

WebTransactions V7.5

Connection to openUTM Applications via UPIC

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

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

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

1	Preface	7
1.1	Product characteristics	7
1.2	Architecture of WebTransactions for openUTM	9
1.3	WebTransactions documentation	11
1.4	Structure and target group of this manual	14
1.5	New features	14
1.6	Notational conventions	15
2	Installing WebTransactions	17
2.1	Installation	17
2.1.1	Windows	18
2.1.1.1	Installation via the user interface	18
2.1.1.2	Silent installation	19
2.1.2	Solaris	21
2.1.3	Linux	22
2.1.4	BS2000/OSD	23
2.1.5	WebLab installation	23
2.2	Licensing	24
3	Example session	25
3.1	Administering the WebTransactions server	25
3.1.1	Setting the browser	26
3.1.2	Starting the administration program	27
3.1.3	Entering licenses	28
3.1.4	Creating users	31
3.1.5	Creating a pool	32

3.1.6	Assigning the pool to a user	34
3.2	Connecting a host application to the Web	35
3.2.1	Creating a project	35
3.2.1.1	Creating a base directory	36
3.2.2	Saving the project	40
3.2.3	Generating templates from FHS formats	42
3.2.3.1	Generating the description file with IFG2FLD	42
3.2.3.2	Generating templates and field files from the description file	43
3.2.4	Define local host application names	46
3.2.5	Starting a session	47
3.3	Editing templates	53
3.3.1	Inserting a drop-down list for selecting a country	54
3.3.2	Replacing command input with buttons	60
3.3.3	Inserting a clickable image	62
3.4	Starting WebTransactions	64
3.4.1	Creating a start template	64
3.4.2	Starting a session with WebLab	68
3.4.3	Alternative ways of starting a WebTransactions application	69
4	Creating the base directory	71
<hr/>		
4.1	Creating a base directory with WebLab	71
4.2	Structure of a base directory	73
5	Generating templates	75
<hr/>		
5.1	Generating templates from FHS formats	76
5.1.1	Using IFG2FLD	77
5.1.2	Using WebLab to generate templates and FLD files from the format description source	79
5.2	Generating templates from FORMANT formats	82
5.2.1	Generating templates and FLD files using WebLab	82
5.3	Structure of the field files (FLD files)	84
5.4	Structure of the generated templates	89

6	Editing templates	99
6.1	Master template UTM.wmt	100
6.2	Designing templates	101
6.2.1	Defining the global layout	101
6.2.2	Customizing the interface	103
6.2.3	Designing the dialog sequence	103
6.3	Special characteristics of FHS/FORMANT partial formats	104
6.3.1	Communications sequence	104
6.3.2	Structure of the master template UTMpartial.wmt	106
6.3.3	Editing with partial format templates	117
6.4	Support for openUTM line mode	120
6.5	Binary data support	124
7	Configuring connections	125
7.1	Aligning WebTransactions and the host	127
7.2	Configuring the WebTransactions side	132
7.2.1	localapps file	132
7.2.2	Addressing the openUTM application using system attributes	133
7.2.3	upicfile	134
7.2.4	Declaring the server computer name	137
7.3	Configuring the openUTM (host) side	138
7.3.1	Adapting the openUTM generation	138
7.3.2	Declaring the client computer	141
7.4	BCMAP entries (BS2000/OSD)	142
8	Controlling communication	145
8.1	openUTM-specific attributes of the system object	145
8.1.1	Overview	146
8.1.2	Interaction between system object attributes and actions/methods	153
8.2	Host objects and attributes	159
8.2.1	Host objects for the individual format fields (host data objects)	159
8.2.2	Host control object WT_HOST_MESSAGE	165
8.2.3	Host control object WT_HOST_GLOBALS	169
8.2.4	Host control objects \$FIRST and \$NEXT	170

8.3	Terminal functions supported by the browser	171
8.3.1	Terminal functions supported	171
8.3.2	Mapping keys in wtKeysUTMFHS.js and wtKeysUTMFormant.js	174
8.3.3	Interaction between wtCommonBrowserFunctions.js and wt<browser>BrowserFunctions.js	179
8.3.4	Using the WT_BROWSER object	183
8.4	Start templates for openUTM	185
8.4.1	The openUTM-specific start template in the start template set (wtstartUTMV4.htm) .	186
8.4.2	WTBean wtcStartUPIC.wtc for the generation of a start template	191
8.5	Creating a new openUTM communication object (wtcUPIC)	193
8.6	Security through the openUTM user concept	195
8.7	RESTART - automatic restart	197
8.8	BADTAC - simulating the BADTAC event service	200
8.9	Automatic conversation chaining	201
8.10	Simulating function keys	202
8.11	Support for KDCSCUR	203
8.12	Targeted logon via specific LTERMs	204
	Glossary	205
	Abbreviations	223
	Related publications	225
	Index	227

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 WebTransactions for openUTM

The figure below illustrates the architecture of WebTransactions for openUTM:

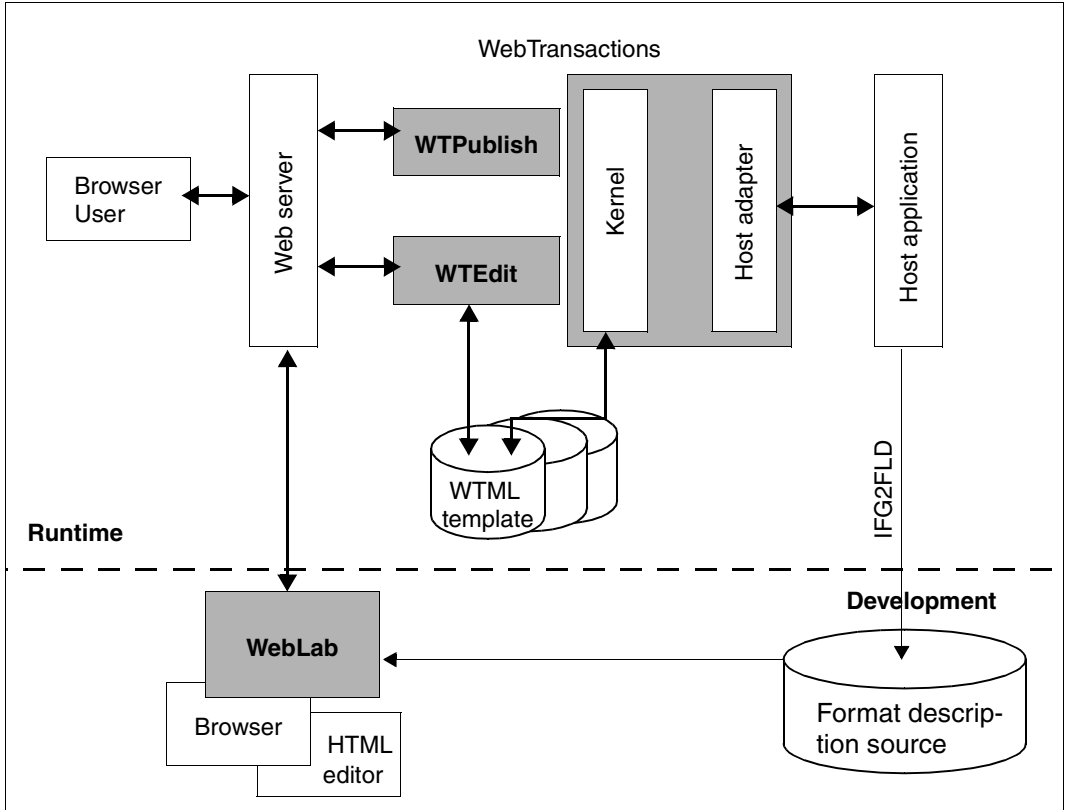


Figure 1: Architecture of WebTransactions for openUTM

Host adapter with Integrated UPIC protocol

WebTransactions for openUTM uses the UPIC protocol at runtime and during the development phase. This protocol is integrated in the host adapter and is used for managing communication between the WebTransactions kernel and the host application.

WebLab

WebLab is the development environment for WebTransactions and is used to carry out all the work necessary for connecting a host application, creating and editing of format-specific templates through to testing of the application. WebLab can be used to generate templates from existing FHS and FORMANT formats. These format-specific templates can be edited using WebLab.

WebLab does not need to be installed on the computer that is running WebTransactions. You can use WebLab on a different computer running the Windows operating system. All the data required in order to run a WebTransactions application is managed on the computer on which WebTransactions is running.

Runtime

At runtime, the templates control the graphical conversion and editing of formats.

Unicode support

The host adapter in WebTransactions for openUTM can also interpret data as Unicode characters at the UPIC interface.

The BS2000/OSD program `IFG2FLD` reads format descriptions from an IFG library and stores them in a format description source. The fields in an IFG library can contain the new attribute `Unicode`.

You can use WebLab to generate templates and field files (FLD files) from the format description source. During this conversion, the Unicode marker is automatically taken into account as of IFG2FLD Version 8.3 (see [chapter “Generating templates” on page 75](#)).

For further information refer to section [“Unicode support” on page 126](#).

Existing templates can be retained unchanged if the only change in a format was the change to Unicode fields. However, the assignment of the value `UTF-8` to the global system object attribute `CHARSET` must be inserted (see [section “Host control object WT_HOST_MESSAGE” on page 165](#)).

You will find information on the rules you must observe for templates in this context in the section [“Unicode support” on page 126](#).

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 (this User Guide)

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.
- A User Guide valid for all the supply units which describes the web frontend of WebTransactions that provides access to the general web services:

Web-Frontend for Web Services

This manual describes:

- The concept of web frontend for object-oriented backend systems.
- The generation of templates for the connection of general web services to WebTransactions.
- The testing and further development of the web frontend for general web services.

1.4 Structure and target group of this manual

This documentation is intended for users who want to use WebTransactions to connect openUTM dialog applications to the Web.

The individual chapters describe the necessary steps. If you have not yet worked with WebTransactions for openUTM, you should first read chapter 3, which presents an example session which will give you an initial insight into working with WebTransactions.

This manual provides all the openUTM-specific information necessary to complement the WebTransactions manuals “Concepts and Functions” and “Template Language”.

Scope of this description

WebTransactions for openUTM runs on the system platforms BS2000/OSD, Solaris, Linux and Windows. This documentation is valid for all WebTransactions platforms. However, if any information applies specifically to any one of these platforms, then this fact is explicitly noted in the text.



1.5 New features

This section only lists the openUTM-specific innovations of the WebTransactions version 7.5. For a general overview of the new features, refer to the WebTransactions manual “Concepts and Functions”.

Type of new feature	Description
New host data object attribute <code>Unicode</code>	page 161
New attribute <code>Unicode</code> at <code>WT_HOST_MESSAGE</code>	page 168

1.6 Notational conventions

The following notational conventions are used in this documentation:

Name	Description
typewriter font	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
bold font	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 Installing WebTransactions

The WebTransactions installation files can be downloaded from the Web.



Detailed information on the hardware and software requirements can be found in the release notice accompanying the product.

2.1 Installation

WebTransactions for openUTM consists of the host adapter via which communications with the openUTM applications transit, the WebTransactions runtime system and the host adapter for dynamic web contents.

WebTransactions for openUTM contains the installation package for the WebLab development environment which you can use to connect host applications to the WWW, edit the appearance of host formats and extend their functionality. You may need to install WebLab explicitly on your development machine (see [section “WebLab installation” on page 23](#)).



Before installing WebTransactions, make sure that the web server and, if necessary, Java are already installed.

Make a note of the Java installation directory together with the following information from the web server configuration:

- root directory for web pages (=document directory)
- CGI directory
- URL prefix for CGI programs

2.1.1 Windows

For Windows, WebTransactions is available as a Windows installer package (msi file) `WebTransactionsUTM75.msi` after it has been downloaded.

2.1.1.1 Installation via the user interface

To perform installation, you must possess Windows administrator rights. There are various ways of starting installation

- Via the **Settings/Control Panel** command in the Start menu.
- Via Windows Explorer.
Double click the msi file or single click this file with the right mouse button and then, in the context menu which appears, select the **Install** command.

Setting the web server settings

When you start `WebTransactionsUTM75.msi` you will see a series of dialog boxes in which you must enter the installation directory and the values for your web server:

- Root directory for web pages (= document directory).
- CGI directory and URL prefix.
- ISAPI directory and ISAPI prefix (optional).
- Directory of the Java2 library `jvm.dll` for Java integration (optional).

When you have entered the values, the installation will be started and the required components will be installed. If you install WebTransactions with an additional host adapter on the same system, these values will be taken over by the new installation.

Selecting components

You can now select all the components you want to install. In the **Select Installation Type** dialog box, select one of the following entries:

Typical or Complete

This will install all the WebTransactions components.

Custom

The installation program proposes the following components:

- WebTransactions runtime system.
- WebTransactions demo applications

2.1.1.2 Silent installation

For a silent installation, use the Windows installer `Msiexec.exe`. You can find a complete description of this command in, for example, the Windows online help. In order to run an installation with `Msiexec.exe` you will require administrator access rights.

Use the `Msiexec.exe` command with the following syntax:

```
Msiexec.exe /I "package" /q  
[INSTALLDIR="install-dir"]  
[DOCUMENTROOTDIR="documentroot-dir"]  
[HTTPSCRIPTSDIR="cgi-dir"]  
[JAVA2SYS="java-dir"]  
[ISPREFIX="isapi-prefix"]  
[URLPREFIX="cgi-prefix"]  
[ISAPICHECK="isapicheck"]  
[JAVA2CHECK="java2check"]
```

The parameters have the following meaning:

package

Path for the package to be installed (e.g. `C:\tmp\WebTransactionsUTM75.msi`).

install-dir

The WebTransactions installation directory.

Default value: `C:\Programme\WebTransactionsV75` or
`C:\Program Files\WebTransactionsV75`

documentroot-dir

Web server document directory.

Default value: `C:\InetPub\wwwroot`

cgi-dir The CGI directory of the web server.

Default value: `C:\InetPub\scripts`

java-dir

Directory of the Java2 library `jvm.dll`. This entry is only necessary when the support for the Java interface is to be installed.

isapi-prefix

URL prefix for ISAPI.

Default value: `scripts`

cgi-prefix

URL prefix for CGI.

Default value: `scripts`

isapicheck

This indicates if the ISAPI interface for WebTransactions is to be installed.

Possible values: Yes | No

Default value: No

java2check

This indicates if the support for the Java interface is to be installed.

Possible values: Yes | No

Default value: No

Example

```
Msiexec.exe /I "C:\tmp\WebTransactionsUTM75.msi" /q  
INSTALLDIR="D:\Program Files\WebTransactionsV75"  
DOCUMENTROOTDIR="C:\Program Files\Apache Group\Apache\htdocs"  
HTTPSCRIPTSDIR="C:\Program Files\Apache Group\Apache\cgi-bin"  
JAVA2SYS="D:\Program Files\Java\jdk1.6.0_13\jre\bin\client"  
URLPREFIX="cgi-bin" JAVA2CHECK="Yes"
```

2.1.2 Solaris

As usual, when you install WebTransactions, you use the installation procedure `pkgadd` with root authorization. To do this, enter the absolute path name of the unpacked product file:

```
pkgadd -d /absolute_path/filename
```

During installation, the following questions are displayed:

1. Should WebTransactions demos be installed?

If you enter `y` (yes) the WebTransactions demo applications are also installed.

2. Your Web Server has a 'document default directory'
Where is this directory?

Enter the corresponding path name.

3. The server uses an URL prefix to access WebTransactions CGI program.
URL prefix:

Enter the URL prefix that is set for CGI programs on your web server.

4. Your Web Server has a `cgi-bin` directory,
in which you install WebTransactions CGI-Program.
Where is this directory?

Here you enter the absolute path to the CGI directory which is configured for your web server.

During the installation, you will then see the URL which you use to start the demo application.

2.1.3 Linux

WebTransactions is available as a compressed archive for downloading and has the suffix `.gz` (for example, `webtransUTMV75.tar.gz`). You must first decompress this file using the command:

```
gunzip -d webtransUTMV75.tar
```

Please note that you must not specify the suffix `.gz`. You can then fetch the installation files from the archive using the `tar` command:

```
tar -xvf webtransUTMV75.tar
```

Start the installation procedure `doinstall` with root authorizations:

```
./doinstall
```

During installation you will be asked the following questions:

You can install WebTransactions into any directory.

```
Where is this directory ? [/opt]
```

You should now enter a different path name if you do not want WebTransactions to be installed under the default path `/opt`.

Your Web Server has a directory for CGI programs.

```
Where is this directory ? [/usr/local/httpd/cgi-bin]
```

Enter the corresponding path name.

Your Web Server uses an URL prefix to access the CGI programs in

```
/usr/local/httpd/cgi-bin
```

```
What is this prefix ? [cgi-bin]
```

Enter the URL prefix used for CGI programs on your web server

```
Are this settings OK ? [y]
```

Confirm your specifications to terminate installation.

2.1.4 BS2000/OSD

Standard installation is performed using the SOLIS procedure. If the IMON product (Installation MONitor) is started in the source system then you can also perform a standard installation using IMON.

To install POSIX, you can use the POSIX installation tool.

2.1.5 WebLab installation

When you install WebTransactions on any platform, the msi file for the installation of WebLab under Windows (`WebLab75.msi`) is written to the web server's document directory that is located below the directory `webtav75`.

Transferring the installer package to the development computer

The WebLab installer package can be downloaded to the required development computer via a browser call specifying the following URL:

`http://web-server/webtav75/wtdownload.htm`

Installing WebLab under Windows

When you have downloaded the WebLab installer package to your development computer, install the msi file as usual via the graphical user interface (see [page 18](#)) or with `Msiexec.exe` (see [page 19](#)).

In both cases, you need only specify the WebLab installation directory.

2.2 Licensing

After installation, you must configure the number of licenses present and the machine-specific activation key. To do this, you require the WebTransactions administration interface and select the **Licences** menu item. For more information on the administration program, see the WebTransactions manual “Concepts and Functions”.

3 Example session

In this chapter, you will learn about what you can do with WebTransactions and become familiar with a number of basic rules for working with WebTransactions. This example session is intended to serve as a procedural description which will show you how you can connect a host application to the Web simply and quickly.

In this example session, you will first use the administration program to create the conditions necessary for your work with WebLab and WebTransactions. Next you will use WebLab to connect the host application to the Web. You will then get to know the ways in which you can make global and format-specific changes in a template.



Please note that all path specifications are based on the assumption that WebTransactions has been installed in the initial directory.

3.1 Administering the WebTransactions server

Once you have installed WebTransactions for openUTM on a computer (see also [page 17](#)), you must create the conditions necessary for your work with WebTransactions and WebLab. To do this, you use the administration program that is described in the WebTransactions manual “Concepts and Functions”.

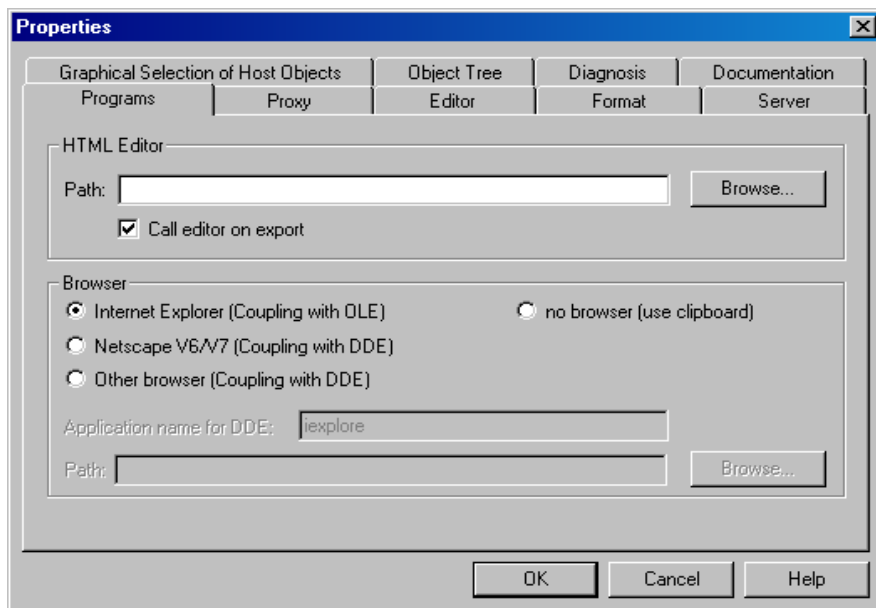
The first step in WebLab is to set the browser that WebLab is to use to operate the WebTransactions application. Your work with the administration program is subdivided into the following steps:

1. Enter the licenses
2. Set up the user
3. Create the pool
4. Assign the pool to the user

3.1.1 Setting the browser

Before you start to work, you should - in WebLab - set the browser which you want WebLab to use to operate the WebTransactions application. This step is only necessary if you are working with WebLab for the first time.

- ▶ Start WebLab with the command **Start/Programs/WebTransactions 7.5/WebLab**. The WebLab main window is displayed on the screen. For a detailed description of the main window and its components, refer to the WebTransactions manual „Concepts and Functions“ and the online help.
- ▶ In WebLab you can now select the **Options/Properties** command. The **Properties** dialog box is displayed on screen with the **Programs** tab open.



- ▶ In the lower section, **Browser**, select the browser which is installed on your computer, and specify how it is to be used by WebLab.
- ▶ Click on **OK** to confirm your settings.

3.1.2 Starting the administration program

- ▶ Choose the **Administration/Server** command to start the administration program initially. The dialog box **Administrate Server** opens on the screen.
- ▶ Under **URL of WTPublish**, click the **Change** button. The **URL of WTPublish** dialog box will be displayed.
- ▶ Select the **Protocol** to be used for the connection.
- ▶ In the other fields, enter the corresponding values for your host:

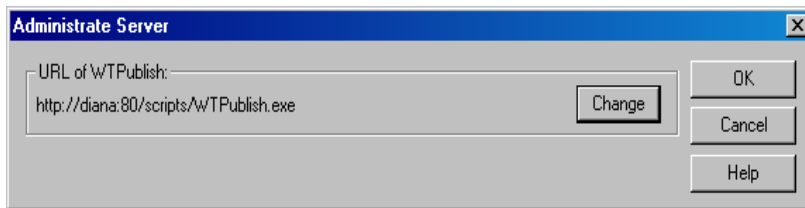
Server Host computer on which WebTransactions runs.

Port Corresponding port number.

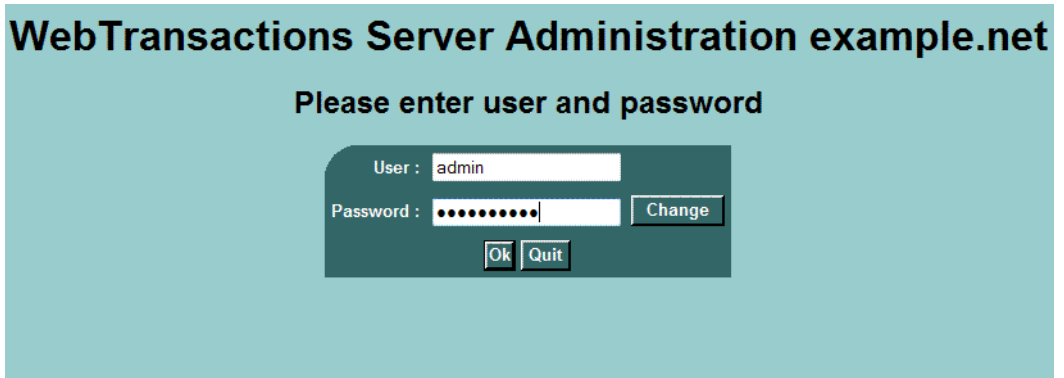
CGI-Path Path for the CGI program `WTPublish.exe`.

Program CGI program.

- ▶ Confirm with **OK**. The values will be entered in the **Administrate Server** dialog box.



- ▶ Confirm with **OK**. The administration program is started and the first window is shown in the browser.



