

# WebTransactions V7.5

Web Frontend for Web Services

## **Comments... Suggestions... Corrections...**

The User Documentation Department would like to know your opinion on this manual. Your feedback helps us to optimize our documentation to suit your individual needs.

Feel free to send us your comments by e-mail to:

[manuals@ts.fujitsu.com](mailto:manuals@ts.fujitsu.com)

## **Certified documentation according to DIN EN ISO 9001:2008**

To ensure a consistently high quality standard and user-friendliness, this documentation was created to meet the regulations of a quality management system which complies with the requirements of the standard DIN EN ISO 9001:2008.

cognitas. Gesellschaft für Technik-Dokumentation mbH

[www.cognitas.de](http://www.cognitas.de)

## **Copyright and Trademarks**

Copyright © Fujitsu Technology Solutions GmbH 2010.

All rights reserved.

Delivery subject to availability; right of technical modifications reserved.

All hardware and software names used are trademarks of their respective manufacturers.

---

# Contents

<b>1</b>	<b>Preface . . . . .</b>	<b>5</b>
<b>1.1</b>	<b>Product characteristics . . . . .</b>	<b>5</b>
<b>1.2</b>	<b>Architecture of the web frontend for web services . . . . .</b>	<b>7</b>
<b>1.3</b>	<b>WebTransactions documentation . . . . .</b>	<b>8</b>
<b>1.4</b>	<b>Structure and target group of this manual . . . . .</b>	<b>10</b>
<b>1.5</b>	<b>New features . . . . .</b>	<b>10</b>
<b>1.6</b>	<b>Notational conventions . . . . .</b>	<b>11</b>
<b>2</b>	<b>Concept of the web frontend . . . . .</b>	<b>13</b>
<b>3</b>	<b>Creating a base directory . . . . .</b>	<b>15</b>
<b>3.1</b>	<b>Creating a base directory with WebLab . . . . .</b>	<b>15</b>
<b>3.2</b>	<b>Structure of a base directory . . . . .</b>	<b>16</b>
<b>4</b>	<b>Generating a template . . . . .</b>	<b>17</b>
<b>5</b>	<b>Testing the web frontend . . . . .</b>	<b>21</b>
<b>5.1</b>	<b>General start template . . . . .</b>	<b>22</b>
<b>5.2</b>	<b>Web frontend for a web service . . . . .</b>	<b>23</b>

<b>6</b>	<b>Editing templates</b> . . . . .	<b>25</b>
<b>6.1</b>	<b>Accessing objects</b> . . . . .	<b>26</b>
6.1.1	Integration via SOAP . . . . .	26
6.1.2	Example of a simple data type . . . . .	28
6.1.3	Example of a complex data type . . . . .	29
<b>6.2</b>	<b>Master template</b> . . . . .	<b>31</b>
<b>6.3</b>	<b>Generated template</b> . . . . .	<b>32</b>
	<b>Glossary</b> . . . . .	<b>43</b>
	<b>Abbreviations</b> . . . . .	<b>61</b>
	<b>Related publications</b> . . . . .	<b>63</b>
	<b>Index</b> . . . . .	<b>65</b>

---

# 1 Preface

Over the past years, more and more IT users have found themselves working in heterogeneous system and application environments, with mainframes standing next to Unix systems and Windows systems and PCs operating alongside terminals. Different hardware, operating systems, networks, databases and applications are operated in parallel. Highly complex, powerful applications are found on mainframe systems, as well as on Unix servers and Windows servers. Most of these have been developed with considerable investment and generally represent central business processes which cannot be replaced by new software without a certain amount of thought.

The ability to integrate existing heterogeneous applications in a uniform, transparent IT concept is a key requirement for modern information technology. Flexibility, investment protection, and openness to new technologies are thus of crucial importance.

## 1.1 Product characteristics

With WebTransactions, Fujitsu Technology Solutions offers a best-of-breed web integration server which will make a wide range of business applications ready for use with browsers and portals in the shortest possible time. WebTransactions enables rapid, cost-effective access via standard PCs and mobile devices such as tablet PCs, PDAs (Personal Digital Assistant) and mobile phones.

WebTransactions covers all the factors typically involved in web integration projects. These factors range from the automatic preparation of legacy interfaces, the graphic preparation and matching of workflows and right through to the comprehensive frontend integration of multiple applications. WebTransactions provides a highly scalable runtime environment and an easy-to-use graphic development environment.

On the first integration level, you can use WebTransactions to integrate and link the following applications and content directly to the Web so that they can be easily accessed by users in the internet and intranet:

- Dialog applications in BS2000/OSD
- MVS or z/OS applications
- System-wide transaction applications based on openUTM
- Dynamic web content

Users access the host application in the internet or intranet using a web browser of their choice.

Thanks to the use of state-of-the-art technology, WebTransactions provides a second integration level which allows you to replace or extend the typically alphanumeric user interfaces of the existing host application with an attractive graphical user interface and also permits functional extensions to the host application without the need for any intervention on the host (dialog reengineering).

On a third integration level, you can use the uniform browser interface to link different host applications together. For instance, you can link any number of previously heterogeneous host applications (e.g. MVS or OSD applications) with each other or combine them with dynamic Web contents. The source that originally provided the data is now invisible to the user.

In addition, you can extend the performance range and functionality of the WebTransactions application through dedicated clients. For this purpose, WebTransactions offers an open protocol and special interfaces (APIs).

Host applications and dynamic Web content can be accessed not only via WebTransactions but also by “conventional” terminals or clients. This allows for the step-by-step connection of a host application to the Web, while taking account of the wishes and requirements of different user groups.

## 1.2 Architecture of the web frontend for web services

The figure below shows the architecture of the web frontend for web services:

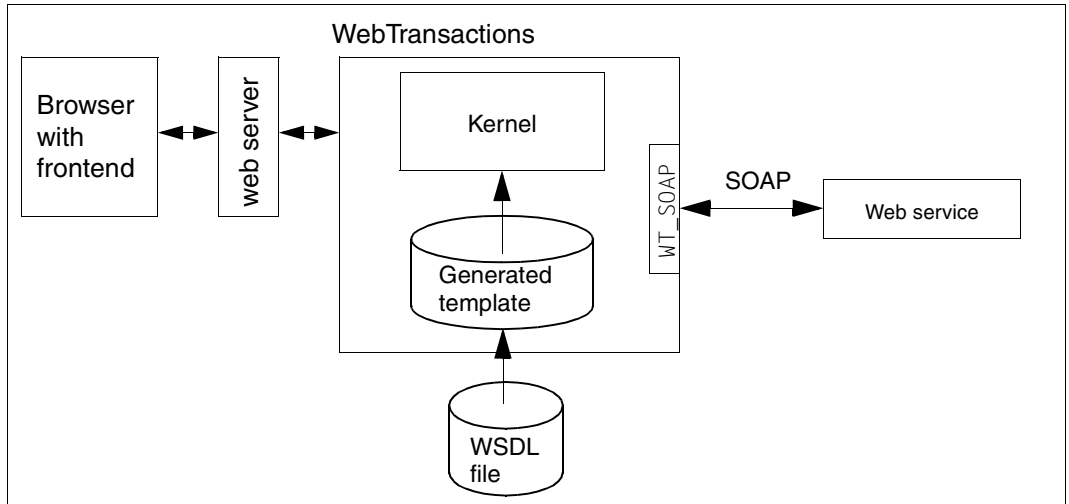


Figure 1: Architecture of the web frontend for web services

WebTransactions uses this web frontend for web services to access general web services.

## 1.3 WebTransactions documentation

The WebTransactions documentation consists of the following documents:

- An introductory manual which applies to all supply units:

### Concepts and Functions

This manual describes the key concepts behind WebTransactions:

- The various possible uses of WebTransactions.
  - The concept behind WebTransactions and the meanings of the objects in WebTransactions, their main characteristics and methods, their interaction and life cycle.
  - The dynamic runtime of a WebTransactions application.
  - The administration of WebTransactions.
  - The WebLab development environment.
- A Reference Manual which also applies to all supply units and which describes the WebTransactions template language WTML. This manual describes the following:

### Template Language

After an overview of WTML, information is provided about:

- The lexical components used in WTML.
- The class-independent global functions, e.g. `escape()` or `eval()`.
- The integrated classes and methods, e.g. `array` or `Boolean` classes.
- The WTML tags which contain functions specific to WebTransactions.
- The WTScript statements that you can use in the WTScript areas.
- The class templates which you can use to automatically evaluate objects of the same type.
- The master templates used by WebTransactions as templates to ensure a uniform layout.
- A description of Java integration, showing how you can instantiate your own Java classes in WebTransactions and a description of user exits, which you can use to integrate your own C/C++ functions.
- The ready-to-use user exits shipped together with WebTransactions.
- The XML conversion for the portable representation of data used for communication with external applications via XML messages and the conversion of WTScript data structures into XML documents.



- A User Guide for each type of host adapter with special information about the type of the partner application:

- **Connection to openUTM applications via UPIC**

- **Connection to OSD applications**

- **Connection to MVS applications**

- All the host adapter guides contain a comprehensive example session. The manuals describe:

- The installation of WebTransactions with each type of host adapter.
    - The setup and starting of a WebTransactions application.
    - The conversion templates for the dynamic conversion of formats on the web browser interface.
    - The editing of templates.
    - The control of communications between WebTransactions and the host applications via various system object attributes.
    - The handling of asynchronous messages and the print functions of WebTransactions.

- A User Guide that applies to all the supply units and describes the possibilities of the HTTP host adapter:

- **Access to Dynamic Web Contents**

- This manual describes:

- How you can use WebTransactions to access a HTTP server and use its resources.
    - The integration of SOAP (Simple Object Access Protocol) protocols in WebTransactions and the connection of web services via SOAP.

