Benutzerhandbuch - Deutsch

FUJITSU

WebTransactions V7.5

Zugriff auf dynamische Web-Inhalte

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 von WebTransactions für HTTP

Folgende Abbildung zeigt die Architektur von WebTransactions für HTTP:

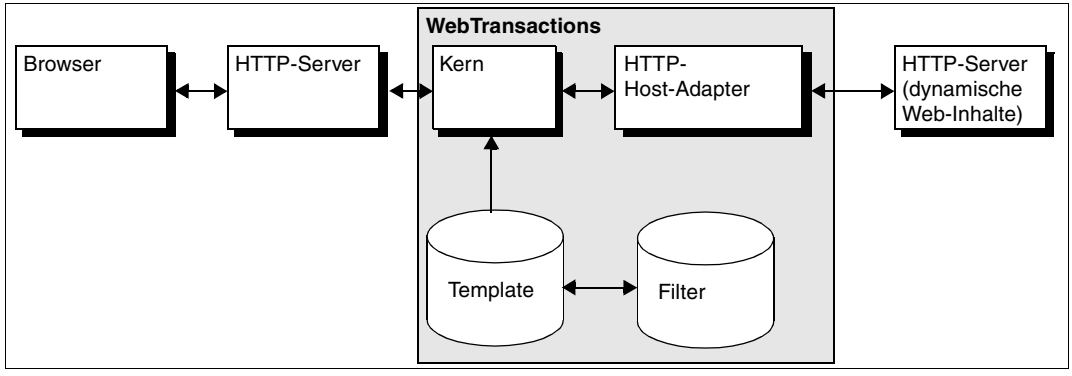


Bild 1: Architektur von WebTransactions für HTTP

HTTP-Host-Adapter

WebTransactions benutzt den HTTP-Host-Adapter, um die Kommunikation zwischen dem Kern von WebTransactions und einem beliebigen HTTP-Server abzuwickeln und auf dynamische Web-Inhalte zuzugreifen.

Der HTTP-Host-Adapter unterstützt den Transport über das HTTP-Protokoll. Die Hostobjekte entsprechen dem einfachen Objektmodell dieses Protokolls mit Nachrichtenkopf und Nachrichtenkörper. Eine weitergehende Interpretation des Nachrichteninhalts, der z.B. in HTML oder in XML kodiert sein kann, wird von diesem Host-Adapter nicht durchgeführt.

Filter

Um die empfangenen Nachrichten zu interpretieren bzw. WebTransactions-Informationen in HTTP-Nachrichten umzuwandeln müssen individuelle Filter eingesetzt werden. Einige Filter werden mitgeliefert (z.B. HTTP-Nachrichten für `WT_REMOTE`). Es ist außerdem möglich, eigene Filter zu verwenden, die z.B. als WTScript Funktionen oder als Userexits implementiert sind (siehe auch Kapitel „Beispiele“ auf Seite 99).

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 das offene Protokoll und die Schnittstellen für die Client-Entwicklung für WebTransactions beschreibt:

  Client-APIs für WebTransactions

  Das Handbuch beschreibt

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

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

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

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

  Web-Frontend für Web-Services

  Das Handbuch beschreibt

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

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

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

1.4 Konzept und Zielgruppe dieses Handbuchs

Diese Dokumentation wendet sich an alle, die mit WebTransactions auf dynamische Web-Inhalte zugreifen wollen.

Die einzelnen Kapitel beschreiben die hierfür notwendigen Schritte. Im abschließenden Kapitel werden diese Schritte nochmals an konkreten Beispielen verdeutlicht.

Das Handbuch ergänzt das einführende WebTransactions-Handbuch „Konzepte und Funktionen“ und das WebTransactions-Referenzhandbuch „Template-Sprache“ um die für den HTTP-Anschluss benötigten Informationen.

Gültigkeit der Beschreibung

WebTransactions for HTTP ist auf den Systemplattformen Windows, Solaris, Linux und BS2000/OSD ablauffähig. Diese Dokumentation gilt für alle Plattformen. Falls sich eine Information speziell auf eine bestimmte Plattform bezieht, wird jeweils ausdrücklich darauf hingewiesen.

1.5 Neue Funktionen

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

Art der Neuerung	Beschreibung
Neues Systemobjekt-Attribut `COMMUNICATION_FILE_NAME`	Seite 20
Neues Systemobjekt-Attribut `COMMUNICATION_FILE_TYPE`	Seite 20
Neues Systemobjekt-Attribut `METHOD`	Seite 20

1.6 Darstellungsmittel

Diese Dokumentation verwendet die folgenden Darstellungsmittel:

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

2 Basisverzeichnis anlegen

Bei der Installation einer der Liefereinheiten von WebTransactions auf dem WebTransactions-Server werden die Dateien und Programme, die für den dynamischen Zugriff auf Web-Inhalte notwendig sind, automatisch mit installiert. Nach der Installation von WebLab auf Ihrem persönlichen Windows-Rechner können Sie dann mit Hilfe von WebLab ein oder mehrere Basisverzeichnisse erzeugen. Ein Basisverzeichnis nimmt alle Dateien auf, die WebTransactions für eine bestimmte WebTransactions-Anwendung konfigurieren.

Bei einer De-Installation von WebTransactions oder beim Installieren einer neuen Produktversion bleiben die individuellen Konfigurationen erhalten.

WebTransactions for HTTP können Sie auf zwei Arten einsetzen:

- Sie können WebTransactions for HTTP für den eigenständigen Zugriff auf einen HTTP-Server verwenden. In diesem Fall können Sie ein oder mehrere Basisverzeichnisse exklusiv für WebTransactions for HTTP anlegen.
- Sie können es auch zusammen mit einer der Liefereinheiten zur Umsetzung von Host-Anwendungen verwenden (z.B. WebTransactions for OSD), um beispielsweise eine Web-Oberfläche für eine bestehende Host-Anwendung zu realisieren, in die auch Zugriffe auf eine Suchmaschine im Web integriert werden sollen. In diesem Fall richten Sie das Basisverzeichnis für HTTP und die Liefereinheit ein, die Sie für die Umsetzung erworben haben. Lesen Sie dazu auch das entsprechende Kapitel „Basisverzeichnis anlegen“ im Handbuch zur jeweiligen Liefereinheit.

Die folgenden Abschnitte beschreiben den ersten Fall und erklären für beide Fälle, welche speziellen Dateien für den HTTP-Zugriff angelegt werden, sowie welche Systemobjekt-Attribute oder Host-Objekte speziell für den HTTP-Zugriff zur Verfügung stehen.

Basisverzeichnis anlegen mit WebLab

Damit Sie ein Basisverzeichnis für eine WebTransactions-Anwendung anlegen können, muss der WebTransactions-Administrator zuvor eine Benutzerkennung für Sie einrichten. Anschließend muss er für diese Benutzerkennung einen oder mehrere Pools freigeben, damit Sie dort ein Basisverzeichnis anlegen können.

Bevor Sie ein Basisverzeichnis erzeugen, empfiehlt es sich außerdem, zunächst ein Projekt zu erstellen, in dem die wichtigsten Daten gespeichert werden, die WebLab beim Arbeiten mit der WebTransactions-Anwendung benötigt. Beim Anlegen des Projekts wird Ihnen dann automatisch die Option angeboten, ein Basisverzeichnis zu erstellen.

Gehen Sie folgendermaßen vor:

- Rufen Sie WebLab auf, z.B. über **Start/Programme/WebTransactions 7.5/WebLab**
- Als Einstieg zum Anlegen eines Basisverzeichnisses gibt es folgende zwei Möglichkeiten:

  - Wählen Sie den Befehl **Projekt/Neu...** und bestätigen Sie die Abfrage, ob ein Basisverzeichnis erstellt werden soll, mit **Ja**.

  oder

  - Wählen Sie den Befehl **Generieren**/**Basisverzeichnis ...** und geben Sie bei der folgenden Abfrage an, dass ein neues Projekt erstellt wird.

  In beiden Fällen wird das Dialogfeld **Verbinden** geöffnet.

- Tragen Sie im Dialogfeld **Verbinden** mit Hilfe der Schaltflächen **Ändern** die Verbindungsparameter ein und bestätigen Sie mit **OK**. Es werden die Parameter eingeblendet, die über das Menü **Optionen**, Menüpunkt **Einstellungen**, Registerblatt **Server** festgelegt wurden.
- Geben Sie im nachfolgenden Dialogfeld Ihre Benutzerkennung mit Passwort an und bestätigen Sie Ihre Angaben mit **OK**.
- Machen Sie im Dialogfeld **Basisverzeichnis erstellen** folgende Einträge:
  - Wählen Sie unter den angebotenen Pools den Pool aus, in dem das Basisverzeichnis angelegt werden soll
  - Tragen Sie den Namen des neuen Basisverzeichnisses ein
  - Aktivieren Sie im Bereich **Host-Adapter** die Checkbox **HTTP**
  - Klicken Sie auf **OK**.

Damit richtet WebLab das Basisverzeichnis mit allen Dateien ein, die für den Ablauf der WebTransactions-Anwendung benötigt werden. Struktur und Inhalt des Basisverzeichnisses sind im WebTransactions-Handbuch „Konzepte und Funktionen“ beschrieben.

Basisverzeichnis auf eine neue Version umstellen

- Wählen Sie **Generieren/Basisverzeichnis aktualisieren**. Das Dialogfeld **Basisverzeichnis aktualisieren** wird geöffnet.
- Wenn nur die Links aus dem Basisverzeichnis in das neue Installationsverzeichnis geändert werden sollen, wählen Sie die Option **Verweise anpassen**. Wählen Sie diese Option, wenn Sie Dateien angepasst haben, die von WebTransactions mitgeliefert oder generiert wurden.
- Wenn alle Dateien, die beim Erzeugen eines Basisverzeichnisses kopiert oder generiert werden, neu erstellt werden sollen, wählen Sie die Option **Dateien überschreiben**.

3 Kommunikation steuern

Dieses Kapitel beschreibt, wie Sie mit WebTransactions for HTTP auf HTTP-Server zugreifen und deren Ressourcen nutzen können. Konkrete Beispiele für die hier dargestellten Konzepte finden Sie im Kapitel „Beispiele“ auf Seite 99.

3.1 Systemobjekt-Attribute

Mit einigen Attributen des Systemobjekts steuern Sie die Kommunikation zwischen WebTransactions und einem HTTP-Server.

Hier werden nur diejenigen Attribute beschrieben, die es speziell für HTTP-Anbindungen gibt oder die zumindest für HTTP-Anbindungen eine spezielle Bedeutung haben. Attribute des Systemobjektes, deren Bedeutung für alle Liefereinheiten von WebTransactions gleich ist, sind im WebTransactions-Handbuch „Konzepte und Funktionen“ beschrieben.

Existiert unterhalb des verwendeten Kommunikationsobjekts ein Objekt `WT_SYSTEM` („privates Systemobjekt“), so müssen die in diesem Abschnitt beschriebenen Attribute dort definiert werden, anderenfalls sind sie als Attribute des globalen Systemobjekts `WT_SYSTEM` zu erklären.

Grundlegende Informationen zum verbindungsspezifischen und globalen Systemobjekt finden Sie im WebTransactions-Handbuch „Konzepte und Funktionen“.

Die Attribute können Sie beim Start von WebTransactions im ersten Template (Start-Template) setzen und für die Sitzung beibehalten oder aktiv steuernd in der Sitzung verändern (siehe WebTransactions-Handbuch „Konzepte und Funktionen“).

3.1.1 Übersicht

Einen Überblick über die Attribute und ihre Wirkung gibt die folgende Tabelle. Die Systemobjekt-Attribute können in folgende Kategorien eingeteilt werden, die in der rechten Spalte der Tabelle angegeben sind:

t (**t**emporary)
Attribute, die während der Kommunikation verwendet werden und die jederzeit in den Templates verändert werden können.

c (**c**ommunication module)
Attribute, die automatisch vom Host-Adapter gesetzt werden.

Attributname	Bedeutung	Erläuterung/Kategorie	
COMMUNICATION_FILE_NAME	Name der Zieldatei	Datei, in die die bei einem receive empfangenen Daten gespeichert werden sollen. Gespeichert wird jeweils der Body der Nachricht. Die Datei muss unterhalb des Basisverzeichnisses liegen, eine Ablage im Unterverzeichnis wwwdocs ist erlaubt. Dieses Attribut können Sie z.B. benutzen, um Bilder zu übertragen. (siehe auch COMMUNICATION_FILE_TYPE unten).	t
COMMUNICATION_FILE_TYPE	Filter beim Speichern	Filter für das Speichern einer Nachricht. Das Attribut enthält den Typ einer Nachricht. Nur wenn der Typ der empfangenen Nachricht (Feld Content-Type im HTTP-Header) mit dem hier angegebenen Wert übereinstimmt, wird der Body der Nachricht in der Datei gespeichert, die Sie mit COMMUNICATION_FILE_NAME angegeben haben (siehe oben).	t
HTTP_RETURN_CODE	Fehlernummer	Rückgabe eines HTTP-Requests. Das Attribut wird durch die Methode receive gesetzt und gibt den Returncode des HTTP-Requests an. Ein Wert von 200 bedeutet OK, jeder andere Wert ist ein Fehler. HTTP_RETURN_CODE ist leer, wenn vom HTTP-Server keine Antwort auf den Request gekommen ist. Eine Übersicht über die möglichen Fehlercodes und ihre Bedeutung finden Sie im Abschnitt „HTTP-Fehlermeldungen“ auf Seite 111.	c
METHOD	Methode für HTTP-Request	Für den mit send()/receive() ausgelösten HTTP-Request wird die hier angegebene Methode benutzt. Andernfalls wird GET oder POST benutzt, je nach Existenz des Objektes Body.	t

Attributname	Bedeutung	Erläuterung/Kategorie	
PASSWORD	Passwort	Dieses Attribut gibt das Passwort für die HTTP-Verbindung an. Zusammen mit dem Attribut USER (siehe unten) bildet es die Authentifizierung am fernen HTTP-Server. Dieses Attribut hat Vorrang vor dem entsprechenden Wert in URL.[1]	t
PROXY	HTTP-Proxy	Name oder IP-Adresse des Rechners, der für die HTTP-Verbindung als HTTP-Proxy verwendet werden soll (siehe auch PROXY_PORT weiter unten)	t
PROXY_PASSWORD	Passwort für den Proxy	Passwort für die HTTP-Verbindung über einen Proxy-Rechner	
PROXY_PORT	Port für HTTP-Proxy	Port des zu verwendenden HTTP-Proxy (siehe auch PROXY); wird kein Port angegeben, ist die Voreinstellung 80	t
PROXY_USER	Benutzername für den Proxy	Benutzername für die HTTP-Verbindung über einen Proxy-Rechner; zusammen mit dem Attribut PROXY_PASSWORD (siehe oben) bildet dieses Attribut die Authentifizierung am fernen HTTP-Server über einen Proxy-Rechner.	t
SSL_CERT_FILE	SSL-Zertifikat	Dateiname, in dem sich ein zu verwendendes Clientseitiges Zertifikat befindet. Wenn der Dateiname nicht absolut angegeben ist, ist er relativ zum Basis-Verzeichnis.	t
SSL_KEY_FILE	Schlüssel zum Zertifikat	Dateiname, in dem sich der private Schlüssel zu dem Zertifikat (siehe SSL_CERT_FILE) befindet. Wenn der Dateiname nicht absolut angegeben ist, ist er relativ zum Basis-Verzeichnis. Wird dieses Attribut nicht angegeben, wird der Wert von SSL_CERT_FILE übernommen. Es wird dann davon ausgegeangen, dass Zertifikat und Schlüssel gemeinsam in einer Datei enthalten sind.	t
SSL_PASSPHRASE	Kennwort für den privaten Schlüssel	Kennwort für die Verwendung mit dem privaten Schlüssel	t

Attributname	Bedeutung	Erläuterung/Kategorie	
SSL_PROTOCOL	Auswahl des SSL-Protokolls	Angabe, welche Version des SSL-Protokolls und des TLS-Protokolls aktiviert werden sollen. OpenSSL unterstützt das SSL-Protokoll in den Versionen 2 und 3 und das TLS-Protokoll Version 1. Mögliche Werte: SSLv2 SSLv3 TLSv1 All Voreinstellung: All Mehrere Protokolle können durch Leerzeichen getrennt angegeben werden.	t
TIMEOUT_HTTP	Timeout für HTTP-Requests	Zeitspanne in Sekunden, die nach dem Aufruf der Methode send auf eine Antwort des HTTP-Servers gewartet wird. Ist das Attribut nicht gesetzt, dann wird als Voreinstellung ein Wert von 60 Sekunden angenommen. Ist TIMEOUT_HTTP größer als das globale Systemobjekt-Attribut TIMEOUT_APPLICATION, wird ein von TIMEOUT_APPLICATION abgeleiteter Wert verwendet: – ist TIMEOUT_APPLICATION > 10: TIMEOUT_HTTP entspricht TIMEOUT_APPLICATION -5 – ist TIMEOUT_APPLICATION > 1: TIMEOUT_HTTP entspricht TIMEOUT_APPLICATION -1 – ist TIMEOUT_APPLICATION =1 TIMEOUT_HTTP entspricht TIMEOUT_APPLICATION.	t
URL	Ziel-URL	Ziel-URL der adressierten HTTP-Ressource. Die Syntax dieses Attributs ist folgende: [http[s]://][*user*[:*password*]@] *machine*[:*port*]/*file*[/*pathinfo*][?*query*] [2] Ist das Protokoll https angegeben, dann ist der Standardwert für *port* 443, anderfalls 80. Die optionalen Werte für *user* und *password* werden durch die Werte der Attribute USER und PASSWORD überschrieben.	t

Attributname	Bedeutung	Erläuterung/Kategorie	
USER	Benutzername	Dieses Attribut gibt den Benutzernamen für die HTTP-Verbindung an. Zusammen mit dem Attribut PASSWORD (siehe oben) bildet es die Authentifizierung am fernen HTTP-Server. Dieses Attribut hat Vorrang gegenüber dem entsprechenden Wert in URL. [1]	t

[1] Benutzername, Kennwort und Port können dem HTTP-Server auch über die URL angegeben werden. Das Format dazu ist:

[**http[s]:**][**//**][*benutzer*[**:***kennwort*]**@**]*machine*[**:***port*]**/...**

[2] Kurzbeschreibung der URL-Syntax:

user - Benutzer
password - Kennwort
machine - Rechnername oder IP-Adresse des HTTP-Servers
port - Portnummer des HTTP-Servers
file - Dateiname auf dem HTTP-Server
pathinfo, query - Informationen, die nur für CGI-Programme relevant sind und diesen über Umgebungsvariable zur Verfügung gestellt werden

3.1.2 Zusammenspiel: Systemobjekt-Attribute und Methoden

Dieser Abschnitt informiert Sie darüber, welche HTTP-spezifischen Attribute des Systemobjekts bei welchen Methodenaufrufen eine Rolle spielen.

open - HTTP-Host-Adapter aktivieren

Ein Aufruf der Methode open initialisiert ein Kommunikationsobjekt für HTTP-Verbindungen. Das HTTP-Protokoll kennt den Begriff „Sitzung“ nicht, daher wird durch open keine Sitzung aufgebaut, sondern nur der HTTP-Host-Adapter für das verwendete Kommunikationsobjekt aktiviert.

An dieser Stelle müssen keine Systemobjekt-Attribute berücksichtigt werden, da diese nur für die jeweiligen HTTP-Requests mit den weiter unten beschriebenen Methoden send und receive eine Rolle spielen. Als einzige Information muss das verwendete Protokoll angegeben werden (HTTP).

Beispiel

```
http_host = new WT_Communication('myHTTP');
http_host.open('HTTP');
```

Dieses Beispiel legt ein neues Kommunikationsobjekt mit dem Namen *myHTTP* unter WT_HOST an. Gleichzeitig wird eine zweite Referenz http_host auf dieses Kommunikationsobjekt angelegt. Dann aktiviert das Beispiel den HTTP-Host-Adapter.

send - HTTP-Request senden

Ein Aufruf der Methode `send` baut eine HTTP-Verbindung auf und verwendet dabei die folgenden Systemobjekt-Attribute:

Systemobjekt-Attribut	Verwendung
`URL`	`send` baut eine HTTP-Verbindung zu der angegebenen HTTP-Ressource auf.
`PROXY` `PROXY_PORT`	Dabei werden diese Systemobjekt-Attribute als HTTP-Proxy berücksichtigt.
`USER` `PASSWORD`	Sind diese Attribute gesetzt, dann werden sie für die Benutzerauthentifizierung genutzt.
`PROXY_USER` `PROXY_PASSWORD`	Sind diese Attribute gesetzt, werden sie für die Benutzerauthentifizierung an den Proxy-Rechner geschickt.
`TIMEOUT_HTTP`	Über dieses Attribut wird der Timeout für die Antwort vom angesprochenen HTTP-Server gesteuert. Voreinstellung: 60 Sekunden

Tabelle 1: Verwendete Systemobjekt-Attribute bei send

Wird die Methode `send` mehrfach hintereinander ausgeführt, so wird die Verbindung aus dem vorigen Aufruf zunächst geschlossen, ein ggf. vorliegendes Ergebnis des vorigen Aufrufs wird verworfen.

Beispiel

```
// Festlegen der URL und des HTTP-Proxy:
http_host.WT_SYSTEM.URL = 'www.mycompany.de/webtransactions';
http_host.WT_SYSTEM.PROXY = 'proxy.mycompany.de';
http_host.WT_SYSTEM.PROXY_PORT = '80';
// Loeschen des Host-Objekts sendData:
delete http_host.sendData;
// HTTP-Request starten:
http_host.send();
```

Dieses Beispiel führt die HTTP-Methode `GET` für die WebTransactions-Homepage aus. Dabei wird der Proxy-Server `proxy.mycompany.de` (Port `80`) verwendet, ein Benutzer wird nicht vorgegeben. Der Timeout für die Serverantwort beträgt 60 Sekunden.

receive - Antwort auf HTTP-Request empfangen

Ein Aufruf der Methode `receive` liefert das Ergebnis einer HTTP-Anfrage im Host-Objekt `receiveData` zurück (siehe auch Abschnitt „Host-Objekte“ auf Seite 27) und verwendet dabei die folgenden Systemobjekt-Attribute:

Systemobjekt-Attribut	Verwendung
`HTTP_RETURN_CODE`	Dieses Attribut wird mit dem Rückgabewert versorgt. Liefert das Empfangen der Antwortdaten einen Rückgabewert ungleich 200, so wird zusätzlich das Systemobjekt-Attribut `ERROR` am globalen Systemobjekt gesetzt. Eine Übersicht über die möglichen Fehlercodes und ihre Bedeutung finden Sie im Abschnitt „HTTP-Fehlermeldungen“ auf Seite 111. Werden keine Antwortdaten empfangen, ist `HTTP_RETURN_CODE` leer und das Systemobjekt-Attribut `ERROR` am globalen Systemobjekt wird gesetzt.
`TIMEOUT_HTTP`	Ist der eingestellte Timer abgelaufen, dann kehrt `receive` ohne Antwort zurück und besetzt die Attribute `ERROR` am globalen Systemobjekt und `HTTP_RETURN_CODE` am Systemobjekt entsprechend.