- ▶ Log on as the `admin` user. This user is set up without a password when WebTransactions is set up. The licensing page is now displayed automatically.



If you are working with the administration program for the first time then, for reasons of security, you should assign a password for the `admin` user after login.

3.1.3 Entering licenses



Administration for server: **example.net**, 0 active sessions, 3 licenses installed

The screenshot shows the WebTransactions administration interface. On the left is a navigation menu with options: Licenses, Users, Pools, Applications, Sessions, Tools, Clusters, and Print. The main content area is divided into two sections: 'Registration' and 'Licenses'.

Registration Section:

Type	Description	Action
Online	To set the number of licenses you need an activation key from the WebTransactions license server. Use the button on the right to register online.	Register
Help Desk	Call our support team +49 (1805) 4040 Use the Info button on the right to get the required informations.	Info

Licenses Section:

Info

Server-Id: **693619ac**

Key: **Invalid**

Licensed Servers: **1**

Licensed concurrent users: **3**

On demand user licenses: **0**

License logging:

Action

[Enter new license](#)

Servers:

Licenses:

On Demand Licenses:

Days:

Key:

[Set](#)

At the bottom of the interface are buttons for [Save](#), [Load](#), [Refresh](#), and [Exit](#).

- ▶ Click the **Register** button.

This opens the registration page:

Type of License: Single Server Cluster

Server-Id:

Number of licenses:

On demand licenses:

Email address:

Key will be sent to this address!

Platform:

Installed adapters:

Reason for registration:

Name:

Company:

Department:

Street:

PC/City:

Country:

Sales Representative:

Remarks:

- ▶ To register licenses for a stand-alone server, click on **Single Server** under **Type of license**.
- ▶ Enter the number of servers that you want to license in the **Number of licences** field.
- ▶ Enter your e-mail address and additional parameters as required.

- ▶ Click **Request Key** to submit the form.
The license key will then soon be sent to the specified e-mail address.
- ▶ Enter the number of acquired licenses and the valid license key notified to you by e-mail in the **Licenses** and **Key** fields of the licensing page.
- ▶ Confirm by clicking **Set** followed by **Save**.
The licenses are activated and the new number of licenses is displayed in the status bar.

3.1.4 Creating users

- ▶ Click on the **Users** menu item to enter new users. The **Users** window is displayed in the browser.

The screenshot shows the WebTransactions administration interface. At the top left is the FUJITSU logo. Below it is a navigation menu with items: Licenses, Users (highlighted), Pools, Applications, Sessions, Tools, Clusters, and Print. At the top right, it says "Administration for server: example.net, 0 active sessions, 3 licenses installed". The main area is titled "Users" and contains a table with columns "Username", "Comment", and "Action".

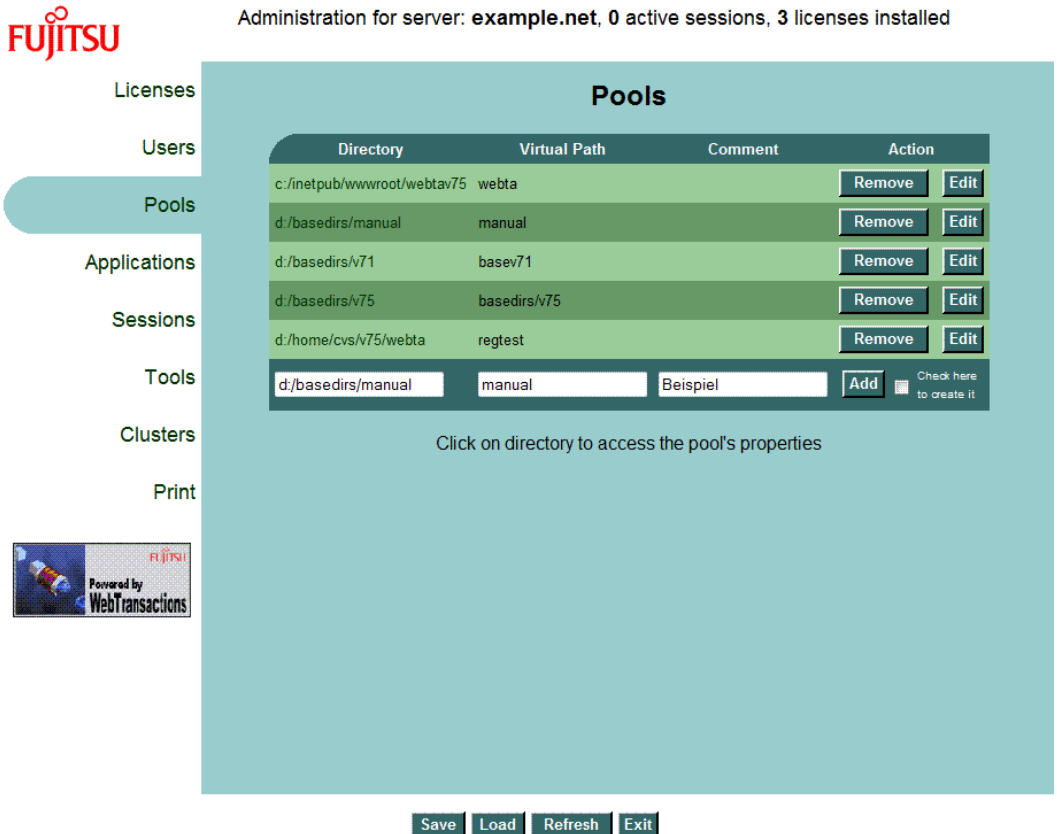
Username	Comment	Action
admin	Administrator	Change Password
frank		Change Password Remove
webtaman		Change Password Remove
<input type="text"/>	<input type="text"/>	Add

Below the table, it says "Click on user name to access the user's properties". At the bottom of the interface are buttons for "Save", "Load", "Refresh", and "Exit".

- ▶ Enter the name of the new user in the **Username** input field in the work area.
- ▶ If you wish, enter a description or comment for the user in the **Comment** field and click on **Add**. The user is now entered for operations with WebTransactions and WebLab. However, as yet the user has no rights. You must assign these.
- ▶ However, you should first click on the **Change Password** button and enter a password for the new user. You proceed in exactly the same way to assign a password for `admin`.

3.1.5 Creating a pool

- ▶ Next, click on the **Pools** menu item to create a directory under which base directories can be created. The **Pools** window is displayed in the browser.



The screenshot shows the administration interface for a server named 'example.net'. The top navigation menu includes Licenses, Users, Pools (highlighted), Applications, Sessions, Tools, Clusters, and Print. The main content area is titled 'Pools' and contains a table with the following data:

Directory	Virtual Path	Comment	Action
c:/inetpub/wwwroot/webtav75	webta		Remove Edit
d./basedirs/manual	manual		Remove Edit
d./basedirs/v71	basev71		Remove Edit
d./basedirs/v75	basedirs/v75		Remove Edit
d./home/cvs/v75/webta	regtest		Remove Edit

Below the table is a form for creating a new pool with the following fields and controls:

- Directory:
- Virtual Path:
- Comment:
- Buttons: Add, Check here to create it

Below the form, there is a note: "Click on directory to access the pool's properties". At the bottom of the interface, there are buttons for Save, Load, Refresh, and Exit.

- ▶ Enter the name of the directory in the **Directory** input field in the work area. Please note that if this directory does not already exist you must select the directory creation option.
- ▶ In the **Virtual Path** entry field, type the name of a directory below the web server's document directory that is allocated to the new pool. This directory corresponds to the start of the virtual path used by the web server to directly (i.e. without it being necessary to call WebTransactions) access the files of WebTransactions applications (e.g. images, entry page etc.) in this directory.



If you want to use base directories with identical names in different pools, the values for **Virtual Path** corresponding to the pools have to be different.

- ▶ You may also enter a description or comment for the pool in the **Comment** field before clicking on the **Add** button. The new pool is now entered for WebTransactions and WebLab operation. You can enter further pools as required.

You can now use WebLab to create the base directories under the pools which you have created in this way. However, as yet no WebLab user can access such a pool since the pool has not yet been assigned to a user.

3.1.6 Assigning the pool to a user

- ▶ In the pools table, click on the pool which you have just created. The **Pool** window with the newly created pool is displayed in the browser.

The screenshot shows the administration interface for a server. At the top, it says "Administration for server: example.net, 0 active sessions, 3 licenses installed". On the left is a navigation menu with items: Licenses, Users, Pools (highlighted), Applications, Sessions, Tools, Clusters, and Print. Below the menu is a small logo for "Powered by WebTransactions".

The main content area is titled "Pool d:/basedirs/manual". It contains a "Users" section with the subtitle "(who can create applications here)". Below this is a green bar that says "No users allowed yet". Underneath is a table with two rows: "frank" and "webtaman". The "webtaman" row is selected. To the right of the table is an "Add" button. Below the table, it says "Click on user name to access the user's properties".

At the bottom of the interface are four buttons: "Save", "Load", "Refresh", and "Exit".

This window displays the users who are permitted to access the new pool. Currently no user is assigned to this pool. A list displays all the users who are permitted to work with WebTransactions on this host.

- ▶ Click on an entry in this list to select the user you have just created and then click on the **Add** button. The selected user is entered as possessing access to this pool. You have now completed the preparations required in order to work with WebTransactions.
- ▶ Click on the **Save** button to save the current WebTransactions configuration.
- ▶ Click on the **Exit** button to terminate the administration program.

- ▶ Exit the browser.

3.2 Connecting a host application to the Web

Once you have performed the preparations for your work with WebTransactions and WebLab, you can use WebTransactions development environment - WebLab - to connect the host application to the WWW. To do this, you must perform the following steps:

1. Create the project
 - Create a base directory
2. Save the project
3. Generate templates from FHS formats
4. Define local application names
5. Start a session

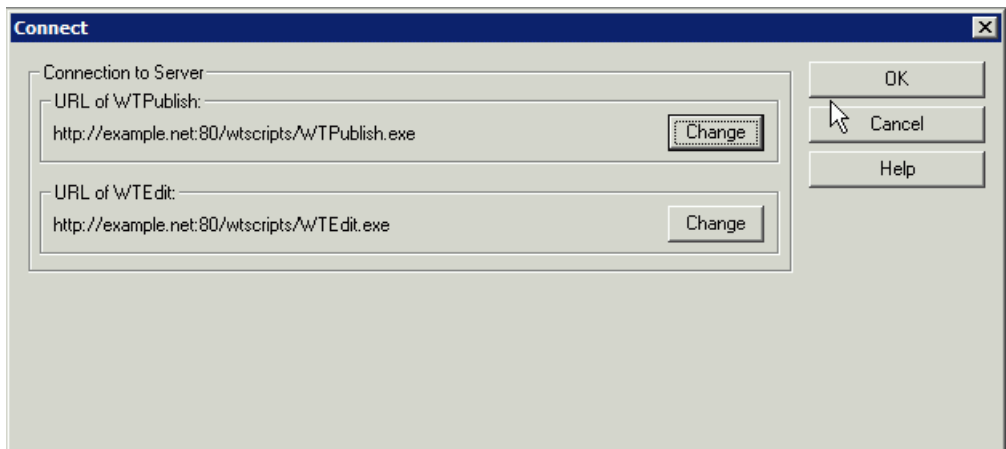
3.2.1 Creating a project

The project stores the most important data that is required by WebLab to generate and edit a WebTransactions application, e.g. the WebTransactions server data.

- ▶ To create a project, choose the **Project/New...** command.
- ▶ In the next dialog box, you are asked whether you want to generate a base directory. Click **Yes**. This opens the **Connect** dialog box, see next section.

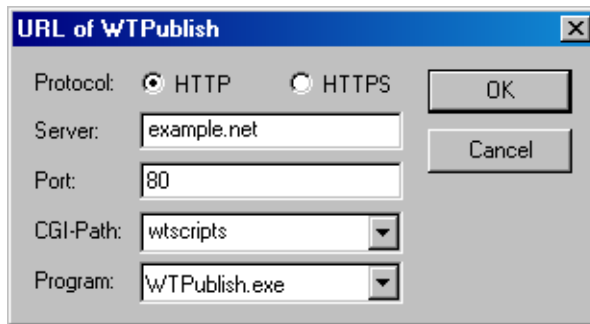
3.2.1.1 Creating a base directory

The base directory is the fundamental requirement for connecting a host application to the web using WebTransactions. This directory contains all the necessary files and links to the programs that constitute a WebTransactions application.



The base directory must always be located on the host on which WebTransactions is running. In the **Connect** dialog box, you enter this WebTransactions server and the paths to the CGI programs `WTPublish.exe` and `WTEdit.exe`.

- `WTEdit.exe` receives all WebLab requests. It performs all the necessary tasks on behalf of WebLab (which may be running on a different host) on the WebTransactions server (e.g. creation of a base directory) and enables WebLab to access running WebTransactions sessions.
- `WTPublish.exe` receives all requests from the browser. It starts new WebTransactions sessions or establishes connections to an open session for each subsequent dialog step.
- ▶ Under **Connection to server - URL of WTPublish**, click **Change**. The **URL of WTPublish** dialog box will be displayed.



- ▶ Select the **Protocol** to be used for the connection; in our example this is **HTTP**.
- ▶ In the **Server** field, enter the name of the host on which WebTransactions is running; in our example this is *example.net*.
- ▶ In the **Port** field, enter the corresponding port number; in our example this is *80*.
- ▶ Enter the path for the CGI program WTPublish; in our example this is *wtscripts*.
- ▶ Enter the name of the CGI program, in our example *WTPublish.exe*, and then confirm with **OK**. These values will now be taken over by the **Connect** dialog box.
- ▶ Repeat this procedure for the entries under **URL of WTEdit**. Once again enter the values for your host; in our example these are:

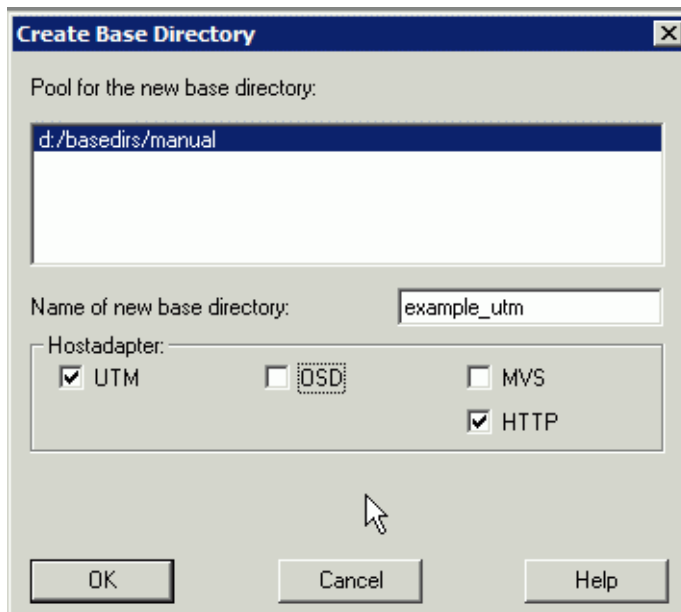
Server	<i>example.net</i>
Port	<i>80</i>
CGI Path	<i>wtscripts</i>
Program	<i>WTEdit.exe</i>

- ▶ When you have finished, in the **Connect** dialog box, click **OK**. The connection to the WebTransactions host computer will now be established with the values entered.

First, however, you must log on to WebTransactions. The **Name and Password** dialog box is opened to allow you to do this.



- ▶ Enter the user name and password that you specified in the [section “Creating users” on page 31](#).
- ▶ Click on **OK** to confirm. The **Create Basedir** dialog box is displayed on the screen.



The upper list of this dialog box displays the pools under which the logged on user is able to create base directories on the WebTransactions server.

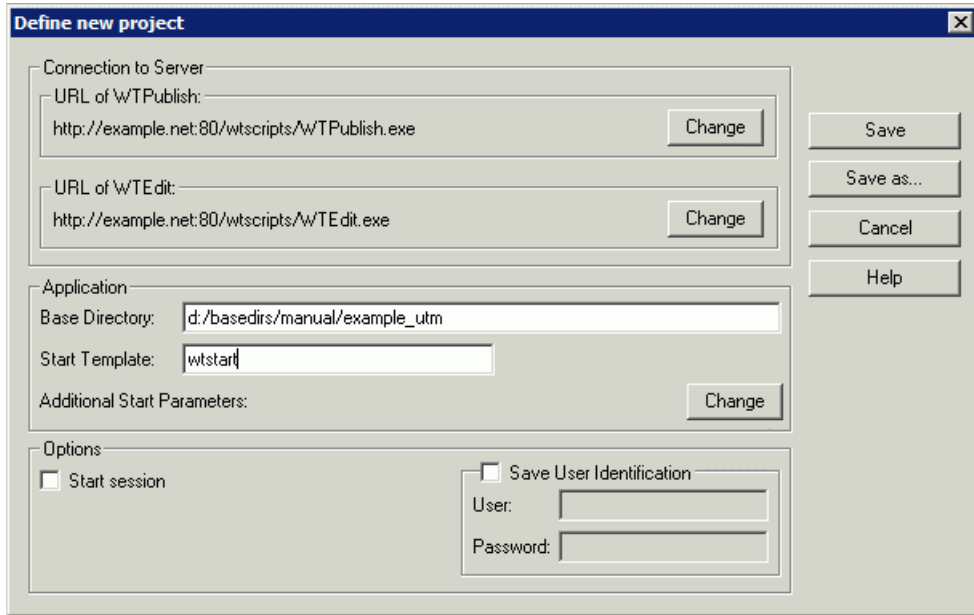
- ▶ In the list, click on the pool which you created in the [section “Creating a pool” on page 32](#).
- ▶ Enter a name in the **Name of new Base Directory** input field, here *example_utm*.
- ▶ Next select the host adapter via which WebTransactions communicates with the host application, here *UTM*. Only those host adapters that are actually installed are displayed. The host adapter for HTTP is preset by default.
- ▶ Click on **OK** to confirm your input. The **Create Basedir** dialog box is closed and the base directory is generated using the specified values at the WebTransactions server. In the WebLab main window, a message window is opened below the work area. This displays the progress of the operation.

The **Define New Project** dialog box is now opened, see next section.

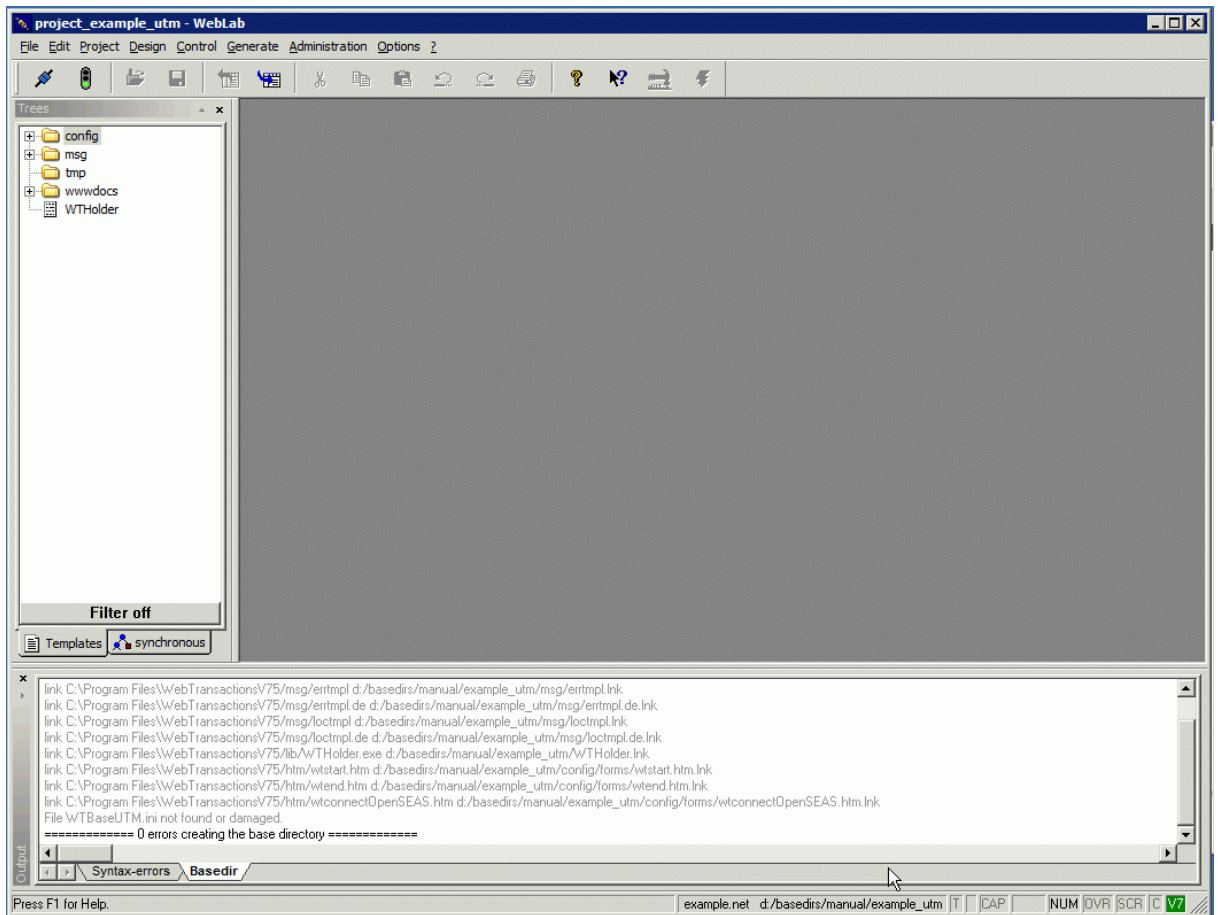
A start template that takes the user directly to the first format in the host application will be created at the end of the example session (see [section “Creating a start template” on page 64](#)).

3.2.2 Saving the project

You define the settings for the newly created project in the **Define New Project** dialog box.



- ▶ In this example, you should accept all the default settings and save the project with **Save as...**
This opens the **Save As** dialog box.
- ▶ In this dialog box you select the directory in which you want to save the project and enter a name for the project file.
- ▶ Click on **Save**.
The project file is created with the suffix `.wtp` in the selected directory. The name of the project file is displayed in the WebLab title bar.



You are then connected with your new base directory. For a full overview of the files and directories that are created, see [section “Structure of a base directory” on page 73](#).

3.2.3 Generating templates from FHS formats

In order to generate templates from FHS formats, you must first create a format description source from the IFG library under OSD using the `IFG2FLD` program. The format description source contains all field names defined originally by the developer of the host formats using the formatting system. It is transferred to the WebLab host, where it is used by WebLab to generate the templates and field files.

3.2.3.1 Generating the description file with IFG2FLD

- ▶ Transfer one of the following LMS libraries from the WebTransactions installation directory `lib` in binary format to the OSD host to the user ID under which the IFG library is located. On the OSD host name this library `WTIFG2FLD.LMS`.