- A User Guide valid for all the supply units which describes the open protocol, and the interfaces for the client development for WebTransactions:

### **Client APIs for WebTransactions**

This manual describes:

- The concept of the client-server interface in WebTransactions.
- The `WT_RPC` class and the `WT_REMOTE` interface. An object of the `WT_RPC` class represents a connection to a remote WebTransactions application which is run on the server side via the `WT_REMOTE` interface.
- The Java package `com.siemens.webta` for communication with WebTransactions supplied with the product.

## **1.4 Structure and target group of this manual**

This manual is intended for anyone who wishes to integrate general web services in WebTransactions, and describes the procedures involved in detail.

It is designed to supplement the introductory WebTransactions manual “Concepts and Functions” and the reference guide “Template Language” to provide all the information you need in order to integrate general web services in WebTransactions.

### **Scope of the manual**

The web frontend for web services runs on Solaris, Linux, Windows and BS2000/OSD. The information provided in this manual applies for all of these system platforms, unless indicated otherwise.



## **1.5 New features**

You will find an overview of all the changes in WebTransactions V7.5 in the WebTransactions manual “Concepts and Functions”.

The support of business objects is omitted.

## 1.6 Notational conventions

The following notational conventions are used in this documentation:

Name	Description
<code>typewriter font</code>	Fixed components which are input or output in precisely this form, such as keywords, URLs, file names
<i>italic font</i>	Variable components which you must replace with real specifications
<b>bold font</b>	Items shown exactly as displayed on your screen or on the graphical user interface; also used for menu items
[ ]	Optional specifications; do not enter the square brackets themselves
{ <i>alternative1</i>   <i>alternative2</i> }	Alternative specifications. You must select one of the expressions inside the curly brackets. The individual expressions are separated from one another by a vertical bar. Do not enter the curly brackets and vertical bars themselves.
...	Optional repetition or multiple repetition of the preceding components
	Important notes and further information
▶	Prompt telling you to do something.
	Refers to detailed information



## 2 Concept of the web frontend

Working with WebTransactions you can incorporate in your WebTransactions application any web service via the `WT_SOAP` interface. Information on integrating the SOAP protocol in WebTransactions is provided in the WebTransactions manual “Access to Dynamic Web Contents”.

In addition to the functions of the `WT_SOAP` interface WebTransactions offers a basic interface, or web frontend, for the integration of web services which are freely available in the network.

The web frontend is implemented in a template, that you can generate with WebLab. It controls the display on the browser and the communication with the web service. WebLab requires for the generation of this template a description file in WSDL format ( **Web Service Description Language**), that describes the operations of the web service.

Via the web frontend you can provide users with the functionality of the web service. Alternatively you profit from it by obtaining an overview of their functionality, and adapting the generated template to suit your needs.

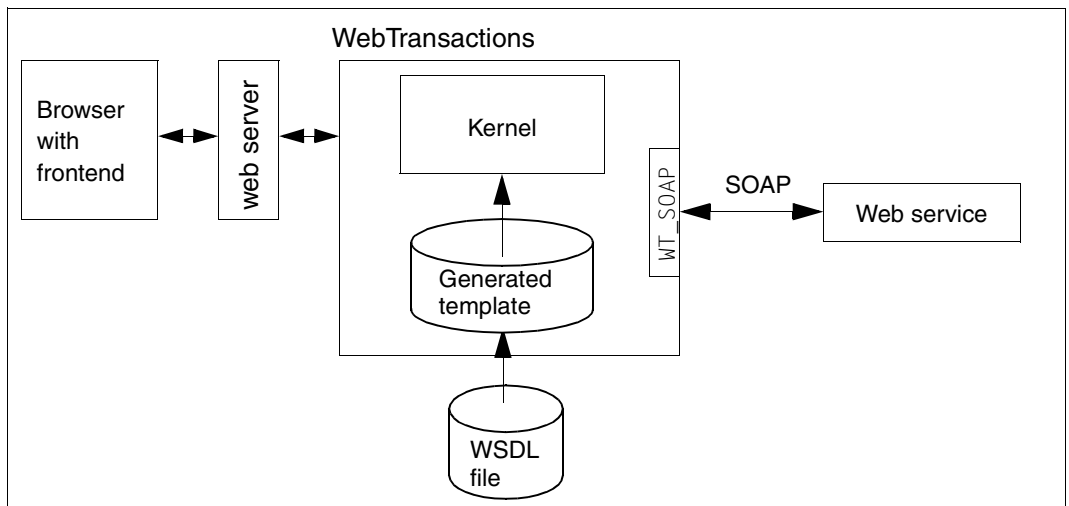


Figure 2: Flowchart for the web frontend

At runtime, WebTransactions uses the template to generate an HTML page for the browser, as well as an instance of the `WT_SOAP` class for web services. It then uses these instances to communicate with the relevant components. It is possible for you to influence the communication process and to subsequently edit the template to suit your needs.

To connect a component to WebTransactions via a web frontend, proceed as follows:

- ▶ Using WebLab, create a base directory.
- ▶ Create or supply the WSDL file.
- ▶ Generate the template for the web frontend.
- ▶ Run the template.
- ▶ Edit the template to suit your needs.

These steps are described in the following chapters.

---

## 3 Creating a base directory

Once you have installed WebTransactions on the WebTransactions server and WebLab on your Windows PC, you can use WebLab to create one or more base directories. A base directory includes all the files which configure WebTransactions for a specific application scenario.

If you de-install WebTransactions or install a new product version, the individual configurations are therefore retained.

### 3.1 Creating a base directory with WebLab

Before you can create a base directory for a WebTransactions application, the WebTransactions administrator must have created a user ID for you and then subsequently released one or more pools for this user ID in which you can create a base directory.

Before you create a base directory, it is recommended that you first create a project to store most important data required by WebLab when working with the WebTransactions application. When creating a project, you are automatically offered the opportunity to create a base directory.

To do this, proceed as follows:

- ▶ Call WebLab, e.g. via **Start/Programs/WebTransactions 7.5/WebLab**
  - ▶ There are two possibilities for starting to create a base directory:
    - ▶ Select the **Project/New...** command and when asked whether you want to create a base directory, answer **Yes**.
  - or
  - ▶ Choose the **Generate/Basedir...** command and specify that a new project is to be created when the relevant query appears.
- In both instances, the **Connect** dialog box is opened.
- ▶ In the **Connect** dialog box, enter the connection parameters and confirm with **OK**.
  - ▶ In the next dialog box, enter your user ID and password and click **OK**.

- ▶ In the **Create Basedir** dialog box, make the following entries:
  - Select the directory under which your base directory is to be created.
  - Enter a name for the new base directory.
  - Click **OK**.

Structure and contents of a base directory are described in the WebTransactions manual “Concepts and Functions” and in [section “Structure of a base directory” on page 16](#).

### Converting a base directory to a new version

- ▶ Select **Generate/Update Base Directory**. This opens the **Update Base Directory** dialog box.
- ▶ If you only want to change the links from the base directory to the new installation directory, select the **Update all links** option. Select this option when you have updated the files that are supplied or generated by WebTransactions.
- ▶ If all files which are copied or generated on creation of the base directory need to be recreated, select the **Overwrite all files** option.

## 3.2 Structure of a base directory

This section only describes the specifics of the base directory of the web frontend for web services.



For general information on the structure of base directories see WebTransactions manual “Concepts and Functions”.

### wsdl subdirectory

This subdirectory is created not with the base directory, but when you use web services. The WSDL files of the web services used, which are required by WebTransactions at runtime are stored in `wsdl`.



---

## 4 Generating a template

Once you have created a base directory for web services or you are connected to such a base directory, you can use WebLab to generate a template with the **Generate/Templates/for WebServices ...** command.

With this command you can generate templates for integrating general web services in WebTransactions.

To generate the required templates, you also need the following:

- a description file (WSDL file)
- the corresponding master template

### WSDL file

In the course of the generation process, you specify one or more description files, known as WSDL files (**WebService Description Language**). Each WSDL file will contain a description of precisely one component. If you specify several WSDL files, a template will be generated for each of the associated components. WSDL files can be stored locally on your own system or under the base directory.

Most web services are already described in WSDL, which means that a WSDL file for the respective service will be readily available on the web.

### Master template

A template is generated for each WSDL file on the basis of the special master template `WSDL.wmt`. This master template is stored in the subdirectory `web1ab` under the WebLab installation directory, and can be customized to suit your needs. The master template tags are described in the WebTransactions manual “Template Language”.

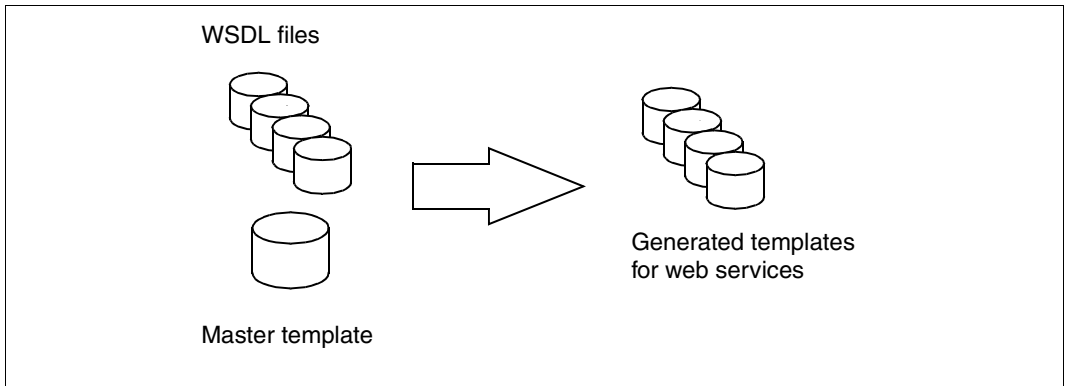


Figure 3: Generating templates for web services

The templates generated for general web services on the basis of WSDL files and a master template have the following predefined name:

```
wtWebService_componentname.htm
```

where *componentname* is dependent on the name of the WSDL file. They are stored in the base directory under `config/forms`, unless specified otherwise. If you wish to use another target directory, this must be located under `basedir/config`.