Tabelle 2: Verwendete Systemobjekt-Attribute bei receive

Nach dem Empfang der Antwort-Daten wird die Verbindung zum HTTP-Server wieder geschlossen. Die Antwortnachricht wird analysiert und der Inhalt wird im Host-Objekt `receiveData` ohne weitere Bearbeitung gespeichert. Handelt es sich um eine mehrteilige Nachricht, so werden die Inhalte der einzelnen Teile in einem Array von Objekten abgelegt (`receiveData.0`, `receiveData.1`, ...).

WebTransactions führt keinerlei Interpretation der empfangenen Inhalte durch. Eine Interpretation dieser Inhalte muss über Filter vorgenommen werden, die individuell für den jeweiligen Inhaltstyp und dessen Struktur angepasst sind (z.B. `text/html`, `text/xml`, etc.).

Wird die Methode `receive` ohne vorheriges `send` ausgeführt, so werden die Aktionen von `send` implizit ausgeführt, wobei die Systemobjekt-Attribute aus der Beschreibung von `send` sowie das Host-Objekt `sendData` automatisch berücksichtigt werden.

Die Kombination aus `send` und `receive` ist gegenüber der ausschließlichen Verwendung von `receive` vor allem dann sinnvoll, wenn man ein nebenläufiges Laden erreichen will, d.h. man kann ein `send` bereits einen Dialogschritt vorher ausführen und nach dem Dialogschritt mit `receive` abfragen, ob die Daten bereits eingetroffen sind. Auf diese Weise lassen sich Wartezeiten sparen.

Beispiel

```
// Festlegen der URL:
http_host.WT_SYSTEM.URL = 'www.mycompany.de/webtransactions';
// Loeschen des Host-Objekts sendData:
delete http_host.sendData;
// HTTP-Request starten:
http_host.send();
// Empfangen der Antwortdaten:
http_host.receive();
```

Dieses Beispiel führt zunächst die HTTP-Methode `GET` für die WebTransactions-Homepage aus. Dabei wird kein Proxy-Server verwendet, ein Benutzer wird nicht vorgegeben. Der Timeout für die Serverantwort beträgt 60 Sekunden. Die angegebene Seite wird empfangen und kann dann über `http_host.receiveData` weiter verarbeitet werden.

close - HTTP-Modul deaktivieren

Ein Aufruf der Methode `close` deaktiviert den HTTP-Host-Adapter. Das HTTP-Modul gibt dadurch seinen intern verwendeten Speicher frei. Sie sollte daher dann aufgerufen werden, wenn keine weiteren HTTP-Requests in der WebTransactions-Anwendung mehr benötigt werden.

3.2 Host-Objekte

WebTransactions wickelt den Datenaustausch mit einem HTTP-Server über die Host-Objekte `sendData` und `receiveData` ab:

– `sendData` können Sie selbst anlegen, wenn Sie Daten an den HTTP-Server übermitteln wollen

– `receiveData` enthält die Antwortdaten des Servers.

Beide Host-Objekte können folgende Attribute besitzen:

Attribut	Bedeutung
`ContentType`	Attribut vom Typ `string`: Typ der HTTP-Nachricht
`Header`	Objekt der Klasse `Array`: Informationen für den HTTP-Server Eine Beschreibung des Attributs `Header` finden Sie im Abschnitt „Das Attribut Header“ auf Seite 35.
`Body`	Attribut vom Typ `string`: Inhalt der HTTP-Nachricht

Tabelle 3: Attribute von Host-Objekten

Zu diesen Attributen wird bei jedem Senden ein Header mit vordefinierten Feldern mitgeschickt. Die Anzahl und Bedeutung der Felder ist abhängig vom jeweiligen HTTP-Request, siehe hierzu auch folgenden Abschnitt.

3.2.1 Nachricht mit dem Host-Objekt sendData senden

Abhängig davon, ob und welche Daten Sie an den HTTP-Server senden wollen, ist das Host-Objekt `sendData` und damit der HTTP-Request aufgebaut. Ein HTTP-Request besteht aus der Methode `GET` oder `POST` mit Argument, gefolgt von einer Protokoll-Version, einem Header und ggf. einer Nachricht. Ein Request kann also folgendermaßen aufgebaut sein:

`POST` *request* `HTTP/1.0 Header senddata`

Der *request* selbst setzt sich zusammen aus:

`/file[/pathinfo][?query]`

Über den HTTP-Host-Adapter können Sie folgende Nachrichtentypen an den HTTP-Server senden:

– ohne Hostobjekt `sendData`

– einteilige Nachrichten

– mehrteilige Nachrichten

Diese unterschiedlichen Nachrichtentypen sind im Folgenden beschrieben.

3.2.1.1 Nachricht ohne sendData-Objekt

Wenn Sie für die erste Verbindungsanfrage noch kein Host-Objekt `sendData` angelegt haben, wird die HTTP-Methode `GET` ausgeführt:

`GET` *request* `HTTP/1.0 Header`

An Informationen wird ein vordefinierter Header mit folgenden Feldern an den HTTP-Server geschickt:

`Host`	enthält den Hostnamen aus der URL (und ggf. die Port-Nummer, falls diese nicht 80 ist).
`User-Agent`	enthält den Text " WebTransactions HTTP/7.5" (versionsabhängig)
`Authorization`	enthält einen mit `uuencode` aus den Attributen `USER` und `PASSWORD` erzeugten Text (siehe auch Attribute des Systemobjekts).
`Proxy-Authorization`	enthält einen mit `uuencode` aus den Attributen `PROXY_USER` und `PROXY_PASSWORD` erzeugten Text (siehe auch Attribute des Systemobjekts).

Beispiel

```
host = new WT_Communication( 'http' );
host_system = host.WT_SYSTEM;
host.open('HTTP');
host_system.URL = "//localhost/webtav75/wtadm_admin.htm";
host.send();
host.receive();
```

Aus diesen Angaben wird folgende Information an den HTTP-Server geschickt:

```
      Hexadecimal:                                      Ascii:
00000: 47 45 54 20 2f 77 65 62 74 61 76 37 35 2f 77 74 !GET /webtav75/wt!.......@./......!
00016: 61 64 6d 5f 61 64 6d 69 6e 2e 68 74 6d 20 48 54 !adm_admin.htm HT!/]_^/]_.>..._...!
00032: 54 50 2f 31 2e 30 0d 0a 48 6f 73 74 3a 20 6c 6f !TP/1.0..Host: lo!.&.......?....%?!
00048: 63 61 6c 68 6f 73 74 0d 0a 55 73 65 72 2d 41 67 !calhost..User-Ag![/%.?..........~!
00064: 65 6e 74 3a 20 57 65 62 54 72 61 6e 73 61 63 74 !ent: WebTransact!.>.....@../>./[.!
00080: 69 6f 6e 73 20 48 54 54 50 2f 37 2e 35 41 30 30 !ions HTTP/7.5AOO!.?>.....&.......!
00096: 0d 0a 0d 0a                                     !....            !....            !
```

3.2.1.2 Einteilige Nachrichten

Wenn Sie eine einteilige Nachricht ohne eigenen Header an den HTTP-Server schicken wollen, dann muss `sendData` ein Objekt der Klasse `Object` sein, mit dem Attribut `Body`:

```
sendData
       Body
```

Im Attribut `Body` steht die eigentliche Nachricht an den HTTP-Server. Aus dieser Angabe formuliert WebTransactions die Methode `POST` und schickt die Daten an den HTTP-Server. An Informationen wird ein vordefinierter Header mit folgenden Feldern und `sendData.Body` an den HTTP-Server geschickt:

`Host`	enthält den Hostnamen aus der URL (und ggf. die Port-Nummer, falls diese nicht 80 ist).
`User-Agent`	enthält den Text " WebTransactions HTTP/7.5" (versionsabhängig)
`Authorization`	enthält einen mit `uuencode` aus den Attributen `USER` und `PASSWORD` erzeugten Text (siehe auch Attribute des Systemobjekts).
`Proxy-Authorization`	enthält einen mit `uuencode` aus den Attributen `PROXY_USER` und `PROXY_PASSWORD` erzeugten Text (siehe auch Attribute des Systemobjekts).
`Content-Length`	enthält die Länge von `sendData.Body`

Der Inhalt im `Body` muss url-codiert werden. Dies bedeutet, dass einige Zeichen durch `'%'`, gefolgt von ihrem hexadezimalen Wert, ersetzt werden (z.B.: `':'` durch `%3A`, `'\'` durch `%5C`).

Beispiel

```
host.sendData = new Object();
host.sendData.ContentType = 'application/x-www-form-urlencoded';
host.sendData.Body = 'question=sense+of+universe&answer=42';
host.WT_SYSTEM.URL = 'www.deepThought.mt/cgi-bin/verify.exe';
host.WT_SYSTEM.USER = 'user27';
host.WT_SYSTEM.PASSWORD = 'pass';
host.send();
...
```

Folgende Werte werden über die vordefinierten Felder im Header mit dem HTTP-Request gesendet:

`Content-Type`	application/x-www-form-urlencoded
`Host`	www.deepThought.mt
`User-Agent`	WebTransactions HTTP/7.5
`Authorization`	Basic dXNlcjI3OnBhc3M=
`Content-Length`	36

Wenn Sie die Nachricht mit einem eigenen Header versorgen wollen, dann brauchen Sie für das Host-Objekt `sendData` zusätzlich die Attribute `ContentType` und/oder `Header`:

– Wenn Sie nur den Typ der Nachricht angeben wollen, verwenden Sie wie bisher das Attribut `ContentType`.

– Nur wenn Sie den HTTP-Server mit weiteren Header-Informationen versorgen wollen, verwenden Sie das Attribut `Header`. `Header` muss ein Objekt vom Typ `Array` sein, siehe hierzu auch Abschnitt „Das Attribut Header“ auf Seite 35.

```
sendData
      Body
      [ContentType]
      [Header]
          0
           Name
           Value
          1
           Name
           Value
```

Der vordefinierte Header wird um das Feld `Content-Type` ergänzt:

`Content-Type` enthält den Wert des Attributs `sendData.ContentType` oder des entsprechenden Elementes aus dem selbst-definierten Header. Existiert dieses Attribut nicht, wird kein solches Header-Feld gesendet

Beachten Sie, dass die Reihenfolge und der Wert der vordefinierten Header-Felder durch einen selbst-definierten Header beeinflusst werden können, siehe auch Abschnitt „Das Attribut Header“ auf Seite 35.

Beispiel

```
host.sendData = new Object();
host.sendData.ContentType = 'application/x-www-form-urlencoded';
host.sendData.Header = new Array();
host.sendData.Header[0] = new Object();
host.sendData.Header[0].Name = 'Pragma';
host.sendData.Header[0].Value = 'nocache';
host.sendData.Header[1] = {Name:'Authorization'};
host.sendData.Header[2] = {Name:'ConTenT-Type', Value:'text'};
host.sendData.Header[3] = {Name:'Content-Length'};
host.sendData.Body = 'question=sense+of+universe&answer=42';
host.WT_SYSTEM.URL = 'www.deepThought.mt/cgi-bin/verify.exe';
host.WT_SYSTEM.USER = 'user27';
host.WT_SYSTEM.PASSWORD = 'pass';
host.send();
...
```

Mit dem Attribut `Header` wird Einfluss genommen auf Reihenfolge, Schreibweise und Inhalt der Header-Felder. Folgende Felder werden mit dem HTTP-Request gesendet:

`Host`	www.deepThought.mt	1
`User-Agent`	WebTransactions HTTP/7.5	1
`Pragma`	nocache	2
`Authorization`	Basic dXNlcjI3OnBhc3M=	3
`ConTenT-Type`	text	4
`Content-Length`	36	3

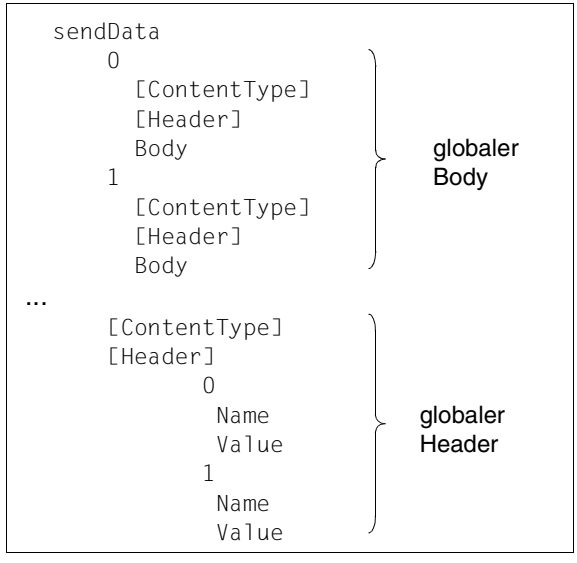
Anmerkungen

1. Vordefiniertes Feld am Beginn der Liste, weil ein entsprechendes Element nicht im Array Header enthalten ist.
2. Zusätzliches Feld aus dem Array Header.
3. Position durch das Element im Array Header gesteuert; Inhalt intern errechnet (vordefiniertes Feld), weil das Attribut Value nicht existiert.
4. Das vordefinierte Feld wurde ignoriert, Position und Wert des Feldes sind aus dem Array Header ermittelt (einschließlich der nicht sinnvollen Schreibweise `ConTenT-Type`).

3.2.1.3 Mehrteilige Nachrichten

Wenn Sie eine mehrteilige Nachricht senden wollen, müssen Sie `sendData` als ein Objekt der Klasse `Array` anlegen. Jede Teilnachricht ist ein Element das Arrays `sendData` also `sendData[0]`, `senddata[1]`, usw..

Jede Teilnachricht (jedes Element) muss dann ein Objekt `Body` enthalten und kann je ein Objekt `ContentType` und `Header` enthalten.



Im globalen Body steht die gesamte Nachricht, aufgeteilt in mehrere Teilnachrichten an den HTTP-Server. Eine Teilnachricht entspricht dabei einer einteiligen Nachricht mit den Attributen `Body`, `ContentType` und `Header`. Für den globalen Body sind folgende Header-Felder vordefiniert:

`Host`	enthält den Hostnamen aus der URL (und ggf. die Port-Nummer, falls diese nicht 80 ist).
`User-Agent`	enthält den Text " WebTransactions HTTP/7.5" (versionsabhängig)
`Authorization`	enthält einen mit `uuencode` aus den Attributen `USER` und `PASSWORD` erzeugten Text (siehe auch Attribute des Systemobjekts).
`Proxy-Authorization`	enthält einen mittels `uuencode` aus den Attributen `PROXY_USER` und `PROXY_PASSWORD` erzeugten Text (siehe auch Attribute des Systemobjekts).

`Content-Type`	enthält den Wert des Attributs `sendData.ContentType` oder des entsprechenden Elementes aus dem selbst-definierten Header. Existiert dieses Attribut nicht, wird folgender Wert gesendet: `multipart/mixed; boundary=<<'<<"<<42>>">>'>>`
`Content-Length`	enthält die Länge von `sendData.body`

Beachten Sie, dass die Reihenfolge und der Wert der vordefinierten Header-Felder durch einen selbst-definierten Header beeinflusst werden können.

Wenn das globale Attribut `ContentType` für den Typ der Gesamtnachricht verwendet wird, muss es immer mit dem Text `multipart` beginnen, wobei Groß- und Kleinschreibung nicht unterschieden werden.

Beispiel

In diesem Beispiel wird ein globaler Header und je ein individueller Header für die einzelnen Teilnachrichten mitgeschickt.

```
host_system.URL = "//localhost/Scripts/Cgitest.exe";
host.sendData = new Array();
host.sendData.Header = new Array();
host.sendData.Header[0] = { Name:'global-header-field', Value:'global
content' };
host.sendData[0] = new Object();
host.sendData[0].ContentType = "application/x-www-form-urlencoded";
host.sendData[0].Header = new Array();
host.sendData[0].Header[0] = { Name:'part1-header-field', Value:'header
part 1' };
host.sendData[0].Body =
"WT_SYSTEM_BASEDIR=%2Fhome1%2Fpuls%2Fhttpd%2Fpulswww&command=Refresh";
host.sendData[1] = new Object();
host.sendData[1].ContentType = "type1";
host.sendData[1].Header = new Array();
host.sendData[1].Header[0] = { Name:'part2-header-field', Value:'any
other' };
host.sendData[1].Body = "WT_SYSTEM_BASEDIR=val1&command=Refresh";
host.send();
```

Aus diesen Angaben werden folgende Informationen an den HTTP-Server gesendet:

```
       Hexadecimal:                                        Ascii:
00000: 50 4f 53 54 20 2f 53 63 72 69 70 74 73 2f 43 67 !POST /Scripts/Cg!
00016: 69 74 65 73 74 2e 65 78 65 20 48 54 54 50 2f 31 !itest.exe HTTP/1!
00032: 2e 30 0d 0a 43 6f 6e 74 65 6e 74 2d 54 79 70 65 !.0..Content-Type!
00048: 3a 20 6d 75 6c 74 69 70 61 72 74 2f 6d 69 78 65 !: multipart/mixe!
00064: 64 3b 20 62 6f 75 6e 64 61 72 79 3d 3c 3c 27 3c !d; boundary=<<'<!
00080: 3c 22 3c 3c 34 32 3e 3e 22 3e 3e 27 3e 3e 0d 0a !<"<<42>>">'>>..!
00096: 48 6f 73 74 3a 20 6c 6f 63 61 6c 68 6f 73 74 0d !Host: localhost.!
00112: 0a 55 73 65 72 2d 41 67 65 6e 74 3a 20 57 65 62 !.User-Agent: Web!
00128: 54 72 61 6e 73 61 63 74 69 6f 6e 73 20 48 54 54 !Transactions HTT!
00144: 50 2f 37 2e 35 0d 0a 43 6f 6e 74 65 6e 74 2d 4c !P/7.5..Content-L!
00160: 65 6e 67 74 68 3a 20 33 35 33 0d 0a 67 6c 6f 62 !ength: 353..glob!
00176: 61 6c 2d 68 65 61 64 65 72 2d 66 69 65 6c 64 3a !al-header-field:!
00192: 20 67 6c 6f 62 61 6c 20 63 6f 6e 74 65 6e 74 0d ! global content.!
00208: 0a 0d 0a 2d 2d 3c 3c 27 3c 3c 22 3c 3c 34 32 3e !...--<<'<<"<<42>!
00224: 3e 22 3e 3e 27 3e 3e 0d 0a 43 6f 6e 74 65 6e 74 !>">>'>>..Content!
00240: 2d 54 79 70 65 3a 20 61 70 70 6c 69 63 61 74 69 !-Type: applicati!
00256: 6f 6e 2f 78 2d 77 77 77 2d 66 6f 72 6d 2d 75 72 !on/x-www-form-ur!
00272: 6c 65 6e 63 6f 64 65 64 0d 0a 43 6f 6e 74 65 6e !lencoded..Conten!
00288: 74 2d 4c 65 6e 67 74 68 3a 20 36 37 0d 0a 70 61 !t-Length: 67..pa!
00304: 72 74 31 2d 68 65 61 64 65 72 2d 66 69 65 6c 64 !rt1-header-field!
00320: 3a 20 68 65 61 64 65 72 20 70 61 72 74 20 31 0d !: header part 1.!
00336: 0a 0d 0a 57 54 5f 53 59 53 54 45 4d 5f 42 41 53 !...WT_SYSTEM_BAS!
00352: 45 44 49 52 3d 25 32 46 68 6f 6d 65 31 25 32 46 !EDIR=%2Fhome1%2F!
00368: 70 75 6c 73 25 32 46 68 74 74 70 64 25 32 46 70 !puls%2Fhttpd%2Fp!
00384: 75 6c 73 77 77 77 26 63 6f 6d 6d 61 6e 64 3d 52 !ulswww&command=R!
00400: 65 66 72 65 73 68 0d 0a 2d 2d 3c 3c 27 3c 3c 22 !efresh..--<<'<<"!
00416: 3c 3c 34 32 3e 3e 22 3e 3e 27 3e 3e 0d 0a 43 6f !<<42>>">>'>>..Co!
00432: 6e 74 65 6e 74 2d 54 79 70 65 3a 20 74 79 70 65 !ntent-Type: type!
00448: 31 0d 0a 43 6f 6e 74 65 6e 74 2d 4c 65 6e 67 74 !1..Content-Lengt!
00464: 68 3a 20 33 38 0d 0a 70 61 72 74 32 2d 68 65 61 !h: 38..part2-hea!
00480: 64 65 72 2d 66 69 65 6c 64 3a 20 61 6e 79 20 6f !der-field: any o!
00496: 74 68 65 72 0d 0a 0d 0a 57 54 5f 53 59 53 54 45 !ther....WT_SYSTE!
00512: 4d 5f 42 41 53 45 44 49 52 3d 76 61 6c 31 26 63 !M_BASEDIR=val1&c!
00528: 6f 6d 6d 61 6e 64 3d 52 65 66 72 65 73 68 0d 0a !ommand=Refresh..!
00544: 2d 2d 3c 3c 27 3c 3c 22 3c 3c 34 32 3e 3e 22 3e !--<<'<<"<<42>>">!
00560: 3e 27 3e 3e                                     !>'>>           !
```

3.2.2 Das Attribut Header

Das Attribut `Header` ist ein Objekt vom Typ `Array`. Über diesen Array können Sie den HTTP-Server beim Senden mit zusätzlichen Header-Informationen versorgen, und beim Empfangen auf die Informationen der zurückgelieferten HTTP-Header-Felder zugreifen.

Der Array ist folgendermaßen aufgebaut:

Feld	Inhalt
n	Nummer des Feldes im Array, beginnend bei 0
n.`Name`	Name des Header-Feldes
n.`Value`	Wert des Header-Feldes

Tabelle 4: Aufbau des Array-Objekts `Header`

Wenn Sie das Attribut `Header` zum Senden von zusätzlichen Header-Informationen nutzen wollen, beachten Sie folgende Hinweise:

– Die Elemente des Arrays `Header` haben Vorrang. Das bedeutet, sie überschreiben die vordefinierten Header-Felder und auch ein zusätzlich gesetztes Attribut `sendData.ContentType`.

Beachten Sie die unterschiedliche Schreibweise des Host-Objekt-Attributs `ContentType` (ohne Bindestrich '-') und des Inhalts des Name-Attributes des Array-Elements `senddata.Header[n].Name='Content-Type'` (mit Bindestrich '-').