Library	Meaning
<code>WTifg2f1dFTP.lms</code>	For transfer with FTP
<code>WTifg2f1dopenFT.lms</code>	For transfer with openFT

- ▶ Log on at the OSD host under this ID.
- ▶ Use the following SDF command to redirect `SYSLST` to a file.
`/ASSIGN=SYSLST description-file`
- ▶ Start `IFG2FLD` with:
`/START=PROGRAM FROM=FILE=*PHASE(LIBRARY=WTIFG2FLD.LMS,ELEMENT=IFG2FLD)`
- ▶ In the `IFG2FLD MAPFILE` command, specify the library containing the formats to be converted:
`MAPFILE LIB.EUROSI.FORMATS`
- ▶ If various user profiles are defined in the IFG library, you must use the `PROFILE` command to select the same profile that was used to prepare the openUTM application for use:
`PROFILE user-profile`
- ▶ You can use the `*ALL` operand of the `CONVERT` command to convert all the formats in the library:
`CONVERT *ALL`

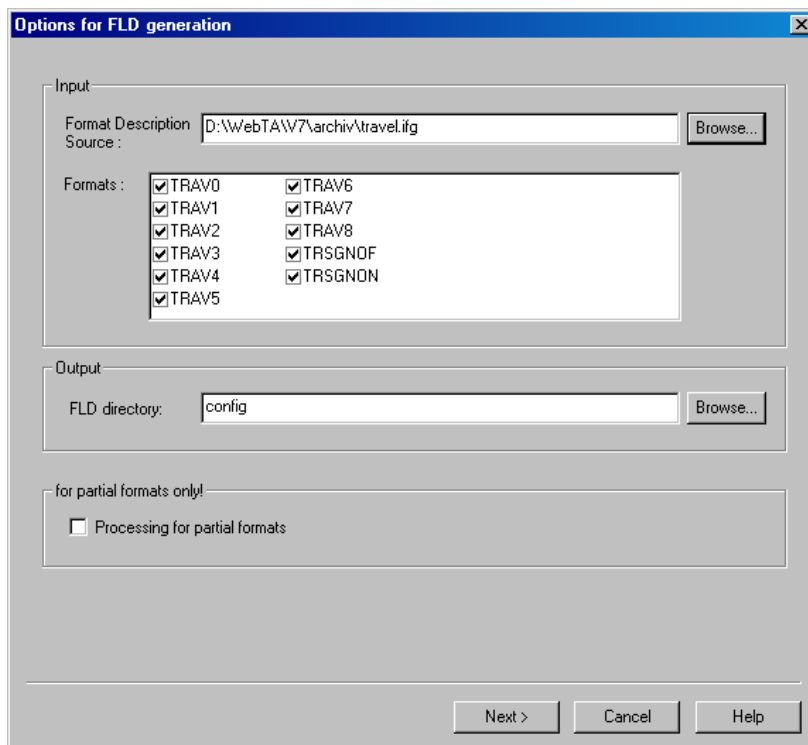
Alternatively, you can select specific formats for conversion:

```
CONVERT TRAV0
CONVERT TRAV1
...
```

- ▶ The `END` command now triggers generation of the description file and terminates
IFG2FLD:
END
- ▶ Enter the following SDF command to reassign `SYSLST` to the system default:
`/ASSIGN=SYSLST *P`
- ▶ Transfer the generated format description source in text format to the system on which WebLab is installed. In this example, the file is stored under
`D:\WebTA\V75\archiv\travel.ifg`.

3.2.3.2 Generating templates and field files from the description file

- ▶ Call WebLab on the development system.
- ▶ Select **Generate/Templates/from IFG library**. This opens the **Options for FLD generation** dialog box.



- ▶ In the **Input** field, enter the name of the format description file (in this case, `D:\WebTA\V75\archiv\travel.ifg`). The **Formats** list then displays the names of all formats whose descriptions can be read by WebLab in the specified format description source.

- ▶ In the list, click on the formats for which templates and FLD files are to be generated.
- ▶ In the **FLD directory** field, define the directory in which the newly generated FLD files are to be stored.
- ▶ Confirm your entries with **Next >**. The **Generate FLD and Template** dialog box then appears on your screen.

Generate FLD and Template

Input : FLD file(s)

Basedir: config

FLD file(s): "TRAV0.fld" "TRAV1.fld" "TRAV2.fld" "TRAV3.fld" "TRAV4.fld" "TR

Options

Master template: C:\Program Files\WebTransactions\75\weblab\UTM.wmt Browse...

Host protocol: UTM Communication object: UTM_0

Generation method

Class templates

Inline script

Display attributes:

No Static Dynamic

Static text First line as menu bar

Display euro symbol

Output :

WTML directory: config/forms Browse...

Use application prefix (same value as Communication object)

Continue with 'Capture from FLD files'

< Back Finish Cancel Help

You can use the options in this dialog box to control the generation of templates and their subsequent conversion.

- ▶ Check the default settings for **Master Template** (UTM.wmt) and **Host Protocol** (UTM).

- ▶ Under **Communication Object**, assign a name to the communication object. In this example, the default setting `UTM_0` is used. If you wish to open several connections within a session, however, it makes sense to name the communication object individually. The name of the communication object must also be specified in the start template.



Please note that this entry is case-sensitive.

- ▶ Under **Generation method**, select *Inline script*.
- ▶ In the **WTML Directory** field, select the subdirectory of the `forms` directory in which the newly generated templates are to be stored.
- ▶ Select **Finish** to generate the FLD files and templates.

The `sample_utm\config` directory then contains the generated field files (`*.fld`), while the `sample_utm\config\forms` directory contains the generated templates (`*.htm`).

The templates are ready to use and can be tested.

In [section “Structure of the generated templates” on page 89](#), you will find a corresponding generated template.

3.2.4 Define local host application names

In the localapps file, you have to define the local application names. Create the following entries in the file `sample_utm\localapps` file:

```
*file: localapps
FREE          UPICPTR0
FREE          UPICPTR1
FREE          UPICPTR2
...
FREE          UPICPTR9
```

Explanation

10 free local application names are defined for the openUTM application. This means that up to 10 WebTransactions sessions can be simultaneously connected to the openUTM application.

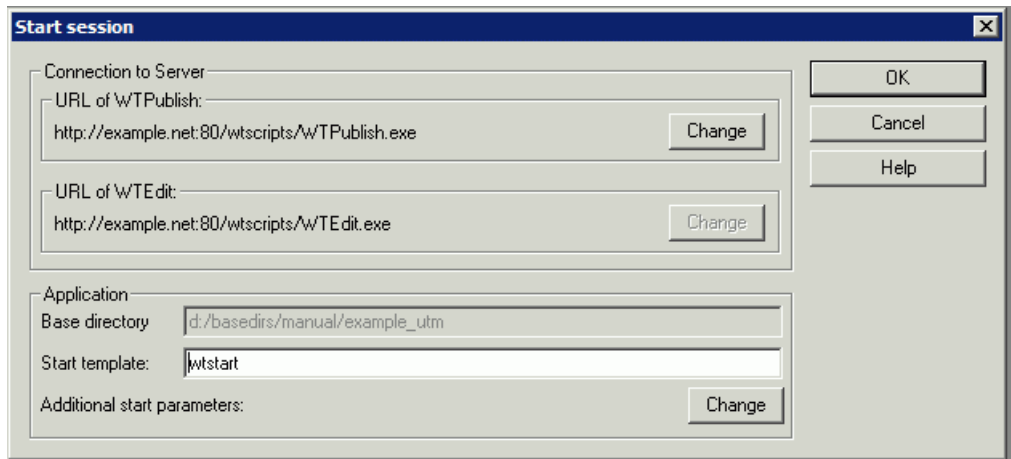


In this example, you specify the configuration parameters of the openUTM application at start of the session. Alternatively, you can enter the parameters into the upicfile, see [section “upicfile” on page 134](#).

3.2.5 Starting a session



Once you have created the base directory, you can start a session to the host application.

- ▶ Choose the **File/Start session** command. The **Start session** dialog box is displayed on the screen.





In this dialog box the connection data such as the web server name, CGI program path and the base directory name have already been taken over from the project settings. You just need to specify the name of the start template with which the host application is to be started.

- ▶ Enter the name of a start template in the **Start Template** dialog box, here `wtstart.htm`. `wtstart` is a supplied start template which is copied into the base directory and can be used for all host applications.
- ▶ Click on **OK** to start the session. The dialog box is closed. The set browser is opened and the general start template `wtstart` is displayed and calls for a new WebTransactions session with the start template `wtstart`. `wtstart` displays a form in the browser window.


 		main menu	
		[WebTransactions: test environment]	
status	base directory:	d:/basedirs/manual/beispiel_utm	
workflow	PROTOCOL:	HTTP ▾	
	private WT_SYSTEM:	<input checked="" type="checkbox"/>	
	name of new communication object:	UTM_0	
	create new communication:	create	
	connect webService	webService	
	terminate session	quit	
system parameters	STYLE:		
	LANGUAGE:		
	DEFAULT_FORMAT:	wtstart	
	TIMEOUT_APPLICATION:	120 (2 minutes) ▾	
	TIMEOUT_USER:	600 (10 minutes) ▾	
	COMMUNICATION_INTERFACE_VERSION:	3.0 or higher ▾	
	WTML_VERSION:	3.0 or higher ▾	
	SEARCH_HOST_OBJECTS:	<input type="checkbox"/>	

In this form of the general start template, you can now enter the connection parameters for WebTransactions in order to set up a new communication object.

- ▶ Select the `UTMV4` entry in the **PROTOCOL** pick list.
- ▶ Specify the name of the communication object, here `UTM_0`. The name of the communication object must correspond to the name which you used when generating the template.
- ▶ Now click on the **create** button to create a new communication object. Your specifications are processed by WebTransactions and the openUTM-specific start template `wtstartUTMV4.htm` continues checking and displays the next form.

 		<h2>UTMV4 communication</h2>		
status	communication object:	WT_HOST.UTM_0		
	connection:	down		
workflow	destination:	main menu ▾	<input type="button" value="go to"/>	
	access host:	<input type="button" value="open"/>	<input type="button" value="open in linemode"/>	
	parameters:	<input type="button" value="update"/>	<input type="button" value="reset"/>	
connection parameters directly	APPLICATION_NAME:	VTV10TRM		
	HOST:	NAME:	HOST001	
		or IP_ADDRESS:	<input type="text"/>	
		with HOST_PORT:	<input type="text"/>	
TAC:	MMENUE			
UPIC_CODE_CONVERSION:	<input checked="" type="checkbox"/>			
... or via upicfile	SYM_DEST:	<input type="text"/>		
additional connection parameters	APPLICATION_PREFIX:	<input type="checkbox"/>		
	CONVERSATION_TAC:	<input type="text"/>		
	CUT_TAC_FIELD:	<input checked="" type="checkbox"/>		
	HOST_CHAR_CODE:	ASCII ▾	<input type="text"/>	
	DISPLAY_EURO:	<input type="checkbox"/>		
	FLD:	TRAV0 ▾		
	BADTAC:	<input type="text"/>		
	LOCAL_APPLICATION:	<input type="text"/>		
	UPIC_TRACE:	<input type="checkbox"/>		
	UPIC_LIB:	<input type="text"/>		
	UTM_PATH:	<input type="text"/>		
	SECURITY_TYPE:	NONE ▾		
	USER:	<input type="text"/>		
	PASSWORD:	<input type="text"/>		
NEW_PASSWORD:	<input type="text"/>			
RESTART:	<input type="checkbox"/>			

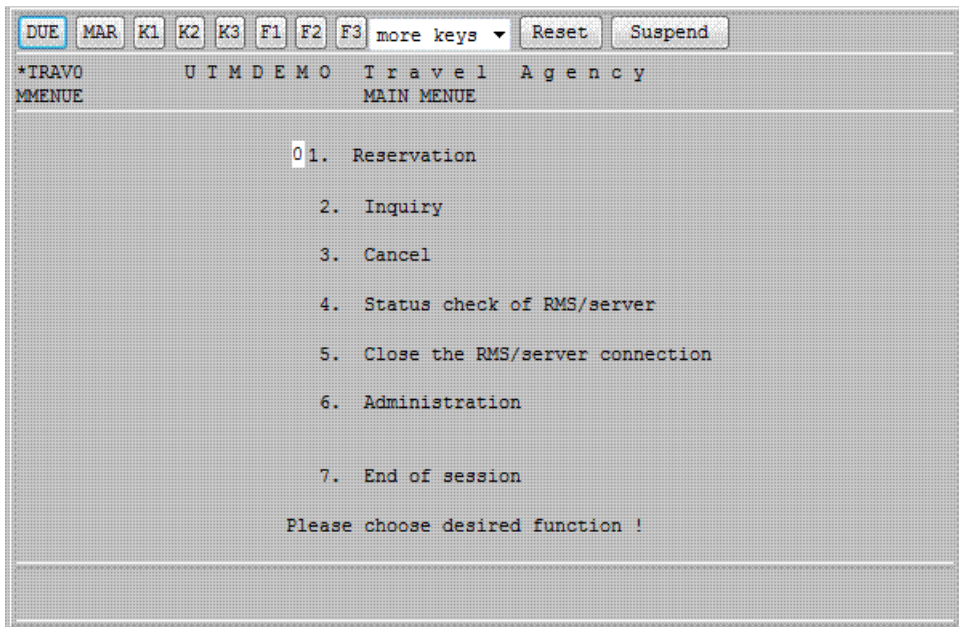
- ▶ In the **APPLICATION_NAME** field, enter the name of the openUTM application, in this example *VTV10TRV*.
- ▶ For **HOST** specify the name of the computer (**NAME** field) on which the openUTM application is running, in this example *HOST0001*.
- ▶ In the **TAC** field, enter the transaction code that starts the conversation in the openUTM application, in this example *MMENUE*.
- ▶ In the **FLD** pick list, select *TRAV0* as the first format description to be used. In the remaining fields, you can set the system object attributes with the same name.
- ▶ Click on **open** to establish a connection with the host application.



UTMV4 communication

status	communication object:	WT_HOSTUTM_0
	connection:	established
workflow	destination:	main menu <input type="button" value="go to"/>
	access host:	<input type="button" value="send"/> <input type="button" value="receive"/> <input type="button" value="close"/> <input type="button" value="enter dialog"/>
	parameters:	<input type="button" value="update"/> <input type="button" value="reset"/>
connection parameters directly	APPLICATION_NAME:	VTV10TRV
	HOST:	NAME: HOST001 or IP_ADDRESS: <input type="text"/> with HOST_PORT: <input type="text"/>
	TAC:	MMENUE
	UPIC_CODE_CONVERSION:	<input checked="" type="checkbox"/>
... or via upicfile	SYM_DEST:	<input type="text"/>
additional connection parameters	APPLICATION_PREFIX:	<input type="text"/>
	CONVERSATION_TAC:	<input type="text"/>
	CUT_TAC_FIELD:	<input checked="" type="checkbox"/>
	HOST_CHAR_CODE:	ASCII <input type="text"/>
	DISPLAY_EURO:	<input type="text"/>
	FLD:	TRAV0 <input type="text"/>
	BADTAC:	<input type="text"/>
	LOCAL_APPLICATION:	<input type="text"/>
	UPIC_TRACE:	<input type="text"/>
	UPIC_LIB:	<input type="text"/>
	UTM_PATH:	<input type="text"/>
	SECURITY_TYPE:	NONE <input type="text"/>
	USER:	<input type="text"/>
	PASSWORD:	<input type="text"/>
NEW_PASSWORD:	<input type="text"/>	
RESTART:	<input type="text"/>	
global host attributes	Padding:	OUT MSG
	Detect:	#f <input type="text"/>
	UnDetect:	#00 <input type="text"/>
	Read:	Modified <input type="text"/>
	FieldLength:	Effective length <input type="text"/>
	Update:	Only <input type="text"/>
[UTM start parameters]	Cursor:	ATTR <input type="text"/>

- ▶ In the **global host attributes** section, you can set the attributes of the WT_HOST_GLOBALS object if necessary.
- ▶ Select **enter dialog**. This displays the first template of the openUTM service (TRAVO).



If you now want to terminate the connection to the host, first you have to terminate the host application. In this example session you enter the value **7** (for **End of Session**) and then click **DUE** or press the return key to send the screen. Processing again branches to the openUTM-specific start template (see also [section “The openUTM-specific start template in the start template set \(wtstartUTMV4.htm\)” on page 186](#)). Select **main menu** and click on the **go to** button to return to the general start template. You can now select **quit** to exit the WebTransactions application.

In the example session you proceed as follows:

- ▶ Enter **1** (for **Reservation**).
- ▶ Click **DUE** or press the return key to send the screen. The next format of the application (trav1.htm) is displayed in the browser (see the following section).

3.3 Editing templates

In this section, we use the example of `trav1.htm` to present a number of ways of enhancing templates.

Interface of the generated template



The interface provided by the automatically generated template resembles an FHS format in terms of appearance and functionality.



The only difference between the generated template and the FHS format is the buttons **DUE**, **MAR**, **K1**, ... **Suspend**. These are always generated as the standard template `wtKeysUTM.htm` is included. A complete listing of the template can be found in [section "Structure of the generated templates" on page 89](#).

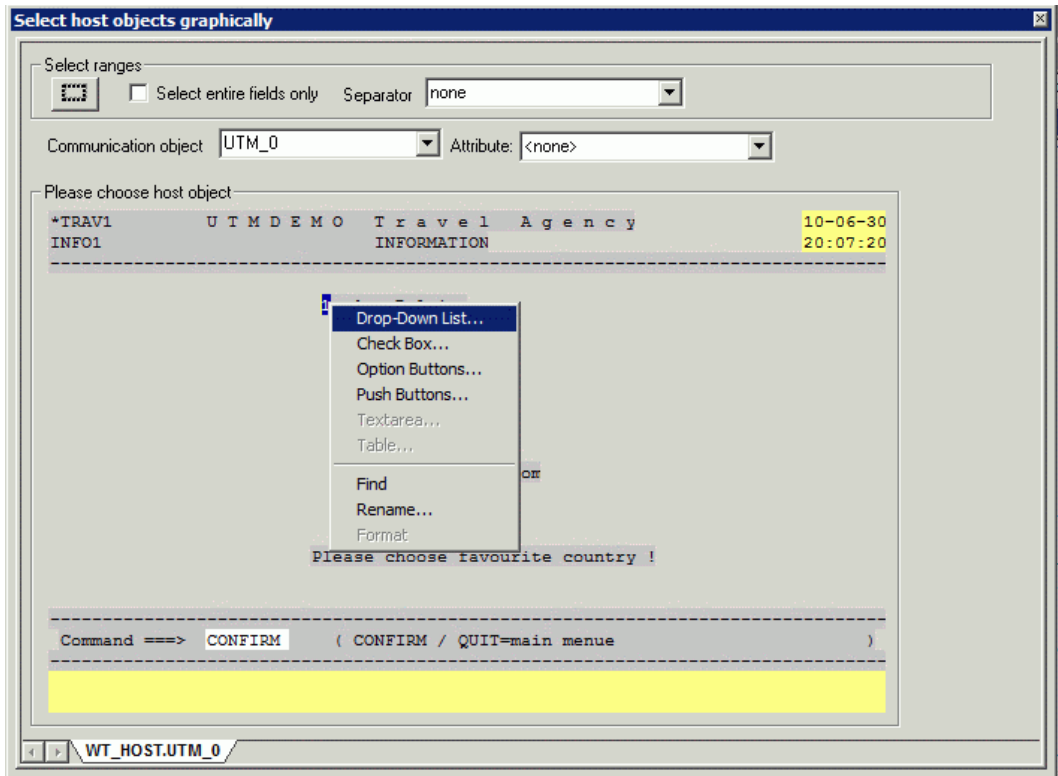
3.3.1 Inserting a drop-down list for selecting a country

An example of an enhanced feature would be the conversion of a value-oriented selection option (selection by entering a number) to a drop-down list, as shown in the diagram below:

Before	After
 <p>1. Belgium 2. France 3. Germany 4. Greece 5. Italy 6. Portugal 7. Spain 8. Switzerland 9. United Kingdom</p>	 <p>Belgium Belgium France Germany Greece Italy Portugal Spain Switzerland United Kingdom</p>

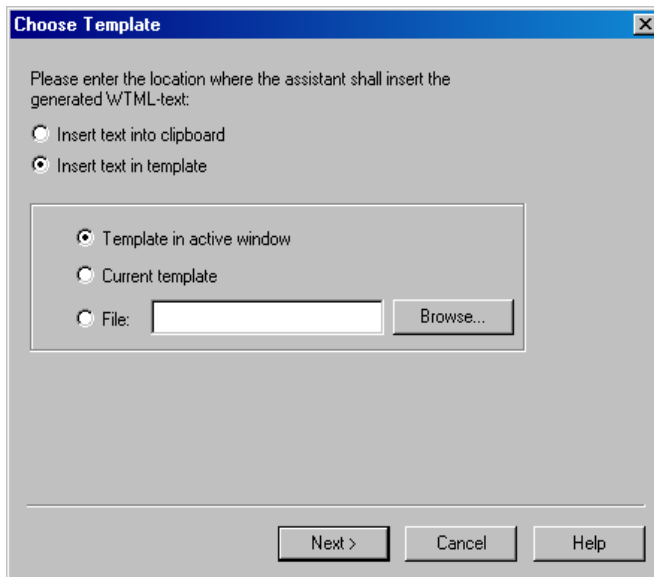
In the FHS format and in the generated template, the user selects the desired country by entering the appropriate number. However, since graphical interface users prefer to make selections via a drop-down list, we will now integrate such a list in our template.

- ▶ Choose the command **File/Open Current Template** to open the format-specific template *TRAVI* in the WebLab work area.
- ▶ Choose the command **Design/Select Hostobjects graphically/From a Communication Object**. The dialogbox **Select host objects graphically** for the current template is displayed on the screen.



This dialog box displays the format as it would appear in an emulation or at a terminal. All the output fields which cannot be edited have a yellow background. The input fields in this format have a white background.

- ▶ Move the mouse pointer to this input field (because of the selection the field becomes blue) and click on the right mouse button to open the context menu.
- ▶ Choose the **Drop-Down List** command from the context menu, see above. The **Choose Template** dialog box is displayed on the screen. This dialog box is the first displayed by a wizard which helps you create a list.



In this dialog box you specify the template in which the list is to be inserted. The option **Template in active window** is preset. If you have not previously opened the active template, choose the **Current template** option.

- ▶ Click on **Next >** to confirm this presetting. The second dialog box, **Assign Values**, is displayed on the screen.

Assign Values

Internal value
9

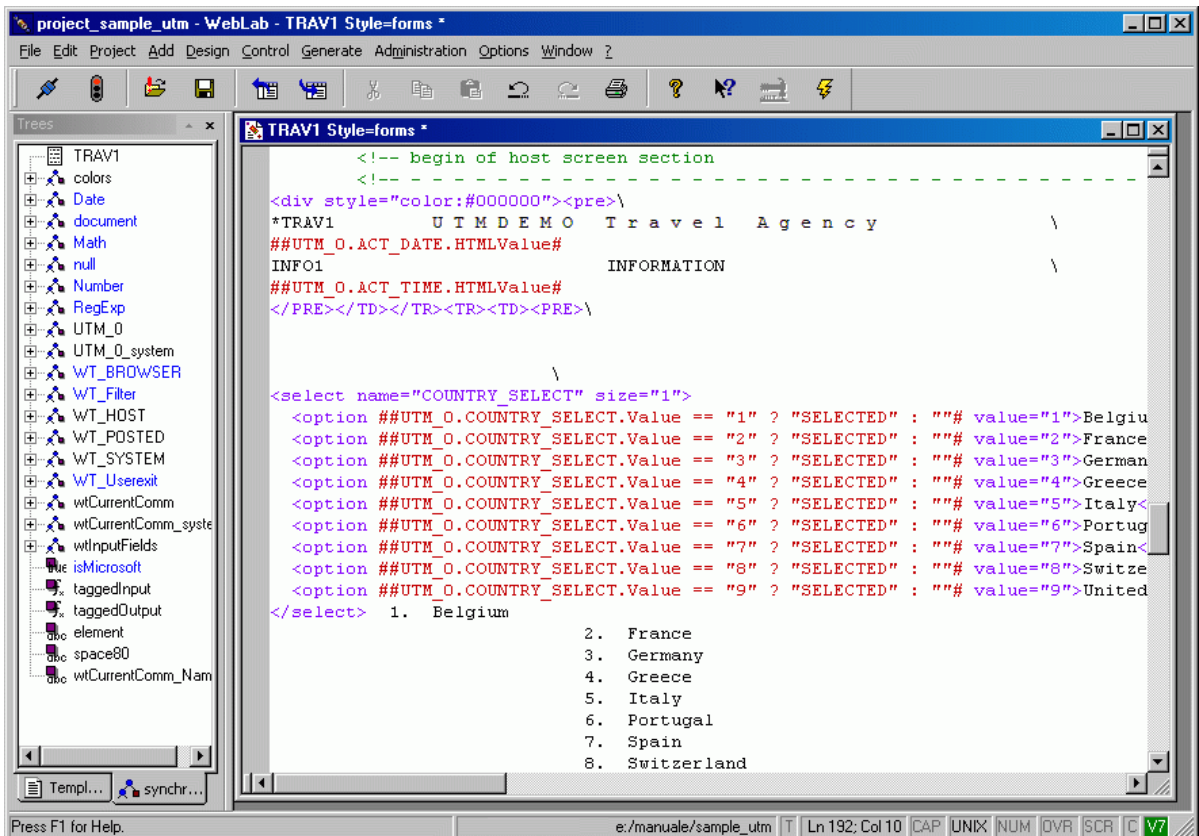
Value shown on user interface:
United Kingdom

Internal	Value on user interface
1	Belgium
2	France
3	Germany
4	Greece
5	Italy
6	Portugal

Do not show control, if external value is empty

In this example, the **Internal value** corresponds to the numerical value which the user must enter to select an item in the field. The **Value on user interface** is the description matching the internal value and corresponds to an entry in the pick list.

