolean` und `string` enthalten `PCDATA`- (parsed character data) Zeichenketten. In diesen `PCDATA`-Teilen wird der Wert des Elements als Text angegeben, d.h. für `number`-Werte als numerische Literale (z.B. `-0.717273E-42`), für `boolean`-Werte als die Literale `true` und `false` und für `string`-Werte als beliebige Zeichenketten.

Das Element `object` kann ein Attribut `class` besitzen, das die Klasse des dargestellten Objekts angibt. In diesem Fall beschreibt das Element die vollständige Struktur des Objekts. Außerdem kann es als erstes Unterelement einen `PCDATA`-Teil besitzen, der den Wert eines `Number`-, `Boolean`- oder `String`-Objekts angibt. Dann muss dieser Text, wie bei den einfachen Datentypen, ein passendes Literal sein.

Im Anschluss an den `PCDATA`-Teil kann das Element `object` eine beliebige Folge von verschachtelten Elementen besitzen, die Attribute oder Methoden des Objekts darstellen.

Wurde ein Objekt bereits innerhalb des Dokuments beschrieben und tritt jetzt eine weitere Referenz innerhalb dieses Dokuments auf dieses Objekt auf, dann wird dies durch das Attribut `reference` angezeigt. Das Attribut enthält den absoluten Namen des referenzierten Objekts. Die enthaltenen Attribute sind identisch mit denen des referenzierten Objekts und werden im XML-Dokument nicht ein zweites Mal aufgeführt.

Das folgende Beispiel zeigt einen Ausschnitt aus `WT_SYSTEM` und dessen Darstellung in `DTDdata`:

```
WT_SYSTEM   (type object, class Object)
   BASEDIR    (type string, value "/home/WebTA")
   CGI      (type object, class Object)
      HTTP_USER_AGENT   (type string, value "Mozilla/4.0")
      REMOTE_HOST   (type string, value "pcfritz")
   FORMAT   (type string, value "wtstart")
   _myArray   (type object, class Array)
      0      (type string, value "the answer is ")
      1      (type number, value 42)
      2      (type boolean, value true)
      att   (type string, value "problem is the question")
      top   (type object, class Object, reference to WT_SYSTEM.CGI)
   _myStringObject   (type object, class String, value "another string")
      att1   (type object, class Boolean, value false)
      att2   (type boolean, value true)
   _myMethod   (type function)
```

Wird diese Datenstruktur entsprechend der `DTDdata` als XML-Dokument dargestellt, so sieht das folgendermaßen aus (die Zeilenumbrüche und Einrückungen entsprechen nicht der tatsächlichen Umsetzung, sondern wurden nur aus Gründen der Lesbarkeit eingefügt):

```
<data>
   <object   name="WT_SYSTEM"   class="Object" >
      <string   name="BASEDIR">
         /home/WebTA
      </string>
      <object   name="CGI"   class="Object">
         <string   name="HTTP_USER_AGENT">
           Mozilla/4.0
```

```
            </string>
            <string   name="REMOTE_HOST">
              pcfritz
            </string>
         </object>
         <string   name="FORMAT">
            wtstart
         </string>
         <object   name="_myArray" class="Array">
            <string   name="0">
              the answer is
            </string>
            <number   name="1">
              42
            </number>
            <boolean   name="2">
              true
            </boolean>
            <string   name="att">
              problem is the question
            </string>
            <object   name="top" reference="WT_SYSTEM.CGI"/>
         </object>
         <object   name="_myStringObject"    class="String">
            another string
            <object   name="att1"   class="Boolean">
              false
            </object>
            <boolean   name="att2">
              true
            </boolean>
         </object>
         <function   name="_myMethod"/>
      </object>
   </data>
```

`uploadData` (oder `data`) überträgt die angegebenen Daten in die angesprochene Web-Transactions-Sitzung. Im Beispiel werden die angegebenen Werte und Attribute von `WT_SYSTEM` übertragen.