Da Bezeichner in WebTransactions keinen Bindestrich enthalten dürfen, in HTTP-Header-Feldern aber Bindestriche in den Namen gebräuchlich sind, wurde die Abbildung mit den zwei Attributen `Name` und `Value` im Header-Array gewählt, um jedes Header-Feld erzeugen zu können.

```
sendData.ContentType='type1';
```

ist also die verkürzte Schreibweise für

```
sendData.Header[0]={Name:'Content-Type', Value:'type1'};
```

oder gar

```
sendData.Header[n]=new Object;
sendData.Header[n].Name='Content-Type';
sendData.Header[n].Value='type1';
```

– Die Header-Felder werden in der Reihenfolge und Schreibweise (Groß-/Kleinschreibung wird nicht unterschieden) erzeugt, in der sie im Array `Header` auftreten. Nicht im Array enthaltene, vordefinierte Felder werden davor eingefügt.

– Ein Element im Array, bei dem nur das Attribut `Name` aber nicht das Attribut `Value` gesetzt ist, bestimmt nur die Position innerhalb der erzeugten Header-Felder. Der Inhalt kann von einem vordefinierten Feld bestimmt werden.

Die so möglichen Reihenfolgen der zu sendenden Header-Felder fasst folgende Tabelle noch einmal zusammen:

	vordefiniertes Feld	**nicht vordefiniertes Feld**
Feld ist nicht im Array `Header` **enthalten**	Feld steht vor den Feldern des Array `Header` Wert: intern	
Feld ist im Array `Header` **mit dem Attribut Name enthalten**	Feld steht an der im Array `Header` festgelegten Position Wert: intern	Feld steht an der im Array `Header` festgelegten Position Wert: leer
Feld ist im Array `Header` **mit den Attributen** `Header` **und** `Value` **enthalten**	Feld steht an der im Array `Header` festgelegten Position Wert: im Array festgelegt	

Tabelle 5: Reihenfolge der zu sendenden Header-Felder

Die Reihenfolge der Header-Felder ist im HTTP-Protokoll nicht festgelegt. Es gibt die Empfehlung, erst die `general header fields`, dann die `request header fields` und schließlich die `entity header fields` zu senden.

3.2.3 Nachricht im Host-Objekt receiveData empfangen

Das empfangene Host-Objekt `receiveData` entspricht im Aufbau dem Host-Objekt `sendData`.

einteilige Nachricht

```
receiveData
      Body
      ContentType
      Header
          0
           Name
           Value
          1
           Name
           Value
```

mehrteilige Nachricht

```
receiveData
      0
        [ContentType]
        [Header]
        Body
      1
        [ContentType]
        [Header]
        Body
      ContentType
      Header
          0
           Name
           Value
          1
           Name
           Value
...
```

`receiveData` enthält immer das Attribut `Header`. Für jedes empfangene Header-Feld legt der Host-Adapter im Array `Header` ein Element an.

3.3 HTTP-Rohdaten weiterverarbeiten

Der HTTP-Host-Adapter selbst verarbeitet nur komplette HTTP-Bodies als Textobjekte in den Host-Objekten `sendData` und `receiveData`.

Die Interpretation der über HTTP ausgetauschten Dokumente muss mit Hilfe spezieller Filter vorgenommen werden. Dieses Kapitel nennt die Möglichkeiten zur Definition eigener Sende- und Empfangsfunktionen mit entsprechender Filterwirkung.

3.3.1 WTScript-Filter

Sie können innerhalb von WebTransactions-Anwendungen beliebige Funktionen in WTScript realisieren, um die Rohdaten weiter zu verarbeiten. Beispiele dazu finden Sie im Kapitel „Beispiele“, Abschnitt „Vorhandenes CGI-Script nutzen“ auf Seite 99 und Abschnitt „Kommunikation über HTTP und Weiterverarbeitung mit WT_Filter“ auf Seite 103

Die Funktionen zum Aufbau bzw. zur Interpretation der HTTP-Nachrichten können folgendes leisten:

- Zusammensetzen von HTTP-Nachrichten aus internen WTScript-Objekten für den Versand zum HTTP-Server (`ContentType` und `Body`)
- Analyse der vom HTTP-Server empfangenen Daten (`Body`) und Aufteilung auf interne WTScript-Objekte.

3.3.2 Userexits

Ebenso ist es möglich, die Interpretation von empfangenen oder den Aufbau von zu sendenden HTTP-Nachrichten über Userexits zu implementieren. Dies ist vor allem dann sinnvoll, wenn entsprechende Bibliotheken bereits zur Verfügung stehen und ggf. nur leicht angepasst werden müssen.

3.3.3 Eingebaute Filter

WebTransactions for HTTP wird mit einer Filterklasse ausgeliefert, die den Inhaltstyp `text/xml` verarbeiten kann, der Klasse `WT_Filter`. Über die Methoden dieser Klasse ist es möglich:

- beliebige WTScript-Objekte in ein XML-Dokument und wieder zurück in ein WTScript-Objekt zu konvertieren, z.B. für den Datenaustausch mit anderen WebTransactions-Anwendungen über `WT_REMOTE`.
- beliebigen XML-Text in einen WTScript-Objektbaum und wieder in XML-Text umzuwandeln, z.B. für die Kommunikation mit Fremdanwendungen.
- beliebigen XML-Text mit einem eingebauten SAX-Parser zu analysieren.
- WTScript-Methodenaufrufe in XML-Dokumenten zu beschreiben und aus diesen wieder Methodenaufrufe zu erzeugen, z.B. für den Aufruf der Schnittstelle `WT_REMOTE` für verteilte WebTransactions-Anwendungen.

Weitere Informationen zur Klasse WT_Filter finden Sie im WebTransactions-Handbuch „Template-Sprache“ (Beschreibung der Klasse und Beispiele), im WebTransactions-Handbuch „Client-APIs für WebTransactions“ und im Abschnitt „Kommunikation über HTTP und Weiterverarbeitung mit WT_Filter“ auf Seite 103.

3.4 Start-Templates für HTTP

Nach dem Start der WebTransactions-Anwendung (über eine Einstiegsseite oder direkte Angabe der URL) müssen in einem Start-Template die Parameter für die Verbindung mit der Partneranwendung gesetzt werden.

WebTransactions stellt Ihnen bereits fertige Start-Templates zur Verfügung, die Sie als Vorlage für eigene Start-Templates verwenden können. Dabei haben Sie zwei Möglichkeiten:

- **Start-Template-Set (sofort ablauffähig)**

  Dieses Start-Template-Set ist sofort ablauffähig. Alle benötigten Parameter können im Dialog eingegeben werden. Es eignet sich sowohl zum Start einer einzelnen Host-Anwendung als auch zum Start mehrerer, in einer WebTransactions-Anwendung integrierter Host- bzw. Partneranwendungen. Das Set besteht aus dem allgemeinen Start-Template `wtstart.htm`, über das Sie z.B. Kommunikationsobjekte anlegen und zwischen verschiedenen parallelen Host-Verbindungen hin- und herschalten können, sowie aus spezifischen Start-Templates für die einzelnen Host-Adapter. Speziell für „WebTransactions for HTTP“ wird das Start-Template `wtstartHTTP.htm` ausgeliefert. Dieses HTTP-spezifische Start-Template wird in Abschnitt „HTTP-spezifisches Start-Template des Start-Template-Sets (wtstartHTTP.htm)“ auf Seite 41 dargestellt. Eine Darstellung des allgemeinen Start-Templates finden Sie im WebTransactions-Handbuch „Konzepte und Funktionen“.

- **einfaches Start-Template (muss angepasst werden)**

  Für den Anschluss an einen einzelnen HTTP-Server können Sie auf das einfache Start-Template `StartTemplateHTTP.htm` aufbauen. Dieses Start-Template ist durch die übersichtliche Struktur und die ausführliche Kommentierung besonders gut geeignet, um die konkreten Startparameter für den unmittelbaren Ablauf der WebTransactions-Anwendung festzulegen. Diese Startparameter müssen dann nicht mehr bei jedem neuen Start wie beim Start-Template-Set eingegeben werden. Das einfache Start-Template ist in Abschnitt „Einfaches Start-Template (StartTemplateHTTP.htm)“ auf Seite 46 dargestellt.

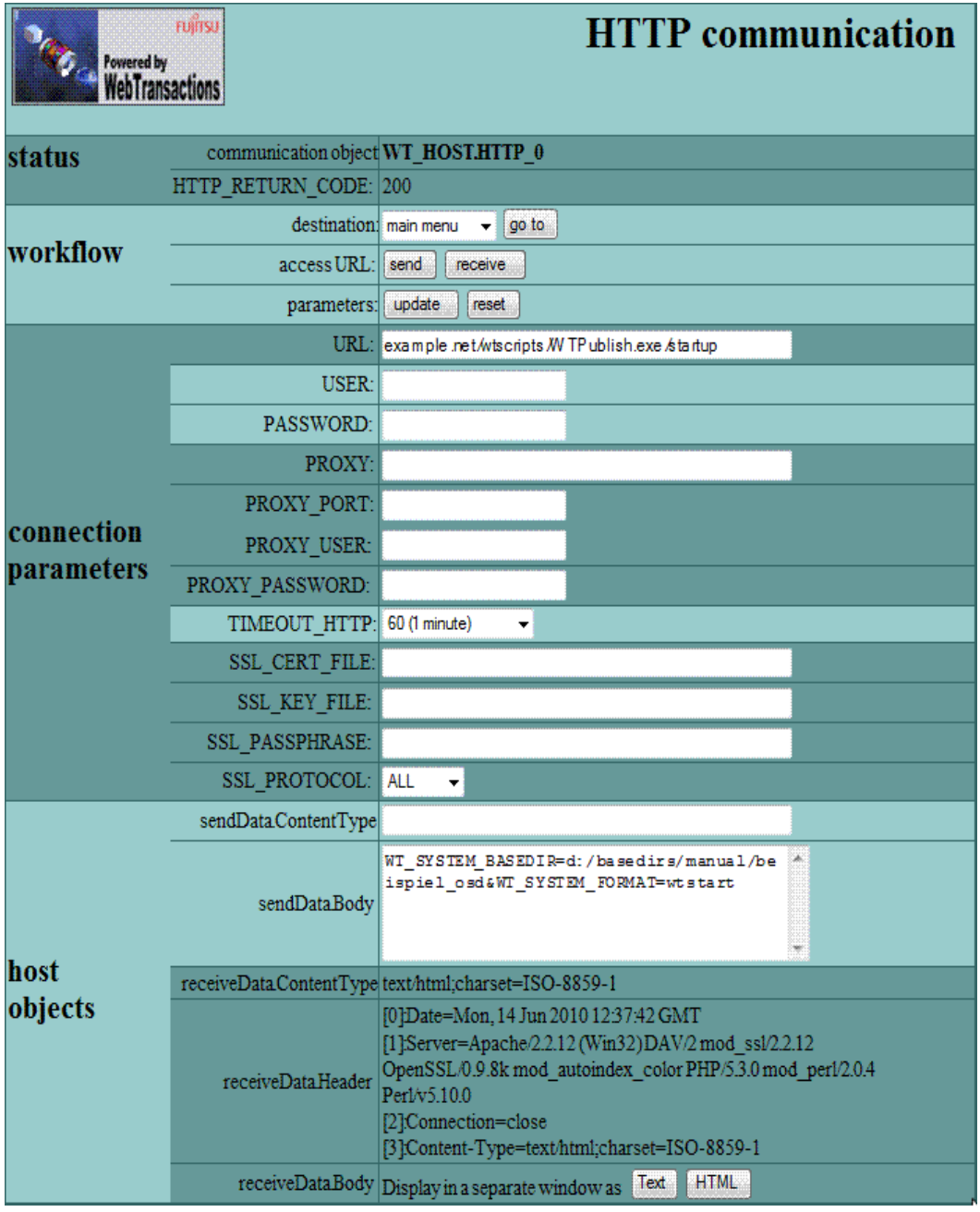
  Dieses einfache Start-Template eignet sich auch als Ausgangsbasis für selbstgeschriebene WTScript-Funktionen, die über HTTP aus dem Web Informationen ermitteln.

3.4.1 HTTP-spezifisches Start-Template des Start-Template-Sets (wtstartHTTP.htm)

Wenn Sie im allgemeinen Start-Template `wtstart.htm` (beschrieben im WebTransactions-Handbuch „Konzepte und Funktionen“) das Protokoll `HTTP` ausgewählt und ein neues Kommunikationsobjekt angelegt haben, wird zum Template `wtstartHTTP.htm` verzweigt.

`wtstartHTTP.htm` erlaubt es, die Verbindungsparameter interaktiv zu setzen und die Kommunikation anzustoßen.

Die folgende Abbildung zeigt die von `wtstartHTTP.htm` erzeugte Oberfläche.



Im Abschnitt **status** werden folgende Informationen angezeigt:

communication object
der Name des zu Grunde liegenden Kommunikationsobjekts

HTTP_RETURN_CODE
Hier wird der Rückgabewert des letzten `receive`-Aufrufs angezeigt. Wurde keine Anwort empfangen, z.B. weil der angesprochene Server nicht verfügbar ist, dann ist `HTTP_RETURN_CODE` leer. Eine Tabelle mit den möglichen Fehlern und deren Bedeutung finden Sie in Abschnitt „HTTP-Fehlermeldungen“ auf Seite 111.

Im Abschnitt **workflow** bestimmen Sie die nächste Aktion.

destination
Hier können Sie aktiv zwischen den verschiedenen Start-Templates wählen. Durch Klick auf `go to` wird auf die ausgewählte Seite verzweigt. Als Vorgabe wird `main menu` angeboten: Damit kann man auf die allgemeine Startseite `wtstart.htm` zurückkehren. Falls mehrere Verbindungen geöffnet sind, werden sie als weitere Einträge zur Auswahl angeboten; verzweigt wird dann auf die jeweiligen Host-Adapter-spezifischen Start-Templates dieser Verbindungen.

access URL
Mit diesen Schaltflächen neben access URL werden die Methoden `send` und `receive` ausgeführt. Dabei werden die weiter unten definierten Verbindungsparameter mitgeschickt und berücksichtigt.

parameters
Mit `reset` werden alle Parameter in den gleichen Zustand versetzt, in dem sie vom Browser empfangen wurden. Mit `update` können die aktuellen Werte der Seite an WebTransactions geschickt werden, ohne dass eine Kommunikation über HTTP stattfindet.

Im Abschnitt **connection parameters** werden die Attribute des Systemobjekts spezifiziert.

URL Hier wird der URL der gewünschten HTTP-Ressource angegeben.

USER, PASSWORD
ggf. eingegebene Werte für `USER` und `PASSWORD` werden zum Zugriff auf geschützte Ressourcen mitgeschickt.

PROXY, PROXY_PORT
Soll die Verbindung über einen Proxy-Server erfolgen, so ist die Internet-Adresse oder ein symbolischer Name unter `PROXY` einzutragen, `PROXY_PORT` gibt den Port des Proxy-Servers an.

PROXY_USER, PROXY_PASSWORD
ggf. eingegebene Werte für `PROXY_USER` und `PROXY_PASSWORD` werden zum Zugriff über einen Proxy-Rechner mitgeschickt.

TIMEOUT_HTTP
Hier kann die Wartezeit auf die vom HTTP-Host-Adapter empfangenen Antworten eingestellt werden.

SSL_CERT_FILE
Hier geben Sie den Dateinamen an, in dem sich ein zu verwendendes Client-seitiges Zertifikat befindet. Wenn der Dateiname nicht absolut angegeben ist, ist er relativ zum Basis-Verzeichnis.

SSL_KEY_FILE
Hier geben Sie den Dateinamen an, in dem sich der private Schlüssel zu dem Zertifikat (siehe CERT_FILE) befindet. Wenn der Dateiname nicht absolut angegeben ist, ist er relativ zum Basis-Verzeichnis

SSL_PASSPHRASE
Hier geben Sie ein Kennwort für die Verwendung mit dem privaten Schlüssel an.

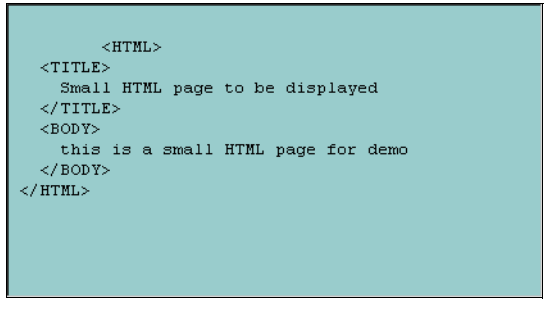
SSL_PROTOCOL
Hier wählen Sie aus, welche SSL- bzw. TLS-Version aktiviert werden soll. OpenSSL unterstützt das SSL-Protokoll in den Versionen 2 und 3 und das TLS-Protokoll Version 1. Mögliche Werte sind:
- `SSLv2`
- `SSLv3`
- `TLSv1`
- `All` (Voreinstellung)

Mehrere Protokolle können durch Leerzeichen getrennt angegeben werden.

Im Abschnitt **host objects** wird das Hostobjekt `sendData` angezeigt. Die Attribute `ContentType` und `Body` von `sendData` können auch modifiziert werden.

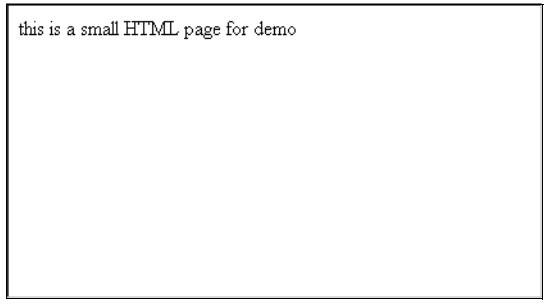
Falls bereits Daten empfangen wurden, d.h. `receiveData` existiert, dann wird zusätzlich der Inhaltstyp von `receiveData` ausgegeben und es stehen in diesem Abschnitt zwei weitere Schaltflächen zur Verfügung:

Text
stellt den Inhalt von `receiveData.Body` in einem eigenen Fenster als HTML-Quellcode dar:



HTML

stellt den Inhalt von `receiveData.Body` in einem eigenen Fenster so dar, wie dies im Browser aussehen würde:



Außerdem werden im Feld **receiveData.Header** die zurückgelieferten Header-Felder angezeigt.

3.4.2 Einfaches Start-Template (StartTemplateHTTP.htm)

Mit dem HTTP-Host-Adapter wird ein einfaches WTML-Dokument ausgeliefert, das alle Anweisungen zur Definition der Verbindungsparameter und der Attribute der Host-Objekte prototypisch enthält. Aus diesem Template lassen sich durch Anpassung schnell WTScripts zum Zugriff auf HTTP-Ressourcen ableiten.

Dieses Template kann sowohl als Start-Template (z.B. für eine Einschritt-Transaktion) als auch mitten in einer Sitzung benutzt werden. Das folgende Beispiel ist reichlich kommentiert, sodass Sie die Bedeutung der einzelnen Abschnitte leicht identifizieren können.

Aufbau des Templates

Der Anfang des Start-Templates besteht aus HTML-Kommentaren, die u.a. den Aufbau der URL zeigen, über welche Sie dieses Start-Template aufrufen können:

```
<wtRem>
   ------------------------------------------------------------------------
   StartTemplateHTTP
   ------------------------------------------------------------------------
   may be called by the url:
      http://<host>/<urlprefix>/WTPublish.exe/<basedirectory>?<starttemplate>
   (e.g.  http://localhost/cgi-bin/WTPublish.exe/c:/base_http?StartTemplateHTTP )
   this document also may be included for an HTTP call while running a
   WebTransactions application:
      <wtInclude name="StartTemplateHTTP">
   ------------------------------------------------------------------------
   Copyright (c) by Fujitsu Technology Solutions GmbH 2010
</wtRem>
```

Der eigentliche Inhalt dieses Start-Templates besteht aus einem OnCreate-Script:

```
...
<wtoncreatescript>
<!--
 ...
//-->
</wtoncreatescript>
```

Code des Templates StartTemplateHTTP.htm

Das OnCreate-Script ist in drei Bereiche unterteilt:

Der erste Teil erzeugt ein neues Kommunikationsobjekt und damit implizit auch ein privates System-Objekt. Außerdem aktiviert er den HTTP-Host-Adapter (`open`).

```
<wtoncreatescript>
<!--
  // ********************************************************************
  // global part of starttemplate which is not dependent of protocol type.
  // these attributes are not relevant, if you use this template as include
  // for a running session.
  // ********************************************************************

  // If a specific style or language is required, you have to set the following
  // attributes to a suitable value. Default value is empty for forms directory.

  // WT_SYSTEM.STYLE = "";
  // WT_SYSTEM.LANGUAGE = "";

  // Application timeout and user timeout are set to standard values.

  // WT_SYSTEM.TIMEOUT_APPLICATION = "120";
  // WT_SYSTEM.TIMEOUT_USER        = "600";

  // If a template should be displayed after TIMEOUT_USER and before
  // terminating the session you have to set the TIMEOUT_FORMAT attribute
  // to that specific template name.

  // WT_SYSTEM.TIMEOUT_FORMAT   = "";

  // Now we have to create a new communication object. This is mandatory also.
  // Further we use the private wt_system object (WT_HOST.<myHandle>.WT_SYSTEM).

  // ****************************************************************
  // specific part of starttemplate for protocol type HTTP
  // ****************************************************************
  // A new communication object has to be created.
  // The name "myHTTPConn" of the object may be changed to any valid symbol.
  host = new WT_Communication("myHTTPConn");

  // Further the private system object (host.WT_SYSTEM) is used.
  host_system                          = host.WT_SYSTEM;

  // The call of open method enables the HTTP adapter
  host.open("HTTP");
```

Der zweite Teil enthält die Zuweisungen für die Attribute des privaten System-Objekts. Diese müssen auf passende Werte gesetzt werden.

```
  // The URL specifies the desired HTTP resource
  // general format is
  // [http[s]:][//][user[:password]@]hostname[:port][/pathinfo[?query]]

     host_system.URL                   = "myURL";
  //   = "localhost/wtadm.htm"  // example for GETting a static file
  //   = "//localhost/Scripts/WTPublish.exe/d:/inetpub/wwwroot/basedir/?wtstartHTTP";
                                // example for calling cgi program with method GET
  //   =
"//localhost/Scripts/WTPublish.exe/d%3A%5Cinetpub%5Cwwwroot%5Cbasedir?wtstartHTTP";
                                // same example but with escape sequences for :(=%3A)
and \(=%5C)
                                // maybe necessary for HTTP daemon
  //   = "//localhost/Scripts/WTPublish.exe/startup";
                                // same example using metod POST
                                // corresponds with content of sendData object

  // For restricted HTTP resources user authorization has to be done.

  // host_system.USER                  = "";
  // host_system.PASSWORD              = "";

  // Address and port of HTTP proxy might be specified.

  // host_system.PROXY                 = "";
  // host_system.PROXY_PORT            = "";

  // For restricted PROXY access user authorization has to be done.

  // host_system.PROXY_USER            = "";
  // host_system.PROXY_PASSWORD        = "";

  // Timeout for HTTP may be determined.

  // host_system.TIMEOUT_HTTP          = "60";

  // if SSL shall be used (see URL: 'https: ...'), the following attributes may
configure the SSL connection
  // host_system.SSL_PROTOCOL          = 'All'; // SSLv2, SSLv3, TLSv1 or All, multiple
values may be defined separated by space
  // host_system.SSL_CERT_FILE         = '';    // file name of certifcate
  // host_system.SSL_KEY_FILE          = '';    // file name of matching key, may be
omitted, if SSL_CERT_FILE contains both: certificate and key
  // host_system.SSL_PASSPHRASE        = '';    // passphrase if needed for the
certifcate

  // For invocation of POST method host object sendData has to be defined.

  // host.sendData                     = new Object();
```

```
  // host.sendData.ContentType         = "application/x-www-form-urlencoded";
  // host.sendData.Body                =
"WT_SYSTEM_BASEDIR=d%3A%5Cinetpub%5Cwwwroot%5Cbasedir&WT_SYSTEM_FORMAT=wtstartHTTP";
            // see comment on system attribute URL
```

Der dritte Teil enthält schließlich Aufrufe der Methoden `send` und `receive`.