- ▶ Enter the internal values and the corresponding descriptions (see Figure on [page 53](#)) in the input fields. Click on the **Add** button to take over a pair of values into the list.
- ▶ When the list is complete, click on **Finish** to confirm. The corresponding HTML code for the conversion of a list is entered along with the corresponding values in the template *TRAV1.htm*.
- ▶ To view this replacement, scroll through the template *TRAV1* until you reach the host section. This section starts with the comment `begin of host screen section`.



WebLab has generated the following code for the input field:

```

<select name="COUNTRY_SELECT" size="1">
  <option ##UTM_0.COUNTRY_SELECT.Value == "1" ? "SELECTED" : ""#
value="1">Belgium</option>
  <option ##UTM_0.COUNTRY_SELECT.Value == "2" ? "SELECTED" : ""#
value="2">France</option>
  <option ##UTM_0.COUNTRY_SELECT.Value == "3" ? "SELECTED" : ""#
value="3">Germany</option>
  <option ##UTM_0.COUNTRY_SELECT.Value == "4" ? "SELECTED" : ""#
value="4">Greece</option>
  <option ##UTM_0.COUNTRY_SELECT.Value == "5" ? "SELECTED" : ""#
value="5">Italy</option>
  <option ##UTM_0.COUNTRY_SELECT.Value == "6" ? "SELECTED" : ""#
value="6">Portugal</option>
  <option ##UTM_0.COUNTRY_SELECT.Value == "7" ? "SELECTED" : ""#
value="7">Spain</option>

```

```

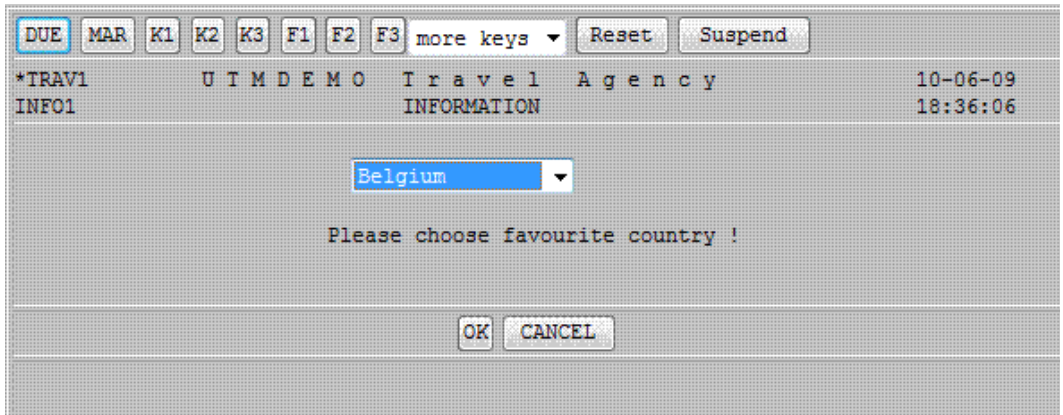
<option ##UTM_0.COUNTRY_SELECT.Value == "8" ? "SELECTED" : ""#
value="8">Switzerland</option>
<option ##UTM_0.COUNTRY_SELECT.Value == "9" ? "SELECTED" : ""#
value="9">United
  Kingdom</option>
</select>

```

WebLab does not simply generate a drop-down list which maps the interface values to the values which are used internally. It also automatically ensures that the value which is preset in the list is the value which is treated as default in the host application.

To this end, every OPTION tag contains an evaluation operator each of which in turn contains an IF query in abbreviated JavaScript notation: the condition is followed by a question mark. This is followed by a value which is the current value if the condition is fulfilled. This is followed by a colon and a value which applies if the condition is not fulfilled.

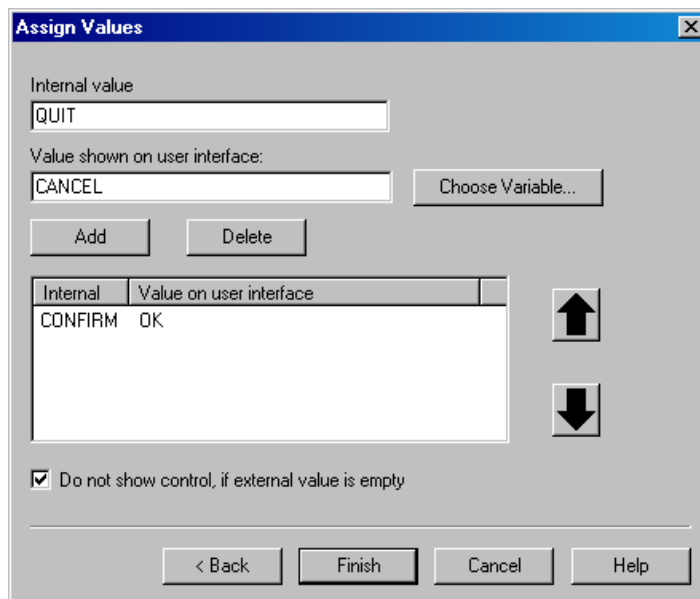
- ▶ Remove the now superfluous static country selection text.
- ▶ Choose the **Update in Browser** command in the WebLab **Control** menu to view the result.



3.3.2 Replacing command input with buttons

The next step involves replacing the text box for command input with appropriate buttons. The procedure here is the same as when generating a drop-down list:

- ▶ In the **Select host objects graphically** dialogbox, position the cursor on the command input field and click your right mouse button to open the context menu.
- ▶ Choose the **Push Buttons...** command. The **Choose Template** dialog box is displayed on the screen. This dialog box is the first displayed by a wizard which helps you create a list.
- ▶ Confirm the default settings in the **Choose Template** dialog box with **Next >**. The **Assign Values** dialog box then appears.
- ▶ Enter the values for the buttons. Under **Internal value**, enter the values to be sent to the host application. In the **Value shown on user interface** field, define the text displayed to the user at the interface.



- ▶ When the list is complete, click on **Finish** to confirm. The corresponding HTML code for the conversion of a list is entered along with the corresponding values in the template *TRAVI.htm*.

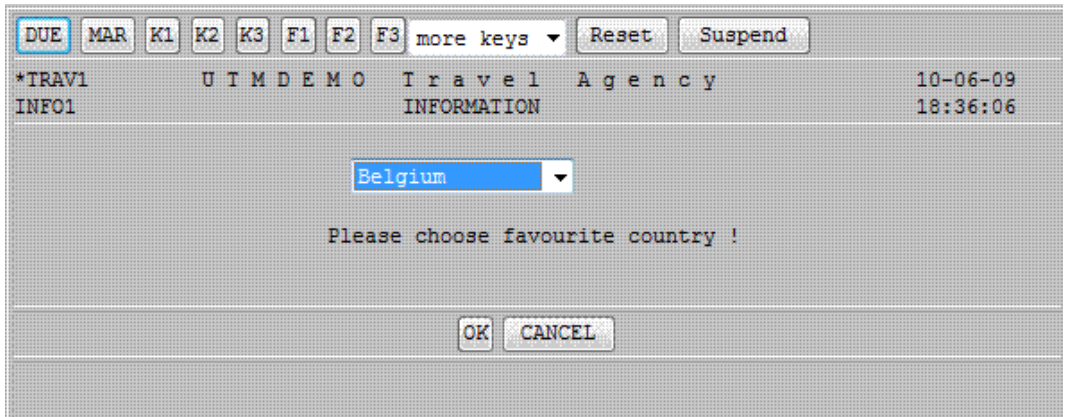
WebLab then generates the following HTML code:

```
Command ===&#62; \
<input type="submit" name="COMMAND" value="OK">

<input type="submit" name="COMMAND" value="CANCEL">
  ( CONFIRM / QUIT=main menue )
```

The corresponding OnReceive or OnReceiveScript tag is automatically modified such that the host data object with the value that corresponds to the pressed key is supplied.

- ▶ Remove the now superfluous static text `Command ===>` and `(CONFIRM / QUIT=main menue)` and align the buttons by adding spaces, for example.
- ▶ Remove any superfluous buttons from the include template `wtKeysUTM.htm` (in this example, it is assumed that the F1, F2, F3 keys are not generated in the openUTM application). `wtKeysUTM.htm` is located in the base directory under `config/forms`. You should remove the corresponding lines in the `wwdocs/javascript/wtKeysUTMFHS.js` file because the keys can also be triggered by the PC keyboard. In `wtKeysUTMFHS.js`, you also can modify the `more keys` selection list. For more details see [section “Mapping keys in wtKeysUTMFHS.js and wtKeysUTMFormant.js” on page 174](#).
- ▶ Choose the **Update in Browser** command of the WebLab **Control** menu to view the result:



3.3.3 Inserting a clickable image

The next enhancement step is to insert an image for country selection whose individual sections are sensitive to mouse activation and branch to various HTML pages ("clickable image"):



The template which performs this conversion is described below:

```
<html>
<head>
<title>Travel Agency - Country</title>
</head>
<wtinclude name="header">
<h2><font color="white">Click the country name you want to travel
to</font></h2>
```

The image (<input Name="EUROPE" type="image">) returns two name/value pairs, namely the x and y coordinates of the position clicked in the image. These are available as attributes of the posted object and are evaluated by a user exit in an OnReceive script (see the WebTransactions manual „Template Language“).

```
<input name="EUROPE" type="image" border="0"
src="##WT_SYSTEM.WWWDOCS_VIRTUAL#/image/europe.gif"><p>
<p>
<p>
<input type="submit" name="COMMAND" value="Cancel">
<p><p><p>
<wtonreceivescript>
<!--
    host = WT_HOST[WT_SYSTEM.HANDLE];
    // Generate an object of class Userexit, Look for
    // the required function in the library WTUserexit.[dll|so].
    // If the function is contained in a different library
    // the library must be specified in the following
    // constructor.
    UserExit = new WT_Userexit();
    // The result of the Userexit is buffered in a
    // local variable.
    country=UserExit.UXEurope(WT_Posted.EUROPE.x, WT_POSTED.EUROPE.y);
    // If the user has clicked outside any
    // sensitive area, the Userexit returns "0" and the
    // old value of the host object is retained.
    if (country != "0")
        host.COUNTRY_SELECT.Value = country;
    // The clicked button is now queried.
    if (WT_POSTED.COMMAND == "Cancel")
        host.COMMAND.Value = "QUIT";
    else
        host.COMMAND.Value = "CONFIRM";
//-->
</wtOnReceiveScript>
```

3.4 Starting WebTransactions

An edited WebTransactions application is started with WebLab as described in [section “Starting a session” on page 47](#). However, you can also use WebLab to create your own start template which brings the user directly to the first format of the host application.

3.4.1 Creating a start template

WebLab also provides you with a special WTBBean for the creation of application-specific start templates. This is a standalone WTBBean.



Before you can access WTBBeans, there must be a connection to a WebTransactions application.

- ▶ Choose the **File/New/wtcStartUPIC** command to call the WTBBean. This opens the dialog box **Add:wtcStartUPIC** which contains four tabs in which you can edit the properties of the WTBBean.

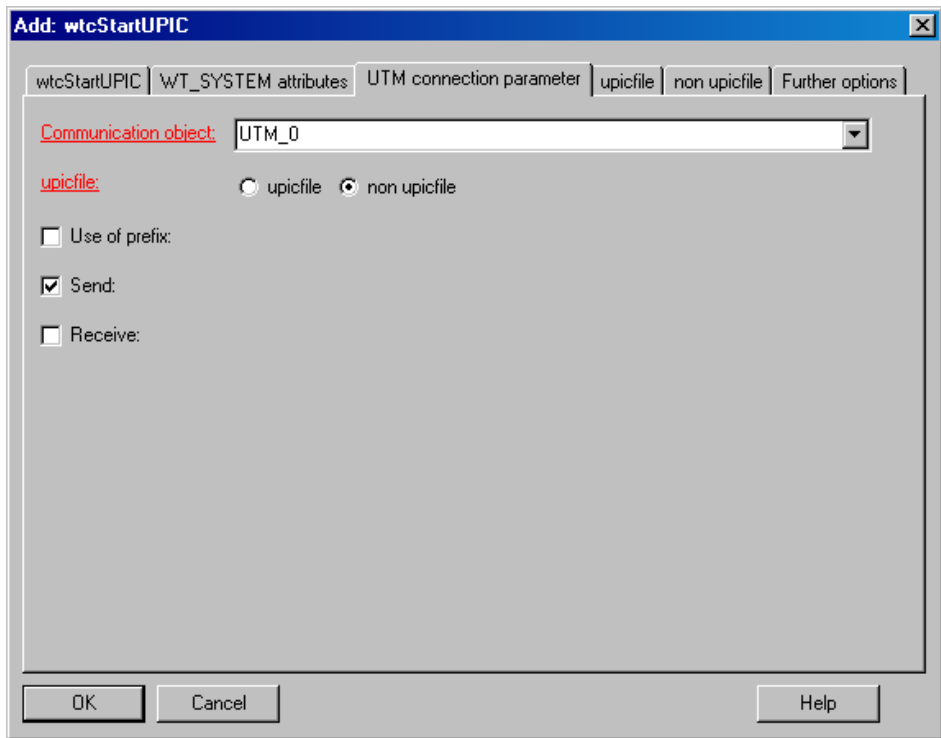
You define the name and directory of the start template in the **wtcStartUPIC** tab. By default, the file name is set to `config/forms/startUPIC.htm`.

- ▶ Under **File name**, enter the directory and name of the start template, in this case *Start_Travel.htm*.

- ▶ Next, choose the **WT_SYSTEM attributes** tab.

In this tab you define the most important attributes of the system object. The default values are sufficient for the example session.

- ▶ Choose the **UTM connection parameter** tab.



- ▶ Enter the name of the communication object, in this case *UTM_0*. You should note that the name of the communication object must be the same as you have defined in the `wtstart` template.
- ▶ In the example session activate the options **non upicfile** and **Send**.
- ▶ Choose the **non upicfile** tab.

wtcStartUPIC | WT_SYSTEM attributes | UTM connection parameter | upicfile | non upicfile | Further options

Host application: VTV10TRV [Select...]

Host name: HOST0001 [Select...]

Host ip address: [Select...]

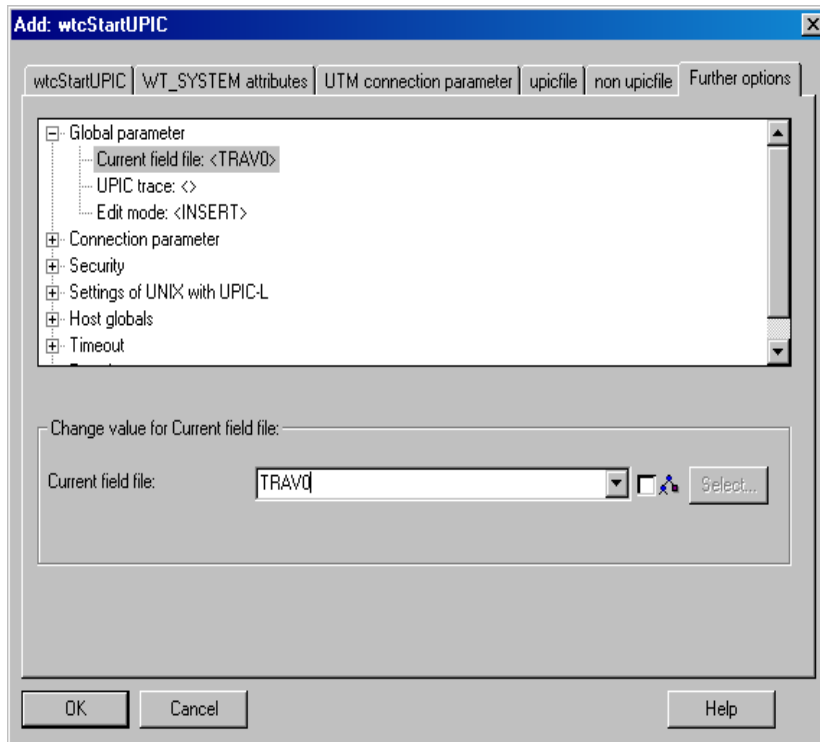
Host port: [Select...]

Start tac: MMENUE [Select...]

Code conversion:

OK Cancel Help

- ▶ In the **Host application** field, enter the name of the openUTM application, in this case *VTV10TRV*.
- ▶ In the **Host name** field, specify the name of the host on which the openUTM application is running, in this case *HOST0001*.
- ▶ For **Start tac** enter the name of the transaction code that starts the conversation in the openUTM application, in this case *MMENUE*.
- ▶ Check the option **Code conversion**.
- ▶ Click on the **Further options** tab.



Here you will see a tree structure in which you can edit all the properties relating to the connection to the openUTM application.

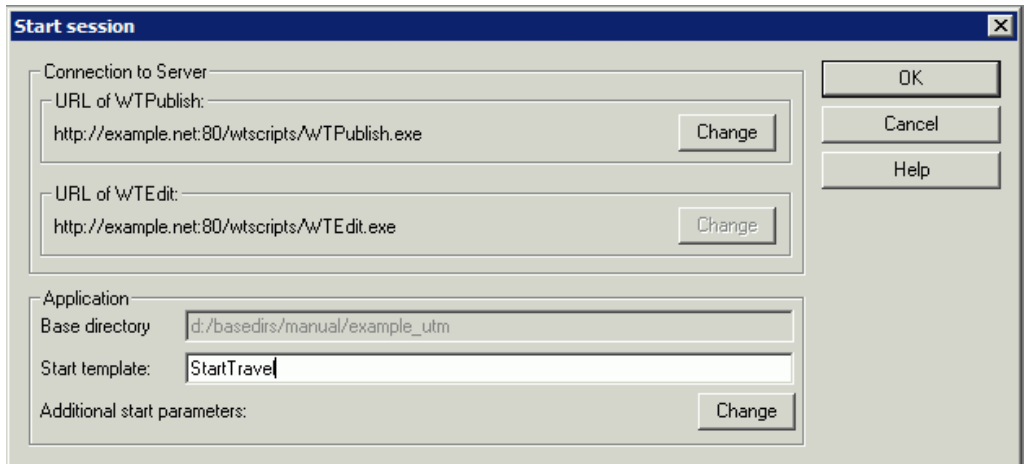
- ▶ In **Global parameter** choose *TRAVO* for **Current field file**.
No further modifications are required for the sample application.
- ▶ Click **OK**. The **Add:wtcStartUPIC** dialog box is closed. The start template *Start_Travel.htm* is generated and displayed in the WebLab work area. The start template is stored in the base directory under `config/forms`.

The start template *Start_Travel.htm* will now make the settings undertaken here every time it is called. You no longer have to make these settings every time you start a session as described in [section “Starting a session” on page 47](#) with `wtstart.htm` and `wtstartUTMV4.htm`.

3.4.2 Starting a session with WebLab

There are two ways of starting a WebTransactions session with the application-specific WebLab start template:

- You select the **File/Start Session** command. In this case the **Start session** dialog box will be displayed.



- ▶ In the **Start Template** input field, enter the name of the application-specific start template.
 - ▶ Confirm with **OK**.
- In the template tree you click with the right mouse button on the application-specific start template. In the context menu which appears, select the command **Start session**.

In both cases WebLab immediately starts the session with the template selected as the start template.

3.4.3 Alternative ways of starting a WebTransactions application

The above example only explains how to start a WebTransactions application from WebLab during development. In productive operation, however, there are other ways of doing this. For a complete description, see the WebTransactions manual “Concepts and Functions”.

- You can give the supplied entry page `wtadm.htm` the name of the start template and provide the user with this.
- You can write your own entry page in which WebTransactions is started via a form or link.

Example

```
<form method="post" action=
    "/cgi-prefix/WTPublish.exe/basedir?startTemplate">
    <input type="submit" value="Start WebTransactions">
</form>
```

- You could also start WebTransactions without an entry page by simply entering the URL directly:

```
http://WebServer/cgi-prefix/WTPublish.exe/basedir?startTemplate
```

For *basedir* you must specify the absolute path of the base directory.

Example

```
http://diana/scripts/WTPublish.exe/d:\webta\apps\
sample_utm?Start_Travel
```

4 Creating the base directory

Once you have installed WebTransactions, 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.

4.1 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** (see [section “Creating a project” on page 35](#)).
- 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.

- ▶ Enter the connection parameters in the **Connect** dialog box and click on **OK**.
- ▶ In the following dialog box, enter your user ID and password and click on **OK**.

- ▶ Enter the following in the **Create basedir** dialog box:
 - from the list of proposed pools, select the pool in which the base directory is to be created
 - enter the name of the new base directory
 - check the **UTM** box in the **Host Adapter** section
 - click on **OK**.
- ▶ Enter the required options in the **Generate Automask** dialog box. These correspond to the options for the generation of format-specific templates.
- ▶ Click on **Generate**. WebLab now creates the base directory together with all the files that are required for the execution of the WebTransactions application. The structure and contents of the base directory are described in the WebTransactions manual “Concepts and Functions” and in [section “Structure of a base directory” on page 73](#).

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.

4.2 Structure of a base directory

This section describes the specific aspects of the base directory that only apply to the openUTM host adapter.

i Information regarding the structure of base directories that applies to all host adapters can be found in the WebTransactions manual “Concepts and Functions”.

The configuration files `localapps` and `upicfile`

The connection to an openUTM application can be realized via configuration files or via system object attributes.

For connection via configuration files, the base directory contains the files `localapps` for the management of local applications and `upicfile` for the configuration of the UPIC connections. You must adapt these files for the requirements of your particular application (see [section “localapps file” on page 132](#) and [section “upicfile” on page 134](#)).

Preparing `upicfile` in the DMS user ID (BS2000/OSD)

UPIC functions running on the OSD system platform cannot read from the hierarchically structured POSIX file system (UFS). If you are using the `upicfile`, the `upicfile` must therefore be copied to the data management system (DMS) under the ID under which the `http` daemon has been started.

Use the POSIX command `bs2cp` to copy `upicfile` to the user ID under which you start the HTTP server. This may be `$TSOS` or another user ID `$UserId`. You must then also store `upicfile` as `$TSOS.UPICFILE` or `$UserId.UPICFILE`.

i The user ID under which you install WebTransactions need not be identical to the user ID under which you start the HTTP server.

Under OSD, you must terminate every line in the `upicfile` with a semi-colon (;). The file must not contain any comments. Furthermore, only entries of type `HD` are permitted.

The config subdirectory

For each converted format (screen), `config` contains a description file `formatname.fld` in which the format’s data fields are described. These files, also known as field files, are created automatically when the templates are generated.

A file with the fixed name `wt1nmode.fld` is provided for line mode. The name of this file must not be changed.

5 Generating templates

The WebLab development environment allows you to generate templates automatically from existing alphanumeric formats. FHS formats must be edited beforehand using the OSD tool IFG2FLD.

When generating templates with WebLab, a template and a field file is created for each format or data structure.