The generated templates can then be used as is or as the basis for more sophisticated visual layouts.

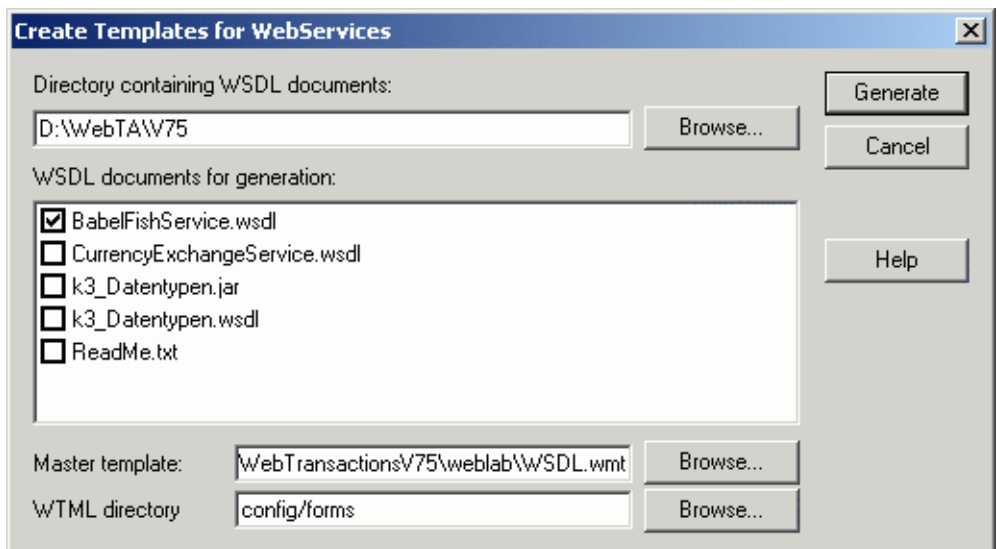
## Generating a template for a general web service with SOAP integration

Before generating a template, make sure that the following files are available:

- the WSDL file
- the master template `WSDL.wmt`

To generate a template for web services, proceed as follows:

- ▶ In WebLab, choose **Generate/Templates/for WebServices** to open the **Create Templates for WebServices** dialog box.



- ▶ Specify the directory containing the WSDL files to be used as the basis for generating the template, either by entering the directory path directly or by clicking **Browse** and making the appropriate selection. The **WSDL documents for Generation** list will then display all files in the specified directory.
- ▶ In **WSDL documents for Generation**, select the WSDL files to be used as the basis for generating the template.
- ▶ Check that the master template is set correctly (`WSDL.wmt`).
- ▶ In **WTML Directory**, enter the subdirectory under the base directory in which the generated template is to be stored.
- ▶ Click **Generate** to generate a template for integrating the web service in accordance with your specifications. In the process, the subdirectory `wsd1` will be created under the base directory and the specified WSDL file will be copied to this subdirectory.



---

## 5 Testing the web frontend

Once you have generated the template for integrating a web service, you are ready to test the web frontend. The generated template can be used as is and form a modifiable web frontend.

The `wsdl` subdirectory must contain the WSDL file for the web service. If the web service is modified in some way since the generation of the template, it is recommended that you regenerate the template with the new WSDL file.

For the flowchart for a web service frontend refer to [figure “Flowchart for the web frontend” on page 13](#).

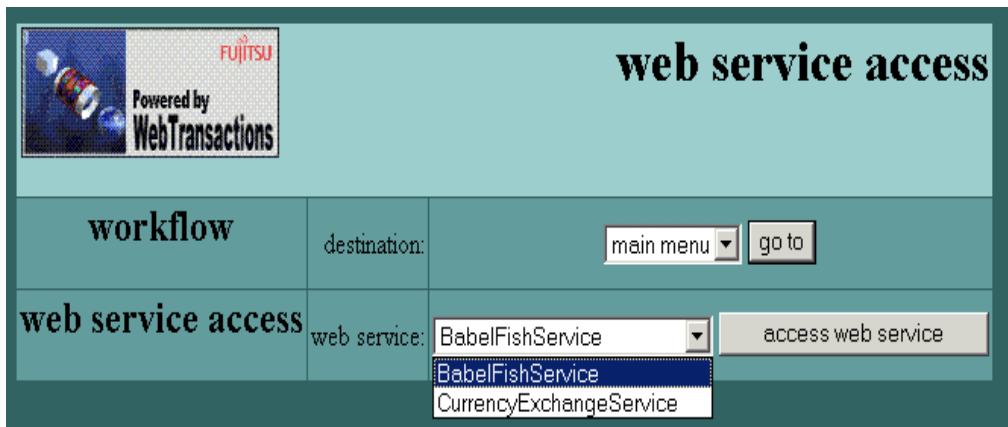
This chapter describes

- the general start template
- the web frontend

## 5.1 General start template

To start a web service frontend, proceed as follows:

- ▶ In WebLab, open a session by selecting **File/Start Session**.
- ▶ In the **Start Session** dialog box, enter the required data on the WebTransactions host and the base directory.
- ▶ Set the start template to `wtstart` and confirm with **OK**. The start template will then be displayed in the predefined browser.
- ▶ Click **WebService**. This opens the start template for the web service in the browser.



The **web service access** list will include all templates

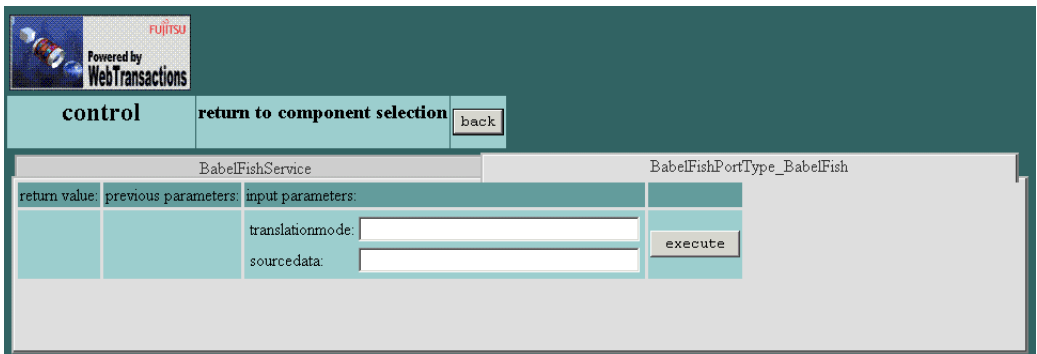
- whose names begin with `wtWebService_`
- which are located in the directory defined by the system object's `STYLE` and `LANGUAGE` attributes
- ▶ Choose a web service and confirm with **access Webservice**.

## 5.2 Web frontend for a web service

A web service is represented by a WebTransactions template. The interface itself is designed with the help of the WTBean `wtcTabs`. It includes a tab sheet for the web service, which is named after the web service in question. Here you can define general settings, such as the proxy settings for the integration via SOAP. Each method also has its own tab sheet containing the following information:

- the output or return value
- the last parameters entered
- the prescribed input parameters

A method is called by clicking the **execute** button.



The screenshot shows a web interface for the BabelFish method. At the top left, there is a logo for Fujitsu and the text "Powered by WebTransactions". Below this, there is a "control" tab and a "return to component selection" button with a "back" button next to it. The main area is divided into two tabs: "BabelFishService" and "BabelFishPortType\_BabelFish". The "BabelFishService" tab is active and contains a table with columns for "return value", "previous parameters", and "input parameters". The "input parameters" column has two rows: "translationmode:" with an input field, and "source data:" with an input field. To the right of these input fields is an "execute" button.

Figure 4: Tab sheet containing the web service BabelFish method





---

## 6 Editing templates

As a template programmer, you can feel free to edit the generated templates in WebLab, e.g. in order to make the resulting interface more attractive to the eye.

When working with WebLab, WebTransactions offers a considerable degree of support, particularly when it comes to things like using objects to transfer parameters, for example. For this purpose, WebTransactions generates an object for each structure used. Once the template is run, these objects are created and displayed in the WebLab object tree, from where they can be easily incorporated in the code using the normal WebLab utilities.

This chapter contains information on the following:

- accessing objects
- enhancing the layout of the generated interfaces
- master templates

At the end of the chapter, you will find a commented listing of the generated template.

## 6.1 Accessing objects

### 6.1.1 Integration via SOAP

In the case of integration via SOAP, each web service is mapped to a `WT_SOAP` object when generating the web frontend.

#### **WT\_SOAP class**

The `WT_SOAP` class is implemented in the template `wtSOAP.htm` and provides an access to web services via the SOAP protocol:

An instance of this class represents a web service which must be described in a WSDL file. `WT_SOAP` analyzes the WSDL file and provides the operations of a web service as methods (proxy methods) of the `WT_SOAP` class.

You will find a detailed description of the `WT_SOAP` class and corresponding methods in the WebTransactions manual “Access to Dynamic Web Contents”.

#### **Representation of a web service by a WT\_SOAP object**

A web service is represented by an object of the `WT_SOAP` class. For each web service an instance of the `WT_SOAP` class is created. This instance is labeled `WSDL_serial_number`.

#### **Calling web service operations**

It is possible to call proxy methods at a `WT_SOAP` object. In WebLab, a method call is inserted simply by dragging it from the object tree into the current template.

### Mapping SOAP data types to WScript data types

During generation, a method call including all the necessary parameters is generated for each web service operation. The table below shows how the parameter data types are mapped:

<b>SOAP data type</b>	<b>WScript data type</b>
string	string
boolean	boolean
float	number
double	number
int	number
short	number
long	number
decimal	number
struct (complexType)	Object
array	Array

## 6.1.2 Example of a simple data type

An entry field and an associated value is generated on the web frontend for each method parameter. If the parameter has a simple data type, the user input will be forwarded directly to the method.