6.5.3 Aufbau des Elements downloadData (DTDdownload)

Das Element `downloadData` gibt ein oder mehrere Objekte an, die von der fernen WebTransactions-Sitzung herunterzuladen sind. Die gewünschten Objekte werden in der Antwort zurückgegeben. Das Element ist entsprechend der folgenden DTD aufgebaut, die ab jetzt mit dem Namen `DTDdownload` bezeichnet wird:

```
<!ELEMENT donwloadData                    EMPTY>
<!ATTLIST downloadData    name    CDATA   #REQUIRED>
```

Das Element `downloadData` verfügt über ein Attribut `name`, das die Namen der zu ladenden Objekte oder Attribute entsprechend der folgenden Syntax angibt:

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.
object1|*object2* *object..wert1*|*wert2*	Mehrere Objekte oder mehrere Attribute unterhalb eines Objekts. Die Objekte *object1* und *object2* bzw. alle Attribute mit den Namen *wert1* oder *wert2*, die in Objekten direkt unterhalb des Datenobjekts *object* enthalten sind. Das Zeichen `|` bindet stärker als das Zeichen `.`, sodass folgendes Beispiel gilt: `WT_HOST|WT_SYSTEM.xyz` liefert `WT_HOST.xyz` **und** `WT_SYSTEM.xyz`

Beispiel

Dieses Beispiel fragt die drei Attribute `BASEDIR`, `SESSION` und `SIGNATURE` der Sitzung ab:

```
<request>
   <downloadData name="WT_SYSTEM.BASEDIR|SESSION|SIGNATURE"/>
</request>
```

Als Antwort auf diese Anforderung könnte dann z.B. folgende Antwort-Nachricht gemäß der DTD `DTDresponse` zurückgeliefert werden (siehe auch Abschnitt „XML-Dokumente in Antwort-Nachrichten“ auf Seite 89):

```
<response>
<data>
   <object name="WT_SYSTEM" class="Object">
      <string name="BASEDIR">C:/basedirs/context
      </string>
      <string name="SESSION">E-935516492-8494
      </string>
      <string name="SIGNATURE">159177851380710585
      </string>
   </object>
</data>
</response>
```

6.5.4 Aufbau des Elements callMethod (DTDmethod)

Das Element `callMethod` gibt die notwendigen Informationen zum Aufruf einer Methode der fernen WebTransactions-Sitzung an. Die entsprechende Methode wird ausgeführt und das Ergebnis der Funktion zurückgegeben. Das Element ist entsprechend der folgenden DTD aufgebaut, die ab jetzt mit dem Namen `DTDmethod` bezeichnet wird:

```
<!ELEMENT callMethod                    ((undefined | number | boolean |
                                        string | object | function)*)>

<!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 callMethod  name        CDATA  #REQUIRED
                      codeBase    CDATA  #IMPLIED>

<!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>
```

Das Wurzelelement `callMethod` besitzt zwei Attribute: `name` und `codeBase`. Das Attribut `name` muss angegeben werden und den absoluten Namen der auszuführenden Methode festlegen. Für eine globale Funktion wird ein einfacher Bezeichner angegeben, eine objekt-spezifische Methode wird durch die Punktnotation dargestellt.

Das Attribut `codeBase` ist optional und kann dazu verwendet werden, einen Template-Namen anzugeben. WebTransactions sucht dann entsprechend der Standard-Suchreihenfolge nach dem Template mit diesem Namen (die Standard-Suchreihenfolge ist im WebTransactions-Handbuch „Konzepte und Funktionen“ beschrieben). Ist die Methode dem System bereits bekannt (z.B. eine eingebaute Funktion oder eine Methode eines Objekts), dann kann dieses Attribut weggelassen werden.

Die Elemente innerhalb des Wurzelelements stellen die Parameter der aufzurufenden Methode dar. Sie können von jedem beliebigen Typ sein: `undefined`, `number`, `boolean`, `string`, `object` oder `function`.

Das jeweilige Attribut `name` dieser Elemente auf der obersten Ebene wird auf eine ganze Zahl gesetzt, die den Index des jeweiligen Parameters in der Parameterliste angibt (also `'0'` für den ersten, `'1'` für den zweiten Parameter usw.). Dieser Name wird für Verweise auf diese Objekte innerhalb der Parameter verwendet. Diese Parameter sind innerhalb des aufrufenden XML-Dokuments definiert, die Parameter-Übergabe an die Methode erfolgt „by value“, d.h. Änderungen an den Parametern innerhalb der Methode haben im aufrufenden XML-Dokument keine Wirkung.

Für tiefere Ebenen der Parameter-Definition muss jeweils ein Attribut `name` angegeben werden, damit wohlgeformte Strukturen benannter Attribute aufgebaut werden können.

Die Definition von Werten und Klassen entspricht dabei der aus der DTD `DTDdata`, die im Abschnitt „Aufbau der Elemente data und uploadData (DTDdata)“ auf Seite 79 beschrieben ist.