- Templates (*.htm files)

The templates contain HTML tags for structuring the Web page and WT tags for communications control. One template is generated for each format and stored by default in the directory *basedir/config/forms*. The template echoes the appearance and functionality of the terminal format. You can adapt the design of a template to meet your own needs, e.g. with WebLab (see [section “Editing templates” on page 53](#)).

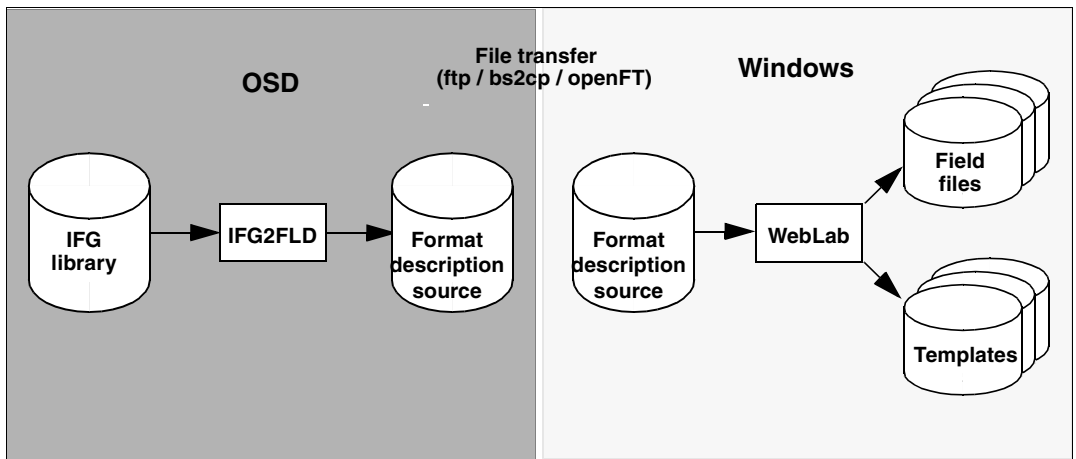
- Field files (*.fld files)

The field files describe the data fields in the formats (attributes, lengths, offsets). One field file is generated for each format and stored in the directory *basedir/config*. The host adapter requires the metadata which these files contain in order to interpret the exchanged data. During communications between WebTransactions and the host application, only the net data is transferred over the network. You should only edit field files when you need to adapt to modifications in the host application. However, it is often easier and safer to generate new field files after modifying the host application.

5.1 Generating templates from FHS formats

The procedure for generating templates from FHS formats consists of two steps:

1. Firstly, you must generate a format description file from the IFG library using the IFG2FLD program under OSD and then transfer this file to the development system. The format description source is the result of an IFG2FLD run in which the format descriptions are read from the IFG library into the source. It contains all field names defined originally by the developer of the host formats using the formatting system.
2. You can then generate templates and field files from the format description source using WebLab under Windows.



Procedure

To generate templates and FLD files from FHS formats, proceed as follows:

- ▶ For the Unix and Windows WebTransactions platforms, transfer one of the following LMS libraries in binary format to the OSD host to the user ID under which the IFG library is located and name this library on the OSD host `WTIFG2FLD.LMS`:

Library	Meaning
<code>WTifg2f1dFTP.lms</code>	For transfer with FTP
<code>WTifg2f1dopenFT.lms</code>	For transfer with openFT

This step is not necessary for the OSD platform, as IFG2FLD is located in the OSD installation directory.

- ▶ Log on at the OSD host under this ID.

- ▶ Run IFG2FLD as described in [section “Using IFG2FLD” on page 77](#).
- ▶ Transfer the generated format description source in text format to the system on which WebLab is installed.
- ▶ When generating the template, specify this format description source in the **FLD and Template generation** dialog box (see [section “Using WebLab to generate templates and FLD files from the format description source” on page 79](#)).

5.1.1 Using IFG2FLD

IFG2FLD is an OSD program and therefore requires the corresponding libraries to be transferred to an OSD user ID. IFG2FLD can then be started from the transferred library using the following command:

```
/START-PROGRAM FROM-FILE=*PHASE(LIBRARY=WTIFG2FLD.LMS,ELEMENT=IFG2FLD)
```

IFG2FLD is called using the START-PROGRAM command and output is written to SYSLST. The following commands are supported by IFG2FLD:

Command	Description
MAPFILE <i>IFG-library</i>	Assigns the IFG library which is to be processed.
PROFILE <i>user-profile</i>	Assigns a user profile for the conversion. Each IFG library contains at least one user profile in order to enable it to process formats. The user profile is a set of default values for controlling IFG, for the format properties, for the properties of the format's fields and for properties which affect programming.
CONVERT <i>format-name</i> [/ <i>version</i>] /ALL]	Takes the specified IFG format over into the format description source. Optionally, you can also specify the version of the format. If you do not specify the version then the most recent format version is entered. If, instead of a version, you specify the value /ALL then all the versions of the format are entered in the format description source.
CONVERT *ALL [/ <i>version</i>] /ALL]	Takes all the specified IFG formats in the IFG library over into the format description source. Optionally, you can also specify the version of the format. Only formats with this version are then entered. If you do not specify the version then the formats with the most recent version are entered in the format description source. If, instead of a version, you specify the value /ALL then all the formats of all versions are entered.
END	Generates an output file and exits IFG2FLD.

Example

To perform a conversion using IFG2FLD, the following steps might be necessary in OSD:

```
/ASSIGN-SYSLST output-file  
/START-PROGRAM FROM-FILE=*PHASE(LIBRARY=WTIFG2FLD.LMS,ELEMENT=IFG2FLD)  
MAPFILE IFG-library  
CONVERT *ALL  
END  
/ASSIGN-SYSLST *P
```

5.1.2 Using WebLab to generate templates and FLD files from the format description source

- ▶ Use the command **File/Connect** to establish a connection to the base directory if required.
- ▶ Select **Generate/Templates/from IFG library**. This opens the **Options for FLD generation** dialog box.

Options for FLD generation

Input

Format Description Source:

Formats :

<input checked="" type="checkbox"/> TRAV0	<input checked="" type="checkbox"/> TRAV6
<input checked="" type="checkbox"/> TRAV1	<input checked="" type="checkbox"/> TRAV7
<input checked="" type="checkbox"/> TRAV2	<input checked="" type="checkbox"/> TRAV8
<input checked="" type="checkbox"/> TRAV3	<input checked="" type="checkbox"/> TRSGNDF
<input checked="" type="checkbox"/> TRAV4	<input checked="" type="checkbox"/> TRSGNDN
<input checked="" type="checkbox"/> TRAV5	

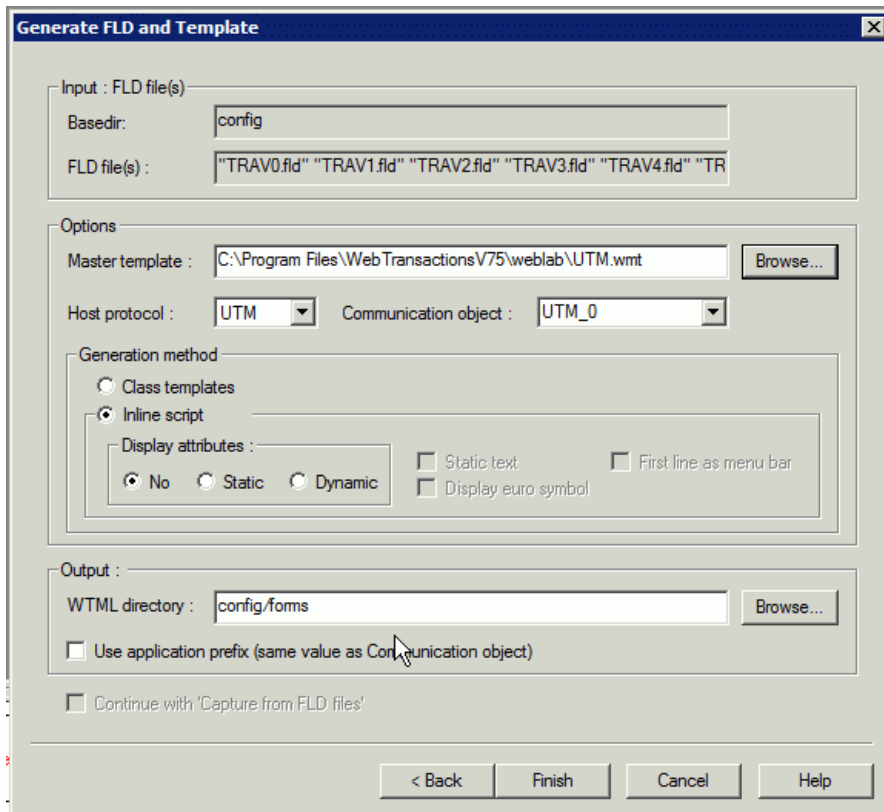
Output

FLD directory:

for partial formats only!

Processing for partial formats

- ▶ In the **Input** group, enter the name of the format description source. The **Formats** list then displays the names of all formats described there.
- ▶ In the list, click on the formats for which templates and FLD files are to be generated.
- ▶ In the **Output** group, select the directory to which the generated FLD files are to be written. If a session is started with this WebTransactions application, the path name is automatically displayed.
- ▶ Confirm by clicking **Next >**. This opens the **Generate FLD and Template** dialog box.



You can use the options in this dialog box to control the generation of format-specific templates. A detailed description of the generation options can be found in the online help system.

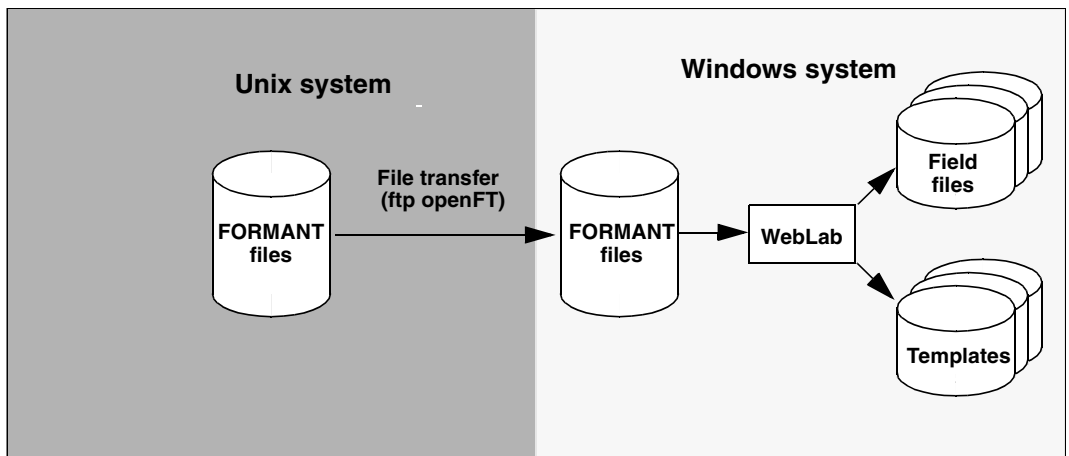
- ▶ In the **Master Template** field, enter the master template you wish to use. This is different for full formats and partial formats:
 - For full formats, you must use the master template `UTM.wmt` (see also [section “Master template UTM.wmt” on page 100](#)).
 - For partial formats, you must use the master template `UTMpartial.wmt` (see [section “Structure of the master template UTMpartial.wmt” on page 106](#)).
- ▶ In the **Output** group, enter the directory to which the generated templates are to be written.
 Default path for templates (HTML output): `basedir/config/forms`

- ▶ If you wish to integrate several host applications which may have the same format names, select the **Use Application Prefix** option. WebLab then inserts a prefix in front of the names of FLD files and templates. The file name is then composed as follows:
`[comobj@]formatname.{fld|htm}`.
- ▶ Confirm the generation options with **Finish**.
- ▶ Use **OK** to close the **Generate FLD and Template** dialog box. WebLab then generates an HTML template and an FLD file for each of the selected formats:
 - HTML templates are interpreted by WebTransactions at runtime and define the interface displayed in the browser.
 - FLD files are special description files. They are required by WebTransactions at runtime, and are used by the WebTransactions development environment WebLab to implement the graphical object selection function. To ensure that they can be accessed by WebTransactions during operation, they must be stored under `basedir\config`.

5.2 Generating templates from FORMANT formats

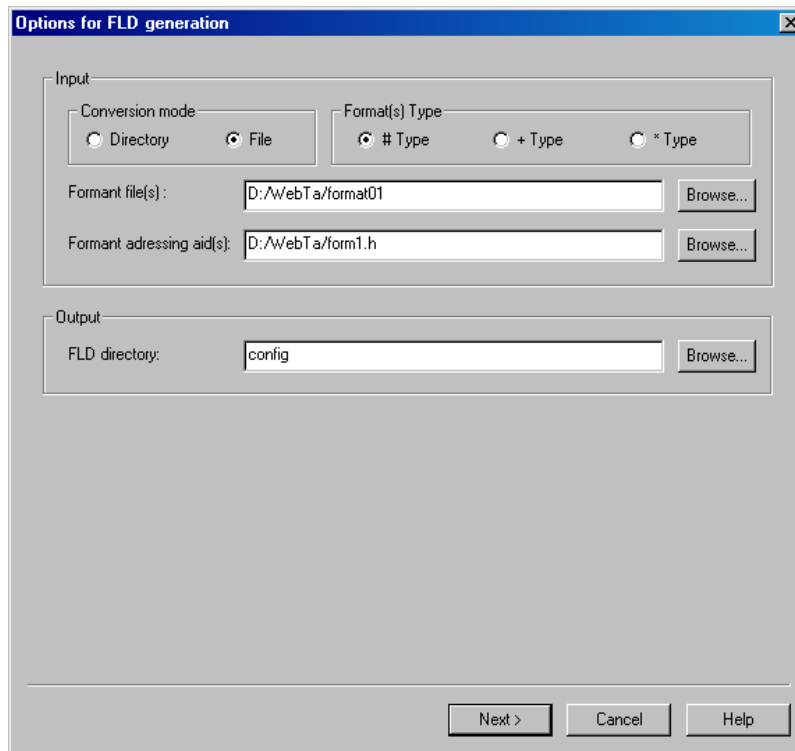
The procedure for generating templates from FORMANT formats consists of three steps:

1. Create the converter format `<format_name>.txt` using `formantconv`.
2. Transfer the FORMANT files (converter format description and addressing aid in the appropriate format (*, +, #) for C `<formatname>a.h`) to the development system. If necessary, for example, when an addressing aid is only available for COBOL, create this addressing aid by saving it from `formantgen`.
3. You can then generate templates and field files from these FORMANT files using WebLab under Windows.



5.2.1 Generating templates and FLD files using WebLab

- ▶ If necessary, use the **File/Connect** to establish a connection to the base directory.
- ▶ Select **Generate/Templates/from FORMANT file (for openUTM)**. This opens the **Option for FLD generation** dialog box.



Conversion mode

Each FORMANT format is defined by a format description file and an addressing aid. WebLab can generate templates and FLD files for both individual formats and groups of formats. In the latter case, the associated description files must be combined into directories.

Format(s) Type

This defines the type of format.

Formant file(s)

Formant addressing aid(s)

Depending on the conversion method chosen, enter a file or directory from which WebLab is to generate the FLD files and templates.

- ▶ In the **FLD directory** field, you must specify the directory `config`. The newly generated FLD files will then be stored in `/config`.
- ▶ Confirm using **Next >**. This opens the **Generate FLD and Template** dialog box (see [page 82](#)).

5.3 Structure of the field files (FLD files)

WebLab saves the structure of an FHS/FORMANT screen as a data record in the field file. This represents the metadata concerning the structure of the data which is exchanged between the host application and WebTransactions. This static metadata is not therefore sent with the message but is stored in a file. It is read the first time WebTransactions processes a format. Your information is then available to the host adapter for the interpretation of the net data (host data objects) from the host application.

The format of the field file corresponds to that of a *.ini file under Windows:

- The file is subdivided into various sections which describe the individual parts of the format (see table on [page 85](#)).
- Sections can be totally omitted if none of the keywords assigned to them are present in the field file.
- Boolean attributes have the value 0 or 1, all other attributes can only assume the values which you have specified in your description.
- Unless specified to the contrary, all numerical values are integer values and > 0 .
- Comments must start in column 1 with a semi-colon (;) and must not exceed 1 line (256 characters) in length.
Blank lines are permitted.

The following sections are supported:

Section name	Contents
FormProperties	The version and format type as well as the global attributes of the format
AttributOffsets	The relative offsets of the underlying attributes in the field attribute block
<Fieldname>	<p>The individual fields are described in sequence here. The description includes data such as name, length and offset. Field names may consist only of letters, digits, underscores (_) and dollar symbols(\$) and must not start with a digit. They are derived from the original names with the following modifications:</p> <ul style="list-style-type: none"> – a hyphen ('-') becomes an underscore – an initial digit is preceded by a dollar symbol ('\$') – all illegal characters are replaced by a dollar symbol – as of the second occurrence of an ambiguous name, a suffix consisting of a dollar symbol and serial number is added
FieldMatching	Establishes the link between IFG or FORMANT field names and generic field names (E_yy_xxx_III)

The following keywords are supported:

The table contains all the keywords. The actual keywords present in any concrete case may vary depending on the type of format (*,+,#) and the IOTYPE of the field.

Keyword	Section	Description
AttributeLength	FormProperties	Length of the global attribute block in # formats
Background Colour	FormProperties	Background color of format: WHITE (normal brightness) or GRAY (half brightness)
BaseVariant	FormProperties	Base variant of the character set (only for 8-bit formats)
CursorControl	FormProperties	Offset of global attribute CURSOR CONTROL
CursorPosition	FormProperties	Offset of global attribute CURSOR POSITION
DateFormat	FormProperties	Display format for fields with DataType=DATE
DecimalSeparator	FormProperties	Decimal separator for numerical fields (DataType=NUMBER)
DigitSeparator	FormProperties	Digit separator for numerical fields (DataType=NUMBER)
FieldsDetect	FormProperties	Offset of global attribute FIELDS DETECTION
FieldsMod	FormProperties	Offset of global attribute FIELDS_MOD

Keyword	Section	Description
FieldsValid	FormProperties	Offset of global attribute FIELDS VALIDATION
FormatLength	FormProperties	Length of format contents
FormatName	FormProperties	Format name
FormattingSystem	FormProperties	Formatting system: FHS, FORMANT
FormatType	FormProperties	Type of format: #, + (no alignment), *
InputKeyClass	FormProperties	Offset of global attribute INPUT KEY CLASS
InputKeyNumber	FormProperties	Offset of global attribute INPUT KEY NUMBER
PopUp	FormProperties	Specifies whether or not the object is a pop-up box. Possible values: 0, 1
Protocol	FormProperties	Type of application protocol, always set to UTM
ScreenDimensions	FormProperties	Screen dimensions in the format <i>linesxcolumns</i> , e.g. 24x080
TimeFormat	FormProperties	Display format for fields with DataType=TIME
UndefinedValues	FormProperties	Offset of global attribute UNDEFINED VALUES
UserexitRc	FormProperties	Offset of global attribute USER EXITROUTINE RC
Version	FormProperties	Version of WebTransactions
AttributeCombination	AttributOffsets	Offset within an attribute block as of which the AttributeCombination attribute starts
AttributeLength	AttributOffsets	Length of the field attribute block in # formats
Color	AttributOffsets	Offset within an attribute block as of which the Color attribute starts
Cursor	AttributOffsets	Offset within an attribute block as of which the Cursor attribute starts
EditRC	AttributOffsets	Offset within an attribute block as of which the EditRC attribute starts
EditState	AttributOffsets	Offset within an attribute block as of which the EditState attribute starts
FieldLength	AttributOffsets	Offset within an attribute block as of which the FieldLength attribute starts
InputControl	AttributOffsets	Offset within an attribute block as of which the FieldInput attribute starts (mandatory input)
InputState	AttributOffsets	Offset within an attribute block as of which the InputState attribute starts (field value type since last output of format)
InputStateAct	AttributOffsets	Offset within an attribute block as of which the InputStateAct attribute starts (identifier of the currently entered field)
Intensity	AttributOffsets	Offset within an attribute block as of which the Intensity attribute starts (field display mode, e.g. bold or normal)
Inverse	AttributOffsets	Offset within an attribute block as of which the Inverse attribute starts (field displayed in inverse video)
NumAttributes	AttributOffsets	Number of field attributes in # formats

Keyword	Section	Description
OutputControl	AttributOffsets	Offset within an attribute block as of which the OutputControl attribute starts
Protection	AttributOffsets	Offset within an attribute block as of which the Protection attribute starts
Underline	AttributOffsets	Offset within an attribute block as of which the Underline attribute starts (field underscored)
Visibility	AttributOffsets	Offset within an attribute block as of which the Visibility attribute starts (field visible)
Align	<Fieldname>	Field alignment: LEFT, RIGHT
AttributOffset	<Fieldname>	Offset to start of field attributes. It is then possible to navigate through the attributes using the relative offsets specified above.
AutolInput	<Fieldname>	Automatic field input. Possible values: Y,N.
Blink	<Fieldname>	Field displayed blinking: 1, 0
Case	<Fieldname>	Letter conversion: UPPER, LOWER, BOTH; (in FHS UPPERCASE)
Color	<Fieldname>	Field foreground color. Possible values: 1 (red), 2 (green), 3 (yellow), 4 (blue), 5 (magenta), 6 (cyan), default (black).
Column	<Fieldname>	Column number
DataOffset	<Fieldname>	Offset to start of field data.
DataType	<Fieldname>	Data type of field: NUMERIC, ALPHANUMERIC, ALPHA, DATE, TIME.
DefaultCursor	<Fieldname>	Specifies whether the cursor is to be positioned in this field if not positioned elsewhere by the program. Possible values: 1, 0.
Detectable	<Fieldname>	Specifies whether the field can be selected. Possible values: 1, 0.
DisplayLength	<Fieldname>	Field length on the screen
FillCharInput	<Fieldname>	Field fill character for data input to openUTM program unit. Possible values as in IFG report; Default: ' '.
FillCharOutput	<Fieldname>	Field fill character for data output from openUTM program unit. Possible values as in IFG report; Default: ' '.
GroupDigit	<Fieldname>	Specifies whether digit grouping is permitted (for fields with DataType=NUMERIC). Possible values: Y/N.
Intensity	<Fieldname>	Intensity of field: BRIGHT, NORMAL
Invers	<Fieldname>	Specifies whether the field is to be displayed in reverse video. Possible values: 1, 0.

Keyword	Section	Description
IOType	<Fieldname>	Type of field: INPUT, OUTPUT, TEXT, FIXTEXT. Fields of type FIXTEXT cannot be accessed via the program, rather only via the FLD file. Their values are output in the Text attribute (see below).
Length	<Fieldname>	Length of the field in the addressing aid. This value must be > 0. The field length in the addressing aid may differ from that on the screen (DisplayLength), e.g. as in the case of date fields (DataType=DATE) or numeric fields (DataType=NUMERIC).
Line	<Fieldname>	Line number
Mandatory	<Fieldname>	Specifies whether a field is mandatory. Possible values: 1, 0.
NumDecimal	<Fieldname>	Number of characters after the decimal separator (for fields with DataType=NUMERIC)
Protection	<Fieldname>	Field write protection: 1, 0
Sign	<Fieldname>	Specifies whether a sign is permitted (for fields with DataType=NUMERIC). Possible values: Y/N.
SignFloat	<Fieldname>	Specifies whether the sign may be located to the right of the number (for fields with DataType=NUMERIC). Possible values: Y/N.
SuppressZero	<Fieldname>	Specifies whether zero suppression is permitted (for fields with DataType=NUMERIC). Possible values: Y/N.
Text	<Fieldname>	Field contents for IOType=FIXTEXT enclosed in single quotes ('text')
Underline	<Fieldname>	Field displayed underscored: 1, 0
UTMControl	<Fieldname>	Specifies whether the field is a openUTM control field: 1, 0
Visibility	<Fieldname>	Specifies whether the data field is visible. Possible values: 1, 0.
<IFG-name> or <FORMANT-name>	FieldMatching	Describes the mapping between IFG or FORMANT field names and generic names in the format E_yy_xxx_III, e.g. BERUF=E10_005_36

5.4 Structure of the generated templates

This section describes templates generated from full FHS or FORMANT formats. For information on the templates generated from partial FHS or FORMANT formats, refer to section [section “Structure of the master template UTMpartial.wmt” on page 106](#).