*Example of a web frontend for a web service*

The purpose of the following web service is to translate a piece of text. The first parameter specifies the direction of the translation process, while the second contains the actual text to be translated.

### Generating the entry fields

Two entry fields are generated:

```
<TABLE BGCOLOR="#99CCCC">
  <TR>
    <TD>translationmode:</TD>
    <TD><INPUT TYPE="TEXT" CLASS="BOX"
NAME="translationmode_Translate" SIZE=32></TD>
  </TR>
  <TR>
    <TD>sourcedata:</TD>
    <TD><INPUT TYPE="TEXT" CLASS="BOX"
NAME="sourcedata_Translate" SIZE=32></TD>
  </TR>
</TABLE>
```

### Calling the web service

```
WSDL_0.service.Translate.port.TranslatePort.operation.Translate(
WT_POSTED.translationmode_Translate, WT_POSTED.sourcedata_Translate);
```

If you wish, you can change the graphical layout of the interface, e.g. by replacing one of the entry fields with a pick list:

```
<TR>
  <TD>translationmode:</TD>
  <TD>
    <SELECT NAME="translationmode_Translate">
      <OPTION VALUE="de_en" >German->English</OPTION>
      <OPTION VALUE="en_de" >English->German</OPTION>
    </SELECT>
  </TD>
</TR>
```

Since this pick list has the same name as the entry field, the operation call remains unchanged.

### 6.1.3 Example of a complex data type

If you wish to incorporate more complex data structures in the interface, you must ensure that the user has some means of entering individual values for the various elements within the data structure at the interface. This is because only one entry field will initially be generated for each complex data type. Proceed as follows:

- ▶ At the appropriate location in the template, replace the `INPUT` tag generated for the entire structure with several `INPUT` tags for each attribute of the structure.
- ▶ After sending, copy this posted data to the object attributes in the `Receive` script before the method call.

*Example of a web frontend for a web service with integration via SOAP*

The following is generated:

```
<wtOnCreateScript>
p0_Object_getTest = new Object(); //Object for the complex data structure
</wtOnCreateScript>

<input type="text" name="p0_Object_getTest" value="" />

<wtOnReceiveScript>

/* Method call; The object for the complex data structure is passed as a
   parameter*/

...getTest( proxyObjects.getTest_p0, ...);

</wtOnReceiveScript>
```

Extend the script as follows:

- ▶ Specify a separate `INPUT` tag for each attribute:

```
...
<input type="text" name="p0_Object_getTest_minutes" value="" />
...
```

- ▶ For each attribute, add the following lines to the receive script before the method call (this example applies for the `minutes` attribute):

```
...
proxyObjects.getTest_p0.minutes = WT_POSTED.p0_Object_getTest_minutes;
...
<input type="text" name="p0_Object_getTest" value="" />
```

This procedure allows you to generate the attribute and assign it the value entered by the user at the interface.

Other substitutions that must be carried out include:

```
<input type="text" name="p0_Object_getTest.minutes" value="">  
<input type="text" name="p0_Object_getTest.seconds" value="">  
<input type="text" name="p0_Object_getTest.hours" value="">  
<input type="text" name="p0_Object_getTest.date" value="">  
<input type="text" name="p0_Object_getTest.month" value="">  
<input type="text" name="p0_Object_getTest.year" value="">
```

## 6.2 Master template

The master template `WSDL.wmt` is supplied:

The supplied master template is installed under `installdir/web1ab`.

The table below lists a number of new master template tags (MT tags), which have been introduced for the WSDL protocol options only:

MT tag	Explanation
<code>%%SOURCE%</code>	Replaced by the name of the copied WSDL file in the base directory (e.g. <code>WSDL/BabelFishService.wsdl</code> )
<code>%%ObjectCreate%</code>	Initiates the generation of objects for complex structures
<code>%%MethodInterface%</code>	Initiates the generation of the interface and further processing of individual methods

The following MT tags may not be used for the generation of web frontends for web services in the master template:

- `%%Lines...%`
- `%%ReceiveCopies%`

## 6.3 Generated template

This section describes the generated template and the presentation of the associated interface for the support for general web services via SOAP integration.

The text below shows the `wtWebService_BabelFishService.htm` template, generated in section [“Generating a template for a general web service with SOAP integration”](#) on [page 19](#).

### Generation options

```
<html>
<wtrem>*****
**</wtrem>
<wtrem>** WTML document: BabelFishService
**</wtrem>
<wtrem>*****
**</wtrem>
<wtrem>**
**</wtrem>
<wtrem>** Document generation based on Master Template :
**</wtrem>
<wtrem>** C:\Programme\WebTransactionsV75\web\lab\WSDL.wmt
**</wtrem>
<wtrem>**
**</wtrem>
<wtrem>** Generated at Mon Jan 10 11:06:12 2010 **</wtrem>
<wtrem>**
**</wtrem>
<wtrem>** Options used by the generator :
**</wtrem>
<wtrem>** - %OPTIONS:
**</wtrem>
<wtrem>**      CommObj = WSDL_0
**</wtrem>
<wtrem>**      Source = wsdl/BabelFishService.wsdl
**</wtrem>
<wtrem>*****
**</wtrem>
<wtrem>** WebTransactions V7.5      Fujitsu Technology Solutions GmbH, 2010
**</wtrem>
<wtrem>*****
**</wtrem>
```



**Creating a proxy object with WT\_SOAP**

```

<wtOnCreateScript>
<!--
include ('wtSOAP');
if ( typeof WT_SYSTEM.WSDL_0 == 'undefined' )
    WT_SYSTEM.WSDL_0 = new Object();
if ( typeof WT_SYSTEM.WSDL_0["BabelFishService"] == 'undefined' )
    WT_SYSTEM.WSDL_0["BabelFishService"] = new Object();
if ( typeof WT_SYSTEM.WSDL_0["BabelFishService"].instance == 'undefined' )
{
    try
    {
        WSDL_0 = WT_SYSTEM.WSDL_0["BabelFishService"].instance = new
WT_SOAP('wsdl/BabelFishService.wsdl');
        WSDL_0.proxyObjects = proxyObjects;
    }
    catch ( exc )
    {
        document.writeln(exc);
        WT_SYSTEM.WSDL_0["BabelFishService"].ret_value = exc;
    }
}
else {
    WSDL_0 = WT_SYSTEM.WSDL_0["BabelFishService"].instance;
    proxyObjects = WSDL_0.proxyObjects;
}
WSDL_0_system = WT_SYSTEM;

//-->
</wtoncreatescript>

<wtif (WSDL_0_system.PROLOG)>
    <wtinclude Name="##WSDL_0_system.PROLOG#">
</wtif>

```

**HTML header: defining the WebTransactions format**

```

<head>
<title>WebTransactions V7.5 - WebService BabelFishService</title>
##WT_SYSTEM.CGI.HTTP_USER_AGENT.indexOf( 'MSIE' ) >= 0 ?
    '<meta http-equiv="Pragma" content="no-cache"/>' :
    '<meta http-equiv="Cache-Control" content="no-cache"/>'#
<wtif (WT_BROWSER.acceptClass)>
    <style type="text/css">
        input {
            font-size:    ##WT_BROWSER.charSize#px;
            font-family:  courier new, monospace;

```

```

    }
    input.box {
        border:      0 solid;
        padding:     1px 0 1px 0;
        margin-left: -1px;
        margin-top:  ##WT_BROWSER.marginTop#px;
        font-size:   ##WT_BROWSER.charSize#px;
        font-family: courier new, monospace;
        color:       #000000;
        background-color: #FFFFFF;
    }
    input.button {
        font-size:   ##WT_BROWSER.charSize#px;
        font-family: courier new, monospace;
        border-width: 1pt;
        margin-left: -1pt;
    }
    select {
        font-size:   ##WT_BROWSER.charSize#px;
        font-family: courier new, monospace;
    }
    pre {
        font-size:   ##WT_BROWSER.charSize#px;
        font-family: courier new, monospace;
        margin:      0;
    }
}
</style>
</wtif>
</head>
<body bgcolor="#336666" text="#000000">
<form WebTransactions name="BabelFishService">
<table cellspacing="1" cellpadding="2">
<tr>
<td colspan="3" align="right">

</td>
</tr>
<tr>
<td>
<wtif (WSDL_0_system.FORMTPL)>
<wtinclude Name="##WSDL_0_system.FORMTPL#">
</wtif>
</td>
</tr>
</table>

```

**Button for general control**

```

<tr bgcolor="#99CCCC">
  <td colspan="3" align="center">
    <h2>control</h2>
  </td>
  <td align="center"><h3>return to component selection</h3>
  </td>
  <td align="center">
    <input type="submit" name="Control" value="back"/>
  </td>
</tr>
</table>

```

**General control**

```

<!-- -----
- -->
<!-- main template control section
-->
<!-- -----
- -->
<wtonreceivescript>
  if (WT_POSTED.Control)
  {
    WT_SYSTEM.FORMAT = WT_SYSTEM.PREVIOUS_CONTROL_FORMAT;
    delete WT_SYSTEM.PREVIOUS_CONTROL_FORMAT;
    exitReceiveProcessing();
  }
  if (WT_POSTED.SetParameter)
  {
    if (WT_POSTED.Proxy)
    {
      if (WT_POSTED.ProxyPort)
        WSDL_0.setProxy(WT_POSTED.Proxy, WT_POSTED.ProxyPort);
      else WSDL_0.setProxy(WT_POSTED.Proxy);
    }
    if (WT_POSTED.ProxyUser)
    {
      if (WT_POSTED.ProxyPwd)
        WSDL_0.setProxyAuthorization(WT_POSTED.ProxyUser,
WT_POSTED.ProxyPwd);
      else WSDL_0.setProxyAuthorization(WT_POSTED.ProxyUser);
    }
  }
</wtonreceivescript>