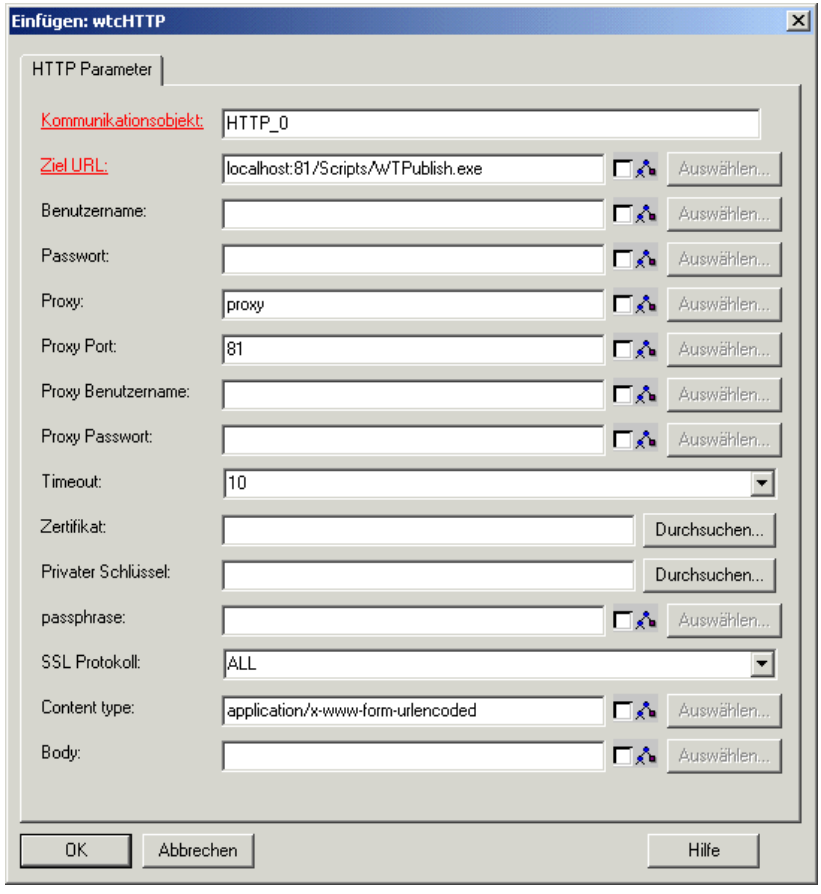
```
  // Now the HTTP resource is requested.
  host.send();
  host.receive();

  // The response receiveData has to be analysed.
  // document.write( host.receiveData.Body );

  // Close HTTP communication
  host.close();

  // If this was the whole job, terminate the session
  // exitSession();

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

3.5 Neues HTTP-Kommunikationsobjekt anlegen (wtcHTTP)

Um in einem Template ein neues HTTP-Kommunikationsobjekt anzulegen, und damit eine Verbindung zu einem HTTP-Server aufzubauen, wird das WTBean `wtcHTTP` mit ausgeliefert. Dieses WTBean können Sie auch dazu nutzen, um mehrere Verbindungen parallel zu öffnen. `wtcHTTP` ist ein inline WTBean, siehe hierzu auch das WebTransactions-Handbuch „Konzepte und Funktionen“.

Damit Sie auf inline WTBeans zugreifen können, muss eine Verbindung zur Host-Anwendung bestehen und das Template geöffnet sein, in das Sie das WTBean einfügen wollen.

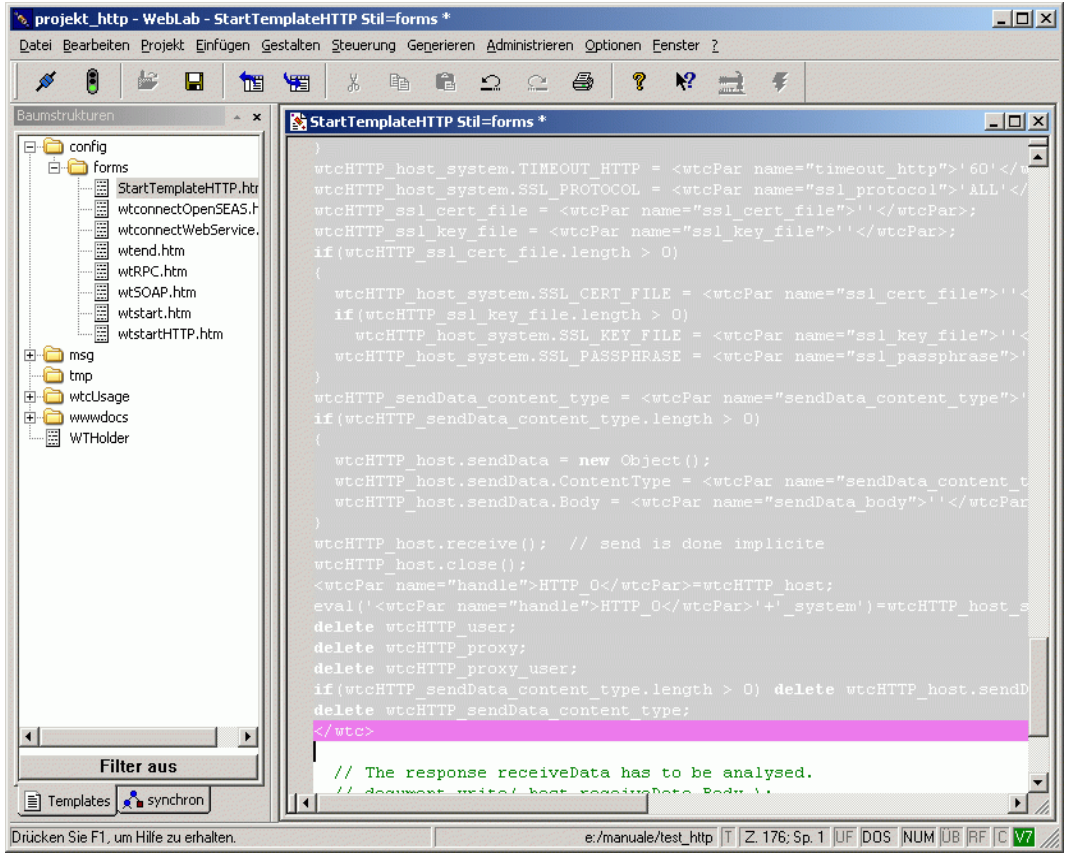
Mit dem Befehl **Einfügen/WTBean/wtcHTTP** rufen Sie das WTBean zur Bearbeitung auf. WebLab generiert das Dialogfeld **Einfügen:wtcHTTP**:



In diesem Dialogfeld können Sie die Parameter für das zu erzeugende Kommunikationsobjekt bearbeiten. Die Parameter, die Sie in diesem Dialogfeld in Eingabefelder eingeben können, entsprechen den Parametern des Start-Templates, siehe hierzu auch Abschnitt „HTTP-spezifisches Start-Template des Start-Template-Sets (wtstartHTTP.htm)“ auf Seite 41. Die Pflichtparameter sind rot dargestellt.

Wenn Sie die Parameter versorgt haben und Ihre Angaben mit **OK** bestätigen, wird aus den Parametern und der Beschreibungsdatei der Code des WTBeans erzeugt und in das geöffnete Template an der Cursor-Position eingefügt.

Das WTBean besteht aus geschützten und ungeschützten Code-Bereichen. Die geschützen Bereiche sind grau hinterlegt dargestellt. Auf diese Bereiche können Sie nur über die Oberfläche des WTBeans Einfluss nehmen. Sie wählen dazu im Kontextmenü der Startzeile des WTBeans (pink hinterlegt) den Befehl **WTBean bearbeiten** (siehe WebTransactions-Handbuch „Konzepte und Funktionen“).



4 Web-Services anschließen über SOAP

Dieses Kapitel beschreibt die Einbettung des SOAP-Protokolls (**S**imple **O**bject **A**ccess **P**rotocol) in WebTransactions:

- Konzept der SOAP-Einbettung in WebTransactions
- Klasse `WT_SOAP`
- Klasse `WT_SOAP_HEADER`

4.1 Konzept der SOAP-Einbettung in WebTransactions

Die Einbettung des SOAP-Protokolls in WebTransactions basiert auf dem WebTransactions-Host-Adapter HTTP sowie den Definitionen der Protokolle SOAP und WSDL (**W**eb **S**ervices **D**escription **L**anguage).

4.1.1 SOAP (Simple Object Access Protocol)

Das XML-basierte SOAP-Protokoll 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 einer im Web erreichbaren fernen Anwendung (Web-Service).

SOAP ist ein XML-basiertes Protokoll, das aus drei Teilen besteht:

- Spezifikation für einen Umschlag (Envelope)

  Der Umschlag definiert ein Regelwerk, das beschreibt,

  – **was** in einer Nachricht enthalten ist,
  – von **wem** die Nachricht **wie** verarbeitet werden soll und
  – **ob** einzelne Daten der Nachricht optional sind oder angegeben werden müssen (mandatory).

  Der Namensraum-Bezeichner (Namespace-Identifier) des Envelope lautet:

  ```
  "http://schemas.xmlsoap.org/soap/envelope"
  ```

  (Unter einem Namensraum ist hier die Menge von Namen zu verstehen, die in einem bestimmten Kontext gültig sind.)

- Satz von Codier- und Ordnungsregeln (Serialisation), der Instanzen von anwendungsspezifischen Datentypen beschreibt.

  Der Namespace-Identifier der Serialisation lautet:

  ```
  "http://schemas.xmlsoap.org/soap/encoding"
  ```

- Konvention, um Remote Procedure Calls (RPC) und eventuelle Antworten auf diese RPCs darzustellen.

SOAP-Services werden innerhalb von WSDL-Dokumenten beschrieben.

4.1.2 Beschreibung der SOAP-Services mit WSDL

WSDL (**W**eb **S**ervice **D**escription **L**anguage) bietet XML-Sprachregeln für die Beschreibung der Fähigkeiten von Web-Services. In einem einzelnen WSDL-Dokument können mehrere SOAP-Services beschrieben sein. Diese SOAP-Services können wiederum auf mehreren Servern verfügbar sein. Die Ein- und Ausgabeparameter der einzelnen Services lassen sich ebenfalls mit WSDL beschreiben.

Die folgende Abbildung skizziert den Aufbau eines WSDL-Dokuments.

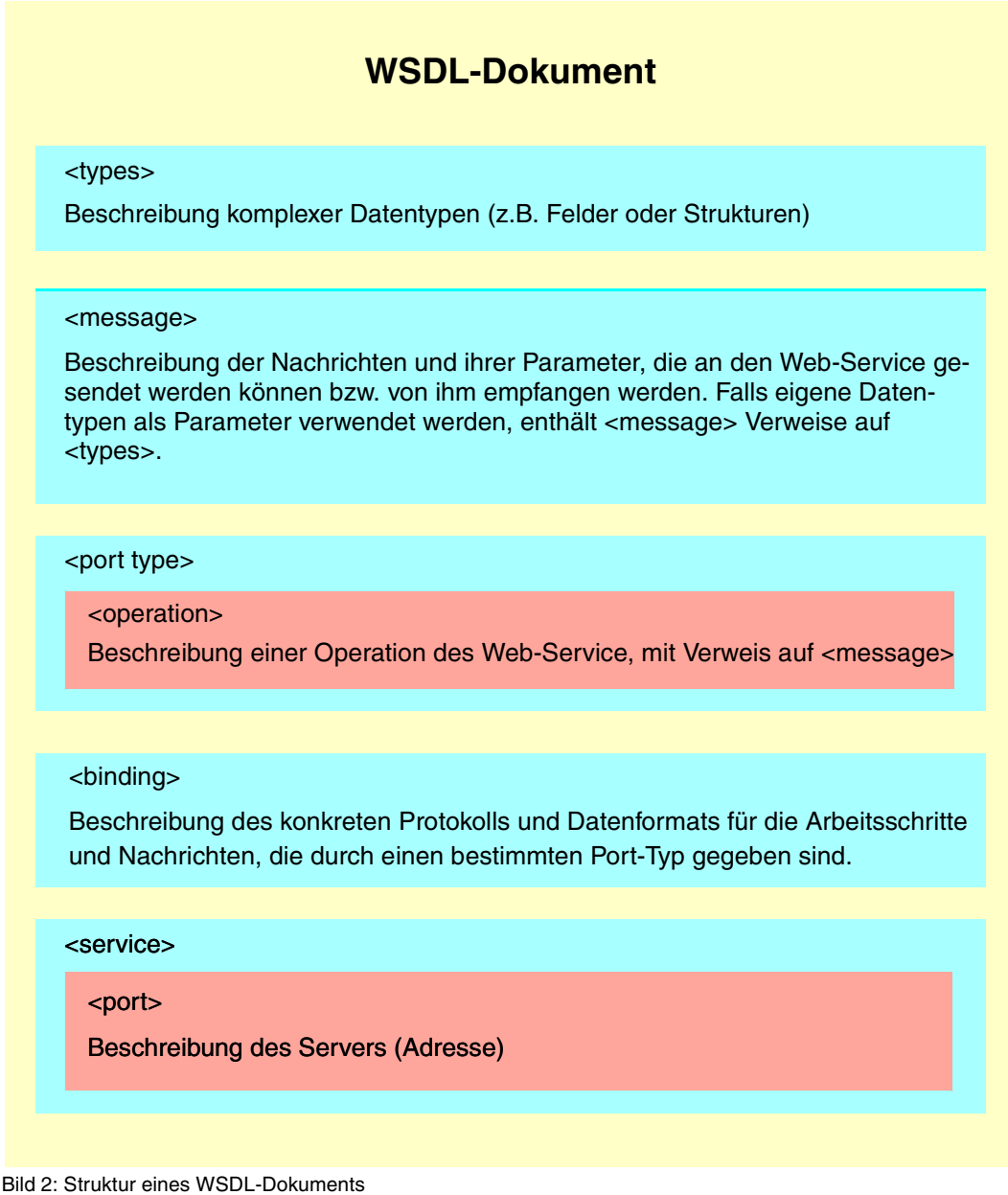


Bild 2: Struktur eines WSDL-Dokuments

4.1.3 UDDI (Universal Description, Discovery and Integration Project)

Das **U**niversal **D**escription, **D**iscovery and **I**ntegration Project (UDDI) ist ein umfassendes, plattform-unabhängiges System zur Dokumentation u.a. von Web-Services. Die Verzeichnisse von UDDI, in denen die Web-Services dokumentiert sind, sind für alle Web-Benutzer frei zugänglich.

4.1.4 SOAP-Unterstützung in WebTransactions

WebTransactions realisiert den client-seitigen Zugriff auf SOAP-Services via HTTP-Protokoll mit Hilfe der in WTML geschriebenen Klasse `WT_SOAP`. Für die SOAP-Unterstützung von WebTransactions werden somit nur der Host-Adapter HTTP und die Klasse `WT_SOAP` benötigt. Web-Services können Sie gemäß den SOAP-Standards V1.1 und V1.2 aufrufen.

Die SOAP-Services sind mittels einer WSDL beschrieben. WebTransactions kann die WSDL entweder aus einer lokalen Datei lesen oder aus dem Netz laden. Auf Basis dieser WSDL wird beim Anlegen eines `WT_SOAP`-Objekts der entsprechende Service als Proxy-Methode dieses Objekts verfügbar gemacht. Hat die Proxy-Methode Aufrufparameter, so werden für diese Parameter Proxy-Objekte angelegt. Bei Ausführung der Proxy-Methode wird die Anfrage-Nachricht über den HTTP-Host-Adapter an den SOAP-Server geschickt und die Antwortnachricht auf demselben Wege empfangen und in WTScript aufbereitet.

Die folgende Abbildung veranschaulicht diesen Sachverhalt.

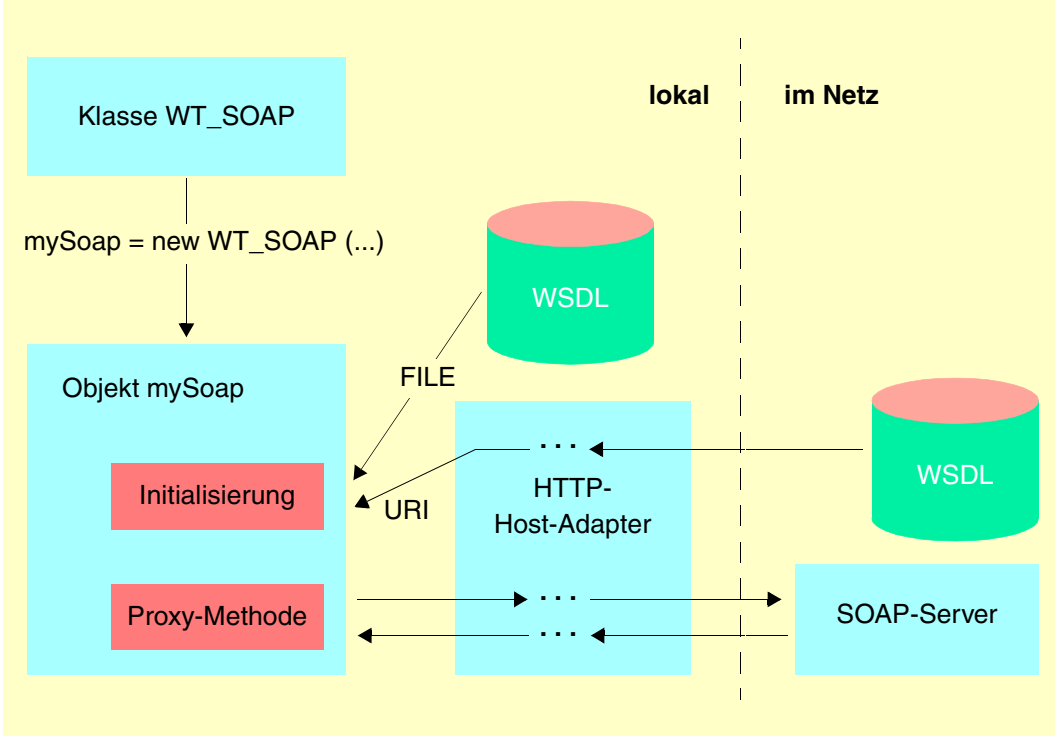


Bild 3: Client-seitige SOAP-Unterstützung mit Hilfe der Klasse WT_SOAP

Um einen Web-Service zu testen, können Sie mit WebLab eine Defaultoberfläche für den Web-Service generieren (**Generieren/Templates/für WebServices**). Dieses Template können Sie aus dem allgemeinen Start-Template `wtstart.htm` aufrufen. Nähere Informationen zum Starten eines Web-Service finden Sie im WebTransactions-Handbuch „Web-Frontend für Web-Services“.

Die SOAP-Unterstützung ist Bestandteil jeder WebTransactions-Liefereinheit. Beim Erzeugen des Basisverzeichnisses für den HTTP-Host-Adapter (siehe Kapitel „Basisverzeichnis anlegen“ auf Seite 15) wird ein Verweis auf das Template `wtSOAP.htm` im Installationsverzeichnis angelegt.

4.2 WT_SOAP - Klasse für client-seitige Nutzung

Die Klasse `WT_SOAP` ist im Template `wtSOAP.htm` implementiert.
Das Template enthält den Konstruktor der Klasse `WT_SOAP`, der aus einer WSDL-Datei eines Web-Services eine Instanz der Klasse erzeugt. An dieser Instanz werden Proxy-Methoden zur Verfügung gestellt, die eine Operation des Web-Services aufrufen. Neben den Proxy-Methoden gibt es noch eine Reihe weiterer Methoden, die Einstellungen für ein Objekt der Klasse `WT_SOAP` vornehmen.

Die Klasse `WT_SOAP` verwendet die Ausnahmenbehandlung (Exception Handling) von WebTransactions (siehe WebTransactions-Handbuch „Template-Sprache“). Wenn Sie die Klasse `WT_SOAP` verwenden, empfiehlt es sich, die Methoden dieser Klasse (einschließlich des Konstruktors) innerhalb eines `try`-Blocks aufzurufen und Fehler in einem `catch()`-Block zu behandeln. Andernfalls werden Fehler immer bis an die Benutzeroberfläche weitergereicht.

Nach der Instanziierung der Klasse `WT_SOAP` stehen einige Attribute und Methoden direkt als Attribute und Methoden des `WT_SOAP`-Objekts zur Verfügung. Die Mehrzahl der Methoden des `WT_SOAP`-Objekts wird jedoch intern genutzt, d.h. während der Instanziierung oder der Ausführung von Proxy-Methoden (siehe Seite 68). In den nachfolgenden Abschnitten sind daher nur die Methoden und Attribute beschrieben, die Bestandteil der Anwenderschnittstelle sind.

Folgende Besonderheiten hinsichtlich der SOAP-Funktionalität sind bei der Klasse `WT_SOAP` zu beachten:

- Der SOAP-Header wird nicht automatisch aus der WSDL generiert. Sie können der Nachricht aber einen Header hinzu fügen, indem Sie diesen in `WT_SOAP.envelope.header` vor Aufruf der Methode einfügen oder Header der Klasse `WT_SOAP_HEADER` erzeugen, die automatisch übernommen werden.
- Die einfachen Datentypen `ID` und `IDREF` werden nicht unterstützt. Die entsprechenden Daten können redundant übergeben werden.
- Es werden nur Nachrichten unterstützt, die das "SOAP-Binding" verwenden und über `http` versendet werden.
- SOAP Body Encoding wird nur gemäß

  `"http://schemas.xmlsoap.org/soap/envelope"`

  unterstützt. Wenn ein anderes Schema für die Serialisation der Daten verwendet wird, kann die Klasse `WT_SOAP` als Basis für eine eigene Klasse dienen.
- Nur Datentyp-Beschreibungen in XML-Schema (XSD) werden automatisch unterstützt.

4.2.1 Struktur eines WT_SOAP-Objekts

Dieser Abschnitt erläutert die Struktur eines WT_SOAP-Objekts anhand der Darstellung eines Beispiels im Objektbaum von WebLab. Das WT_SOAP-Objekt wurde aus dem WSDL-Dokument generiert, das im Abschnitt „Beispiel: WSDL-Dokument“ auf Seite 64 dargestellt ist.

Die einzelnen Komponenten eines WT_SOAP-Objekts sind in den darauffolgenden Abschnitten (ab Seite 66) näher erläutert.

4.2.1.1 Darstellung im Objektbaum von WebLab

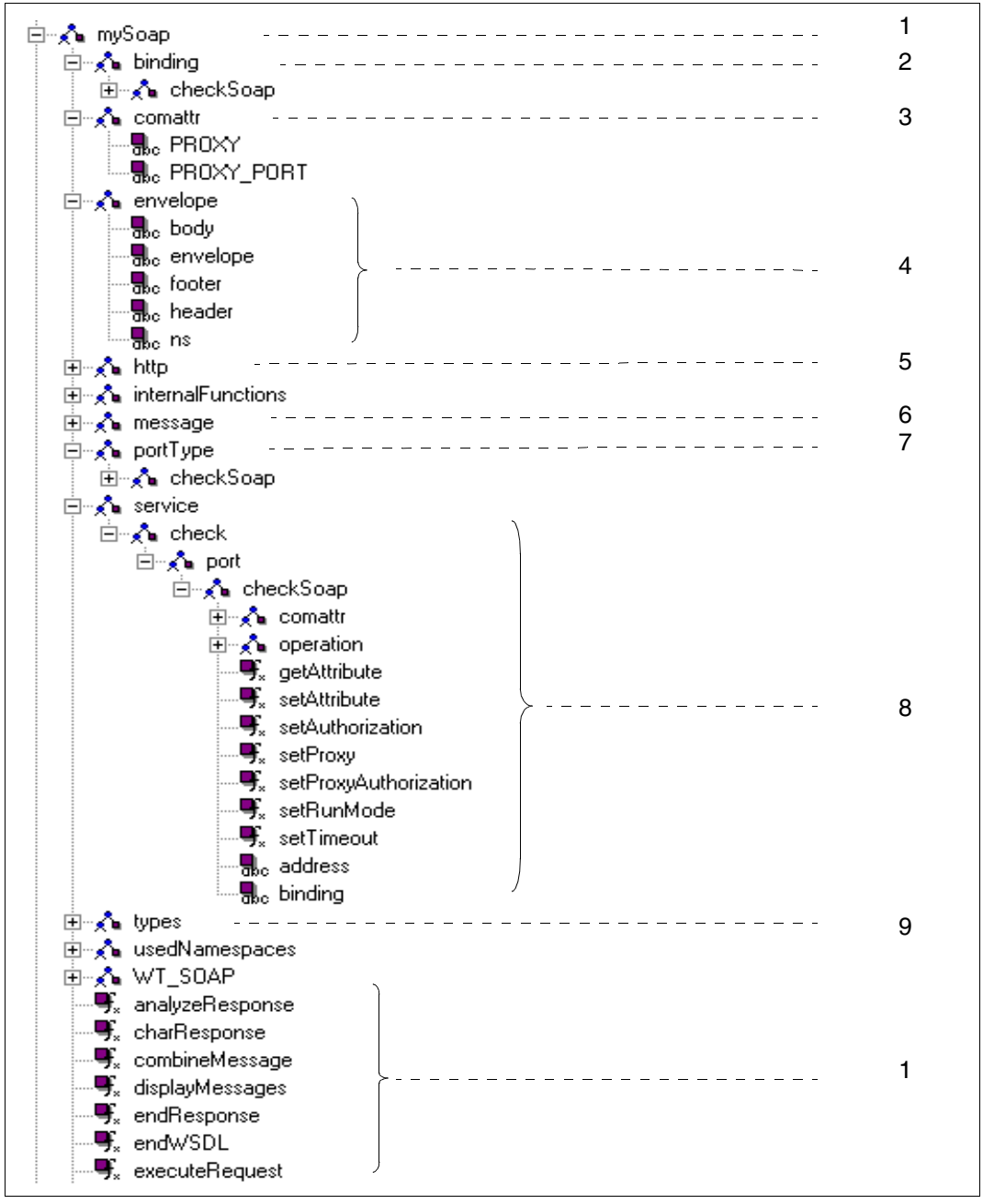


Bild 4: Darstellung eines WT_SOAP-Objekts im Objektbaum von WebLab

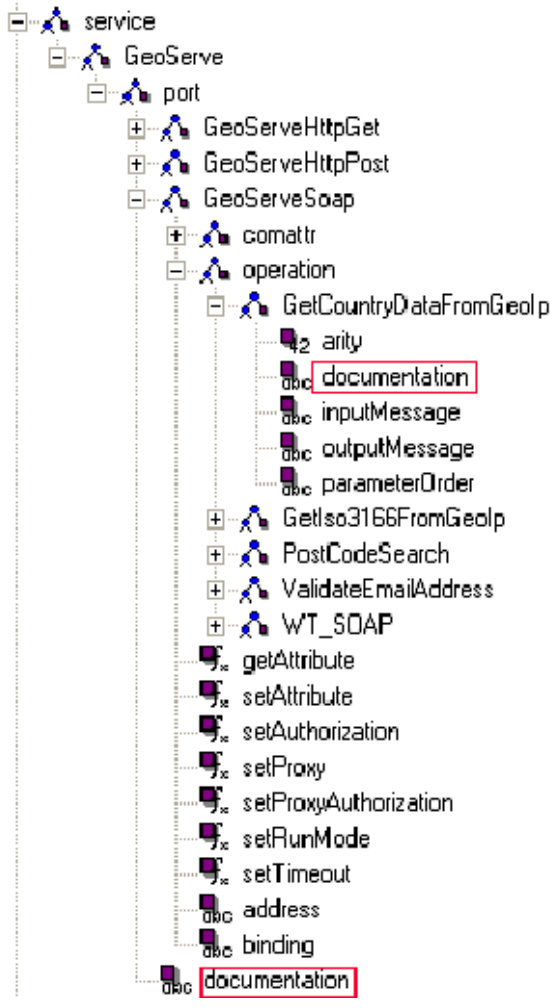
Erläuterung (siehe auch Abschnitt „Attribute an WT_SOAP“ auf Seite 89)

1. Objekt `mySoap` nach der Instanziierung. Die unter `WT_SOAP` definierten Methoden können auf Instanzen der `WT_SOAP`-Klasse und damit auch auf `mySoap` angewendet werden (siehe Abschnitte „Methoden eines Objekts der Klasse WT_SOAP“ auf Seite 71 und „Methoden zur Konfiguration des Zugriffs an der Unterklasse WT_SOAP_COM_FUNCTIONS“ auf Seite 84).
2. Abbild der `<binding>`-Section aus dem WSDL-Dokument.
3. Das Objekt `commattr` enthält die Attribute, die den HTTP-Zugriff steuern.
4. Das Objekt `envelope` enthält in den Attributen `envelope`, `header` und `body` die Textteile, aus denen die SOAP-Nachricht beim Aufruf einer Proxy-Methode oder der Methode `executeRequest` zusammengesetzt wird.
5. Das Objekt `http` enthält das Kommunikationsobjekt für den HTTP-Host-Adapter.
6. Das Objekt `message` ist Abbild aus der `<message>`-Section des WSDL-Dokuments.
7. Das Objekt `portType` enthält ein Objekt für jede Operation, die in der `<portType>`-Section der WSDL beschrieben ist. (hier `checkSoap`).
8. Das Objekt `service` enthält ein Objekt für jeden Service, den der Web-Service anbietet. Für jeden Service existiert in der WSDL eine eigene `<service>...</service>`-Section (im Beispiel: `check`). `check` enthält ein Objekt `port` und unterhalb dieses Objekts befinden sich die Informationen aus der `<port>`-Section des WSDL-Dokuments (hier `checkSoap`). `checkSoap` enthält seinerseits die Objekte `commattr`, `operation`, `adress` sowie einige Methoden, mit denen sich port-spezifische Einstellungen vornehmen lassen. Das wichtigste Objekt ist `operation`, denn hier befinden sich die Proxy-Methoden zum Aufruf der Operationen des Web-Services. Das Objekt `comattr` enthält die port-spezifischen Attribute für den HTTP-Zugriff, die beim Aufruf der Proxy-Methode verwendet werden. `adress` enthält die URI des Web-Services.

   In der WSDL können Sie bei der Definition eines Services und einer Operation Kommentare mittels eines `documentation`-Elementes einfügen. Diese Informationen werden in die Struktur von `WT_SOAP` aufgenommen. Beim Erzeugen von generischen Oberflächen für Web-Services ist es hilfreich, solche Informationen anzubieten. Deshalb werden die Kommentare an folgenden Stellen hinterlegt:
   – Wenn ein `documentation`-Element innerhalb des service-Tags in der WSDL gefunden wird, unter `WT_SOAP.service.`*<Name des Services>*`.port.documentation`.
   – Wenn in den operation-Tags in der WSDL ein entsprechender Kommentar gefunden wird, unter dem Namen der Operation.

   Innerhalb eines `documentation`-Elements kann Text, eine beliebige XML-Struktur oder beides vorkommen. In der Praxis kommt nur einfacher Text vor. Deshalb wird nur der Textanteil innerhalb des `documentation`-Elements als String in die Struktur von `WT_SOAP` übernommen.

Beispiel



9. Das Objekt `types` wird verwendet als Ablage für Datentypen, die in dem WSDL-Dokument beschrieben sind.

Web-Services auf verschiedenen Rechnern

Ein Web-Service kann auf verschiedenen Rechnern angeboten werden. Die `<service>`-Section des WSDL-Dokuments enthält dann mehrere `<port>`-Sections. Da die verschiedenen Server in der Regel unterschiedlich parametrisiert sind (Proxy etc.), spiegelt sich dies auch in der Struktur des `WT_SOAP`-Objekts unter `service`/*servicename*/`port` wider. So enthält das Objekt `port` in diesem Fall mehrere Objekte mit unterschiedlichen port-spezifischen Einstellungen.

Ebenso müssen Proxy-Methoden zum Aufruf des Web-Services, die im Objekt-Baum unter `service`/*servicename*/`port`/*portname*/`operation` angeboten werden, mehrfach vorhanden sein. Dies ist jedoch nicht redundant im eigentlichen Sinn, da jeder *portname* mit einer anderen Adresse im Internet verbunden ist und daher der Aufruf an einem anderen Port auch mit einem anderen Server im Internet kommuniziert.

Die Methoden und Attribute zum Einstellen von Rechnern und Zugangsberechtigungen für den Zugriff auf den Web-Service stehen sowohl unter dem `WT_SOAP`-Objekt als auch unterhalb von `service`/*servicename*/`port`/*portname* zur Verfügung. Auf diese Weise können Sie diese Einstellungen wahlweise global oder port-spezifisch vornehmen.

4.2.1.2 Beispiel: WSDL-Dokument

Im Folgenden ist ein Web-Service dargestellt, mit dem die Rechtschreibung von Texten überprüft werden kann. Dieses WDSL-Dokument bildet die Basis für den auf Seite 60 abgebildeten Objektbaum. Das WTScript im Abschnitt „Beispiel: Rechtschreibung von Texten überprüfen“ auf Seite 93 verwendet diese WSDL und ruft den Web-Service auf.