Each generated template is based on the master template you specify on generation. The standard master template UTM.wmt supplied with the product is described in [section “Master template UTM.wmt” on page 100](#).

This section describes the structure of a generated template using an example.

The comments are the expansion of %%GenerationInfo%.

```
<html>

<wtrem>*****</wtrem>
<wtrem>**   WTML document: TRAV3                               **</wtrem>
<wtrem>*****</wtrem>
<wtrem>**
<wtrem>** Document generation based on Master Template :     **</wtrem>
<wtrem>**   C:\Programme\WebTransactionsV75\web\lab\UTM.wmt    **</wtrem>
<wtrem>**
<wtrem>** Generated at Wed Jun 09 13:54:57 2010              **</wtrem>
<wtrem>**
<wtrem>** Options used by the generator :                       **</wtrem>
<wtrem>**   - %OPTIONS:                                           **</wtrem>
<wtrem>**     CommObj = UTM_0                                       **</wtrem>
<wtrem>**     NationalVariant = International - PartialFormatMode = No **</wtrem>
<wtrem>**   - %LINES:                                             **</wtrem>
<wtrem>**     TaggedInput = Enabled - TaggedOutput = Enabled      **</wtrem>
<wtrem>**     DisplayAttributes = No - CursorInProtectedField = No **</wtrem>
<wtrem>**     Generate = Inline                                       **</wtrem>
<wtrem>**     CellsDelimiter = "-"                                   **</wtrem>
<wtrem>**     CellsDelimiterReplace = </PRE></TD></TR><TR><TD><PRE>\ **</wtrem>
<wtrem>**   - %RECEIVES:                                           **</wtrem>
<wtrem>**     Parameters not specified                               **</wtrem>
<wtrem>*****</wtrem>
<wtrem>** WebTransactions V7.5                               Fujitsu Technology Solutions 2010 **</wtrem>
<wtrem>*****</wtrem>
```

References to the communication object and the system object attribute that belongs to it are created in order to ensure uniform access to the connection parameters and host objects. UTM_0 is the name selected in the **Communication Object** field at generation.

```

<wtoncreatescript>
<!--
  {{{WebLab(assignCommunicationObject)
  UTM_0 = WT_HOST.active || WT_HOST.UTM_0;
  if (UTM_0.WT_SYSTEM != null)
    UTM_0_system = UTM_0.WT_SYSTEM;    // communication specific system object
  else
    UTM_0_system = WT_SYSTEM;          // global system object
  }}}
  // propagate communication object to included WTML documents ////////////////
  wtCurrentComm = UTM_0;
  wtCurrentComm_system = UTM_0_system;
  //-->
</wtoncreatescript>

```

The `<head>` tag is used to define the header for the HTML page. This contains the title as well as a number of global values.

```

<head>
<title>WebTransactions V7.5 - application ##UTM_0_system.SYM_DEST#</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>

```

In the BODY section, a form with the current format (<form>) and a table are opened. This table frames the entire format.

To use the dialog, <wtInclude> calls the templates wtKeysUTM.htm and wtBrowserFunctions.htm in order to support as precise a 1:1 display as possible. Their controls therefore become part of the form.

```

<body bgcolor="#C0C0C0">
<form WebTransactions name="TRAV3">
  <table frame="border" rules="all">
    <tr>
      <td>
        <wtinclude name="wtBrowserFunctions">
        <wtinclude name="wtKeysUTM">
        <wtif (UTM_0_system.FORMTPL)>
          <wtinclude Name="##UTM_0_system.FORMTPL#">
          </wtif>
        </td>
      </tr>
      <tr>
        <td>

```

This wtOnCreate script defines the display attributes for the host objects. The taggedOutput() function is used to process the output fields and the taggedInput() function is used to process the input fields.

```

<wtoncreatescript>
  <!--
    colors = new Array('RED','GREEN','YELLOW','BLUE','MAGENTA','CYAN',
'WHITE');
    space80 = "
";

    function taggedInput( hostObject )
    {
      if ( hostObject.Protected == 'Y' )

```

```

        {
            taggedOutput( hostObject );
            return;
        }
        currentLength = hostObject.Length;
        input = '<input type='+ (hostObject.Visible == 'N' ? "password"
"text" );
        if ( WT_BROWSER && (WT_BROWSER.is_ie || WT_BROWSER.is_ns61up) )
        {
            input += ' class="box" style="width:' + (currentLength *
                WT_BROWSER.charWidth + 1) + 'px';
            input += (hostObject.Blinking == 'Y' ? '; background-
color:#FFC0C0' : '');
            input += (hostObject.Underline == 'Y' ? ( hostObject.Intensity
== 'N' ?
                '; color:#A0A0FF' : '; color:#0000A0' ) :
                ( hostObject.Intensity == 'N' ? '; color:#A0A0A0' : ''
)) + ''';
        }
        input += ' name="' + hostObject.Name + '" size="' + currentLength
            + ' maxlength="' + currentLength
            + ' value="' + hostObject.Value
            + (hostObject.DataType == 'Numeric'? "
numeric="1": '')
            + (hostObject.Detectable == 'Y'? "
markable="1": '')
            + '>';
        document.write( input );
    }

function taggedOutput( hostObject )
{
    if ( hostObject.Protected == 'N' )
    {
        taggedInput( hostObject );
        return;
    }
    if ( hostObject.Visible == 'Y' )
    {
        output = hostObject.HTMLValue;
        if ( hostObject.Inverse == 'Y' )
        {
            if ( hostObject.Color && hostObject.Color.toUpperCase() !=
'N')
                output = '<font color=#000000 style=\"background-color:' +
                    colors[hostObject.Color-1] +
'\">>' + output + '</font>';
        }
    }
}

```

```

else if ( hostObject.Color && hostObject.Color.toUpperCase() !=
'N')
    output = '<font color=' + colors[hostObject.Color-1] + '>' +
        output + '</font>';
    if (hostObject.Intensity == 'H')
        output = '<b>' + output + '</b>';
    if (hostObject.Blinking == 'Y')
        output = '<i>' + output + '</i>';
    if (hostObject.Underlined == 'Y')
        output = '<u>' + output + '</u>';
    document.write( output );
}
else
{
    document.write( space80.substr(0,hostObject.Length));
}
}
//-->
</wtoncreatescript>

```

In this section, the contents of the format are mapped to HTML tags and the table is closed.

```

<!-- -----
----- -->
<!-- begin of host screen section
-->
<!-- -----
----- -->
<div style="color:#000000"><pre>\
*TRAV3      U T M D E M O   T r a v e l   A g e n c y           \
##UTM_0.ACT_DATE.HTMLValue#
INFO3              INFORMATION                               \
##UTM_0.ACT_TIME.HTMLValue#
</PRE></TD></TR><TR><TD><PRE>\

        Golf course \
##UTM_0.GOLF_COURSE.HTMLValue# in \
##UTM_0.GOLF_COUNTRY.HTMLValue#
        From \
##UTM_0.FIRST_DAY_YEAR.HTMLValue#-\
##UTM_0.FIRST_DAY_MONTH.HTMLValue#-\
##UTM_0.FIRST_DAY_DAY.HTMLValue# to \
##UTM_0.LAST_DAY_YEAR.HTMLValue#-\
##UTM_0.LAST_DAY_MONTH.HTMLValue#-\
##UTM_0.LAST_DAY_DAY.HTMLValue# for \
##UTM_0.PERSONS_NBR.HTMLValue# persons

```

```

                                Hotels                                Category EURO/day

                                \
<input type="##(UTM_0.HOTEL_SELECT.Visible == 'N') ? 'password' : 'text'#"
##( WT_BROWSER.acceptClass ) ? 'class="box" style="width:9px"' : '#'
name="HOTEL_SELECT" size="1" maxlength="1"
value="##UTM_0.HOTEL_SELECT.Value#"/> 1. \
##UTM_0.HOTEL_NAME$000.HTMLValue# \
##UTM_0.HOTEL_CATEGORY$000.HTMLValue# \
##UTM_0.HOTEL_PRICE$000.HTMLValue#
                                \
##UTM_0.HOTEL_ADDRESS$000.HTMLValue#

                                2. \
##UTM_0.HOTEL_NAME$001.HTMLValue# \
##UTM_0.HOTEL_CATEGORY$001.HTMLValue# \
##UTM_0.HOTEL_PRICE$001.HTMLValue#
                                \
##UTM_0.HOTEL_ADDRESS$001.HTMLValue#

                                3. \
##UTM_0.HOTEL_NAME$002.HTMLValue# \
##UTM_0.HOTEL_CATEGORY$002.HTMLValue# \
##UTM_0.HOTEL_PRICE$002.HTMLValue#
                                \
##UTM_0.HOTEL_ADDRESS$002.HTMLValue#

</PRE></TD></TR><TR><TD><PRE>\
    Command ==&#62; \
<input type="##(UTM_0.COMMAND.Visible == 'N') ? 'password' : 'text'#" ##(
WT_BROWSER.acceptClass ) ? 'class="box" style="width:65px"' : '#'
name="COMMAND" size="8" maxlength="8" value="##UTM_0.COMMAND.Value#"/> (
CONFIRM / QUIT=main menue )
</PRE></TD></TR><TR><TD><PRE>\
##UTM_0.WARNING_AREA.HTMLValue#
##UTM_0.INFO_AREA.HTMLValue#
<wtoncreatescript>
<!--
    wtInputFields = {HOTEL_SELECT:UTM_0.HOTEL_SELECT,COMMAND:UTM_0.COMMAND};
//-->
</wtoncreatescript></pre></div>
                                <!-- - - - - -
                                -->

```

```

        <!-- end of host screen section
-->
        <!-- -----
----- -->
        </td>
</tr>
</table>

```

A loop is used across the input fields for all the browsers which ignore the `maxLength` attribute in the `<input>` tag. The loops add the client-side attribute `maxLength` to the appropriate script object as this is not available automatically. For all browsers which do not accept attributes in the `<input>` tag, the attribute `markable` is set, provided that the appropriate host object is `markable`.

```

<script type="text/javascript">
  <!--
  <wtoncreatescript>
  <!--
    for( element in wtInputFields )
    {
      if (!WT_BROWSER.acceptMaxLength)
        document.writeln('wtSetMaxLength('\', element, '\',', wtInputFields[
element ].Length, ' ');');
      if (!WT_BROWSER.acceptInputAttributes)
      {
        //if (wtInputFields[ element ].DataType == 'NUMERIC')
        // document.writeln('wtSetInputAttribute("numeric", "', element,
');');
        if (wtInputFields[ element ].Detectable == 'Y')
          document.writeln('wtSetInputAttribute("markable", "', element,
');');
      }
    }
  //-->
</wtoncreatescript>
//-->
</script>
</form>

```

The focus in the Web browser window is then set to the field in whose corresponding screen field the cursor was positioned (provided that the field is an input field).

```
<wtrem** initial focus selection *****>
<script type="text/javascript">
<!--
<wtif( UTM_0[UTM_0.WT_HOST_MESSAGE.CursorField] &&
      UTM_0[UTM_0.WT_HOST_MESSAGE.CursorField].IOType == 'INPUT') >
  wtSetFocus('##UTM_0.WT_HOST_MESSAGE.CursorField#');
<wtelse>
  wtBuildFieldList();
</wtif>
//-->
</script>
```

In a wtOnReceiveScript, the posted values for the individual fields are written back to the corresponding host objects.

```
<wtrem** Script executed after post of HTML page *****>
<wtonreceivescript>
<!--
  //{{WebLab(processPostedData)
  UTM_0.HOTEL_SELECT.Value = WT_POSTED.HOTEL_SELECT;
  UTM_0.COMMAND.Value = WT_POSTED.COMMAND;
```

The selected fields are determined.

```
wtMarkedFields = WT_POSTED.wt_markedFields.split(',');
for( i=1; i<wtMarkedFields.length; i++)
{
  UTM_0[wtMarkedFields[i]].InputState = 'D';
  UTM_0[wtMarkedFields[i]].InputStateAct = 'D';
}
```

If the Suspend key is not activated, there is one call each to send() and receive() in order to synchronize the WebTransactions dialog cycles and the host dialog steps. Finally, the setNextPage() function defines the next template to be processed.

```
//}}
//{{WebLab(processHostCommunication)
if ( WT_POSTED.wt_special_key == 'Suspend' || UTM_0_system.SUSPEND )
{
  UTM_0_system.SUSPEND = false;
}
else
{
  try {
```



```
        UTM_0.send();
        UTM_0.receive();
        if( UTM_0_system.APPLICATION_PREFIX )
            setNextPage( UTM_0_system.APPLICATION_PREFIX + '@' + UTM_0_system.FLD
);
        else
            setNextPage( UTM_0_system.FLD );
    }
    catch (e) {
        if ( WT_SYSTEM.COMMUNICATION_ERROR_FORMAT )
            setNextPage( WT_SYSTEM.COMMUNICATION_ERROR_FORMAT );
    }
    }
    //}}
//-->
</wtonreceivescript>
</body>
<wtif (UTM_0_system.EPILOG)>
    <wtinclude Name="##UTM_0_system.EPILOG#">
</wtif>
</html>
```

6 Editing templates

The interface of automatically generated templates echoes the appearance and functionality of the terminal formats. This interface can be edited generally in the master template or specifically in the individual templates:

- The openUTM-specific master template `UTM.wmt` defines the general layout for fixed areas. It is used when generating format-specific templates and can be adapted for specific applications (see [section “Master template UTM.wmt” on page 100](#)).
- It is possible to alter the graphical layout of each generated template, e.g. convert a selection menu into a drop-down list, by editing the templates directly. This is carried out using the template language, which is described in detail in the WebTransactions manual “Template Language”.

The WebLab editor provides a particularly comfortable and convenient way of editing templates. For basic information on WebLab refer to the WebTransactions manual “Concepts and Functions”. A detailed description can also be found in the WebLab extensive online help texts.

The following is simply a brief presentation of the design possibilities - which are by and large the same for all WebTransactions product variants. The focal point of this chapter resides in the openUTM-specific characteristics: templates for FHS and FORMANT partial formats and the openUTM line mode.

6.1 Master template UTM.wmt

WebTransactions uses master templates as a model for the generation of format-specific templates. They therefore ensure a consistent layout. Like any other template, master templates can contain fixed HTML areas and any WTML tags and WTScripts. However, in master templates you can also use special master template tags, known as MT tags, which are described in the WebTransactions manual “Template Language”.

The use of master templates is especially effective in the case of host applications in which large numbers of formats possess a similar structure: e.g. a fixed subdivision into header, workspace and footer. In such cases, it is sufficient to define the layout once in a master template and then apply this master template when generating the format-specific templates. All generated templates are thus automatically structured as defined.

WebTransactions for openUTM is supplied with the standard master templates `UTM.wmt` and `UTMpartial.wmt` for partial formats. These can be used in their current form or adapted individually as you wish. `UTMpartial.wmt` is described on [section “Structure of the master template UTMpartial.wmt” on page 106](#).

The standard master templates already contain all the WTML tags and WTScripts that are the same for all the templates of the supply unit in question, for example, checks to establish whether a connection-specific system object exists.

Via the WebLab graphical user interface, you can specify which master template is to be used for generation. You can define certain generation options (e.g. the generation method) both in the master template or directly in WebLab. The specifications which you make in WebLab override the corresponding specifications in the master template.

6.2 Designing templates

The interface of format-specific templates echoes the appearance and functionality of the terminal representation. If you wish to alter the graphical layout of the generated templates, e.g. convert a selection menu into a drop-down list, you must edit the templates directly. This is carried out using the template language, which is described in detail in the WebTransactions manual “Template Language”.

The WebLab editor provides a particularly comfortable and convenient way of editing templates. For basic information on WebLab refer to the WebTransactions manual “Concepts and Functions”. A detailed description can also be found in the WebLab extensive online help texts.

There are no limits to the ways in which you can design your WebTransactions templates: moving, clickable images, Java scripts and applets, ActiveX controls, video and audio sequences etc. can all be inserted. The WebTransactions template language is open: you can use all the language resources supported by your Web browser.

6.2.1 Defining the global layout

There are three options for defining a global layout for all templates:

- include templates
- use master templates
- the system object attributes `EPILOG`, `FORMTPL` and `PROLOG`

Including templates

In order to apply certain design features globally to all templates, e.g. insert company logos or general information on all pages, it is possible to create the corresponding objects in a separate file and incorporate this in the templates using include tags. This makes it easier to manage the templates. If modifications are required, they need only be made once in the central include template.

Using master templates

You can also define the global layout in master templates. To do this, you must edit the master template before generating the format-specific templates.

In the **Generate FLD and Template** field of WebLab, enter the master template to be used for the generation. Some generation options (e.g. the generation method) can be defined either in the master template or directly with WebLab.

The settings in the master template are transferred to the dialog box as default values. These can be modified in the dialog box if you wish, in which case your new values override the corresponding specifications in the master template. This means that the values displayed in the dialog box always apply when generating a template.

System object attributes: EPILOG, FORMTPL and PROLOG

You can also use the attributes `EPILOG`, `FORMTPL` and `PROLOG` to make global template design definitions. The master template generates these attribute calls in the templates. The attributes are then evaluated the next time the generated template is executed.

The attributes each have the name of a template which is executed at different times depending on the attribute:

<code>PROLOG</code>	at the start of the current template
<code>FORMTPL</code>	before execution of the DataForm tag in the current template
<code>EPILOG</code>	at the end of the current template

6.2.2 Customizing the interface

It is possible to incorporate standard enhancements in the interface automatically using the WebLab wizards. These wizards replace generated input fields (INPUT tags of type “Text”) with graphical elements, such as drop-down lists, radio buttons, check boxes or pushbuttons (an example can be found in [section “Editing templates” on page 53](#)).

WebLab provides you with support for the most important HTML tags (command: **Add/HTML**). However, you can also design the HTML user interface with an HTML editor quite independently of the generated templates and then use the WebLab **Design/Merge** command within a wizard to convert the interface into templates.

WTBeans

You use WTBeans to edit the functionality of your templates and define how data is displayed in the browser.

WTBeans are re-usable components which you use, for example, to design the GUI of a WebTransactions application or establish a connection to a host application. A distinction is made between inline and standalone WTBeans. An inline WTBean corresponds to a part of a template and is inserted in an existing template. A standalone WTBean corresponds to an autonomous template, for example the WTBean that is used to generate the start template.

You can insert inline WTBeans in your templates or use a standalone WTBean to generate a separate template. For more information, refer to the WebTransactions manual “Concepts and Functions”.

6.2.3 Designing the dialog sequence

WebTransactions not only offers you options for giving your host applications a “face-lift”. It also allows you to redesign the dialog sequence. The strict 1:1 correspondence between HTML pages and host formats thus no longer applies. With WebTransactions you can actively modify the dialog strategies provided by the host applications: input/output elements can be filtered out or added, and dialog steps can be combined or split up.

6.3 Special characteristics of FHS/FORMANT partial formats

WebTransactions supports openUTM applications that use partial formats of the FHS and FORMANT formatting systems with the master template for partial formats

`UTMpartial.wmt`. If your openUTM application works with partial formats, you need only generate partial templates from the partial formats using the `UTMpartial.wmt` master template.

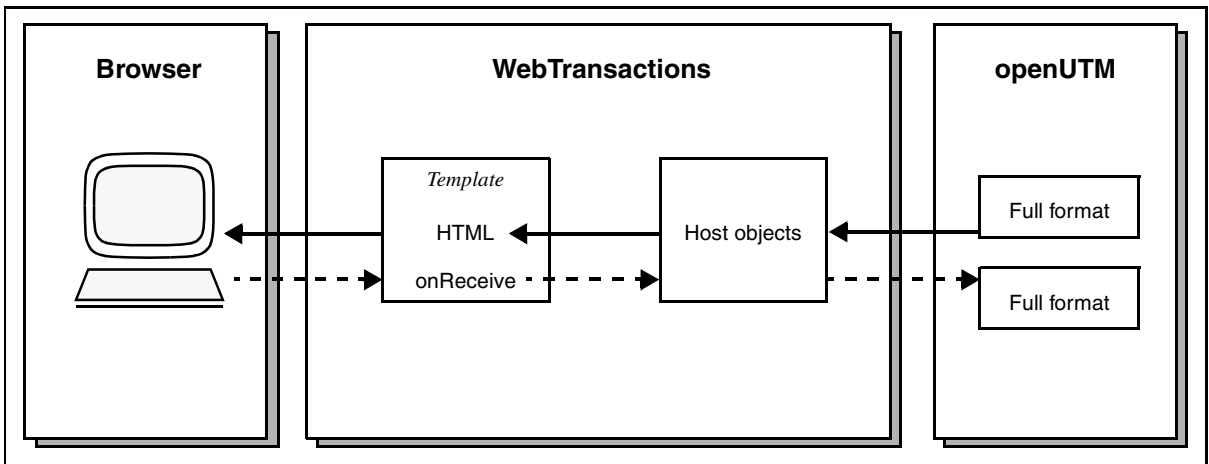
These partial format templates differ from the full format templates. This section describes

- the communications sequence for partial formats
- the structure of the master template `UTMpartial.wmt`
- the structure of partial format templates

and provides a number of comments concerning the enhancement of partial formats.

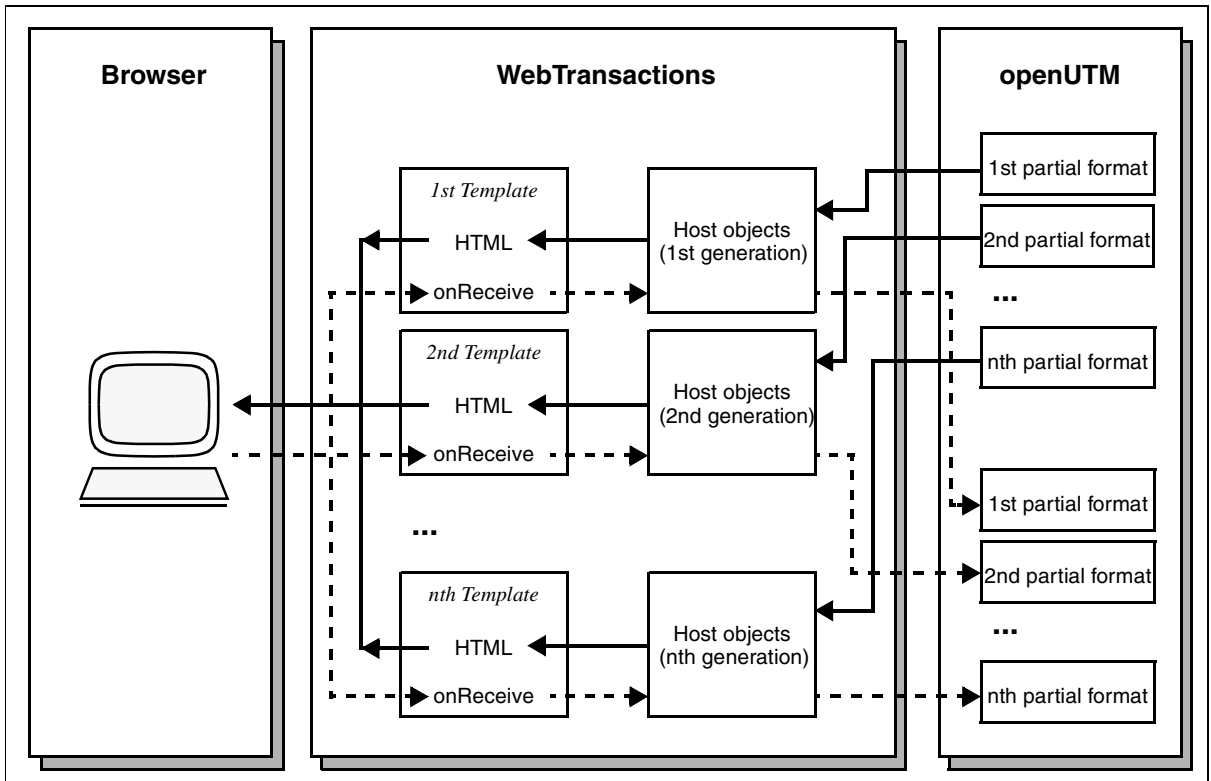
6.3.1 Communications sequence

In the case of full formats, the openUTM application alternately sends and receives messages. The data path is as follows:



The openUTM application sends a message. WebTransactions saves this as a host data object. The template controls the generation of the HTML page which is then sent to the browser. The returned data is copied to the host data objects in accordance with the OnReceive tag. These host data objects are sent in a message to the openUTM application.