```

## web service header

```

    <!-- -----
- -->
    <!-- begin of service operations section
-->
    <!-- -----
- -->
    <wtc name="wtcTabs" type="inline" version="7.5">
<wtcTitle>Tab Control <wtcPar
name="objectname">ComponentTabs</wtcPar></wtcTitle>
<script language="Javascript"
src="##WT_SYSTEM.WWDOCS_VIRTUAL#/javascript/wtcTabs_browserFunctions.js"
type="text/javascript">
</script>
<script>
<wtcPar name="objectname">ComponentTabs</wtcPar> = new wtcTabs(<wtcPar
name="width">950</wtcPar>,<wtcPar name="tabHeight">20</wtcPar>,'<wtcPar
name="ident">compTabs</wtcPar>',<wtcPar
name="foregroundColor">'#333333'</wtcPar>,<wtcPar
name="backgroundColor">'#DDDDDD'</wtcPar>,<wtcPar
name="borderColor">'#FFFFFF'</wtcPar>,<wtcPar
name="foregroundColorDimmed">'#999999'</wtcPar>,<wtcPar
name="backgroundColorSelectedDimmed">'#CCCCCC'</wtcPar>,<wtcPar
name="borderColorDimmed">'#DDDDDD'</wtcPar>);
</script>
<!-- ----->
<!-- begin of list of tabs -->
<!-- ----->
<wtcList name="Tab-List" minLength="1">
<wtcTitle>List of Tabs</wtcTitle>

```

## Tab sheets for controlling access via HTTP

```

<!------->
<!--      begin of list item      -->
<!------->
<wtcListItem>
  <script language="JavaScript" type="text/javascript">
<wtcPar name="objectname">ComponentTabs</wtcPar>.addTab(<wtcPar
name="titlename">'BabelFishService'</wtcPar>, <wtcPar
name="titlewidth">150</wtcPar>);
  </script>
<wtcBlock>
<wtcTitle><wtcPar name="titlename">'BabelFishService'</wtcPar></wtcTitle>
  <table cellspacing="3" cellpadding="2">
    <tr bgcolor="#669999">
      <td>
        further parameters:
      </td>
      <td>
        HTTP parameters:
      </td>
    </tr>
    <tr bgcolor="#99CCCC">
      <td>
        <table>
          <tr>
            <td>##WT_POSTED.Proxy || '&nbsp;'</td>
            <td>&nbsp;</td>
          </tr>
          <tr>
            <td>##WT_POSTED.ProxyPort || '&nbsp;'</td>
            <td>&nbsp;</td>
          </tr>
          <tr>
            <td>##WT_POSTED.ProxyUser || '&nbsp;'</td>
            <td>&nbsp;</td>
          </tr>
          <tr>
            <td>&nbsp;</td>
            <td>&nbsp;</td>
          </tr>
        </TABLE>
      </td>
      <td>
        <TABLE>
          <tr>
            <td>Proxy:</td>
            <td><input type="text" name="Proxy" size="32" /></td>
          </tr>
        </TABLE>
      </td>
    </tr>
  </table>

```

```

        </tr>
        <tr>
            <td>Proxy-Port:</td>
            <td><input type="text" name="ProxyPort" size="6"/></td>
        </tr>
        <tr>
            <td>Proxy-User:</td>
            <td><input type="text" name="ProxyUser" size="32"/></td>
        </tr>
        <tr>
            <td>Proxy-Password:</td>
            <td><input type="password" name="ProxyPwd" size="32"/></td>
        </tr>
    </table>
</td>
<td>
    <input type="submit" name="SetParameter" value="set parameters"/>
</td>
</tr>
</table>
</wtcBlock>
</div>
</wtcListItem>
<!------->
<!-- end of list item      -->
<!------->

```

There are blocks such as the one shown below for each method

```

<!------->
<!-- begin of list item      -->
<!------->
<wtcListItem>
    <script language="JavaScript" type="text/javascript">
        <wtcPar name="objectname">ComponentTabs</wtcPar>.addTab(<wtcPar
name="titlename">'BabelFishPortType_BabelFish'</wtcPar>, <wtcPar
name="titlewidth">120</wtcPar>);
    </script>
    <wtcBlock>
        <wtcTitle><wtcPar
name="titlename">'BabelFishPortType_BabelFish'</wtcPar></wtcTitle>

```

### User interface

- displaying return information
- displaying the last parameters specified
- generating entry fields for the method input parameters
- button for initiating the method

```
<table cellspacing="3" cellpadding="2">
  <tr bgcolor="#669999">
    <td>
      return value:
    </td>
    <td>
      previous parameters:
    </td>
    <td>
      input parameters:
    </td>
  <tr>
    <td>&nbsp;&nbsp;&nbsp;</td>
  </tr>
```

### Displaying return information

```
<tr bgcolor="#99CCCC">
  <td>
    <table>

<tr><td>##WT_SYSTEM.WSDL_0["BabelFishService"].ret_BabelFishPortType_BabelFish#</td>
</tr>
    </table>
  </td>
```

### Displaying the last parameters specified

```
<td>
  <table bgcolor="#99CCCC">
    <tr>
      <td>
        ##WT_POSTED.translationmode_BabelFishPortType_BabelFish ||
'&nbsp;&nbsp;&nbsp;';'#
      </td>
    </tr>
    <tr>
      <td>
        ##WT_POSTED.sourcedata_BabelFishPortType_BabelFish ||
'&nbsp;&nbsp;&nbsp;';'#
      </td>
```

```

        </tr>
    </table>
</td>

```

### Generating entry fields for the method input parameters

```

<td>
    <table bgcolor="#99CCCC">
        <tr>
            <td>translationmode:</td>
            <td><input type="text"
name="translationmode_BabelFishPortType_BabelFish" size="32"/></td>
        </tr>
        <tr>
            <td>sourcedata:</td>
            <td><input type="text"
name="sourcedata_BabelFishPortType_BabelFish" size="32"/></td>
        </tr>
    </table>
</td>

```

### Button for initiating the method

```

        <td>
            <input type="submit" name="BabelFishPortType_BabelFish"
value="execute"/>
        </td>
    </tr>
</table>

```

### Calling the proxy method with type-specific parameters (corresponding sections generated for each method)

```

<wtOnReceiveScript>
    if ( WT_POSTED.BabelFishPortType_BabelFish)
    {
        try {
            WT_SYSTEM.WSDL_0["BabelFishService"].ret_BabelFishPortType_BabelFish
= WSDL_0.service.BabelFishService.port.BabelFishPort.operation.BabelFish(
WT_POSTED.translationmode_BabelFishPortType_BabelFish,
WT_POSTED.sourcedata_BabelFishPortType_BabelFish);
        }
        catch ( exc ) {

            WT_System.WSDL_0["BabelFishService"].ret_BabelFishPortType_BabelFish = exc;}
        exitReceiveProcessing();
    }

```



```

        </wtonreceivescript>
        </wtcBlock>
    </div>
</wtcListItem>
<!------->
<!-- end of list item          -->
<!------->

```

### End of the template: activating the tab sheets

```

</wtcList>
<!------->
<!-- end of list of tabs      -->
<!------->
</span>
<script language="JavaScript" type="text/javascript">
<wtcPar name="objectname">ComponentTabs</wtcPar>.adjust();
<wtcPar name="objectname">ComponentTabs</wtcPar>.redefineFocus();
<wtcPar
name="objectname">ComponentTabs</wtcPar>.activate(##WT_POSTED['wtcTabs_<wtcPa
r name="ident">compTabs</wtcPar>_ActiveTab' ]||0#);
</script>
</wtc>

    <!-------
- -->
    <!-- end of service operations section
-->
    <!-------
- -->

    </form>
</body>
<wtif (WSDL_0_system.EPILOG)>
    <wtinclude Name="##WSDL_0_system.EPILOG#">
</wtif>
</html>

```



---

# Glossary

A term in *->italic* font means that it is explained somewhere else in the glossary.

## active dialog

In the case of active dialogs, WebTransactions actively intervenes in the control of the dialog sequence, i.e. the next *->template* to be processed is determined by the template programming. You can use the *->WTML* language tools, for example, to combine multiple *->host formats* in a single *->HTML* page. In this case, when a host *->dialog step* is terminated, no output is sent to the *->browser* and the next step is immediately started. Equally, multiple interactions between the Web *->browser* and WebTransactions are possible within **one and the same** host dialog step.

## array

*->Data type* which can contain a finite set of values of one data type. This data type can be:

- *->scalar*
- a *->class*
- an array

The values in the array are addressed via a numerical index, starting at 0.

## asynchronous message

In WebTransactions, an asynchronous message is one sent to the terminal without having been explicitly requested by the user, i.e. without the user having pressed a key or clicked on an interface element.

## attribute

Attributes define the properties of *->objects*.

An attribute can be, for example, the color, size or position of an object or it can itself be an object. Attributes are also interpreted as *->variables* and their values can be queried or modified.

### **Automask template**

A WebTransactions *->template* created by WebLab either implicitly when generating a base directory or explicitly with the command **Generate Automask**. It is used whenever no format-specific template can be identified. An Automask template contains the statements required for dynamically mapping formats and for communication. Different variants of the Automask template can be generated and selected using the system object attribute `AUTOMASK`.

### **base directory**

The base directory is located on the WebTransactions server and forms the basis for a *->WebTransactions application*. The base directory contains the *->templates* and all the files and program references (links) which are necessary in order to run a WebTransactions application.

### **BCAM application name**

Corresponds to the openUTM generation parameter `BCAMAPPL` and is the name of the *->openUTM application* through which *->UPIC* establishes the connection.

### **browser**

Program which is required to call and display *->HTML* pages. Browsers are, for example, Microsoft Internet Explorer or Mozilla Firefox.

### **browser display print**

The WebTransactions browser display print prints the information displayed in the *->browser*.

### **browser platform**

Operating system of the host on which a *->browser* runs as a client for WebTransactions.