```
<?xml version="1.0" encoding="utf-8"?>
<definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:s="http://www.w3.org/2001/XMLSchema"
   xmlns:s0="http://ws.cdyne.com/"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
   xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
   targetNamespace="http://ws.cdyne.com/"
   xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://ws.cdyne.com/">
      <s:element name="CheckTextBody">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="BodyText" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="LicenseKey" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="CheckTextBodyResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="DocumentSummary"
                       nillable="true" type="s0:DocumentSummary" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="DocumentSummary">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="unbounded" name="MisspelledWord"
                     type="s0:Words" />
          <s:element minOccurs="0" maxOccurs="1" name="ver" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="body" type="s:string" />
          <s:element minOccurs="1" maxOccurs="1" name="MisspelledWordCount"
                     type="s:int" />
        </s:sequence>
      </s:complexType>
      <s:complexType name="Words">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="unbounded" name="Suggestions"
                     type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="word" type="s:string" />
          <s:element minOccurs="1" maxOccurs="1" name="SuggestionCount" type="s:int" />
        </s:sequence>
```

```
      </s:complexType>
     <s:element name="DocumentSummary" nillable="true" type="s0:DocumentSummary" />
      <s:element name="string" nillable="true" type="s:string" />
    </s:schema>
  </types>
  <message name="CheckTextBodySoapIn">
    <part name="parameters" element="s0:CheckTextBody" />
  </message>
  <message name="CheckTextBodySoapOut">
    <part name="parameters" element="s0:CheckTextBodyResponse" />
  </message>
  <portType name="checkSoap">
    <operation name="CheckTextBody" parameterOrder="parameters">
      <input message="s0:CheckTextBodySoapIn" />
      <output message="s0:CheckTextBodySoapOut" />
    </operation>
  </portType>
  <binding name="checkSoap" type="s0:checkSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
    <operation name="CheckTextBody">
      <soap:operation soapAction="http://ws.cdyne.com/CheckTextBody"
                      style="document" />
      <input>
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal" />
      </output>
    </operation>   </binding>
  <service name="check">
    <port name="checkSoap" binding="s0:checkSoap">
      <soap:address location="http://ws.cdyne.com/SpellChecker/check.asmx" />
    </port>
  </service>
</definitions>
```

4.2.2 Konstruktor der Klasse WT_SOAP

Der Konstruktor analysiert die übergebene WSDL-Datei und erzeugt daraus ein WTScript-Objekt. Zusätzlich wird unter `WT_HOST` ein Kommunikationsobjekt mit dem Namen `WT_SOAP_`*erste-freie-nummer* angelegt. Über dieses Kommunikationsobjekt werden sämtliche HTTP-Zugriffe abgewickelt, u.a. das Laden des WSDL-Dokuments als URI (**U**niform **R**esource **I**dentifier).

`WT_SOAP()`
`WT_SOAP(`*init_document*`)`
`WT_SOAP(`*init_document*`, `*http_proxy_host*`)`
`WT_SOAP(`*init_document*`, `*http_proxy_host*`, `*http_proxy_port*`)`

Rückgabewert

Instanz der Klasse `WT_SOAP` für die SOAP-Unterstützung

Parameter

init_document
: spezifiziert den URI oder die Datei, die das WSDL-Dokument enthält. Dateinamen müssen relativ zum Basisverzeichnis angegeben werden.

http_proxy_host
: spezifiziert den PROXY-Host, der bereits beim Laden des WSDL-Dokuments und darüber hinaus auch für den eigentlichen SOAP-Request vom HTTP-Host-Adapter verwendet werden soll.

http_proxy_port
: spezifiziert den PROXY-Port, der bereits beim Laden des WSDL-Dokuments und darüber hinaus auch für den eigentlichen SOAP-Request vom HTTP-Host-Adapter verwendet werden soll.

Beispiel

```
//Initialize WT_SOAP-Object from file (file is in
                      <basedir>/wsdl/check.wsdl)
mySoap = new WT_SOAP('wsdl/check.wsdl', 'proxy','81');
```

Aufruf des Konstruktors ohne Parameter

Werden für eine Initialisierung neben *http_proxy_host* und *http_proxy_port* noch weitere Einstellungen für das Laden des WSDL-Dokuments benötigt, dann müssen Sie den Konstruktor ohne Argumente aufrufen. Danach benutzen Sie die Methoden der Klasse `WT_SOAP_COM_FUNCTIONS` (siehe Seite 84), um alle benötigten Einstellungen vorzunehmen. Danach initialisieren Sie das Objekt mit Hilfe der Methode `WT_SOAP.initFromWsdlUri` (siehe Seite 71).

Beispiel

```
mySoap = new WT_SOAP();
mySoap.setProxy ('proxy.company.com','81');
mySoap.setProxyAuthorization('puser','ppass');
mySoap.initFromWsdlUri('http://url');
```

Exceptions

Im Fehlerfall kann der Konstruktor die Exceptions `SOCKET`, `HTTP`, `FILE` oder `WSDL` werfen (siehe Abschnitt „Exceptions“ auf Seite 87).

4.2.3 Proxy-Methoden

Die bei der Initialisierung des `WT_SOAP`-Objekts zum Aufruf des Web-Services via Interpretation des WSDL-Dokuments dynamisch generierten Methoden entstehen gemäß der Struktur des WSDL-Objekts unterhalb von:

`service/`*`servicename`*`/port/`*`portname`*`/operation`

(Siehe „Darstellung im Objektbaum von WebLab“ auf Seite 60.)

Bietet der Web-Service mehrere Funktionen gleichen Namens an, so werden diese als Array angelegt. Die einzelnen Methoden sind als Elemente dieses Array zugreifbar.

Aufruf und Parameterübergabe bei einer Proxy-Methode

Um die Schreibweise beim Aufruf einer Proxy-Methode abzukürzen, können Sie auch eine Referenz anlegen.

Beispiel

```
myOperations =
mySoap.service.check.port.checkSoap.operation;
myOperations.CheckTextBody(. . .);
```

Parameter werden beim Aufruf der Proxy-Methode wie folgt übergeben:

– Sind die Parameter von einfachem Datentyp, dann können sie „by value“ als Argumente beim Aufruf der Proxy-Methode übergeben werden, z.B.:

```
myOperations.SendMail('recipients@domain','sender@domain,'subject',
                                                    'this is the Message');
```

– Für komplexe Datentypen werden entsprechend strukturierte Proxy-Objekte als globale Objekte unter dem globalen Objekt `proxyObjects` angelegt, die mit den konkreten Daten gefüllt und als Argument beim Aufruf übergeben werden können. Der Name dieser Proxy-Objekte wird aus dem Namen der zu sendenden Nachricht (siehe Attribut `inputMessage`) und dem Namen des Parameters zusammengesetzt.

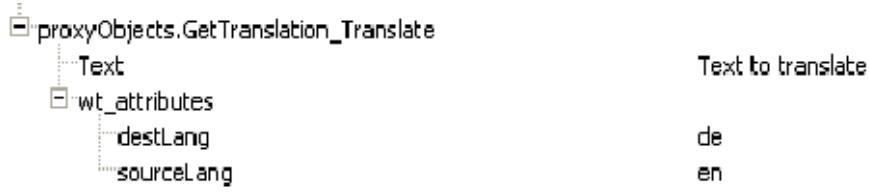
– Attribute in einer Nachricht oder Antwort werden im Objekt `wt_attributes` abgelegt. Das hat an der Benutzeroberfläche folgende Auswirkungen:

  – Die Proxy-Objekte können an jeder Stelle das Objekt `wt_attributes` enthalten, das die Attribute beinhaltet.

    Wenn es sich bei dem Element um einen einfachen Datentyp handelt, wird dieser in den entsprechenden Objekt-Datentyp (String, Boolean, Number) umgewandelt, um die Attribute unterbringen zu können.

Beispiel

Der Eingabeparameter einer Methode zum Übersetzen von Text – Quell- und Zielsprache in Attributen – sieht folgendermaßen aus:



Daraus wird folgende Nachricht erzeugt:

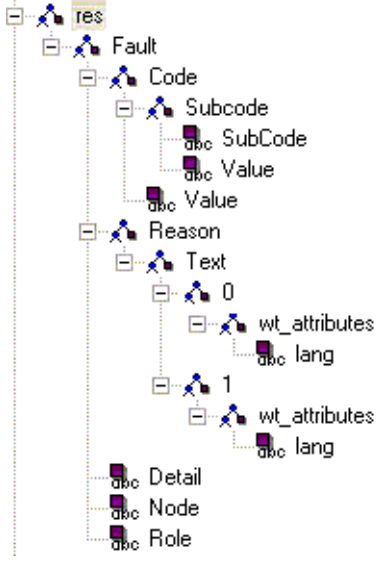
….

```
<wt_tns0:GetTranslation>
   <Translate destLang="de" sourceLang="en">
      <Text>Text to translate</Text>
   </Translate>
</wt_tns0:GetTranslation>
```

…

– Bei der Analyse der Antwort werden Attribute ebenfalls unter dem Objekt `wt_attributes` abgelegt.

Beispiel



Erzeugen der SOAP-Nachricht ohne sie zu senden

Benötigt der Web-Service eine Funktionalität, die von der Klasse `WT_SOAP` nicht unterstützt wird, dann rufen Sie die Proxy-Methode im Modus `PREPARE` auf (siehe Abschnitt „Methode setRunMode“ auf Seite 72). In diesem Fall präpariert die Proxy-Methode die HTTP-Nachricht, ohne sie zu senden. Durch Manipulation der Daten unter `WT_SOAP.envelope` können Sie die Nachricht nachbearbeiten und anschließend mit `WT_SOAP.executeRequest` versenden. Das Ergebnis kann mit `WT_SOAP.anlyseResponse` in eine WTScript-Datenstruktur umgewandelt werden.

Ein Beispiel finden Sie im Abschnitt „Methode setRunMode“ auf Seite 72.

Setzen von HTTP-Headern

Beim Ausführen einer Proxy-Methode werden automatisch die HTTP-Header `Content-Type` und `SOAPAction` erzeugt.

Wenn Sie der Nachricht weitere HTTP-Header hinzufügen wollen, können Sie diese mit der Funktion `setHTTPHeader()` hinzufügen. Nach dem Aufruf dieser Funktion wird jeder weiteren Nachricht dieser Header mitgegeben.

`setHTTPHeader (`*name*`, `*value*`)`
`setHTTPHeader (`*headerObj*`)`

---

name Name des Headerfeldes

value Wert des Headerfeldes

headerObj
Objekt mit den Attributen *name* und *value*, die Name und Wert des gewünschten Header-Feldes enthalten.

Rückgabewert einer Proxy-Methode

Eine Proxy-Methode liefert als Ergebnis ein Objekt zurück, das die Antwort des SOAP-Services enthält:

- Bei Erfolg liefert die Proxy-Methode eine Struktur zurück, die der Definition von `<output message=`*outputmessage*`>` im zugehörigen WSDL-Dokument entspricht.
- Im Fehlerfall sollte der Web-Service eine Fault-Nachricht zurückschicken, die in eine entsprechende Struktur verwandelt wird. Die Struktur enthält neben dem Pflichtattribut `faultcode` die optionalen Attribute `faultstring`, `faultfactor` und `detail`.

  Sendet der Web-Service eine Nachricht, deren Aufbau nicht dem einer Fault-Nachricht entspricht, dann wird der Rückgabewert der Proxy-Methode analog der zurückgelieferten Nachricht strukturiert.

Exceptions

Im Fehlerfall können die Proxy-Methoden die Exceptions SOCKET, HTTP oder PARAMETER werfen (siehe Abschnitt „Exceptions“ auf Seite 87).

4.2.4 Methoden eines Objekts der Klasse WT_SOAP

Die Methoden, die in diesem Abschnitt beschrieben werden, stehen nach der Instanziierung der Klasse WT_SOAP direkt an der Instanz zur Verfügung. Zusätzlich zu diesen Methoden können auch alle Methoden der Klasse WT_SOAP_COM_FUNCTIONS (siehe Abschnitt „Methoden zur Konfiguration des Zugriffs an der Unterklasse WT_SOAP_COM_FUNCTIONS“ auf Seite 84) an einem Objekt der Klasse WT_SOAP aufgerufen werden.

4.2.4.1 Methode initFromWSDLUri

Es ist möglich, dass zur Erzeugung einer Instanz der Klasse WT_SOAP dem Konstruktor nicht die entfernte WSDL-Datei als Parameter übergeben werden kann. Diese Situation ergibt sich beispielsweise dann, wenn zur Übertragung der WSDL-Datei zusätzliche Parameter an der HTTP-Schnittstelle benötigt werden. In solchen Fällen können Sie mit der Methode initFromWSDLUri die Analyse der WSDL-Datei nachholen.

---

initFromWsdlUri(*uri*)

Rückgabewert

kein Rückgabewert

Parameter

uri spezifiziert den URI (**U**niform **R**esource **I**dentifier) des WSDL-Dokuments, das die Beschreibung der Web-Services enthält.

Exceptions

Im Fehlerfall kann initFromWsdlUri die Exceptions SOCKET, HTTP oder WSDL werfen (siehe Abschnitt „Exceptions“ auf Seite 87).

Beispiel

```
mySoap = new WT_SOAP();
mySoap.setProxy ('proxy.company.com','81');
mySoap.setProxyAuthorization('puser','ppass');
mySoap.initFromWsdlUri('http://url');
```

4.2.4.2 Methode setRunMode

Die Methode `setRunMode` legt den Modus für die Ausführung von Proxy-Methoden fest (siehe Abschnitt „Proxy-Methoden“ auf Seite 68).