In the case of partial formats, the data takes the following path:



openUTM sends the 1st partial format and WebTransactions saves it in host data objects. The 1st template generates the basic HTML structure. The contents of the host data object are saved below the communication object in the `wtPartialFormats` array. In the same way, the 2nd - nth partial formats are then received in sequence and stored in the form of host data objects (2nd - nth generation). HTML is generated and the host data objects are saved. The entire generation result is sent to the browser as an HTML page.

The posted data is now sent to the host application in a number of steps:

First the 1st host data object generation is restored from the backup. The associated posted data is copied to the host data objects and these are then sent to the openUTM application (OnReceive tags of 1st template). In the same way, the 2nd to nth generations of host data objects are restored, filled with the corresponding posted data and sent to the openUTM application (OnReceive tags of the 2nd - nth generations).

6.3.2 Structure of the master template UTMpartial.wmt

If partial formats are used in your application, you must use the master template `UTMpartial.wmt` when generating templates. `UTMpartial.wmt` has a similar structure to `UTM.wmt`. This section focuses on the partial-format-specific details of `UTMpartial.wmt`.

For the sake of clarity, the template has been divided into numbered sections which are explained separately. These sections are also referred to under [section “Editing with partial format templates” on page 117](#).

Section (1)

The template begins with a header containing generation information. The first `wtOnCreateScript` sets the variables `%%CommObj%` and `%%CommObj%_system` with a reference to the current communication object and the system object to be used for communication.

The edit mode (insert or overwrite) is set to `isOverwrite` according to the specification in `EDIT_MODE`.

Below the communication object, an array `wtPartialFormats` is created in which the prefix `FORMAT_SEQ` and the contents of the partial formats are saved. The first template used for a sequence of partial formats initializes the `wtPartialFormats` object. If the object is found and the `current` attribute is `< 0`, this indicates that the page has been recalled following interruption with the `Suspend` key. If this occurs, the `Suspend` flag is set. This controls whether data is to be retrieved from the host or from the backup in `wtPartialFormats` when setting up the page.

```
%%GlobalSettings Protocol="UTM"%
<!--
  //{{WebLab(assignCommunicationObject)
  %%CommObj% = WT_HOST.active || WT_HOST.%%CommObj%;
  if (%%CommObj%.WT_SYSTEM != null)
    %%CommObj%_system = %%CommObj%.WT_SYSTEM; // communication specific
system object
  else
    %%CommObj%_system = WT_SYSTEM; // global system object
  //}}
  // propagate communication object to included WTML documents ///////////////
wtCurrentComm = %%CommObj%;
wtCurrentComm_system = %%CommObj%_system;
if ( wtCurrentComm_system.EDIT_MODE )
{
  if ( typeof wtCurrentComm_system.isOverwrite == 'undefined' &&
wtCurrentComm_system.EDIT_MODE.match(/OVERWRITE/) )
    wtCurrentComm_system.isOverwrite = true;
  else if (wtCurrentComm_system.EDIT_MODE == 'OVERWRITE')
    wtCurrentComm_system.isOverwrite = true;
```

```

    else if (wtCurrentComm_system.EDIT_MODE == 'INSERT')
        wtCurrentComm_system.isOverwrite = false;
    } else
        wtCurrentComm_system.isOverwrite = false;

    // create array for contents of all partial formats //////////////////////////////////////
    if( ! %%CommObj%.wtPartialFormats )
    {
        // first format in a sequence of partial formats
        %%CommObj%.wtPartialFormats = new Array;
        %%CommObj%.wtPartialFormats.current = 0;
        %%CommObj%.wtPartialFormats[ 0 ] = new Object;
        %%CommObj%.wtPartialFormats[ 0 ].FORMAT_SEQ =
%%CommObj%_system.FORMAT_SEQ;
        %%CommObj%.wtPartialFormats[ 0 ].Contents =
%%CommObj%.WT_HOST_MESSAGE.Contents;
    }
    else if( %%CommObj%.wtPartialFormats.current < 0 )
    {
        // wtPartialFormats exits but current<0 means recovery from suspend
        %%CommObj%.wtPartialFormats.suspended = true;
        %%CommObj%.wtPartialFormats.current = 0;
    }
}
//-->
</wtOnCreateScript>

```

Section (2)

In addition, the `FORMAT_SEQ` attribute is provided with the index of the partial message and the message buffer of the host adapter (`%%CommObj%.WT_HOST_MESSAGE.Contents`) is updated, the corresponding partial message being taken from the backup in `wtPartialFormats` (see [Section \(5\)](#)).

```

<wtOnReceiveScript>
<!--
    // restore partial format //////////////////////////////////////
    if( %%CommObj%.wtPartialFormats.current < 0 )
        %%CommObj%.wtPartialFormats.current = 0;
    else
    {
        %%CommObj%.wtPartialFormats.current++;
        %%CommObj%_system.FORMAT_SEQ =
            %%CommObj%.wtPartialFormats[ %%CommObj%.wtPartialFormats.current
].FORMAT_SEQ;
        %%CommObj%.WT_HOST_MESSAGE.Contents =
            %%CommObj%.wtPartialFormats[ %%CommObj%.wtPartialFormats.current
].Contents;
    }
}

```

```

    }
  //-->
</wtOnReceiveScript>

```

Section (3)

This section contains code which may only be run once per HTML page. This must always occur in the first partial format. This decision can be made using the `FORMAT_SEQ` system object attribute. In the case of full formats, this attribute is empty and for partial formats the host adapter supplies a sequential number.

The following steps are carried out:

- execution of a PROLOG template, if available.
- output of the basic HTML framework.
- output of CSS definitions, if the browser supports style sheets.
- opening of an HTML form.
- opening of a table which frames the entire format.
- calling of the centrally modifiable templates `wtBrowserFunctions.htm` (cursor control) and `wtKeysUTM.htm` (functions for operating the page).
- definition of the functions `taggedInput()` and `taggedOutput()` which can be used to prepare unprotected (`input`) and protected fields.

```

<wtRem** page header only generated for first partial format *****>
<wtIf ( %%CommObj%_system.FORMAT_SEQ == "1" || %%CommObj%_system.FORMAT_SEQ
== " " )>
<wtIf (%%CommObj%_system.PROLOG)>
  <wtInclude Name="###%CommObj%_system.PROLOG#">
</wtIf>
<html>
<head>
<title>WebTransactions V7.5 - application
###%CommObj%_system.SYM_DEST#</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="#COCOCO">
<form WebTransactions>
    <table frame="border" rules="all">
        <tr>
            <td>
                <wtInclude name="wtBrowserFunctions">
                <wtInclude name="wtKeysUTM">
                <wtIf (%%CommObj%_system.FORMTPL)>
                    <wtInclude Name="###%CommObj%_system.FORMTPL#">
                </wtIf>
            </td>
        </tr>
        <tr>
            <td>
                <wtOnCreateScript>
                <!--
                    colors = new Array('RED','GREEN','YELLOW','BLUE','MAGENTA', 'CYAN',
'WHITE');
                    space80 = "
";

                    wtInputFields = new Object();
function taggedInput( hostObject )
    {
        if ( hostObject.Protected == 'Y' )

```

```

        {
            taggedOutput( hostObject );
            return;
        }
        currentLength = hostObject.Length;
        input = '<input type=' + (hostObject.Visible == 'N' ?
"password" : "text" );
        if ( WT_BROWSER && (WT_BROWSER.is_ie || WT_BROWSER.is_ns61up) )
        {
            input += ' class="box" style="width:' + (currentLength *
WT_BROWSER.charWidth + 1) + 'px';
            input += (hostObject.Blinking == 'Y' ? '; background-
color:#FFC0C0' : '');
            input += (hostObject.Underline == 'Y' ? ( hostObject.Intensity
== 'N' ? '; color:#A0A0FF' : '; color:#0000A0' ) :
( hostObject.Intensity
== 'N' ? '; color:#A0A0A0' : '' )) + ''';
        }
        input += ' name="_' + %%CommObj%_system.FORMAT_SEQ+ '_' +
hostObject.Name
            + ' size="' + currentLength
            + ' maxlength="' + currentLength
            + ' value="' + hostObject.Value
            + (hostObject.DataType == 'Numeric'?''
numeric="1:''')
            + (hostObject.Detectable == 'Y'?''
markable="1:''')
            + '>';
        document.write( input );
    }

function taggedOutput( hostObject )
{
    if ( hostObject.Protected == 'N' )
    {
        taggedInput( hostObject );
        return;
    }
    if ( hostObject.Visible == 'Y' )
    {
        output = hostObject.HTMLValue;
        if ( hostObject.Inverse == 'Y' )
        {
            if ( hostObject.Color && hostObject.Color.toUpperCase() !=
'N')
                output = '<font color="#000000" style="\background-color:'
+ colors[hostObject.Color-1] + '\">'+output+'</font>';
        }
    }
}

```

```

        else if ( hostObject.Color && hostObject.Color.toUpperCase() !=
'N')
            output = '<font color="' + colors[hostObject.Color-1] + '">'
+ output + '</font>';
            if (hostObject.Intensity == 'H')
                output = '<b>' + output + '</b>';
            if (hostObject.Blinking == 'Y')
                output = '<i>' + output + '</i>';
            if (hostObject.Underlined == 'Y')
                output = '<u>' + output + '</u>';
            document.write( output );
        }
    else
    {
        document.write( space80.substr(0,hostObject.Length));
    }
}
//-->
</wtOnCreateScript>
<!-- -----
----- -->
<!-- begin of host screen section
-->
<!-- -----
----- -->
</wtIf>

```

Section (4)

As with full formats, this section contains all constant elements of the format as well as HTML input tags for input fields.

```
<div style="color:#000000"><pre>\
%%LINES CellsDelimiter="-" TaggedInput=Enabled
TaggedOutput=Enabled%</pre></div>
<wtOnCreateScript>
<!--
        if( %%CommObj[%CommObj%.WT_HOST_MESSAGE.CursorField] &&
            %%CommObj[%CommObj%.WT_HOST_MESSAGE.CursorField].IOType ==
'INPUT' )
            wtCursor = '_' + %%CommObj%_system.FORMAT_SEQ + '_' +
                %%CommObj%.WT_HOST_MESSAGE.CursorField;
        //-->
</wtOnCreateScript>
<script type="text/javascript">
<!--
<wtOnCreateScript>
<!--
        for( element in wtInputFields )
        {
            if (!WT_BROWSER.acceptMaxLength)

document.writeln('wtSetMaxLength('_',%%CommObj%_system.FORMAT_SEQ,'_',
element, ',', wtInputFields[ element ].Length, ');');
            if (!WT_BROWSER.acceptInputAttributes)
            {
                //if (wtInputFields[ element ].DataType == 'NUMERIC')
                // document.writeln('wtSetInputAttribute("numeric",
"_' ,%%CommObj%_system.FORMAT_SEQ,'_', element, ');');
                if (wtInputFields[ element ].Detectable == 'Y')
                    document.writeln('wtSetInputAttribute("markable",
"_' ,%%CommObj%_system.FORMAT_SEQ,'_', element, ');');
            }
        }
        //-->
</wtOnCreateScript>
//-->
</script>
```


Section (5)

This section contains code which may only be executed once at the end of an HTML page (similar to [Section \(3\)](#)). This must always occur in the last partial format. The decision can be made using the `FORMAT_SEQ` system object attribute. For full formats, this attribute is empty and for the last partial format of the sequence, the host adapter supplies the value `LAST`.

All the HTML elements opened in [Section \(3\)](#) are closed (except `<html>`).

The cursor is positioned, as specified by the application, in an input field.

```
<wtIf ( %%CommObj%_system.FORMAT_SEQ == "LAST" ||
%%CommObj%_system.FORMAT_SEQ == " " )>
<!-- -----
-->
        <!-- end of host screen section
-->
        <!-- -----
----- -->
        </td>
    </tr>
</table>
</form>
<wtRem** initial focus selection *****>
<script type="text/javascript">
<!--
    <wtIf( wtCursor ) >
        wtSetFocus('##wtCursor#');
    <wtElse>
        wtBuildFieldList();
    </wtIf>
    //-->
</script>
</body>
</wtIf>
```

Section (6)

In this section, the data posted by the browser is copied to the host objects. If the `Suspend` key was activated, the partial format is saved once again to `wtPartialFormats`. Otherwise, the partial message is sent to the host.

When the last partial message is sent to the host, the entire backup in `wtPartialFormats` is deleted and the next message from the host is read.

```
<wtRem** Script executed after post of HTML page *****>
<wtOnReceiveScript>
<!--
```

```

    if (%%CommObj%_system.EDIT_MODE
&&%%CommObj%_system.EDIT_MODE.match(/USER/))
        %%CommObj%_system.isOverwrite = (WT_POSTED.wt_isOverwrite=='1');
    //{{WebLab(processPostedData)
    %%OnReceiveCopies%
    wtMarkedFields = WT_POSTED.wt_markedFields.split(',');
    elementPrefix = '_' + %%CommObj%_system.FORMAT_SEQ + '_';
    for( i=1; i<wtMarkedFields.length; i++)
        if ( wtMarkedFields[i].indexOf(elementPrefix) == 0 )
            {
                var elementName = wtMarkedFields[i].substr(elementPrefix.length);
                %%CommObj%[elementName].InputState = 'D';
                %%CommObj%[elementName].InputStateAct = 'D';
            }
    //}}
    //{{WebLab(processHostCommunication)
    if ( WT_POSTED.wt_special_key == 'Suspend' || %%CommObj%_system.SUSPEND )
        {
            %%CommObj%_system.SUSPEND = false;
            %%CommObj%.wtPartialFormats[ %%CommObj%.wtPartialFormats.current
].Contents =
                %%CommObj%.WT_HOST_MESSAGE.Contents;
            if( %%CommObj%_system.FORMAT_SEQ == "LAST" ||
%%CommObj%_system.FORMAT_SEQ == " " )
                {
                    %%CommObj%.wtPartialFormats.current = -1;
                    %%CommObj%_system.FORMAT_SEQ = %%CommObj%.wtPartialFormats[ 0
].FORMAT_SEQ;
                    %%CommObj%.WT_HOST_MESSAGE.Contents = %%CommObj%.wtPartialFormats[ 0
].Contents;
                }
        }
    else
        {
            try {
                %%CommObj%.send();
                if( %%CommObj%_system.FORMAT_SEQ == "LAST" ||
%%CommObj%_system.FORMAT_SEQ == " " )
                    {
                        delete %%CommObj%.wtPartialFormats;
                        %%CommObj%.receive();
                        if( %%CommObj%_system.APPLICATION_PREFIX )
                            setNextPage( %%CommObj%_system.APPLICATION_PREFIX + '@' +
%%CommObj%_system.FLD );
                        else
                            setNextPage( %%CommObj%_system.FLD );
                    }
            }

```

```

    }
    catch (e) {
        if ( WT_SYSTEM.COMMUNICATION_ERROR_FORMAT )
            setNextPage( WT_SYSTEM.COMMUNICATION_ERROR_FORMAT );
    }

    }
    ///}
//-->
</wtOnReceiveScript>

```

Section (7)

If the current partial message is not the last one, the next partial message is read from the backup (restoring the page after `Suspend`) or retrieved from the host.

The WTML document for displaying the next partial format is included.

```

<wtIf ( %%CommObj%_system.FORMAT_SEQ != "LAST" &&
%%CommObj%_system.FORMAT_SEQ != " " )>
<wtOnCreateScript>
<!--
    if( %%CommObj%.wtPartialFormats.suspended )
    {
        %%CommObj%.wtPartialFormats.current++;
        %%CommObj%_system.FORMAT_SEQ =
            %%CommObj%.wtPartialFormats[ %%CommObj%.wtPartialFormats.current
].FORMAT_SEQ;
        %%CommObj%.WT_HOST_MESSAGE.Contents =
            %%CommObj%.wtPartialFormats[ %%CommObj%.wtPartialFormats.current
].Contents;
    }
    else
    {
        try {
            %%CommObj%.receive();
        }
        catch (e) {
            if ( WT_SYSTEM.COMMUNICATION_ERROR_FORMAT )
                setNextPage( WT_SYSTEM.COMMUNICATION_ERROR_FORMAT );
        }
        %%CommObj%.wtPartialFormats.current++;
        %%CommObj%.wtPartialFormats[ %%CommObj%.wtPartialFormats.current ] = new
Object;
        %%CommObj%.wtPartialFormats[ %%CommObj%.wtPartialFormats.current
].FORMAT_SEQ =
            %%CommObj%_system.FORMAT_SEQ;

```

```

        %%CommObj%.wtPartialFormats[ %%CommObj%.wtPartialFormats.current
    ].Contents =
        %%CommObj%.WT_HOST_MESSAGE.Contents;
    }
//-->
</wtOnCreateScript>
<wtInclude name=
"##( %%CommObj%_system.APPLICATION_PREFIX ?
        %%CommObj%_system.APPLICATION_PREFIX + '@' :
        '' ) +
        %%CommObj%_system.FLD#">

```

Section (8)

In this section, the `wtPartialFormats` storage object is reset, `EPILOG` is called if required, and the last open tag `<html>` is closed.

```

<wtElse>
<wtOnCreateScript>
<!--
    // current=-1 signals end of format sequence
    %%CommObj%.wtPartialFormats.current = -1;
    %%CommObj%_system.FORMAT_SEQ = %%CommObj%.wtPartialFormats[ 0 ].FORMAT_SEQ;
    %%CommObj%.WT_HOST_MESSAGE.Contents = %%CommObj%.wtPartialFormats[ 0
    ].Contents;
//-->
</wtOnCreateScript>
<wtIf (%%CommObj%_system.EPILOG)>
    <wtInclude Name="##%%CommObj%_system.EPILOG#">
</wtIf>
</html>
</wtIf>

```

6.3.3 Editing with partial format templates

In each individual case, the possibilities for retouching and editing partial format templates depend to a large extent on the way the partial formats are used in your host application. For this reason, this section can make no claims to completeness. It simply outlines a number of possibilities in order to provide pointers towards possible editing.

It is possible to make this change in the master template described above (see the [section “Structure of the master template UTMpartial.wmt” on page 106](#)) in which case your changes apply to all generated templates or in the templates already generated.

For example, if you want to convert the layout to include HTML buttons and list boxes and map the data received from the browser to the format interface (field file), you should restrict yourself to [Section \(4\)](#) and [Section \(6\)](#).

If you want to modify the general layout of the page (header, footer, background etc.) you can also edit [Section \(3\)](#) and [Section \(5\)](#).

You should leave the remaining sections unchanged to avoid disrupting the sequence of communications with the openUTM application.

Editing with individual partial formats

If you want to edit the partial format templates individually, you can of course insert convenient HTML tags to edit the user interface. However, you cannot make any basic changes to the subdivision of the format (screen). For example, you cannot position the fields from the second partial format in front of those of the first partial format.

New template for a screen’s partial template sequence

This section illustrates how a combination of partial formats (a logical screen) can be freely redesigned. In this case, the generated templates must also be edited as it is generally not sufficient to modify the master template alone.

The two cases below illustrate two fundamentally different situations:

- In the first example, the partial template sequence has a clear starting point, i.e. in this host application, the 1st partial format occurs as such in this one screen only.
- In the second example, there is no identifiable partial format in the sequence of partial formats from which it is possible to deduce the logical screen (combination of partial formats). The logical screen can only be deduced from the entire combination itself. Not until the last partial format has been ascertained can the logical screen be unambiguously identified.

Unique start of partial format sequence

If you know that a given partial format is always followed by a certain sequence of partial formats, then you can store all the associated templates in sequence in a file with the name of the 1st partial format. You thus eliminate the include tags and combine multiple templates to create one large one.

To do this, copy the successor template to the position of the include tag in the predecessor template. The IF tags in [Section \(3\)](#), [Section \(5\)](#) and [Section \(7\)](#) can be cancelled within the template sequence, as the order of templates is fixed. In [Section \(7\)](#), however, the ELSE branch must be retained.

The body of [Section \(3\)](#) is inserted once at the start of the overall template, and [Section \(5\)](#) appears at the end.

In this type of overall template, it is easier to rearrange the display of the contents of the different partial formats on the HTML page. However, here again you must take account of the generation of host data objects which is active at the moment of generation. You may find it necessary to restore a generation at generation time (see [Section \(2\)](#)) or buffer a number of template object values for later use.

Any combination to end of partial format sequence

If, despite the fact that the partial formats in your host application may be present in almost any order, you still want to change the display of the individual combinations in the browser, then you are advised to proceed as follows.

The partial format templates remain as individual templates. However, they do not generate HTML, but simply save the values received at the host application for further processing. In addition, occurrences of individual templates in the template sequence are recorded in a special attribute. When all the partial format templates have been worked through, a new template (combination template) is called. This generates the HTML page from the buffered data of the entire partial format sequence and contains the OnReceive tags.

1. Delete the sections which generate HTML, namely from [Section \(3\)](#), [Section \(4\)](#) and [Section \(5\)](#) of the partial format templates. These sections are recombined in the combination template or are adapted as required.
2. Replace section (4) with a `wtOnCreate` script which copies the contents of the host data objects to unambiguous attributes of a template object. For example, you can use `FORMAT_SEQ` as a level in this object structure:

```
collect = new Object;
collect['_'+host_system.FORMAT_SEQ] = new Object;
collect['_'+host_system.FORMAT_SEQ].object = %%ComObj%.object.Value;
...
```

(In a generated template, `%%CommObj%` is replaced by the current communication object.)

3. Delete [Section \(6\)](#) and write the corresponding statements to the combination template.
4. Add a statement defining the sequence of templates in a template object attribute.

```
if ( COMBINATION_TEMPLATE )
    COMBINATION_TEMPLATE += %%CommObj%_system.FLD;
else
    COMBINATION_TEMPLATE = %%CommObj%_system.FLD
```

(In a generated template, %%CommObj% is replaced by the current communication object.)

5. Add an include tag that calls a template for the received sequence of partial formats.

```
<wtInclude Name="##COMBINATION_TEMPLATE#">
```

6. The `COMBINATION_TEMPLATE` for the sequence in question is now able to generate the complete HTML page. To do this, the data stored in the template object is accessed. It contains an `OnReceive` script for all the partial formats which occur in the sequence to make it possible to send the posted data back to the openUTM application in individual partial messages. Finally there is a `receive` tag which receives the next message from the openUTM application.

6.4 Support for openUTM line mode

WebTransactions supports messages in openUTM line mode. If an unformatted message is issued by the openUTM application, WebTransactions automatically uses the special line mode template `wtlmode.htm` for display purposes. When the base directory is created, this template is stored as a link or copy in `basedir\config\forms`. It appears in the browser as illustrated below:

Terminal screen:

Send Data Reset Data

Next TAC:

CONVERSATION_TAC:

UTM Partner directly:

or next SYM_DEST:

UTM User:

Stop conversation:

Startpage:

Terminate session:

The line mode template contains the following dialog components:

Terminal screen:

This displays the last message received from the openUTM application. Commands can also be entered here in the same way as on the terminal. Data between the first modified position and the next end mark is taken into consideration.