### **buffer**

Definition of a record, which is transmitted from a *->service*. The buffer is used for transmitting and receiving messages. In addition there is a specific buffer for storing the *->recognition criteria* and for data for the representation on the screen.

### **capturing**

To enable WebTransactions to identify the received *->formats* at runtime, you can open a *->session* in *->WebLab* and select a specific area for each format and name the format. The format name and *->recognition criteria* are stored in the *->capture database*. A *->template* of the same name is generated for the format. Capturing forms the basis for the processing of format-specific templates for the WebTransactions for OSD and MVS product variants.

**capture database**

The WebTransactions capture database contains all the format names and the associated *->recognition criteria* generated using the *->capturing* technique. You can use *->WebLab* to edit the sequence and recognition criteria of the formats.

**CGI**

(Common Gateway Interface)

Standardized interface for program calls on *->Web servers*. In contrast to the static output of a previously defined *->HTML* page, this interface permits the dynamic construction of HTML pages.

**class**

Contains definitions of the *->properties* and *->methods* of an *->object*. It provides the model for instantiating objects and defines their interfaces.

**class template**

In WebTransactions, a class template contains valid, recurring statements for the entire object class (e.g. input or output fields). Class templates are processed when the *->evaluation operator* or the `toString` method is applied to a *->host data object*.

**client**

Requestors and users of services in a network.

**cluster**

Set of identical *->WebTransactions applications* on different servers which are interconnected to form a load-sharing network.

**communication object**

This controls the connection to an *->host application* and contains information about the current status of the connection, the last data to be received etc.

**conversion tools**

Utilities supplied with WebTransactions. These tools are used to analyze the data structures of *->openUTM applications* and store the information in files. These files can then be used in WebLab as *->format description sources* in order to generate WTML templates and *->FLD files*. COBOL data structures or IFG format libraries form the basis for the conversion tools. The conversion tool for DRIVE programs is supplied with the product DRIVE.

**daemon**

Name of a process type in Unix system/POSIX systems which runs in the background and performs no I/O operations at terminals.

### **data access control**

Monitoring of the accesses to data and ->*objects* of an application.

### **data type**

Definition of the way in which the contents of a storage location are to be interpreted. Each data type has a name, a set of permitted values (value range), and a defined number of operations which interpret and manipulate the values of that data type.

### **dialog**

Describes the entire communication between browser, WebTransactions and ->*host application*. It will usually comprise multiple ->*dialog cycles*. WebTransactions supports a number of different types of dialog.

- ->*passive dialog*
- ->*active dialog*
- ->*synchronized dialog*
- ->*non-synchronized dialog*

### **dialog cycle**

Cycle that comprises the following steps when a ->*WebTransactions application* is executed:

- construct an ->*HTML* page and send it to the ->*browser*
- wait for a response from the browser
- evaluate the response fields and possibly send them to the ->*host application* for further processing

A number of dialog cycles are passed through while a ->*WebTransactions application* is executing.

### **distinguished name**

The Distinguished Name (DN) in ->*LDAP* is hierarchically organized and consists of a number of different components (e.g. "country, and below country: organization, and below organization: organizational unit, followed by: usual name"). Together, these components provide a unique identification of an object in the directory tree.

Thanks to this hierarchy, the unique identification of objects is a simple matter even in a worldwide directory tree:

- The DN "Country=DE/Name=Emil Person" reduces the problem of achieving a unique identification to the country DE (=Germany).
- The DN "Organization=FTS/Name=Emil Person" reduces it to the organization FTS.
- The DN "Country=DE/Organization=FTS/Name=Emil Person" reduces it to the organization FTS located in Germany (DE).

**document directory**

->*Web server* directory containing the documents that can be accessed via the network. WebTransactions stores files for download in this directory, e.g. the WebLab client or general start pages.

**Domain Name Service (DNS)**

Procedure for the symbolic addressing of computers in networks. Certain computers in the network, the DNS or name server, maintain a database containing all the known host names and *IP numbers* in their environment.

**dynamic data**

In WebTransactions, dynamic data is mapped using the WebTransactions object model, e.g. as a ->*system object*, host object or user input at the browser.

**EHLLAPI****Enhanced High-Level Language API**

Program interface, e.g. of terminal emulations for communication with the SNA world. Communication between the transit client and SNA computer, which is handled via the TRANSIT product, is based on this interface.

**EJB****(Enterprise JavaBean)**

This is a Java-based industry standard which makes it possible to use in-house or commercially available server components for the creation of distributed program systems within a distributed, object-oriented environment.

**entry page**

The entry page is an ->*HTML page* which is required in order to start a ->*WebTransactions application*. This page contains the call which starts WebTransactions with the first ->*template*, the so-called start template.

**evaluation operator**

In WebTransactions the evaluation operator replaces the addressed ->*expressions* with their result (object attribute evaluation). The evaluation operator is specified in the form ##*expression*#.

**expression**

A combination of ->*literals*, ->*variables*, operators and expressions which return a specific result when evaluated.

**FHS****Format Handling System**

Formatting system for BS2000/OSD applications.

**field**

A field is the smallest component of a service and element of a *->record* or *->buffer*.

**field file (\*.fld file)**

In WebTransactions, this contains the structure of a *->format* record (metadata).

**filter**

Program or program unit (e.g. a library) for converting a given *->format* into another format (e.g. XML documents to *->WTScript* data structures).

**format**

Optical presentation on alphanumeric screens (sometimes also referred to as screen form or mask).

In WebTransactions each format is represented by a *->field file* and a *->template*.

**format type**

(only relevant in the case of *->FHS* applications and communication via *->UPIC*) Specifies the type of format: #format, +format, -format or \*format.

**format description sources**

Description of multiple *->formats* in one or more files which were generated from a format library (FHS/IFG) or are available directly at the *->host* for the use of “expressive” names in formats.

**function**

A function is a user-defined code unit with a name and *->parameters*. Functions can be called in *->methods* by means of a description of the function interface (or signature).

**holder task**

A process, a task or a thread in WebTransactions depending on the operating system platform being used. The number of tasks corresponds to the number of users. The task is terminated when the user logs off or when a time-out occurs. A holder task is identical to a *->WebTransactions session*.

**host**

The computer on which the *->host application* is running.

**host adapter**

Host adapters are used to connect existing *->host applications* to WebTransactions. At runtime, for example, they have the task of establishing and terminating connections and converting all the exchanged data.



**host application**

Application that is integrated with WebTransactions.

**host control object**

In WebTransactions, host control objects contain information which relates not to individual fields but to the entire *->format*. This includes, for example, the field in which the cursor is located, the current function key or global format attributes.

**host data object**

In WebTransactions, this refers to an *->object* of the data interface to the *->host application*. It represents a field with all its field attributes. It is created by WebTransactions after the reception of host application data and exists until the next data is received or until termination of the *->session*.

**host data print**

During WebTransactions host data print, information is printed that was edited and sent by the *->host application*, e.g. printout of host files.

**host platform**

Operating system of the host on which the *->host applications* runs.

**HTML**

(Hypertext Markup Language)  
See *->Hypertext Markup Language*

**HTTP**

(Hypertext Transfer Protocol)  
This is the protocol used to transfer *->HTML* pages and data.

**HTTPS**

(Hypertext Transfer Protocol Secure)  
This is the protocol used for the secure transfer of *->HTML* pages and data.

**hypertext**

Document with links to other locations in the same or another document. Users click the links to jump to these new locations.

**Hypertext Markup Language**

(Hypertext Markup Language)  
Standardized markup language for documents on the Web.

### Java Bean

Java programs (or *->classes*) with precisely defined conventions for interfaces that allow them to be reused in different applications.

### KDCDEF

openUTM tool for generating *->openUTM applications*.

### LDAP

(Lightweight **D**irectory **A**ccess **P**rotocol)

The X.500 standard defines DAP (Directory Access Protocol) as the access protocol. However, the Internet standard “LDAP” has proved successful specifically for accessing X.500 directory services from a PC.

LDAP is a simplified DAP protocol that does not support all the options available with DAP and is not compatible with DAP. Practically all X.500 directory services support both DAP and LDAP. In practice, interpretation problems may arise since there are various dialects of LDAP. The differences between the dialects are generally small.

### literal

Character sequence that represents a fixed value. Literals are used in source programs to specify constant values (“literal” values).

### master template

WebTransactions template used to generate the Automask and the format-specific templates.

### message queuing (MQ)

A form of communication in which messages are not exchanged directly, rather via intermediate queues. The sender and receiver can work at separate times and locations. Message transmission is guaranteed regardless of whether or not a network connection currently exists.

### method

Object-oriented term for a *->function*. A method is applied to the *->object* in which it is defined.

### module template

In WebTransactions, a module template is used to define *->classes*, *->functions* and constants globally for a complete *->session*. A module template is loaded using the `import()` function.

### MT tag

(Master Template tag)

Special tags used in the dynamic sections of *->master templates*.

**multitier architecture**

All client/server architectures are based on a subdivision into individual software components which are also known as layers or tiers. We speak of 1-tier, 2-tier, 3-tier and multitier models. This subdivision can be considered at the physical or logical level:

- We speak of logical software tiers when the software is subdivided into modular components with clear interfaces.
- Physical tiers occur when the (logical) software components are distributed across different computers in the network.

With WebTransactions, multitier models are possible both at the physical and logical level.

**name/value pair**

In the data sent by the *->browser*, the combination, for example, of an *->HTML* input field name and its value.

**non-synchronized dialog**

Non-synchronized dialogs in WebTransactions permit the temporary deactivation of the checking mechanism implemented in *->synchronized dialogs*. In this way, *->dialogs* that do not form part of the synchronized dialog and have no effect on the logical state of the *->host application* can be incorporated. In this way, for example, you can display a button in an *->HTML* page that allows users to call help information from the current host application and display it in a separate window.

**object**

Elementary unit in an object-oriented software system. Every object possesses a name via which it can be addressed, *->attributes*, which define its status together with the *->methods* that can be applied to the object.