```
setRunMode("mode")
```

Rückgabewert

kein Rückgabewert

Parameter

mode spezifiziert den Ausführungsmodus von Proxy-Methoden.

Für *mode* können Sie folgende Werte angeben:

- `PREPARE`:
  Eine nachfolgend aufgerufene Proxy-Methode bereitet lediglich alle Datenstrukturen für den HTTP-Zugriff vor, führt aber den Zugriff nicht durch.
- `DOCUMENT`
  Die Parameter der Proxy-Methode werden als XML-Dokumente verstanden und unverändert direkt nach `<Body>` in die SOAP-Nachricht eingefügt. Auch das Ergebnis wird nicht auf WTScript-Datentypen abgebildet, sondern als `string` zur Verfügung gestellt.
- `RUN`
  Der Modus wird wieder zurückgesetzt. D.h., die Proxy-Methoden führen den Web-Service aus. Die Parameter müssen als WTScript-Datentypen übergeben werden.

Exceptions

Die Methode `setRunMode` wirft keine Exceptions.

Beispiel: setRunMode("PREPARE")

```
try{
mySoap = new WT_SOAP('wsdl/check.wsdl','proxy','81');
}
catch (e)
{
//do something
}
myOperations =
mySoap.service.check.port.checkSoap.operation;
mySoap.setRunMode('prepare'); //Just build the message, do not send it
```

```
//Fill Parameter with values
proxyObjects.CheckTextBodySoapIn_parameters.LicenceKey="0";
proxyObjects.CheckTextBodySoapIn_parameters.BodyText = "This is a sample text";
try{
//call proxy method
    myOperations.CheckTextBody(proxyObjects.CheckTextBodySoapIn_parameters);
}
catch (e)
{
}
//Set additional HTTP-header
mySoap.setHTTPHeader( 'Additional-Header-Field','something');
try {
    mySoap.executeRequest();//Send Request
    myRet=mySoap.analyzeResponse(); //and convert answer into a WTScript-Object
}
catch (e)
{
}
```

4.2.4.3 Methode executeRequest

Die Methode `executeRequest` führt den aktuellen Request aus. Die Nachricht wird aus den Attributen `envelope`, `header`, `HeaderBlocks` und `body` aus dem Objekt `envelope` der Instanz zusammengesetzt und über den HTTP-Host-Adapter verschickt (siehe Abschnitt „Attribute an WT_SOAP“ auf Seite 89). Sie benötigen diese Methode im Zusammenhang mit der Methode `setRunMode("PREPARE")`, die eine Nachricht an den Web-Service aufbaut aber nicht abschickt (siehe Abschnitt „Methode setRunMode“ auf Seite 72).

```
executeRequest()
```

Rückgabewert

kein Rückgabewert

Exceptions

Im Fehlerfall kann `executeRequest` die Exceptions `SOCKET`, `HTTP` oder `PARAMETER` werfen (siehe Abschnitt „Exceptions“ auf Seite 87).

4.2.4.4 Methode executeGetRequest

SOAP V1.2 empfiehlt die Verwendung der HTTP-Methode `GET`, falls Informationen nur geholt und keine Header übermittelt werden sollen. Die Antwort kommt wie gewohnt als SOAP Envelope zurück.

`WT_SOAP` bietet dafür die Methode `executeGetRequest`.

`WT_SOAP` ruft die angegebene URL auf.
Wenn der Server mit einer SOAP-Nachricht antwortet, wird die Antwort analysiert und daraus, wie beim Ausführen einer Proxy-Methode, ein WTScript-Objekt erzeugt.
Wenn der Server nicht mit einer SOAP-Nachricht antwortet, sondern beispielsweise mit einem beliebigen HTML-Dokument, wird der Inhalt dieses Dokuments zurückgegeben.

```
executeGetRequest(url)
```

url URL, die den vollständigen Request enthält

Exceptions

Wenn Sie versuchen einen HTTP-Header zu setzen, der bereits von `WT_SOAP` gesetzt wird, wirft die Methode eine Exception `USAGE`.

4.2.4.5 Methode analyseResponse

Die Methode `analyseResponse` analysiert die Antwort des letzten Requests (d.h. den Inhalt von `<WT_SOAP>.http.receiveData.Body`) und baut daraus eine WTScript-Datenstruktur auf. Sie benötigen diese Methode im Zusammenhang mit der Methode `setRunMode("PREPARE")`, die eine Nachricht an den Web-Service aufbaut aber nicht abschickt und der Methode `executeRequest`, die eine Nachricht abschickt (siehe Abschnitt „Methode setRunMode“ auf Seite 72).

```
analyseResponse()
```

Rückgabewert

WTScript-Datenstruktur

Exceptions

Die Methode `analyseResponse` wirft keine Exceptions.

4.2.4.6 Methode setSOAPVersion

Da es bisher noch keinen von der W3C verabschiedeten Stand von WSDL gibt, der SOAP V1.2 unterstützt, werden Web-Services auch weiterhin in WSDL-Dateien der Version 1.1 beschrieben. Daher kann `WT_SOAP` anhand der WSDL-Datei nicht erkennen, ob eine Nachricht für die SOAP-Version 1.1 oder 1.2 erzeugt werden soll.
Die Methode `setSOAPVersion` definiert, in welchem Format Nachrichten ausgetauscht werden sollen.

```
setSOAPVersion(<version>)
```

Rückgabewert

eingestellte Version

Parameter

<version>

Für *version* können Sie folgende Werte angeben:

1.1	Erzeugt Nachrichten im alten SOAP-Format und erwartet Antworten im alten SOAP-Format. Dieser Wert ist voreingestellt.
1.2	Erzeugt Nachrichten im SOAP-Format der Version1.2 und erwartet Antworten im gleichen Format.

Wenn Sie einen anderen Wert angeben, wird der Standardwert `1.1` gesetzt.

Exceptions

Die Methode `setSOAPVersion` wirft keine Exceptions.

Beispiel

```
mySoap = new WT_SOAP('wsdl/test1.wsdl');  //Create WT_SOAP object
mySoap.setSOAPVersion("1.2");             //Set Version
```

4.2.4.7 Methode addHeader

Die Methode `addHeader` setzt den angegebenen Headerblock für die folgenden Nachrichten. Alle definierten Header werden in dem Array `WT_SOAP.envelope.HeaderBlocks` aufbewahrt. Beim Erzeugen einer Nachricht wird aus den Objekten XML erzeugt und in den Header eingefügt.

Einmal hinzugefügte Header bleiben für alle weiteren Aufrufe von Proxy-Methoden gültig. Wenn sie nicht mehr benötigt werden, müssen Sie sie mit `removeAllHeaders` entfernen.

```
addHeader(wtsoapHeader1[,wtsoapHeader2[,…]])
```

Rückgabewert

Index des ersten hinzugefügten Headers.

Parameter

wtsoapHeader1,
wtsoapHeader2,…
: Objekte der Klasse `WT_SOAP_HEADER`, die der Nachricht angefügt werden sollen.

4.2.4.8 Methode removeAllHeaders

Die Methode `removeAllHeaders` löscht alle Header.

```
removeAllHeaders()
```

4.2.4.9 Methode getHeaderObjects

Wenn die Antwort auf eine SOAP-Nachricht Informationen im SOAP-Header enthält, können Sie u.a. mit der Methode `getHeaderObjects` auf die Header zugreifen.

Die Methode `getHeaderObjects` gibt ein Array von Objekten zurück, wobei jedes Objekt eine Instanz der Klasse `WT_SOAP_HEADER` ist und ein Element innerhalb des SOAP-Headers repräsentiert.

```
getHeaderObjects()
```

Beispiel

Eine Antwort eines Web-Services enthält folgenden Header:

```
<env:Header>
  <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
          env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           env:mustUnderstand="true">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
   <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/employees"
          env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           env:mustUnderstand="true">
   <n:name>Heinz Mustermann</n:name>
  </n:passenger>
 </env:Header>
```

Der Aufruf

```
myObjTree= mySoap.getHeaderObjects();
```

liefert folgende Datenstruktur:

4.2.4.10 Methode getHeaderObjectTree

Wenn die Antwort auf eine SOAP-Nachricht Informationen im SOAP-Header enthält, können Sie u.a. mit der Methode `getHeaderObjectTree` auf die Header zugreifen.

Die Methode `getHeaderObjectTree` liefert ein WTScript-Objekt zurück, das den gesamten Header repräsentiert. Das Objekt wird auf die gleiche Weise erzeugt, wie das Antwortobjekt einer Proxymethode.

```
getHeaderObjectTree()
```

Beispiel

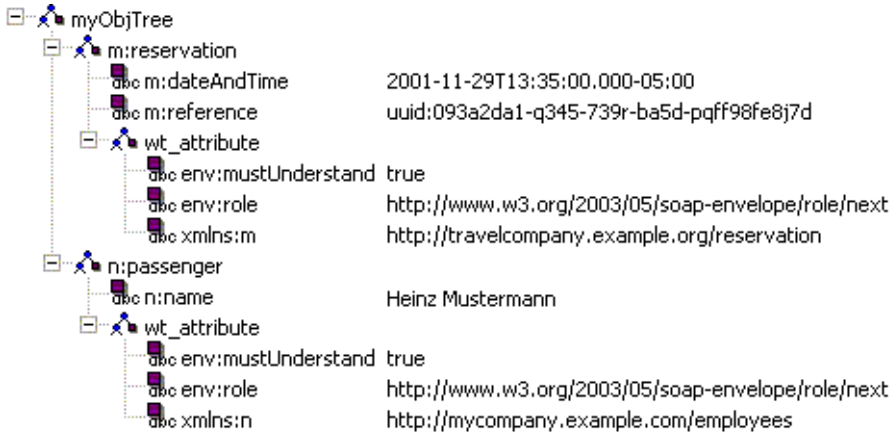
Eine Antwort eines Web-Service enthält folgenden Header:

```
<env:Header>
  <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
          env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           env:mustUnderstand="true">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
   <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/employees"
          env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           env:mustUnderstand="true">
   <n:name>Heinz Mustermann</n:name>
  </n:passenger>
 </env:Header>
```

Der Aufruf

```
myObjTree= mySoap.getHeaderObjectTree();
```

liefert folgende Datenstruktur:



4.2.4.11 Methode createProxysWithPrefix

Operationen, die ein Web-Service anbietet, können sowohl Eingabeparameter haben, die beim Aufruf angegeben werden müssen, als auch solche, die nicht unbedingt erforderlich sind. Diese Information kann mit der Methode `createProxysWithPrefix` auch in den Namen der Proxy-Objekte und Ihrer Attribute sichtbar gemacht werden.

Um dabei die Struktur der Proxy-Objekte nicht zu zerstören, wird diese Information als Präfix im Attributnamen abgelegt. Ein `m_` vor dem Namen kennzeichnet ein obligatorisches Attribut, ein `o_` ein optionales. Dieses Präfix spielt keine Rolle bei der Erzeugung von Nachrichten.

Die Methode `createProxysWithPrefix` schaltet diesen Modus ein oder aus.

Die Methode muss aufgerufen werden, bevor die WSDL analysiert wird. Deshalb muss der Konstruktor zunächst ohne Parameter aufgerufen werden, und die Funktion `initFromWsdl()` verwendet werden

`createProxysWithPrefix (`*bMode*`)`

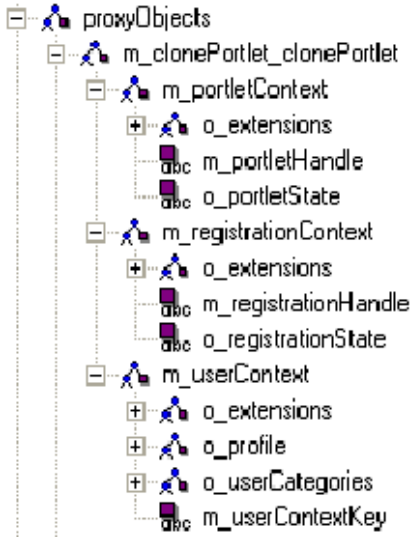
Parameter

bMode boolscher Wert.

true	Die Proxy-Objekte werden mit Präfixen erzeugt.
false	Die Proxy-Objekte werden ohne Präfixe erzeugt. Voreinstellung.

Beispiel

```
mySoap = new WT_SOAP();
mySoap.setProxy ('proxy.company.com','81');
mySoap.setProxyAuthorization('puser','ppass');
mySoap.createProxyWithPrefix(true);
mySoap.initFromWsdlUri('http://url');
```



4.2.5 Methoden zur Konfiguration des Zugriffs an der Unterklasse WT_SOAP_COM_FUNCTIONS

Die Methoden der Klasse `WT_SOAP_COM_FUNCTIONS` sind sowohl direkt unterhalb des `WT_SOAP`-Objekts als auch unterhalb eines jeden Ports verfügbar. Je nachdem, an welchem Objekt die folgenden Methoden aufgerufen werden, wirken sie entweder global oder nur spezifisch für den jeweiligen Port.

4.2.5.1 Methode setAuthorization

Falls ein Web-Service geschützt ist, verwenden Sie die Methode `setAuthorization`, um die Benutzerberechtigung für den Zugriff auf den Web-Service zu setzen. Diese Werte werden in die Attribute `USER` und `PASSWORD` des HTTP-Host-Adapters übernommen und bei Aufruf der Proxy-Methode vom HTTP-Host-Adapter mitgeschickt (siehe Abschnitt „Übersicht“ auf Seite 20).

`setAuthorization(`*user*`, `*password*`)`

Rückgabewert

kein Rückgabewert

Parameter

user spezifiziert den Benutzer, der mitgeschickt werden soll.

password
spezifiziert das Passwort für den Benutzer *user*.

4.2.5.2 Methode setProxy

Die Methode `setProxy` bestimmt Proxy-Host und Proxy-Port. Zu diesem Zweck setzt `setProxy` die Attribute `PROXY` und `PROXY_PORT` für diesen Host-Adapter.

`setProxy(`*http_proxy_host*`, `*http_proxy_port*`)`

Rückgabewert

kein Rückgabewert

Parameter

http_proxy_host.
spezifiziert den Wert für das Attribut `PROXY`.

http_proxy_port.
spezifiziert den Wert für das Attribut `PROXY_PORT`.

4.2.5.3 Methode setProxyAuthorization

Die Methode `setProxyAuthorization` vergibt eine Benutzerberechtigung für den Zugriff auf den Proxy-Host, über den der HTTP-Host-Adapter auf den Web-Service zugreifen soll. `setProxyAuthorization` setzt die Attribute `PROXY_USER` und `PROXY_PASSWORD` des HTTP-Host-Adapters.

`setProxyAuthorization(`*http_proxy_user*`, `*http_proxy_password*`)`

Rückgabewert

kein Rückgabewert

Parameter

http_proxy_user
spezifiziert den Wert für das Attribut `PROXY_USER`.

http_proxy_password
spezifiziert den Wert für das Attribut `PROXY_PASSWORD`.

4.2.5.4 Methode setTimeout

Die Methode `setTimeout` spezifiziert den Timeout-Wert für den Host-Adapter, indem sie das Attribut `TIMEOUT_HTTP` für den HTTP-Host-Adapter setzt.

`setTimeout (`*timeout*`)`

Rückgabewert

kein Rückgabewert

Parameter

timeout. spezifiziert den Timeout-Wert für den HTTP-Host-Adapter.

Bitte beachten Sie, dass das Attribut `TIMEOUT_HTTP` nicht größer als `TIMEOUT_APPLICATION` werden kann. Das heisst sie müssen eventuell auch das Attribut `TIMEOUT_APPLICATION` setzen.

4.2.6 Exceptions

Bei Fehlern, die bewirken, dass der SOAP-Service nicht antworten kann, wird eine Ausnahme (Exception) geworfen und die Proxy-Methode liefert kein Ergebnis. Solche Fehlerursachen können z.B. Parameter-Fehler beim Aufruf der Proxy-Methode oder eine nicht zustande gekommene Verbindung zum Proxy- oder SOAP-Server sein.

Ein Exception-Objekt ist wie folgt aufgebaut:

`type` Entweder `SoapError`, falls der Fehler ausschließlich von der Klasse `WT_SOAP` erkannt wurde, oder einer der anderen Fehlertypen von WebTransactions.

`soapCode`
enthält den SOAP-Fehlercode. Folgende Werte sind möglich:

- `SOCKET`
  Fehler beim Zugriff auf den Web-Service (Zugriff nicht möglich).

  Mögliche Ursachen:
  keine Netzwerk-Verbindung, falscher `PROXY`-Server, falscher `PROXY_PORT` etc.

- `HTTP`
  Fehler beim Zugriff auf den Web-Service (z.B. Zugriff erfolgte, aber `HTTP` meldete einen Fehler).

  Mögliche Ursachen:
  `PROXY_AUTHORIZATION` wird benötigt; der Service kann auf dem adressierten System nicht gefunden werden etc.

- `FILE`
  Der Konstruktor konnte die angegebene WSDL-Datei nicht finden.

- `WSDL`
  Das WSDL-Dokument enthält Elemente, die nicht korrekt interpretiert werden konnten.

- `PARAMETER`
  Die Struktur der Übergabeparameter der Proxy-Methode stimmt nicht mit der in der WSDL geforderten überein.

- `PARSE`
  Während der Ausführung ist ein Syntaxfehler aufgetreten

- `WT_SOAP_HEADER`
  Beim Erzeugen eines Elements der Klasse `WT_SOAP_HEADER` ist ein Fehler aufgetreten.

– `VERSION_MISMATCH`
  Wenn Sie an einen Web-Service, der eine Nachricht im Format SOAP V1.2 erwartet, eine Nachricht im falschen Format senden, antwortet der Server mit einem speziellen Fehlercode. Wenn `WT_SOAP` bei Ausführung einer Proxy-Methode oder der Methode `ExecuteRequest` eine solche Antwort erhält, wird die Ausnahme `VERSION_MISMATCH` mit folgender Bedeutung geworfen:
  Eine Nachricht im falschen Format wurde einem Web-Service zugestellt. Wenn vorhanden, werden für zukünftige Änderungen die Versionen von SOAP ausgegeben, die der Service unterstützt.

`soapText`
enthält erläuternden Text zum aufgetretenen Fehler.

Die Attribute `soapCode` und `soapText` existieren nur, falls gilt:
`type == 'soapError'`

4.2.7 Attribute an WT_SOAP

Die folgende Liste führt die Attribute auf, die Sie für die Nutzung der Klasse `WT_SOAP` benötigen. Zu den Attributen von `WT_SOAP` siehe auch Seite 61.

`service`
: wird als Attribut eines Objekts der Klasse `WT_SOAP` angelegt. `service` enthält ein Objekt für jeden Service, den der Web-Service anbietet. Es dient als Schnittstelle für den Aufruf der Proxy-Methoden.

`envelope`
: wird als Attribut eines Objekts der Klasse `WT_SOAP` angelegt. `envelope` enthält in den Attributen `Envelope`, `Header`, `HeaderBlocks` und `Body` die Textteile, aus denen die SOAP-Nachricht beim Aufruf einer Proxy-Methode oder der Methode `executeRequest` zusammengesetzt wird.

  Das Attribut `envelope` enthält seinerseits die folgenden Attribute

  `envelope`
  : enthält das `Envelope`-Tag mit allen Attributen. Die Attribute werden beim Instanziieren versorgt. Das schließende Tag wird beim Absenden der Nachricht anschließend an den `Body` erzeugt.

  `header`
  : enthält den Inhalt des SOAP-Headers und ist nach der Instanziierung leer. `header` wird beim Absenden der Nachricht mit den passender `Header`-Tags umgeben.

  `HeaderBlocks`
  : Falls Sie Header-Blöcke mit der Funktion `addHeader()` hinzugefügt haben, werden diese Header hier abgelegt und in die Nachricht nach dem Inhalt von `header` eingefügt.

  `body` enthält den Inhalt des SOAP-Body und wird beim Aufruf der Proxy-Methode erzeugt. Beim Absenden der Nachricht wird das passende `Body`-Tag generiert.

`http` enthält eine Referenz auf das verwendete HTTP-Kommunikationsobjekt und wird normalerweise von `WT_SOAP` versorgt; `http` kann für Sonderfälle vom Entwickler benutzt werden.

Die folgende Liste führt Attribute auf, die Sie nur zum Auffinden von Information über die WSDL benötigen:

`binding`
: Abbild aus der `<binding>`-Sektion des WSDL-Dokuments.

`comattr`
: enthält die Attribute, die den HTTP-Zugriff steuern (`http.WT_SYSTEM`). Sie werden von den Methoden der Klasse `WT_SOAP_COM_FUNCTIONS` (siehe Abschnitt „Methoden zur Konfiguration des Zugriffs an der Unterklasse WT_SOAP_COM_FUNCTIONS“ auf Seite 84) dort gespeichert.

`message`
: Abbild aus der `<message>`-Section des WSDL-Dokuments.

`portType`
: Abbild aus der `<portType>`-Sektion des WSDL-Dokuments (wird intern verwendet) und enthält ein oder mehrere Objekte aus WSDL, die die Operationen widerspiegeln, die in der `<portType>`-Section der WSDL beschrieben sind.

`types`
: wird verwendet als Ablage für komplexe Datentypen, die in dem WSDL-Dokument beschrieben sind.

4.2.8 Datentypen für den SOAP-Request im SOAP-Body

Die SOAP-Datentypen werden gemäß den folgenden Tabellen auf WTScript-Datentypen abgebildet. Die Datentypen sind gemäß dem XSD-Schema aufgeführt.

Komplexe Datentypen

SOAP-Datentyp	WTScript-Datentyp
– SOAP-Datentyp `array` – Elemente aus der `<types>`-Section der WSDL mit dem Attribut `maxOccurs > 1`	Array
alle Datentypen, die in der `<types>`-Section mit dem Element `complexType` definiert sind	Object

Einfache Datentypen

SOAP-Datentyp	WTScript-Datentyp
anyURI	string
base64Binary	string
binary	derzeit nicht unterstützt
boolean	boolean
byte	number
date	string
dateTime	string
decimal	number
double	number
duration	string
ENTITIES	derzeit nicht unterstützt
ENTITY	string
float	number
gDay	string
gMonth	string
gMonthDay	string
gYear	string
gYearMonth	string
gYearMonth	string
hexBinary	string
ID	derzeit nicht unterstützt
IDREF	derzeit nicht unterstützt
IDREFS	derzeit nicht unterstützt
int	number
integer	number
language	string
long	number
Name	string
NCName	string
negativeInteger	number
NMTOKEN	string
NMTOKENS	string

SOAP-Datentyp	WTScript-Datentyp
nonNegativeInteger	number
nonPositiveInteger	number
normalizedstring	string
NOTATION	derzeit nicht unterstützt
positiveInteger	number
QName	string
recurringDuration	string
short	number
string	string
time	string
timeDuration	string
token	string
unsignedByte	number
unsignedInt	number
unsignedLong	number
unsignedShort	number

Alle Daten werden beim Versenden der Nachricht mit der Methode `toString` in eine Zeichenkette konvertiert.

Eine Überprüfung der Werte-Konvertierung erfolgt nicht. Beispielsweise wird nicht geprüft, ob bei Aufruf einer Operation, die den SOAP-Datentyp `decimal` verlangt, der Proxy-Methode auch wirklich ganze Zahlen mitgegeben wurden (siehe Beispiel zum Konstruktor auf Seite 67).

4.2.9 Beispiel: Rechtschreibung von Texten überprüfen

Im folgenden WTScript verwenden Sie die WSDL, die im Abschnitt „Beispiel: WSDL-Dokument“ auf Seite 64 dargestellt ist. Sie rufen den Web-Service auf, mit dem die Rechtschreibung von Texten überprüft werden kann. Das Ergebnis wird in einer Tabelle dargestellt.