Beispiele:

- Aufruf einer globalen Funktion z.B. `eval`:

  ```
  <callMethod name="eval">...</callMethod>
  ```

- Aufruf einer selbst geschriebenen Funktion `myFunction`. Hier muss das Template (im Beispiel `MyFunctions.htm`) mit angegeben werden, in dem die Funktion steht:

  ```
  <callMethod name="myFunction" codeBase="MyFunctions.htm">
     ...
  </callMethod>
  ```

- Aufruf einer objektspezifischen Methode für ein Objekt der eingebauten Klassen; der Name der gerufenen Methode muss in dem Fall auch den Namen des rufenden Objekts enthalten.

  `myComm` ist ein Kommunikationsobjekt, also von der Klasse `WT_Communication`

  ```
  <callMethod name="WT_HOST.myComm.send"/>
  ```

- Aufruf einer selbstdefinierten Methode `myMethod` an einer selbstdefinierten Klasse. Hier muss das Template (im Beispiel: `MyMethods.htm`) mit angegeben werden, in dem die Methode steht:

  `myObject` ist ein Objekt der selbstdefinierten Klasse

  ```
  <callMethod name="myObject.myMethod" codeBase="MyMethods.htm">
     ...
  </callMethod>
  ```

- Das folgende Beispiel zeigt den einfachen Aufruf einer globalen Funktion mit einem Parameter:

```
<request>
   <callMethod name="eval">
      <string name="0">2*21</string>
   </callMethod>
</request>
```

Und hier die Antwort-Nachricht:

```
<response>
   <data>
      <number name="">42
      </number>
   </data>
</response>
```

6.5.5 Aufbau des Elements createObject (DTDcreate)

Das Element `createObject` erzeugt einen Konstruktor-Aufruf in der angesprochenen fernen WebTransactions-Sitzung. Das Element ist entsprechend der folgenden DTD aufgebaut, die ab jetzt mit dem Namen `DTDcreate` bezeichnet wird:

```
<!ELEMENT createObject                    ((undefined | number | boolean |
                                          string | object | function)*)>
<!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 createObject  name        CDATA  #REQUIRED
                        constructor CDATA  #REQUIRED
                        codeBase    CDATA  #IMPLIED>
<!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>
```

Das Attribut `name` diese Elements bezeichnet den Namen des neuen Objekts. Das zusätzlich verfügbare Attribut `constructor` des Elements gibt die gewünschte Konstruktorfunktion an. Schließlich kann das optionale Attribut `codeBase` dazu verwendet werden, ein WTML-Dokument der fernen WebTransactions-Sitzung anzugeben, dessen Funktionen vor der Ausführung bereitgestellt werden. In diesem Dokument sind dann der Konstruktor und die Methoden der Klasse definiert.

```
<createObject name="myObj" constructor="myCons" codeBase="remoteTest.htm">
    Argumente des Konstruktors
    ...
</createObject>
```

Die Elemente innerhalb von `createObject` geben die Argumente des Konstruktors an. Als Ergebnis wird das erzeugte Objekt als XML-Dokument zurückgegeben (siehe Abschnitt „XML-Dokumente in Antwort-Nachrichten“ auf Seite 89).

Beispiel

Das folgende Beispiel erzeugt ein Objekt vom Typ `Date` mit dem Wert 31.12.1999:

```
<createObject name="myDate" constructor="Date">
<number   name="0">
   1999
</number>
<number   name="1">
   11
</number>
<number   name="2">
   31
</number>
</createObject>
```

Die Antwort-Nachricht (siehe folgenden Abschnitt) sähe dann z.B. so aus:

```
<response>
<data>
   <object name="myDate" class="Date">
   </object>
</data>
</response>
```

6.6 XML-Dokumente in Antwort-Nachrichten

Alle Antwort-Nachrichten auf `WT_REMOTE`-Anforderungen besitzen, außer bei Verbindungsfehlern, immer das Mime-Format `text/xml`. Die gelieferten XML-Dokumente entsprechen der folgenden DTD (`DTDresponse`):

```
<!ELEMENT response                      (data* error*)>
<!ELEMENT data                          ((undefined | number | boolean | string
                                        | object | function)*)>
<!ELEMENT undefined                     EMPTY>
<!ELEMENT number                        (#PCDATA)>
<!ELEMENT boolean                       (#PCDATA)>
<!ELEMENT string                        (#PCDATA)>
<!ELEMENT object                        (#PCDATA? (undefined | number | boolean
                                        |string | object | function)*)>
<!ELEMENT function                      EMPTY>
<!ELEMENT error                         (#PCDATA?)>
<!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>
<!ATTLIST error       document   CDATA  #IMPLIED
                      line       CDATA  #IMPLIED
                      column     CDATA  #IMPLIED
                      message    CDATA  #REQUIRED>
```