**openUTM**

**(Universal Transaction Monitor)**

Transaction monitor from Fujitsu Technology Solutions, which is available for BS2000/OSD and a variety of Unix platforms and Windows platforms.

**openUTM application**

A *->host application* which provides services that process jobs submitted by *->clients* or other *->host applications*. openUTM responsibilities include transaction management and the management of communication and system resources. Technically speaking, the UTM application is a group of processes which form a logical unit at runtime.

openUTM applications can communicate both via the client/server protocol *->UPIC* and via the emulation interface (9750).

### **openUTM-Client (UPIC)**

The openUTM-Client (UPIC) is a product used to create client programs for openUTM. openUTM-Client (UPIC) is available, for example, for Unix platforms, BS2000/OSD platforms and Windows platforms.

### **openUTM program unit**

The services of an *->openUTM application* are implemented by one or more openUTM program units. These can be addressed using transaction codes and contain special openUTM function calls (e.g. KDCS calls).

### **parameter**

Data which is passed to a *->function* or a *->method* for processing (input parameter) or data which is returned as a result of a function or method (output parameter).

### **passive dialog**

In the case of passive dialogs in WebTransactions, the dialog sequence is controlled by the *->host application*, i.e. the host application determines the next *->template* which is to be processed. Users who access the host application via WebTransactions pass through the same dialog steps as if they were accessing it from a terminal. WebTransactions uses passive dialog control for the automatic conversion of the host application or when each host application format corresponds to precisely one individual template.

### **password**

String entered for a *->user id* in an application which is used for user authentication (*->system access control*).

### **polling**

Cyclical querying of state changes.

### **pool**

In WebTransactions, this term refers to a shared directory in which WebLab can create and maintain *->base directories*. You control access to this directory with the administration program.

### **post**

To send data.

### **posted object (wt\_Posted)**

List of the data returned by the *->browser*. This *->object* is created by WebTransactions and exists for the duration of a *->dialog cycle*.

**process**

The term “process” is used as a generic term for process (in Solaris, Linux and Windows) and task (in BS2000/OSD).

**project**

In the WebTransactions development environment, a project contains various settings for a ->*WebTransactions application*. These are saved in a project file (suffix *.wtp*). You should create a project for each WebTransactions application you develop, and always open this project for editing.

**property**

Properties define the nature of an ->*object*, e.g. the object “Customer” could have a customer name and number as its properties. These properties can be set, queried, and modified within the program.

**protocol**

Agreements on the procedural rules and formats governing communications between remote partners of the same logical level.

**protocol file**

- openUTM-Client: File into which the openUTM error messages as are written in the case of abnormal termination of a conversation.
- In WebTransactions, protocol files are called trace files.

**roaming session**

->*WebTransactions sessions* which are invoked simultaneously or one after another by different ->*clients*.

**record**

A record is the definition of a set of related data which is transferred to a ->*buffer*. It describes a part of the buffer which may occur one or more times.

**recognition criteria**

Recognition criteria are used to identify ->*formats* of a ->*terminal application* and can access the data of the format. The recognition criteria selected should be one or more areas of the format which uniquely identify the content of the format.

**scalar**

->*variable* made up of a single value, unlike a ->*class*, an ->*array* or another complex data structure.

### service (openUTM)

In *->openUTM*, this is the processing of a request using an *->openUTM application*. There are dialog services and asynchronous services. The services are assigned their own storage areas by openUTM. A service is made up of one or more *->transactions*.

### service application

*->WebTransactions session* which can be called by various different users in turn.

### service node

Instance of a *->service*. During development and runtime of a *->method* a service can be instantiated several times. During modelling and code editing those instances are named service nodes.

### session

When an end user starts to work with a *->WebTransactions application* this opens a WebTransactions session for that user on the WebTransactions server. This session contains all the connections open for this user to the *->browsers*, special *->clients* and *->hosts*.

A session can be started as follows:

- Input of a WebTransactions URL in the browser.
- Using the `START_SESSION` method of the `WT_REMOTE` client/server interface.

A session is terminated as follows:

- The user makes the corresponding input in the output area of this *->WebTransactions application* (not via the standard browser buttons).
- Whenever the configured time that WebTransactions waits for a response from the *->host application* or from the *->browser* is exceeded.
- Termination from WebTransactions administration.
- Using the `EXIT_SESSION` method of the `WT_REMOTE` client/server interface.

A WebTransactions session is unique and is defined by a *->WebTransactions application* and a session ID. During the life cycle of a session there is one *->holder task* for each WebTransactions session on the WebTransactions server.

### SOAP

(originally **S**imple **O**bject **A**ccess **P**rotocol)

The *->XML* based SOAP protocol provides a simple, transparent mechanism for exchanging structured and typecast information between computers in a decentralized, distributed environment.

SOAP provides a modular package model together with mechanisms for data encryption within modules. This enables the uncomplicated description of the internal interfaces of a *->Web-Service*.

**style**

In WebTransactions this produces a different layout for a *->template*, e.g. with more or less graphic elements for different *->browsers*. The style can be changed at any time during a *->session*.

**synchronized dialog**

In the case of synchronized dialogs (normal case), WebTransactions automatically checks whether the data received from the web browser is genuinely a response to the last *->HTML* page to be sent to the *->browser*. For example, if the user at the web browser uses the **Back** button or the History function to return to an “earlier” HTML page of the current *->session* and then returns this, WebTransactions recognizes that the data does not correspond to the current *->dialog cycle* and reacts with an error message. The last page to have been sent to the browser is then automatically sent to it again.

**system access control**

Check to establish whether a user under a particular *->user ID* is authorized to work with the application.

**system object (wt\_System)**

The WebTransactions system object contains *->variables* which continue to exist for the duration of an entire *->session* and are not cleared until the end of the session or until they are explicitly deleted. The system object is always visible and is identical for all name spaces.

**TAC**

See *->transaction code*

**tag**

*->HTML*, *->XML* and *->WTML* documents are all made up of tags and actual content. The tags are used to mark up the documents e.g. with header formats, text highlighting formats (bold, italics) or to give source information for graphics files.

**TCP/IP**

(Transport **C**ontrol **P**rotocol/**I**nternet **P**rotocol)

Collective name for a protocol family in computer networks used, for example, in the Internet.

### template

A template is used to generate specific code. A template contains fixed information parts which are adopted unchanged during generation, as well as variable information parts that can be replaced by the appropriate values during generation.

A template is a *->WTML* file with special tags for controlling the dynamic generation of a *->HTML* page and for the processing of the values entered at the *->browser*. It is possible to maintain multiple template sets in parallel. These then represent different *->styles* (e.g. many/few graphics, use of Java, etc.).

WebTransactions uses different types of template:

- *->Automask templates* for the automatic conversion of the *->formats* of MVS and OSD applications.
- Custom templates, written by the programmer, for example, to control an *->active dialog*.
- Format-specific templates which are generated for subsequent post-processing.
- Include templates which are inserted in other templates.
- *->Class templates*
- *->Master templates* to ensure the uniform layout of fixed areas on the generation of the Automask and format-specific templates.
- Start template, this is the first template to be processed in a WebTransactions application.

### template object

*->Variables* used to buffer values for a *->dialog cycle* in WebTransactions.

### terminal application

Application on a *->host* computer which is accessed via a 9750 or 3270 interface.

### terminal hardcopy print

A terminal hardcopy print in WebTransactions prints the alphanumeric representation of the *->format* as displayed by a terminal or a terminal emulation.

### transaction

Processing step between two synchronization points (in the current operation) which is characterized by the ACID conditions (**A**tomicity, **C**onsistency, **I**solation and **D**urability). The intentional changes to user information made within a transaction are accepted either in their entirety or not at all (all-or-nothing rule).



**transaction code/TAC**

Name under which an openUTM service or ->*openUTM program unit* can be called. The transaction code is assigned to the openUTM program unit during configuration. A program unit can be assigned several transaction codes.

**UDDI**

(**U**niversal **D**escription, **D**iscovery and **I**ntegration)

Refers to directories containing descriptions of ->*Web services*. This information is available to web users in general.

**Unicode**

An alphanumeric character set standardized by the International Standardisation Organisation (ISO) and the Unicode Consortium. It is used to represent various different types of characters: letters, numerals, punctuation marks, syllabic characters, special characters and ideograms. Unicode brings together all the known text symbols in use across the world into a single character set.

Unicode is vendor-independent and system-independent. It uses either two-byte or four-byte character sets in which each text symbol is encoded. In the ISO standard, these character sets are termed UCS-2 (Universal Character Set 2) or UCS-4. The designation UTF-16 (Unicode Transformation Format 16-bit), which is a standard defined by the Unicode Consortium, is often used in place of the designation UCS-2 as defined in ISO. Alongside UTF-16, UTF-8 (Unicode Transformation Format 8 Bit) is also in widespread use. UTF-8 has become the character encoding method used globally on the Internet.

**UPIC**

(**U**niversal **P**rogramming **I**nterface for **C**ommunication)

Carrier system for openUTM clients which uses the X/Open interface, which permits CPI-C client/server communication between a CPI-C-Client application and the openUTM application.

**URI**

(**U**niform **R**esource **I**dentifier)

Blanket term for all the names and addresses that reference objects on the Internet. The generally used URIs are->*URLs*.

**URL**

(**U**niform **R**esource **L**ocator)

Description of the location and access type of a resource in the ->*Internet*.

**user exit**

Functions implemented in C/C++ which the programmer calls from a ->*template*.

**user ID**

User identification which can be assigned a password (->*system access control*) and special access rights (->*data access control*).

**variable**

Memory location for variable values which requires a name and a ->*data type*.

**visibility of variables**

->*Objects* and ->*variables* of different dialog types are managed by WebTransactions in different address spaces. This means that variables belonging to a ->*synchronized dialog* are not visible and therefore not accessible in a ->*asynchronous dialog* or in a dialog with a remote application.