```
<wtinclude name="wtSOAP">
<wtoncreatescript>
<!--
  try {

    mySoap = new WT_SOAP("wsdl/check.wsdl",
     'proxy.my_company.net','81');

    obj= new Object();
    obj.BodyText = "This is the latest speling chek";
    obj.LicenseKey = 0;
    myResult = mySoap.service.check.port.checkSoap.operation.CheckTextBody(obj);
      }
    catch( e )
  {
    document.write( '<P>Exception: ', e);
exitTemplate();
  }
//-->
</wtoncreatescript>

<wtrem>Ergebnis auswerten</wtrem>
<h3>Original Text</h3>
##myResult.CheckTextBodyResponse.DocumentSummary.body#
<h3>Misspelled Words</h3>
<table border="1">
<tr>
<th>word</th> <th> Possible corrections</th>
</tr>
<wtoncreatescript>
<!--
 for (i=0;i<myResult.CheckTextBodyResponse.DocumentSummary.MisspelledWord.length;i++)
 {
     document.write("<tr><td>");
     document.write
(myResult.CheckTextBodyResponse.DocumentSummary.MisspelledWord[i].word);
     document.write("</td><td>");

document.write(myResult.CheckTextBodyResponse.DocumentSummary.MisspelledWord[i].Suggest
ions.toString().slice(1,-1).replace(/,/g," ").replace(/"/g,""));
     document.write("</td></tr>");
 }
//-->
</wtoncreatescript>
</table>
```

4.3 WT_SOAP_HEADER - Klasse zur Unterstützung von SOAP-Headern

In SOAP Version 1.2 gewinnen Header durch das so genannte SOAP-Prozessmodell an Bedeutung. In diesem Modell geht es im Wesentlichen darum, dass an der Ausführung eines Web-Services mehrere Rechner beteiligt sein können, die unterschiedliche Rollen spielen: Der Body einer Nachricht ist für den letzen Rechner (ultimateReceiver) in der Kette bestimmt. Die Header, dagegen, werden dazu benutzt, Informationen an alle – also auch die dazwischen liegenden – Rechner zu übermitteln. Dieser Mechanismus dient beispielsweise dazu, Transaktionen mit Web-Services zu realisieren oder Sicherheitsaspekte zu implementieren.

Eine genaue Beschreibung, wie die Header von den einzelnen Rechnern behandelt werden sollen und welche Rolle die einzelnen Attribute der Header spielen, finden Sie in der Dokumentation SOAP Version 1.2 im Abschnitt 2 „SOAP Processing Model“.

Die Klasse `WT_SOAP_HEADER` dient dazu, solche Header in korrekter Form zu erzeugen.

Sie repräsentiert ein XML-Element innerhalb des Header-Blocks von `SOAP`.

Objekte der Klasse `WT_SOAP_HEADER` haben die Attribute `name`, `data`, `role`, `namespace`, `nsPrefix` und `bRelay`. Durch direkten Zugriff auf diese Elemente können Sie die einzelnen Eigenschaften der Klasse später ändern.

4.3.1 Konstruktor der Klasse WT_SOAP_HEADER

Die Klasse `WT_SOAP_HEADER` hat nur einen Konstruktor:

```
WT_SOAP_HEADER (name,data[,encodingStyle[,bMustUnderstand[,
                role[,namespace[,nsPrefix[,bRelay]]]]]])
```

Parameter

name Name des Header-Elements. Hier müssen Sie einen gültigen XML-Tag Namen angeben. Ein gültiger XML-Tag Name beginnt mit einem Buchstaben oder einem Unterstrich. Als Folgezeichen können Ziffern, Buchstaben, Unterstriche, Bindestriche und Punkte verwendet werden.

data Inhalt des Headerelements. Geben Sie einen wohlgeformten XML-Text an.

encodingStyle

Hier können Sie den Wert des Attributes `encodingStyle` für einen SOAP-Header-Block angeben. Als Wert ist jede URI erlaubt, die Aussagen macht, wie das Header-Element codiert ist:
Dabei sind folgende Konstanten für in SOAP definierte Werte möglich:

SOAP1.1	Codierung wie in SOAP 1.1
SOAP1.2	Kodierung wie in SOAP1.2
none	Es wird keine Aussage über die Kodierung gemacht.

Wenn dieser Parameter fehlt, oder ein Leerstring angegeben ist, wird dieses Attribut beim Erzeugen der Nachricht ignoriert.

bMustUnderstand

kann folgende Werte annehmen:

true	Der Rechner, der mit `role` adressiert wurde, muss den Header verstehen. Wenn Sie `role` nicht angeben oder für `role` einen Leer-String angeben, wird das Attribut beim Erzeugen der Nachricht weggelassen. Das Weglassen des Attributes hat die gleiche Bedeutung wie das Setzen des Attributes auf `ultimateReceiver`.
false	Der Header muss nicht unbedingt verstanden werden. Voreinstellung

role SOAP V1.2 definiert Rollen für den Rechner, für den der Header bestimmt ist. Die Rolle wird als String in Form einer URL angegeben. Wenn Sie `role` nicht angeben, oder für `role` einen Leerstring angeben, wird das Attribut weggelassen.
Für die in SOAP V1.2 vordefinierten Rollen können Sie Schlüsselworte angeben:

none	Der Header darf von keinem Rechner verarbeitet werden, der ein SOAP-Knoten ist. Die Recher können den Header nur ansehen.
next	Der Header muss vom nächsten Rechner verarbeitet werden.
ultimate Receiver	Der Header muss vom dem Rechner verarbeitet werden, der den Web-Service ausführt. Voreinstellung
anyUri	Falls der Rechner eine vom Web-Service definierte Rolle hat, geben Sie hier den URI dieser Rolle an.

namespace

In diesem Parameter geben Sie den Namensraum eines Headers an.

nsPrefix

Präfix, den das Header-Element bekommt. Wenn Sie hier nichts angeben, oder wenn der Parameter fehlt, so wird intern ein Präfix erzeugt. Präfixe, die WT_SOAP intern verwendet, beginnen mit `wt_`. Geben Sie solche Präfixe hier **nicht** an.

Ein gültiger Präfix für ein XML-Tag beginnt mit einem Buchstaben oder einem Unterstrich. Als Folgezeichen können Ziffern, Buchstaben, Unterstriche, Bindestriche und Punkte verwendet werden.

bRelay

Hier können Sie `true` oder `false` angeben, je nachdem ob ein Header weitergereicht werden soll, wenn der Vermittlungsrechner (nach dem SOAP-Prozessmodell), für den der Header bestimmt war, ihn nicht verarbeiten kann.

Beispiele

Im folgenden Beispiel 1 wird ein Header erzeugt, der eine Transaktionsnummer enthält:

```
myHead1= new WT_SOAP_HEADER("transaction ",
                "5",
                "http://example.com/encoding ",
                true,
                "",
                "http://thirdparty.example.org/transaction");
```

Der so definierte Header wird folgendermaßen in die Nachricht übernommen:

```
<SOAP-ENV:Header>
  <wt_tns0:transaction
      SOAP-ENV:mustUnderstand="true"
      SOAP-ENV:encodingStyle="http://example.com/encoding"
      xmlns:wt_tns0="http://thirdparty.example.org/transaction">
          5
  </wt_tns0:transaction >
</SOAP-ENV:Header><env:Header>
   <wt_tns0:transaction
           xmlns:wt_hns0="http://thirdparty.example.org/transaction"
           env:encodingStyle="http://example.com/encoding"
           env:mustUnderstand="true" >5</<wt_tns0:transaction>
 </env:Header>
```

Im folgenden Beispiel 2 wird ein Header erzeugt, der Elemente mit Reservierungsnummer und Datum enthält. Da diese Elemente ebenfalls qualifiziert mit dem selben Namensraum angegeben werden wie das Header-Element selbst, müssen Sie das zu verwendende Präfix mit angeben:

```
myHead1=new WT_SOAP_HEADER("transactionreservation",
    '<m:reference>uuid:q345-739r-ba5d-pqff98fe8j7d</m:reference>
     <m:dateAndTime>2001-11-29T13:36:50.000-05:00</m:dateAndTime>',
"",
     true,
     "Next",
     "http://travelcompany.example.org/reservation",
     "m");
```

Der so definierte Header wird folgendermaßen in die Nachricht übernommen:

```
<SOAP-ENV:Header>
  <m:reservation
    SOAP-ENV:mustUnderstand="true"
    SOAP-ENV:role="Next"
    xmlns:m="http://travelcompany.example.org/reservation">
     <m:reference>uuid:q345-739r-ba5d-pqff98fe8j7d</m:reference>
     <m:dateAndTime>2001-11-29T13:36:50.000-05:00</m:dateAndTime>
  </m:reservation>
</SOAP-ENV:Header><env:Header>
   <m:reservation
     xmlns:m="http://travelcompany.example.org/reservation"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           env:mustUnderstand="true">
    <m:reference>uuid:q345-739r-ba5d-pqff98fe8j7d</m:reference>
    <m:dateAndTime>2001-11-29T13:36:50.000-05:00</m:dateAndTime>
  </m:reservation>
 </env:Header>
```

Exceptions

Der Konstruktor wirft eine Ausnahme, wenn sowohl `bMustUnderstand` als auch `bRelay` den Werte `true` haben, da sich diese beiden Werte laut Spezifikation ausschließen.

5 Beispiele

Dieses Kapitel gibt Ihnen einen Überblick über die Möglichkeiten von WebTransactions, auf HTTP-Server zuzugreifen.

5.1 Vorhandenes CGI-Script nutzen

Das folgende Beispiel zeigt einen Aufruf der HTTP-Methode `POST` (Senden von Daten an einen HTTP-Server) zum Aufruf eines CGI-Scripts. Dieses CGI-Script (`www.deepThought.mt/cgi-bin/verify.exe`) erwartet zwei Parameter: eine Frage (`question`) und eine Antwort (`answer`), die es analysieren soll.

Um die Bedienung zu vereinfachen, wird vom Aufrufer ein Objekt `sendFORM` bereitgestellt, das die Parameter als Attribute enthält, z.B.: `sendFORM.question = 'sense+of+universe'`.

Die Sendefunktion `sendForm` legt dazu ein Objekt `sendData` mit den passenden Attributen `ContentType` und `Body` der benötigten HTTP-Nachricht an. Für jedes Attribut von `sendFORM` werden der Name und der Wert als Name/Value-Paar an das Attribut `Body` angehängt:

```
function sendForm()
{
  this.sendData = new Object();
  this.sendData.ContentType = 'application/x-www-form-urlencoded';
  this.sendData.Body = '';
  for( attr in this.sendFORM )
  {
    this.sendData.Body += ( this.sendData.Body ? '&' : '' ) +
                          attr + '=' + this.sendFORM [attr];
  }
  return this.send();
}
```

Die universelle Empfangsfunktion `receiveForm` interpretiert den `Body` der empfangenen HTTP-Nachricht und definiert ein Muster zur Suche nach HTML-Eingabefeldern (im Format `<INPUT TYPE="TEXT" NAME="`*feldname*`" VALUE="`*feldwert*`">`). Die beiden Klammern enthalten den Namen und den Wert. Mit diesem Muster wird das Attribut `Body` des Objekts `receiveData` durchsucht, und für jedes gefundene Eingabefeld wird ein Attribut des Objekts `receiveFORM` mit dem Namen des Eingabefelds und dem empfangenen Wert erzeugt.

```
function receiveForm()
{
  if( ! this.receive() )
    return null;
  this.receiveFORM = new Object();
  fieldPattern =
    /<INPUT.*+TYPE=.*TEXT.+NAME=\W*(\w+).+VALUE=["'](\w*)["'].*>/ig
  while( field = fieldPattern.exec( this.receiveData.Body ) )
  {
    this.receiveFORM[field[1]] = field[2];
  }
  return this;
}
```

Das Kommunikationsobjekt wird erzeugt:

```
host = new WT_Communication('myHTTP');
host.open('HTTP');
```

Diese Methoden werden nun in unser Kommunikationsobjekt eingehängt.

```
host.sendForm = sendForm;
host.receiveForm = receiveForm;
```

Jetzt kann die HTTP-Anfrage folgendermaßen formuliert werden:

```
host.sendFORM = new Object();
host.sendFORM.question = 'sense+of+universe';
host.sendFORM.answer = 42;
host.WT_SYSTEM.URL = www.deepThought.mt/cgi-bin/verify.exe;
host.sendForm();
host.receiveForm();
...
```

Mit den durch den Aufruf von `receiveForm` empfangenen und im Objekt `receiveFORM` abgelegten Attributen (Eingabefelder der Antwort) kann jetzt beliebig weitergearbeitet werden.

5.2 Informationen aus dem Web nutzen

Dieses Beispiel zeigt, wie der HTTP-Host-Adapter genutzt werden kann, um aktuelle Daten aus dem WWW zu holen und für die WebTransactions-Anwendung nutzbar zu machen. Die folgende Funktion holt einmal pro Sitzung die aktuellen Umrechnungskurse von US-Dollar und Yen von einer Website und speichert sie in den Systemobjekt-Attributen `WT_SYSTEM.dollar_factor` und `WT_SYSTEM.yen_factor`.

```
<wtoncreatescript>
    <!--
    /* Funktion, um Dollar und Yen-Kurs zu holen */
    function getEuroToDollarYen ()
    {
        var i,j;
        // Das machen wir aber nur einmal pro Sitzung
        if (!WT_SYSTEM.dollar_factor)
        {
            // Wenn das Kommunikationsobjekt noch nicht da ist, wird es erzeugt
            if (!WT_HOST.HTTPCON)
                db = new WT_Communication("HTTPCON");
            else
                db = WT_HOST.HTTPCON;
            // HTTP-Verbindung vorbereiten
            db.open ("HTTP");
            db.WT_SYSTEM.URL =
                "http://waehrungen.onvista.de/devisenkurse.html";
            db.WT_SYSTEM.PROXY = "proxy.my_company.net";
            db.WT_SYSTEM.PROXY_PORT = "81";
            db.WT_SYSTEM.TIMEOUT_HTTP = "10";
            // HTTP-Request schicken und Ergebnis auswerten
            if (db.receive())
            {
                // Suchen nach dem Dollar-Kurs
                var cellBegin = '<td align="right">';
                i = db.receiveData.Body.indexOf('Euro-US Dollar');
                //Der aktuelle Dollarkurs steht in der 3.Tabellenspalte
                for (z=0; z<3;z++)
                  i = db.receiveData.Body.indexOf(cellBegin,i) + cellBegin.length;
                j = db.receiveData.Body.indexOf(' ',i);
                help=db.receiveData.Body.substring(i,j);
                // Speichern des Dollar-Kurses als number
                WT_SYSTEM.dollar_factor =
                    db.receiveData.Body.substring(i,j).replace(/,/,".") * 1;
                if (WT_SYSTEM.dollar_factor == NaN)
                    delete WT_SYSTEM.dollar_factor;
                // Und wenn wir schon mal da sind, holen wir auch gleich den Yen-Kurs
                i = db.receiveData.Body.indexOf('Euro-Japanischer Yen');
                for (z=0; z<3;z++)
                  i = db.receiveData.Body.indexOf(cellBegin,i) + cellBegin.length;
                j = db.receiveData.Body.indexOf(' ',i);
                WT_SYSTEM.yen_factor =
```

```
                    db.receiveData.Body.substring(i,j).replace(/,/,".") * 1;
                if (WT_SYSTEM.yen_factor == NaN)
                    delete WT_SYSTEM.yen_factor;
            }
                // Aufräumen
                db.close();
        }
    }
    getEuroToDollarYen();
    //-->
</wtoncreatescript>
##WT_SYSTEM.dollar_factor#
<br>
##WT_SYSTEM.yen_factor#
```

5.3 Kommunikation über HTTP und Weiterverarbeitung mit WT_Filter

Dieses Beispiel beschreibt eine WebTransactions-Anwendung für den HTTP-Host-Adapter am Beispiel der mitgelieferten WebTransactions-Client-API WT_RPC für WT_REMOTE. Die Verwendung der Klasse WT_RPC ist ausführlich beschrieben im WebTransactions-Handbuch „Client-APIs für WebTransactions“. Dort finden Sie auch ein Beispiel für die Verwendung dieser Klasse. Das vorliegende Kapitel beschäftigt sich dagegen mit der Definition der Klasse WT_RPC als ein Beispiel für die Kommunikation über HTTP und die Weiterverarbeitung der empfangenen Nachrichten mit den mitgelieferten Filtern.

5.3.1 Grundkonzept der Klasse WT_RPC

Mit Hilfe des HTTP-Host-Adapters ist es möglich, in WebTransactions beliebige von Web-Servern zur Verfügung gestellte Informationen zu nutzen. Hier tritt die WebTransactions-Anwendung als Client einer Web-Applikation auf. Auf der anderen Seite stellt WebTransactions als Server seine Informationen schon immer über das HTTP-Protokoll zur Verfügung. Neben der HTML-Schnittstelle bietet WebTransactions mit WT_REMOTE eine zweite Serverschnittstelle. Über diese Schnittstelle kann ein Client eine WebTransactions-Anwendung von außen steuern.

Die Schnittstelle WT_REMOTE und der HTTP-Host-Adapter als Basismechanismen von WebTransactions erlauben somit eine Verteilung von WebTransactions-Anwendungen über das Netz. Eine WebTransactions-Anwendung kann bestimmte Aufgaben an eine andere delegieren. Die Basismechanismen erlauben es dem Client z.B. Methodenaufrufe in Form von XML-Dokumenten zu formulieren und an die ferne WebTransactions-Anwendung zu schicken. Diese schickt dann wiederum als XML-Dokument das Ergebnis zurück. Das XML-Dokument kann dann in der Client-Anwendung in entsprechende Datenobjekte umgewandelt werden. Die Aufgabe der hier vorgestellten Klasse WT_RPC ist es, im Client den Aufbau des Methodenaufrufs, den fernen Aufruf und die Aufbereitung des Ergebnisses abzuwickeln.

Die Klasse WT_RPC unterstützt einen entfernten Aufruf von WebTransactions-Funktionen. Einem WT_RPC-Objekt werden die Adresse (URL und Basisverzeichnis für WebTransactions) der fernen WebTransactions-Anwendung und die Namen der fernen Funktionen übergeben. Danach können die fernen Funktionen als Methoden des WT_RPC-Objekts aufgerufen werden.

Die folgende Grafik soll dies an einem schematischen Beispiel verdeutlichen. Der Client benötigt das Dokument `WT_RPC.htm`, das die Definition der Klasse `WT_RPC` enthält. Mit Hilfe dieser Klasse wird eine Verbindung definiert und die ferne Funktion `f` im Dokument `calc.htm` bekannt gegeben; jetzt kann die lokale Methode `rtw.f` als Stellvertreter der fernen Funktion `f` aufgerufen werden.

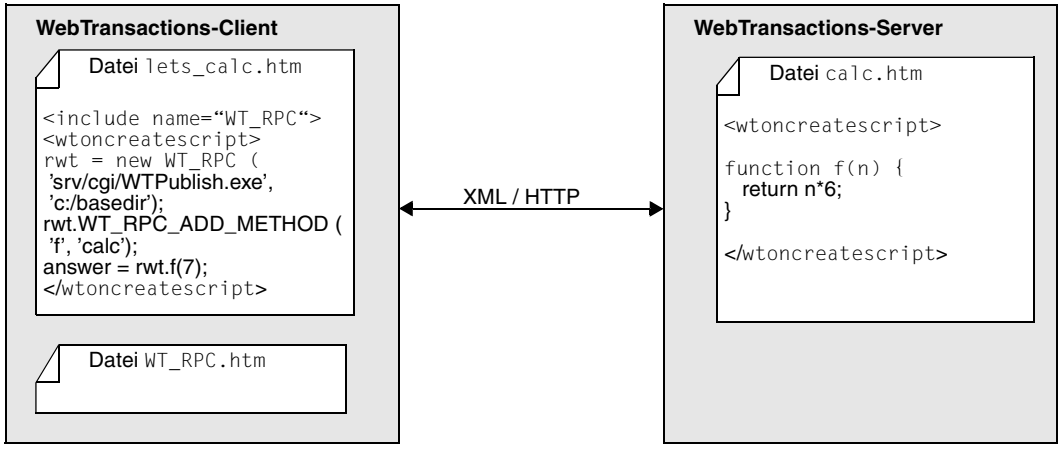


Bild 5: Funktionalität der Klasse `WT_RPC`

5.3.2 Implementierung der Klasse WT_RPC

Die folgenden Abschnitte beschreiben die Implementierung des Konstruktors und der Methoden der Klasse WT_RPC, so wie sie mit WebTransactions ausgeliefert wird. Ein Beispiel für die Verwendung dieser Klasse finden Sie im WebTransactions-Handbuch „Client-APIs für WebTransactions“.

Der WT_RPC-Konstruktor

Der Konstruktor WT_RPC legt ein neues Kommunikationsobjekt mit einem Namen WT_RPC_n an, wobei n=0,1,2... der kleinste Index ist, für den ein entsprechendes Objekt noch nicht existiert. Über dieses Kommunikationsobjekt wird die HTTP-Verbindung zur entfernten WebTransactions-Anwendung abgewickelt. Es werden die Methoden WT_RPC_OPEN, WT_RPC_CLOSE, WT_RPC_INVOKE und WT_RPC_ADD_METHOD eingehängt, die weiter unten beschrieben sind. Ggf. angegebene Verbindungsdaten für die ferne WebTransactions-Anwendung werden in den Attributen WT_URL und WT_BASEDIR hinterlegt und die ferne WebTransactions-Anwendung wird gestartet:

```
// constructor for the remote procedure object //////////////
function WT_RPC( urlOfWebTA, basedir )
{
  // create new communication object WT_RPC_n
  var i;
  for ( i = 0; WT_HOST[ 'WT_RPC_' + i ] != null; i++ );
  this.WT_COM_OBJECT = new WT_Communication( 'WT_RPC_' + i );
  // define methods
  this.WT_RPC_OPEN       = WT_RPC_OPEN;
  this.WT_RPC_CLOSE      = WT_RPC_CLOSE;
  this.WT_RPC_INVOKE     = WT_RPC_INVOKE;
  this.WT_RPC_ADD_METHOD = WT_RPC_ADD_METHOD;
  // start remote WebTransactions application
  if( urlOfWebTA && basedir )
    this.WT_RPC_OPEN( urlOfWebTA, basedir );
  else
    this.WT_CONNECTED = false;
}
```

Methode WT_RPC_OPEN - ferne WebTransactions-Anwendung starten

Diese Methode startet die ferne WebTransactions-Sitzung und bereitet das Kommunikationsobjekt für Methodenaufrufe vor. Wurden alle Aktionen erfolgreich ausgeführt, gibt die Funktion `true`, anderenfalls `false` zurück.

Im Einzelnen werden folgende Aktionen ausgeführt.

- Zunächst wird eine ggf. laufende ferne WebTransactions-Anwendung beendet und die Verbindung geschlossen.
- Dann werden die Parameter, sofern sie angegeben wurden, in die Attribute `WT_URL` und `WT_BASEDIR` übernommen.
- Ist jetzt keine Adresse für die ferne WebTransactions-Anwendung vorhanden (durch Parameterübergabe aus diesem oder einem früheren Aufruf), so wird die Methode mit negativem Rückgabewert beendet.
- Andernfalls wird durch Aufruf der Methode `com.open` der HTTP-Host-Adapter für das Kommunikationsobjekt aktiviert.
- Es wird das Attribut `URL` zum Start der entfernten WebTransactions-Anwendung aufgebaut.
- Die `WT_REMOTE`-Aktion `START_SESSION` veranlasst die entfernte WebTransactions-Anwendung, ihre Sitzungsparameter auszugeben.
- Das Objekt `sendData` wird gelöscht, um die Ausführung der Methode `GET` sicherzustellen.
- Mit `send` und `receive` erfolgt nun der eigentliche Zugriff über das Netz.
- Wurde diese Aktion erfolgreich durchgeführt (`HTTP_RETURN_CODE` hat den Wert 200 und XML-Daten wurden empfangen), so kann das Objekt `sendData` für die folgenden Methodenaufrufe vorbereitet werden.

  Es soll sich um eine mehrteilige HTTP-Nachricht handeln.

  – Im ersten Teil werden die Sitzungsdaten hinterlegt, mit denen WebTransactions die gewünschte entfernte Sitzung wieder findet. Dieser Teil erhält den Mime-Typ `application/x-www-form-urlencoded`, die benötigten Werte erhält man durch Analyse mit `WT_Filter` aus der Antwort des vorangegangenen Aufrufs.

  – Im zweiten Nachrichtenteil soll der eigentliche Nutzinhalt geschickt werden. Dieser wird als XML-Dokument codiert und erhält daher den Mime-Typ `text/xml`.

```
// method to start new remote session and connect to it //////////////
function WT_RPC_OPEN( urlOfWebTA, basedir )
{
  // terminate existing remote WebTransactions application
  if( this.WT_CONNECTED )
    this.WT_RPC_CLOSE();
```

```
  // store new connection definition if redefined
  if( urlOfWebTA && basedir )
  {
    this.WT_URL = urlOfWebTA;
    this.WT_BASEDIR = basedir;
  }

  // return immediately if connection parameters missing
  if( ! ( this.WT_URL && this.WT_BASEDIR ) )
    return this.WT_CONNECTED = false;
  // start remote session
  var com = this.WT_COM_OBJECT;
  com.open( 'HTTP' );
  com.WT_SYSTEM.URL =
    this.WT_URL + '/startup?'+
    'WT_SYSTEM_BASEDIR=' + this.WT_BASEDIR +
    '&WT_REMOTE=START_SESSION';
  if( com.sendData )
    delete com.sendData;
  com.send();
  com.receive();
  if( !WT_SYSTEM.ERROR && com.WT_SYSTEM.HTTP_RETURN_CODE == 200
      && com.ReceiveDate.ContentType='text/xml' )
  {
    //store session parameters
    var remoteWtSystem;
    WT_Filter.XMLToDataObject(com.receiveData.Body,'remoteWtSystem');
    this.WT_SESSION_PARAMS ='WT_SYSTEM_BASEDIR=' + this.WT_BASEDIR +
     '&WT_SYSTEM_FORMAT_STATE=IGNORE'+
     '&WT_SYSTEM_SESSION=' + remoteWtSystem.SESSION +
     '&WT_SYSTEM_SIGNATURE=' + remoteWtSystem.SIGNATURE;
    // prepare sendData
    com.sendData = new Array();
    com.sendData[0] = new Object();
    com.sendData[0].ContentType =
      'application/x-www-form-urlencoded';
    com.sendData [0].Body =
      this.WT_SESSION_PARAMS + '&WT_REMOTE=PROCESS_COMMANDS';
    com.sendData[1] = new Object();
    com.sendData[1].ContentType = 'text/xml';
    // prepare communication object for method invocation
    com.WT_SYSTEM.URL = this.WT_URL;
    this.WT_CONNECTED = true;
  }
  else
    this.WT_CONNECTED = false;
  return this.WT_CONNECTED;
}
```

Methode WT_RPC_CLOSE - ferne WebTransactions-Anwendung beenden

Diese Methode beendet eine entfernte WebTransactions-Anwendung und deaktiviert das HTTP-Kommunikationsmodul. Im Einzelnen werden die folgenden Aktionen ausgeführt.

- Es wird das Attribut `URL` zum Zugriff auf die entfernte WebTransactions-Anwendung aufgebaut.
- Die `WT_REMOTE`-Aktion `EXIT_SESSION` beendet die entfernte WebTransactions-Anwendung.
- Das Objekt `sendData` wird gelöscht, um die Ausführung der Methode `GET` sicherzustellen.
- Mit `send` und `receive` erfolgt nun der eigentliche Zugriff über das Netz.
- Abschließend wird die Methode `close` aufgerufen, um die Ressourcen des HTTP-Kommunikationsmoduls freizugeben:

```
// terminate remote session and close connection ///////////
function WT_RPC_CLOSE()
{
  if( this.WT_CONNECTED )
  {
    var com = this.WT_COM_OBJECT;
    com.WT_SYSTEM.URL = this.WT_URL + '?' +
                        this.WT_SESSION_PARAMS +
                        '&WT_REMOTE=EXIT_SESSION';
    if( com.sendData )
      delete com.sendData;
    com.send();
    com.receive();
    com.close();
    this.WT_CONNECTED = false;
  }
}
```

Methode WT_RPC_INVOKE - ferne Funktion aufrufen

Die Methode `WT_RPC_INVOKE` ruft eine ferne Funktion mit Namen *functionName* auf. Dazu wird mit Hilfe des Filters `methodCallToXML` ein passendes XML-Dokument erzeugt. Der Filter erhält mit `functionName` den Namen der Methode, mit `codeBase` den Namen des implementierenden WTML-Dokumentes auf der fernen Maschine und mit `argArray` ein Array mit den gewünschten Parametern für den Methodenaufruf. Dieses wird an die entfernte WebTransactions-Anwendung geschickt und ausgeführt. Das Ergebnis wird als XML-Dokument empfangen und durch den Filter `XMLToDataObject` in ein WTScript-Datenobjekt überführt, das zurückgegeben wird. Besteht keine Verbindung zu einer fernen WebTransactions-Anwendung, so wird `null` zurückgegeben.

```
// prepare XML document for method call and send it to remote session
function WT_RPC_INVOKE( functionName, codeBase, argArray )
{
  if( this.WT_CONNECTED )
  {
    var returnValue;
    var com = this.WT_COM_OBJECT;
    com.sendData[1].Body =
      WT_Filter.methodCallToXML( functionName, argArray, codeBase );
    com.send();
    com.receive();
    WT_Filter.XMLToDataObject( com.receiveData.Body, 'returnValue' );
    return returnValue;
  }
  else
    return null;
}
```

Methode WT_RPC_ADD_METHOD - ferne Funktion definieren

Die fernen Funktionen sollen nicht nur mit Hilfe der Methode WT_RPC_INVOKE aufrufbar sein, indem Name, implementierendes WTML-Dokument und Parameter-Array als Argumente angegeben werden. Es soll darüber hinaus möglich sein, eine entfernte Funktion direkt über eine gleichnamige lokale Methode des WT_RPC-Objekts aufzurufen. Solche Proxy-Methoden können mittels der Methode WT_RPC_ADD_METHOD angelegt werden. WT_RPC_ADD_METHOD erzeugt eine neue Funktion, die WT_RPC_INVOKE mit dem passenden Funktions- und Dokumentnamen aufruft. Die Parameter dieser neuen Funktion werden als Array an WT_RPC_INVOKE weitergereicht. Das neue Funktionsobjekt wird mit dem Namen *functionName* am WT_RPC-Objekt hinterlegt.

```
// bind function to WT_RPC object ////////////////////
function WT_RPC_ADD_METHOD( functionName, codeBase )
{
  this[functionName] = new Function(
   '{return this.WT_RPC_INVOKE ("' + functionName + '", "' +
   codeBase + '", arguments );}' );
}
```

6 Anhang

Der Anhang enthält:

- Übersicht über die HTTP-Fehlermeldungen nach RFC 2086
- WSDL-Schema

6.1 HTTP-Fehlermeldungen

In diesem Abschnitt finden Sie die HTTP-Fehlermeldungen, die beim Verbindungsaufbau mit einem HTTP-Server auftreten können.

Übersicht nach RFC 2068

Die folgende Tabelle liefert einen Kurzüberblick über die HTTP-Fehlercodes in HTTP/ 1.1, deren Namen in der RFC 2068 und deren Bedeutung:

Code / Name	Bedeutung
1xx	Informative Antwortcodes (nur in HTTP/ 1.1)
200 OK	Anforderung erfolgreich
201 Created	Anforderung ausgeführt, neuer Inhalt wurde erzeugt
202 Accepted	Anforderung akzeptiert, noch nicht beendet
203 Non-Authoritative Information	Meta-Informationen von dritter Seite geändert
204 No Content	erfolgreich, aber kein Inhalt
205 Reset Content	Inhalte erfolgreich geändert
206 Partial Content	teilweise GET-Operation erfolgreich
300 Multiple Choices	lokalisierte Versionen verfügbar
301 Moved Permanently	verschoben
302 Moved Temporarily	zeitweise verschoben
303 See Other	unter anderer URI verfügbar

Tabelle 6: HTTP-Fehlercodes nach RFC 2068

Code / Name	Bedeutung
304 Not Modified	bedingter Zugriff erfolgreich, aber keine Änderung der Ressource
305 Use Proxy	Zugriff nur über Proxy
400 Bad Request	ungültige Syntax der Anforderung
401 Unauthorized	Authentifizierung erforderlich
402 Payment Required	reserviert für künftige Erweiterungen
403 Forbidden	Zugriff verboten
404 Not Found	angeforderte Ressource nicht gefunden
405 Method Not Allowed	angeforderte Methode für angegebene Ressource nicht erlaubt
406 Not Acceptable	Antwort-Einheiten passen nicht zur Anforderung
407 Proxy Authentication Required	Authentifizierung über Proxy erforderlich
408 Request Timeout	Timeout für Client-Anforderung
409 Conflict	Konflikt mit Status der Ressource
410 Gone	Ressource nicht länger verfügbar
411 Length Required	Content-Length muss im Header angegeben werden
412 Precondition Failed	Voraussetzung im Anforderungs-Header nicht erfüllt
413 Request Entity Too Large	Anforderungs-Einheit zu lang
414 Request-URI Too Long	Anforderungs-URI länger als vom Server erlaubt
415 Unsupported Media Type	ungültiges Anforderungsformat für angeforderte Methode
500 Internal Server Error	unerwarteter HTTP-Server-Fehler
501 Not Implemented	benötigte Funktionalität nicht vorhanden
502 Bad Gateway	ungültige Antwort von übergeordnetem Server
503 Service Unavailable	Server kann Anforderung derzeit nicht bearbeiten
504 Gateway Timeout	Timeout bei Antwort von übergeordnetem Server
505 HTTP Version Not Supported	nicht unterstützte HTTP-Version der Anforderung

Tabelle 6: HTTP-Fehlercodes nach RFC 2068

6.2 WSDL-Schema

Das nachfolgend aufgeführte WSDL-Schema zeigt die Spezifikation von WSDL 1.1, wie sie vom World Wide Web Consortium (W3C) zur Verfügung gestellt wird. Passagen, die von der Klasse `WT_SOAP` nicht unterstützt werden, sind in blauer Schrift dargestellt.

```
<schema xmlns="http://www.w3.org/2000/10/XMLSchema"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        targetNamespace="http://schemas.xmlsoap.org/wsdl/"
        elementFormDefault="qualified">
   <element name="documentation">

   <complexType mixed="true">
         <choice minOccurs="0" maxOccurs="unbounded">
            <any minOccurs="0" maxOccurs="unbounded"/>
         </choice>
         <anyAttribute/>
      </complexType>
   </element>
   <complexType name="documented" abstract="true">
      <sequence>
         <element ref="wsdl:documentation" minOccurs="0"/>
      </sequence>
   </complexType>
   <complexType name="openAtts" abstract="true">
      <annotation>
         <documentation>
         This type is extended by  component types
         to allow attributes from other namespaces to be added.
         </documentation>
      </annotation>
      <sequence>
         <element ref="wsdl:documentation" minOccurs="0"/>
      </sequence>
      <anyAttribute namespace="##other"/>
   </complexType>
   <element name="definitions" type="wsdl:definitionsType">
      <key name="message">
         <selector xpath="message"/>
         <field xpath="@name"/>
      </key>
      <key name="portType">
         <selector xpath="portType"/>
         <field xpath="@name"/>
      </key>
       <key name="binding">
         <selector xpath="binding"/>
         <field xpath="@name"/>
      </key>
      <key name="service">
         <selector xpath="service"/>
         <field xpath="@name"/>
```

```
        </key>
       <key name="import">
              <selector xpath="import"/>
              <field xpath="@namespace"/>
        </key>
        <key name="port">
           <selector xpath="service/port"/>
           <field xpath="@name"/>
        </key>
     </element>
<complexType name="definitionsType">
        <complexContent>
           <extension base="wsdl:documented">
              <sequence>
                 <element ref="wsdl:import" minOccurs="0" maxOccurs="unbounded"/>
                 <element ref="wsdl:types" minOccurs="0"/>
                 <element ref="wsdl:message" minOccurs="0" maxOccurs="unbounded"/>
                 <element ref="wsdl:portType" minOccurs="0" maxOccurs="unbounded"/>
                 <element ref="wsdl:binding" minOccurs="0" maxOccurs="unbounded"/>
                 <element ref="wsdl:service" minOccurs="0" maxOccurs="unbounded"/>
                 <any namespace="##other" minOccurs="0" maxOccurs="unbounded">
                    <annotation>
                       <documentation>to support extensibility elements </documentation>
                    </annotation>
                 </any>
              </sequence>
              <attribute name="targetNamespace" type="uriReference" use="optional"/>
              <attribute name="name" type="NMTOKEN" use="optional"/>
           </extension>
        </complexContent>
    </complexType>
     <element name="import" type="wsdl:importType"/>
     <complexType name="importType">
        <complexContent>
     <extension base="wsdl:documented">
     <attribute name="namespace" type="uriReference" use="required"/>
        <attribute name="location" type="uriReference" use="required"/>
     </extension>
    </complexContent>
    </complexType>
     <element name="types" type="wsdl:typesType"/>
     <complexType name="typesType">
         <complexContent>
     <extension base="wsdl:documented">
     <sequence>
     <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
     </extension>
    </complexContent>
    </complexType>
     <element name="message" type="wsdl:messageType">
         <unique name="part">
```

```
        <selector xpath="part"/>
        <field xpath="@name"/>
    </unique>
 </element>
 <complexType name="messageType">
    <complexContent>
 <extension base="wsdl:documented">
 <sequence>
 <element ref="wsdl:part" minOccurs="0" maxOccurs="unbounded"/>
</sequence>
    <attribute name="name" type="NCName" use="required"/>
 </extension>
</complexContent>
</complexType>
 <element name="part" type="wsdl:partType"/>
 <complexType name="partType">
    <complexContent>
 <extension base="wsdl:openAtts">
 <attribute name="name" type="NMTOKEN" use="optional"/>
    <attribute name="type" type="QName" use="optional"/>
    <attribute name="element" type="QName" use="optional"/>
 </extension>
</complexContent>
</complexType>
 <element name="portType" type="wsdl:portTypeType"/>
 <complexType name="portTypeType">
    <complexContent>
 <extension base="wsdl:documented">
 <sequence>
 <element ref="wsdl:operation" minOccurs="0" maxOccurs="unbounded"/>
</sequence>
    <attribute name="name" type="NCName" use="required"/>
 </extension>
</complexContent>
</complexType>
 <element name="operation" type="wsdl:operationType"/>
 <complexType name="operationType">
    <complexContent>
  <extension base="wsdl:documented">
    <choice>
       <group ref="wsdl:one-way-operation"/>
       <group ref="wsdl:request-response-operation"/>
       <group ref="wsdl:solicit-response-operation"/>
       <group ref="wsdl:notification-operation"/>
    </choice>
    <attribute name="name" type="NCName" use="required"/>
 </extension>
</complexContent>
</complexType>
 <group name="one-way-operation">
    <sequence>
       <element ref="wsdl:input"/>
```

```
    </sequence>
 </group>
 <group name="request-response-operation">
    <sequence>
       <element ref="wsdl:input"/>
       <element ref="wsdl:output"/>

        <element ref="wsdl:fault" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
 </group>
 <group name="solicit-response-operation">
    <sequence>
       <element ref="wsdl:output"/>
       <element ref="wsdl:input"/>
       <element ref="wsdl:fault" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
 </group>
 <group name="notification-operation">
    <sequence>
       <element ref="wsdl:output"/>
    </sequence>
 </group>
 <element name="input" type="wsdl:paramType"/>
 <element name="output" type="wsdl:paramType"/>
 <element name="fault" type="wsdl:faultType"/>
 <complexType name="paramType">
    <complexContent>
 <extension base="wsdl:documented">
 <attribute name="name" type="NMTOKEN" use="optional"/>
    <attribute name="message" type="QName" use="required"/>
 </extension>
</complexContent>
</complexType>
 <complexType name="faultType">
    <complexContent>
 <extension base="wsdl:documented">
 <attribute name="name" type="NMTOKEN" use="required"/>
    <attribute name="message" type="QName" use="required"/>
 </extension>
</complexContent>
</complexType>
 <complexType name="startWithExtensionsType" abstract="true">
    <complexContent>
 <extension base="wsdl:documented">
 <sequence>
 <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
</sequence>
 </extension>
</complexContent>
</complexType>
 <element name="binding" type="wsdl:bindingType"/>
 <complexType name="bindingType">
```

```
      <complexContent>
    <extension base="wsdl:startWithExtensionsType">
    <sequence>
    <element name="operation" type="wsdl:binding_operationType" minOccurs="0"
                                                                  maxOccurs="unbounded"
/>
  </sequence>
      <attribute name="name" type="NCName" use="required"/>
      <attribute name="type" type="QName" use="required"/>
   </extension>
  </complexContent>
  </complexType>
   <complexType name="binding_operationType">
      <complexContent>
    <extension base="wsdl:startWithExtensionsType">
    <sequence>
    <element name="input" type="wsdl:startWithExtensionsType" minOccurs="0"/>
      <element name="output" type="wsdl:startWithExtensionsType" minOccurs="0"/>
      <element name="fault" minOccurs="0" maxOccurs="unbounded">
         <complexType>
            <complexContent>
    <extension base="wsdl:startWithExtensionsType">
    <attribute name="name" type="NMTOKEN" use="required"/>
         </extension>
  </complexContent>
  </complexType>
      </element>
  </sequence>
      <attribute name="name" type="NCName" use="required"/>
   </extension>
  </complexContent>
  </complexType>
   <element name="service" type="wsdl:serviceType"/>
   <complexType name="serviceType">
       <complexContent>
    <extension base="wsdl:documented">
    <sequence>
    <element ref="wsdl:port" minOccurs="0" maxOccurs="unbounded"/>
      <any namespace="##other" minOccurs="0"/>
  </sequence>
      <attribute name="name" type="NCName" use="required"/>
   </extension>
  </complexContent>
  </complexType>
   <element name="port" type="wsdl:portType"/>
   <complexType name="portType">
      <complexContent>
    <extension base="wsdl:documented">
    <sequence>
    <any namespace="##other" minOccurs="0"/>
  </sequence>
      <attribute name="name" type="NCName" use="required"/>
```

```
      <attribute name="binding" type="QName" use="required"/>
    </extension>
  </complexContent>
  </complexType>
  <attribute name="arrayType" type="string"/>
</schema>
```

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*: `wtOncCreateScript` 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
Client-APIs für WebTransactions
Benutzerhandbuch

WebTransactions
Anschluss an openUTM-Anwendungen über UPIC
Benutzerhandbuch

WebTransactions
Anschluss an OSD-Anwendungen
Benutzerhandbuch

WebTransactions
Anschluss an MVS-Anwendungen
Benutzerhandbuch

WebTransactions
Web-Frontend für Web-Services
Benutzerhandbuch

Sonstige Literatur

[1] **SOAP Version 1.2**
Part 1: Messaging Framework
http://www.w3.org/TR/soap12-part1/

[2] **SOAP Version 1.2**
Part 2: Adjuncts
http://www.w3.org/TR/soap12-part2

[3] Web Services Description Language (WSDL) 1.1
http://www.w3.org/TR/wsdl

Stichwörter

X

Z