Das Wurzelelement kann eine Folge von Elementen `data` und eine Folge von Elementen `error` enthalten. Das Element `data` entspricht im Aufbau der `DTDdata`, die im Abschnitt „Aufbau der Elemente data und uploadData (DTDdata)“ auf Seite 79 beschrieben ist. Mit diesem Element wird das Ergebnis eines Download, der Erzeugung eines Objekts oder des Aufrufs einer Methode zurückgeliefert.

Treten während der Bearbeitung in der fernen WebTransactions-Sitzung Fehler auf, so werden die Informationen über jeden Fehler als Elemente `error` an den Client übertragen. Diese Elemente `error` können folgende Informationen enthalten, sofern diese ermittelt werden können:

– Dokument (Attribut `document`)
– Zeile (Attribut `line`)
– Spalte (Attribut `column`)

Die Fehlernummer wird immer im Attribut `message` zurückgeliefert, der Fehlertext ist als `PCDATA` im Element `error` enthalten (siehe auch zweites Beispiel im Abschnitt „Antwort-Nachrichten für PROCESS_COMMANDS“ auf Seite 91).

6.6.1 Antwort-Nachricht für START_SESSION

Das XML-Dokument, das als Antwort auf `START_SESSION` zurückgeliefert wird, hat die folgende Form:

```
<response>
   <data>
      <object name="WT_SYSTEM" class="Object">
         <string name="SESSION">sessionID</string>
         <string name="SIGNATURE">signatur</string>
         <string name="BASEDIR">basisverzeichnis</string>
      </object>
   </data>
</response>
```

6.6.2 Antwort-Nachricht für EXIT_SESSION

Die Antwort-Nachricht für `EXIT_SESSION` besteht aus einem leeren Element `response` als Bestätigung

```
<response/>
```

6.6.3 Antwort-Nachrichten für PROCESS_COMMANDS

Für jede Aktion `downloadData`, `createObject` und `callMethod` in der Anforderungs-Nachricht wird ein Element `data` in der Antwort-Nachricht zurückgeliefert. Die Aktionen `data` und `uploadData` in der Anforderungs-Nachricht werden in der Antwort-Nachricht nicht berücksichtigt (siehe folgendes erstes Beispiel). Außerdem kann die Antwort-Nachricht für jeden aufgetreten Fehler ein Element `error` enthalten (zweites Beispiel).

Beispiele

Die folgenden Beispiele sollen das Zusammenspiel von Anforderungs- und Antwort-Nachricht erläutern, zusätzlich zu den bereits bei den Anforderungs-Nachrichten aufgeführten Beispielen.

- Dies ist ein Beispiel für mehrere Anforderungen in einer Nachricht. Dabei ist eine der Methoden `uploadData`, die keine Anwort erzeugt:

```
<request>
   <createObject name="NewString" constructor="String">
      <string name="0">hallo welt</string>
   </createObject>
   <uploadData>
      <string name="NewString">Hello world</string>
   </uploadData>
   <downloadData name="NewString"/>
</request>
```

  Die zugehörige Antwort-Nachricht sieht dann z.B. so aus:

```
<response>
   //Antwort auf createObject
   <data>
      <object name="NewString" class="String">hallo welt
      </object>
   </data>
   //Antwort auf downloadData
   <data>
      <string name="NewString">Hello world
      </string>
   </data>
</response>
```

- Dies ist ein Beispiel für eine Fehlermeldung im Element `error`, hier z.B. ein Schreibfehler beim Funktionsnamen:

```
<request>
   <callMethod name="evla">
      <string name="0">2*21</string>
   </callMethod>
</request>
```

Die entsprechende Antwort-Nachricht:

```
<response>
   <data>
      <undefined name=""/>
   </data>
   <error message="216">Error: unknown function - function "evla" not
   defined or not of type function; correct WTML document.
   </error>
</response>
```

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

WebTransactions-Handbücher

Unter der Web-Adresse *http://manuals.ts.fujitsu.com* stehen Ihnen sämtliche Handbücher zum Download zur Verfügung.

WebTransactions
Konzepte und Funktionen
Einführung

WebTransactions
Template-Sprache
Referenzhandbuch

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

Stichwörter