**web server**

Computer and software for the provision of ->*HTML* pages and dynamic data via ->*HTTP*.

**web service**

Service provided on the Internet, for example a currency conversion program. The SOAP protocol can be used to access such a service. The interface of a web service is described in ->*WSDL*.

**WebTransactions application**

This is an application that is integrated with ->*host applications* for internet/intranet access. A WebTransactions application consists of:

- a ->*base directory*
- a start template
- the ->*templates* that control conversion between the ->*host* and the ->*browser*.
- protocol-specific configuration files.

**WebTransactions platform**

Operating system of the host on which WebTransactions runs.

**WebTransactions server**

Computer on which WebTransactions runs.

**WebTransactions session**

See ->*session*

**WSDL**

(**Web Service Definition Language**)

Provides ->*XML* language rules for the description of ->*web services*. In this case, the web service is defined by means of the port selection.

**WTBean**

In WebTransactions ->*WTML* components with a self-descriptive interface are referred to as WTBeans. A distinction is made between inline and standalone WTBeans:

- An inline WTBean corresponds to a part of a WTML document
- A standalone WTBean is an autonomous WTML document

A number of WTBeans are included in of the WebTransactions product, additional WTBeans can be downloaded from the WebTransactions homepage [ts.fujitsu.com/products/software/openseas/webtransactions.html](http://ts.fujitsu.com/products/software/openseas/webtransactions.html).

**WTML**

(WebTransactions Markup Language)

Markup and programming language for WebTransactions ->*templates*. WTML uses additional ->*WTML tags* to extend ->*HTML* and the server programming language ->*WTScrip*t, e.g. for data exchange with ->*host applications*. WTML tags are executed by WebTransactions and not by the ->*browser* (serverside scripting).

**WTML tag**

(WebTransactions Markup Language-Tag)

Special WebTransactions tags for the generation of the dynamic sections of an ->*HTML* page using data from the ->*host application*.

**WTScrip**t

Serverside programming language of WebTransactions. WTScript

are similar to client-side Java scrip

t

in that they are contained in sections that are introduced and terminated with special tags. Instead of using ->*HTML-SCRIPT* tags you use ->*WTML-Tags*: `wtOnCreateScript` and `wtOnReceiveScript`. This indicates that these scrip

t

s are to be implemented by WebTransactions and not by the ->*browser* and also indicates the time of execution. OnCreate scrip

t

s are executed before the page is sent to the browser. OnReceive scrip

t

s are executed when the response has been received from the browser.

**XML**

(eXtensible Markup Language)

Defines a language for the logical structuring of documents with the aim of making these easy to exchange between various applications.

**XML schema**

An XML schema basically defines the permissible elements and attributes of an XML description. XML schemas can have a range of different formats, e.g. DTD (Document Type Definition), XML Schema (W3C standard) or XDR (XML Data Reduced).



---

## Abbreviations

BO	<b>B</b> usiness <b>O</b> bject
CGI	<b>C</b> ommon <b>G</b> ateway <b>I</b> nterface
DN	<b>D</b> istinguished <b>N</b> ame
DNS	<b>D</b> omain <b>N</b> ame <b>S</b> ervice
EJB	<b>E</b> nterprise <b>J</b> ava <b>B</b> ean
FHS	<b>F</b> ormat <b>H</b> andling <b>S</b> ystem
HTML	<b>H</b> ypertext <b>M</b> arkup <b>L</b> anguage
HTTP	<b>H</b> ypertext <b>T</b> ransfer <b>P</b> rotocol
HTTPS	<b>H</b> ypertext <b>T</b> ransfer <b>P</b> rotocol <b>S</b> ecure
IFG	<b>I</b> nteraktiver <b>F</b> ormat <b>G</b> enerator
ISAPI	<b>I</b> nternet <b>S</b> erver <b>A</b> pplication <b>P</b> rogramming <b>I</b> nterface
LDAP	<b>L</b> ightweight <b>D</b> irectory <b>A</b> ccess <b>P</b> rotocol
LPD	<b>L</b> ine <b>P</b> rinter <b>D</b> aemon
MT-Tag	<b>M</b> aster- <b>T</b> emplate- <b>T</b> ag
MVS	<b>M</b> ultiple <b>V</b> irtual <b>S</b> torage
OSD	<b>O</b> pen <b>S</b> ystems <b>D</b> irection
SGML	<b>S</b> tandard <b>G</b> eneralized <b>M</b> arkup <b>L</b> anguage
SOAP	<b>S</b> imple <b>O</b> bject <b>A</b> ccess <b>P</b> rotocol

## Abbreviations

---

SSL	<b>S</b> ecure <b>S</b> ocket <b>L</b> ayer
TCP/IP	<b>T</b> ransport <b>C</b> ontrol <b>P</b> rotocol/ <b>I</b> nternet <b>P</b> rotocol
Upic	<b>U</b> niversal <b>P</b> rogramming <b>I</b> nterface for <b>C</b> ommunication
URL	<b>U</b> niform <b>R</b> esource <b>L</b> ocator
WSDL	<b>W</b> eb <b>S</b> ervices <b>D</b> escription <b>L</b> anguage
wtc	<b>W</b> eb <b>T</b> ransactions <b>C</b> omponent
WTML	<b>W</b> eb <b>T</b> ransactions <b>M</b> arkup <b>L</b> anguage
XML	<b>e</b> Xtensible <b>M</b> arkup <b>L</b> anguage

---

# Related publications

## WebTransactions manuals

You can download all manuals from the Web address <http://manuals.ts.fujitsu.com>.

**WebTransactions**  
**Concepts and Functions**

Introduction

**WebTransactions**  
**Template Language**

Reference Manual

**WebTransactions**  
**Client APIs for WebTransactions**

User Guide

**WebTransactions**  
**Connection to openUTM Applications via UPIC**

User Guide

**WebTransactions**  
**Connection to OSD Applications**

User Guide

**WebTransactions**  
**Connection to MVS Applications**

User Guide

**WebTransactions**  
**Access to Dynamic Web Contents**

User Guide





---

# Index

## A

access  
  objects 25  
active dialog 43, 46  
architecture  
  web frontend 7  
array 43  
asynchronous message 43  
attribute 43  
automask template 44

## B

base data type 43  
base directory 44  
  converting to a new version 16  
  creating 15  
basic interface  
  general web frontend 13  
BCAM application name 44  
BCAMAPPL 44  
browser 44  
browser display print 44  
browser platform 44  
buffer 44

## C

capture database 45  
capturing 44  
CGI (Common Gateway Interface) 45  
class 45  
  templates 45  
  WT\_SOAP 26  
client 45  
cluster 45  
communication object 45

concept of the web frontend 13

conversion tools 45

## D

daemon 45  
data  
  dynamic 47  
data access control 46  
data type 46  
  WT\_SOAP 27  
dialog 46  
  active 46  
  non-synchronized 46, 51  
  passive 46, 52  
  synchronized 46, 55  
  types 46  
dialog cycle 46  
distinguished name 46  
document directory 47  
Domain Name Service (DNS) 47

## E

edit  
  templates 25  
EHLLAPI 47  
EJB 47  
entry page 47  
evaluation operator 47  
expression 47

## F

FHS 47  
field 48  
field file 48  
filter 48

fld file 48  
flowchart  
    frontend 13  
format 48  
    #format 48  
    \*format 48  
    +format 48  
    -format 48  
format description source 48  
format type 48  
function 48

**G**  
generate  
    template 19  
generated template 18, 31, 32

**H**  
holder task 48  
host 48  
host adapter 48  
host application 49  
host control object 49  
host data object 49  
host data print 49  
host platform 49  
HTML 49  
HTTP 49  
HTTPS 49  
hypertext 49  
Hypertext Markup Language (HTML) 49

**I**  
inline WtBean 59

**J**  
Java Bean 50

**K**  
KDCDEF 50

**L**  
LDAP 50  
literals 50

**M**  
master template 18, 25, 50, 56  
    tag 50  
message queuing 50  
method 50  
module template 50  
MT tag 31, 50  
multitier architecture 51

**N**  
name/value pair 51  
non-synchronized dialog 46, 51

**O**  
object 51  
    accessing 25  
openUTM 51  
    application 51  
    Client 52  
    program unit 52  
    service 54  
operations 46

**P**  
parameter 52  
passive dialog 46, 52  
password 52  
polling 52  
pool 52  
posted object 52  
posting 52  
process 53  
project 53  
property 53  
protocol 53  
protocol file 53

**R**  
recognition criteria 53  
record 53  
record structure 48

**S**  
scalar 53

service (openUTM) 54  
service node 54  
session 54  
    WebTransactions 54  
SOAP 54  
SOAP protocol 13  
standalone WTBear 59  
start template 22, 56  
style 55  
subdirectory  
    wsdl 16  
synchronized dialog 46, 55  
system access control 55  
system object 55

**T**  
TAC 57  
tag 55  
TCP/IP 55  
template 56  
    class 45  
    edit 25  
    generating 19  
    master 56  
    object 56  
    start 56  
    testing 22  
terminal application 56  
terminal hardcopy printing 56  
test  
    template 22  
Thread 48  
transaction 56  
transaction code/TAC 57

**U**  
UDDI 57  
Unicode 57  
UPIC 57  
URI 57  
URL 57  
user exits 57  
user ID 58  
UTM see openUTM

**V**  
value range of a data type 46  
variable 58  
visibility 58

**W**  
web frontend  
    architecture 7  
    basic interface 13  
    concept 13  
    flowchart 13  
    web service 23  
web server 58  
web service 13, 58  
WebTransactions  
    session 54  
WebTransactions application 58  
WebTransactions platform 58  
WebTransactions server 58  
WSDL 58  
wsdl  
    subdirectory 16  
WSDL file 13, 17  
WT\_SOAP 26  
    data types 27  
WTBear 59  
WTML 59  
WTML tag 59  
WTScrip 59  
WWW browser 44  
WWW server 58

**X**  
XML 59  
XML schema 59