At the start of the conversation, the first 8 characters (maximum) are used as the TAC to address the conversation; for this purpose, you must select blank spaces in the **CONVERSATION_TAC** field. The remaining data will be sent to the program unit as a message. During the conversation, the complete data will be sent to openUTM as a message. The end marker is represented by the ~ (tilde) character.

The form can be confirmed using the “Enter” key or the **Send Data** button. Line breaks can be inserted in the text area using the key combination, “Shift+Enter”.

Next TAC

Here you can enter the next TAC. To do this, select the TAC option in the **CONVERSATION_TAC** field.

CONVERSATION_TAC

Here select how the TAC is to be given to address the next openUTM conversation:

blank space The TAC must be at the start of the message (see **Terminal screen**).

TAC The TAC must be entered in the **TAC** field.

SYM_DEST The TAC is determined from the corresponding entry in the `upicfile` (see **or next SYM_DEST**).

UTM Partner directly

Here you can enter the address of the openUTM application directly. Enter the name of the host application in the first field and the name of the computer on which the host application runs in the second. Instead of entering the computer name, you can enter its IP address in the third field.

or next SYM_DEST

Here you can specify a symbolic destination name from the `upicfile`, in order to continue with a different conversation. These conversations must start with a program unit that does not expect any input, as no `send` is performed.

UTM User

Each command that is entered in line mode runs in a separate conversation. You can specify individual authentication levels for these conversations (`NONE|USER|PASSWORD`), enter user name and password and change the password.

Stop Conversation

This button can be used to stop a conversation in an emergency.

Startpage

Here you can return to the start template, `wtstart.htm`, if the new conversation that is to be started requires additional specifications. In this way, it is possible to start a conversation that starts with a program unit that expects some input (send on conversation start).

Terminate session

To exit line mode and terminate the entire session, use the **Quit** button.

Adapting the line mode template

You can also copy and modify the line mode template `wtlnmode.htm`. However, the name of the line mode template must not be changed.

The structure of unformatted messages is described in the file `wtlnmode.fld`, which is located in the `basedir/config` directory. The message contents are written to the host data object `CONTENTS`, which has a length of 32767 bytes.

Sending unformatted messages to openUTM

When sending a message to a openUTM program unit, `CONTENTS.Value` is set to the appropriate value and the `send` method is called. The message sent by WebTransactions consists only of the contents of `CONTENTS` up to the first binary zero. Binary zeros are used as fill characters in the `CONTENTS` field. If a value shorter than 32767 bytes is assigned to `CONTENTS.Value`, the rest of the field is padded with zeros. This makes it possible to send unformatted messages of different lengths.

Up to the version WebTransactions V3.0 the content of the `FLD` attribute was decisive, whether to send a formatted or an unformatted message. As of version WebTransactions V4.0 the decisive factor is the property `FormatType=-` in the `FLD` file, the `FLD` attribute is pointing to.

If customer-specific changes are made in `wtlnmode.fld` or `wtlnmode.htm`, you must make sure that these two files remain consistent on a version change.

Receiving unformatted messages from openUTM

When receiving unformatted messages, WebTransactions stores the contents in `CONTENTS.Value`. If several unformatted partial messages are sent to WebTransactions, these are collated in the `CONTENTS.value` up to a length of 32767 bytes by calling the `receive` method. If further unformatted messages are subsequently received, WebTransactions handles them in the same way as partial formats by setting the `FORMAT_SEQ` attribute to 1. To retrieve the complete message, the `receive` method must be called repeatedly until `FORMAT_SEQ` reaches the value `LAST`.

If it is not known from the outset how many partial messages have been received, the entire message can be structured in a loop:

```
for( host.receive(), msg=host.CONTENTES.Value;
    host.WT_SYSTEM.FORMAT_SEQ!=" " && host.WT_SYSTEM.FORMAT_SEQ!=" LAST";
    host.receive(), msg+=host.CONTENTES.Value );
```

Example: Adapted line mode template

```
<html> (1)
  <head>
    <title>Bulletin</title>
  </head>
  <body>
    <pre>##UTM_0.CONTENTES.Value#</pre>
    <wtDataForm>
      <input type="submit" name="GO" value="Fortsetzen">
    </wtDataForm>
  </body>
</html>
```

```
<wtOnReceiveScript> (2)
<!--
  UTM_0.CONTENTES.Value=" ";
  WT_SYSTEM.SYM_DEST="MAIN ENTER">;
  UTM_0.receive();
  setNextPage(UTM_0.WT_SYSTEM.FLD);
/-->
</wtOnReceiveScript>
```

The example above depicts a line mode template which simply makes one output and then continues with a specific conversation.

An HTML page is generated in area (1). This outputs the contents of the message from the `Value` attribute of the host data object `CONTENTES`. The page contains a button for the continuation of the session. If this is pressed, WebTransactions continues processing with the `wtOnReceiveScript` tag (area (2)).

In area (2), `CONTENTES` is deleted in order to prevent conversation chaining. A particular conversation is selected by specifying the `SYM_DEST` attribute. `receive` implicitly opens the conversation and retrieves the next message.

In the connection specific system object, `receive` supplies the `FLD` attribute with the format name. If the received format is another line mode message, the value will again be `wtlnmode`. In order to ensure that the format can definitely be presented to the appropriate template, you should use `setNextPage()` to ensure that the format name is taken as the next template to be executed.

6.5 Binary data support

Using WebTransactions for openUTM, you can also send any kind of data, including binary data. These data can be read from a file for sending and written to a file on reception.

The name of the file is specified in the system attribute, `COMMUNICATION_FILE_NAME`. This file will then be used as input to the next `send` or for output from the following `receive`. The name specified in `COMMUNICATION_FILE_NAME` only ever applies for the immediately following communication step. After use, the attribute is reset, i.e. the next input/output is handled by host data objects.

Example

Using WebTransactions, a SESAM/SQL database is to be accessed via openUTM, which also contains picture files. This database is used for the administration of personnel data and contains, in addition to the personal details (date of birth, address, etc.), picture files with passport photos of employees.

The following code example shows how these photos can be stored in the database and how they can be viewed.

```
<wtOnCreateScript>
// storing the photo, using own TAC if required
UTM_0_SYSTEM.COMMUNICATION_FILE_NAME = "john smith.gif";
    UTM_0.send();
    ...
// picture retrieval and display
UTM_0_SYSTEM.COMMUNICATION_FILE_NAME = "john smith.gif";
    UTM_0.receive();
    ...
</wtOnCreateScript>
```

Processing of binary data with WTML

Binary data can also be represented in Hexadecimal form. This is done using the host data object, `HexStringValue`. This attribute converts any binary character string to the usual hexadecimal representation (00 to FF).

7 Configuring connections

WebTransactions forms the link between Web browsers and host applications. While WebTransactions functions as a server from the point of view of the Web browser, it acts as a client as far as the host application is concerned.

To communicate with the host application, the supply unit “WebTransactions for openUTM” uses the UPIC (Universal Programming Interface for Communications) client/server communications protocol. UPIC is a complex communications protocol which offers more than the simple exchange of net data. For example, connecting via UPIC allows WebTransactions to use the openUTM user concept and the openUTM automatic restart facility. Since this protocol also permits the transfer of information about format names and openUTM function keys no changes to the openUTM program units are required for the Web connection.

All the communication components - browser, HTTP daemon with WebTransactions, openUTM application and the associated data storage facilities - can be distributed as required over a variety of platforms (multitier architecture). WebTransactions controls the transaction-monitored information exchange between client and server and guarantees reliability/availability, data security and performance even in complex distributed structures.

UPIC-R and UPIC-L

UPIC is available as UPIC-R (remote) and, additionally for Solaris and Linux, as UPIC-L (local).

You can use UPIC-L if WebTransactions and the openUTM host application are both present on the same Unix host. In this case, you declare the UPIC-L library via the system object's `UPIC_LIB` attribute and the path under which the openUTM application is installed via the `UTM_PATH` attribute.

In all other cases, UPIC-R is used.

UPIC libraries

The UPIC libraries are components of the WebTransactions product:

- For BS2000/OSD, the UPIC library (UPIC-R) is an integrated part of the WTHolder program.
- In the case of Solaris and Linux, two UPIC libraries are shipped with WebTransactions (for UPIC-R and UPIC-L respectively). If you want to work with UPIC-L, you must set the path name of this library in the system object's `UPIC_LIB` attribute. If you do not set this attribute, WebTransactions always uses a default UPIC-R library which is an integrated part of the WTHolder program.
- In the case of Windows, a UPIC-R library (`upicws32.d11`) is shipped with WebTransactions. Under Windows, UPIC is dynamically linked by default from this library.

Unicode support

The host adapter in WebTransactions for openUTM can also interpret data as Unicode characters at the UPIC interface.

The BS2000/OSD program `IFG2FLD` reads format descriptions from an IFG library and stores them in a format description source. The fields in an IFG library can contain the new attribute `Unicode`. As of Version 8.3, `IFG2FLD` calculates the offsets of the individual fields in the UPIC buffer dependently of this attribute.

You can use WebLab to generate templates and field files (FLD files) from the format description source. During this conversion, the Unicode marker is automatically taken into account as of `IFG2FLD` Version 8.3 (see [chapter “Generating templates” on page 75](#)).

FLD files

The FLD files must be re-generated for all the formats that contain Unicode fields.

Templates

Existing templates can be retained unchanged if the only change in a format was the change to Unicode fields. However, the assignment of the value `UTF-8` to the global system object attribute `CHARSET` must be inserted (see [section “Host control object WT_HOST_MESSAGE” on page 165](#)).

WebTransactions for openUTM creates data encoded in UTF-8, which is passed straight through to the browser and back. To achieve this, the following rules must be observed with respect to new templates:

- The templates must use `charset` to inform the browser what character encoding is to be used to display the data and to send the response. The browser receives this information
 - either in the meta tag:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```
 - or by setting the attribute `charset` in the HTTP header field `Content-Type`.
- The templates are only allowed to contain ASCII characters. If a template contains special characters (such as German umlauts) directly in the form of ANSI characters in character strings, these must be replaced by HTML escape sequences. The escape sequences themselves are made up of ASCII characters only and can therefore be used in UTF-8.
- String operations can produce incorrect results if UTF-8 characters are included in the string. Operations of this sort do not occur in the templates generated by WebTransactions.

ASCII-to-EBCDIC conversion

WebTransactions applications that have previously used the ASCII-to-EBCDIC conversion facility integrated in UPIC must be converted to use the ASCII-to-EBCDIC conversion facility provided by WebTransactions (see [“HOST_CHAR_CODE” on page 149](#)).

7.1 Aligning WebTransactions and the host

To establish communications via UPIC-R, you must declare the communication partners - WebTransactions and the openUTM application - to one another. This must be done both at the client and at the server.

To provide its services, UPIC uses the facilities of the platform-specific communication system which permits access to the transport system and manages the transport connections:

- in the case of BS2000/OSD this is BCAM. For BCAM (BS2000/OSD), appropriate BCMAP entries are only required in exceptional cases (see [section “BCMAP entries \(BS2000/OSD\)” on page 142](#)).
- For Solaris, Linux and Windows, the required transport system components are integrated in UPIC; the CMX (Solaris and Linux) and PCMX (Windows) transport systems must therefore no longer be used.

The platform-specific communication system is also responsible for host-side communications between the transport system and the openUTM host application.

In practice, there are three types of coupling:

WebTransactions platform	Host platform
Solaris, Linux or Windows	BS2000/OSD
Solaris, Linux or Windows	Solaris, Linux or Windows
BS2000/OSD	BS2000/OSD

Since the configuration is very similar in Solaris, Linux and Windows systems, these will be considered together. Theoretically, there is the fourth possible combination of WebTransactions under BS2000/OSD and openUTM host application under Solaris, Linux or Windows. However, such a combination is unlikely to occur in practice and is not therefore described here.

For all these variations, the following is required on the WebTransactions side:

- For the local WebTransactions application:

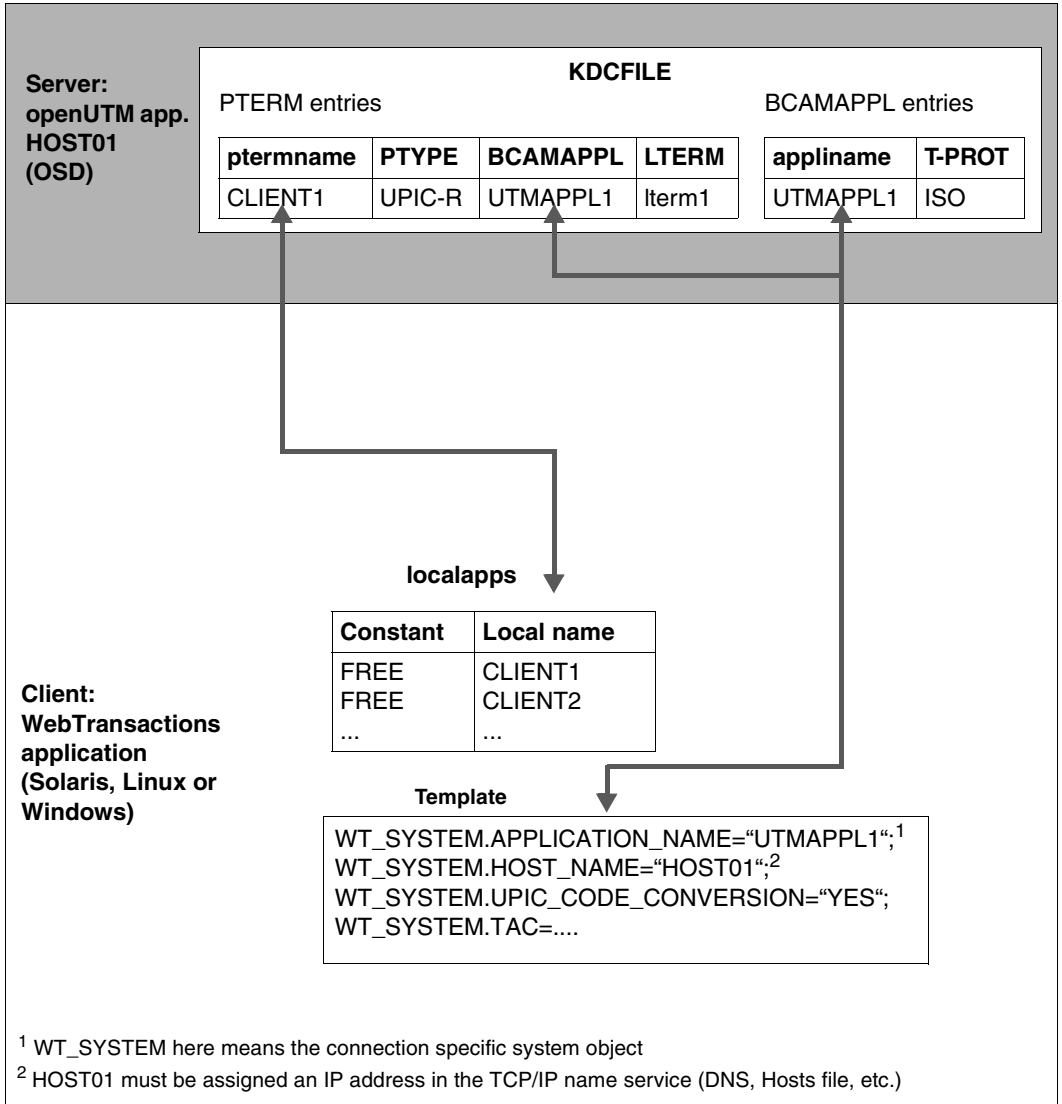
Entries in the `localapps` file, see [page 132](#)

- for the openUTM application:

Entries in the `upicfile` (see [page 134](#)) or provision of the corresponding system attributes `APPLICATION_NAME`, `HOST_NAME`, `TAC`, ... (see [page 133](#)).

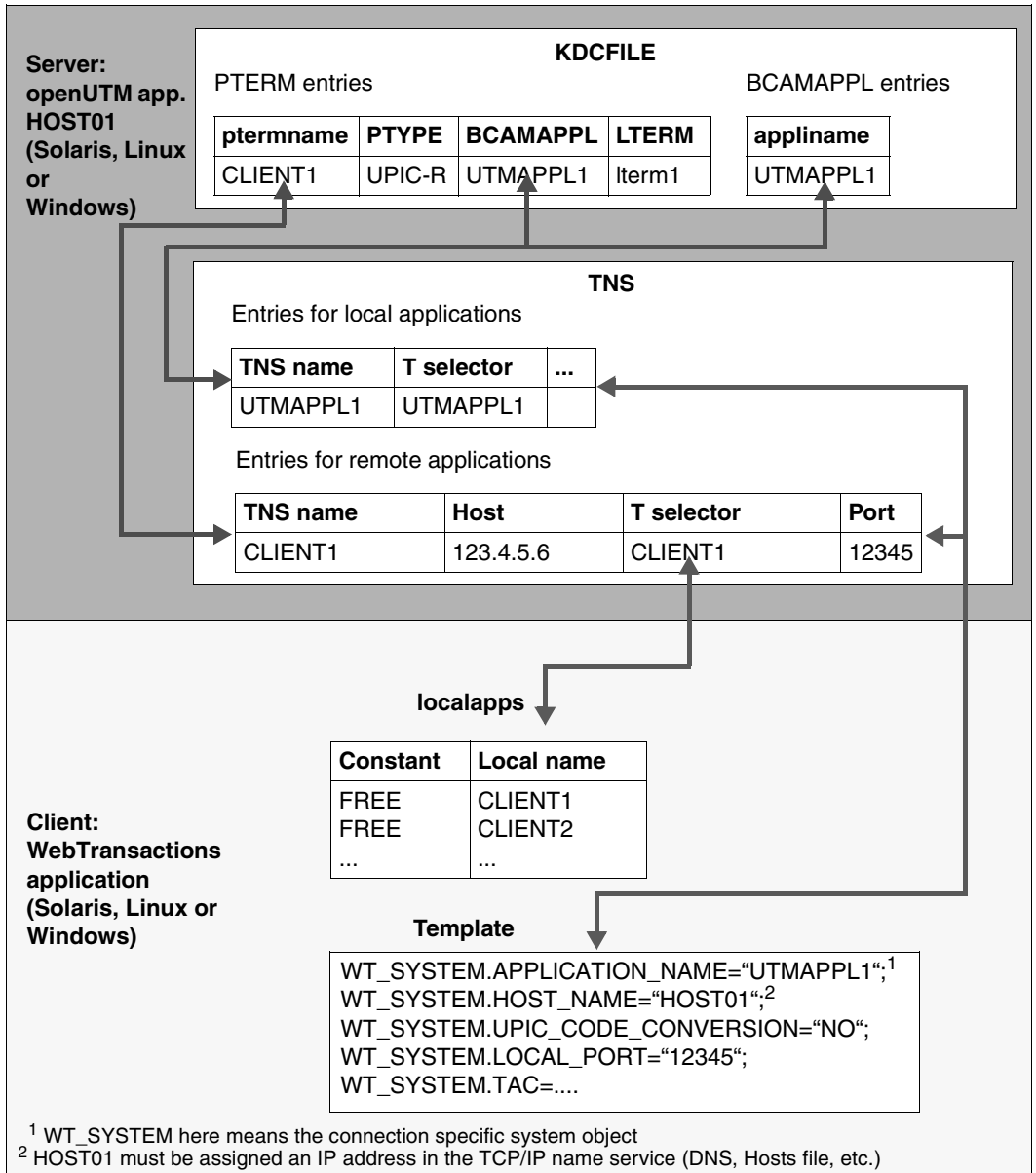
The figures on the following pages illustrate the various possible combinations and the relations between the entries. For more detailed information and examples for the individual entries, refer to [section “Configuring the WebTransactions side” on page 132](#) and [section “Configuring the openUTM \(host\) side” on page 138](#) which follow the figures.

WebTransactions under Solaris, Linux or Windows – openUTM host under BS2000/OSD



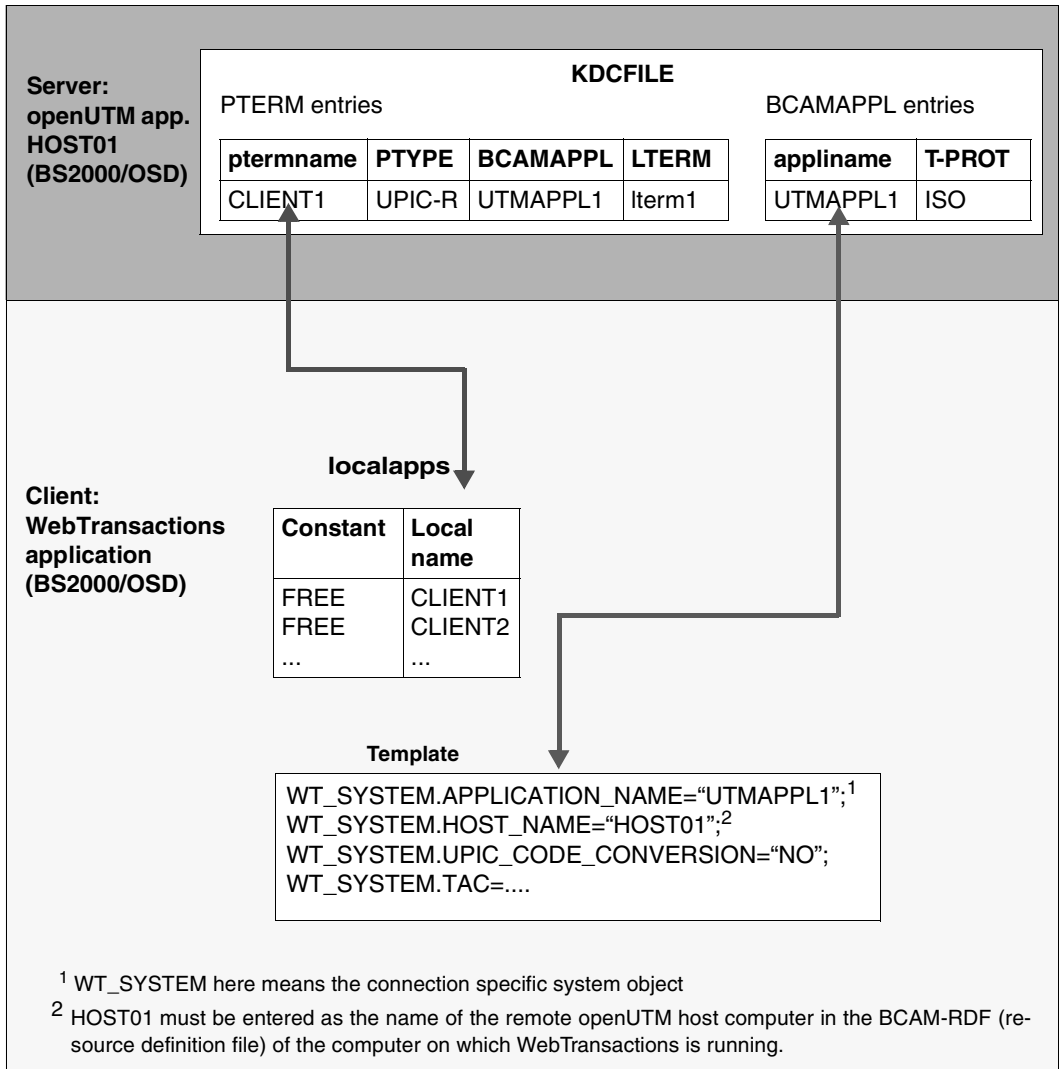
In this example, the parameters for the openUTM application were entered in the system object attributes. Alternatively, you can use the `upicfile`, see [page 134](#). In this case, you must provide the system object attribute, `SYM_DEST`.

WebTransactions under Solaris, Linux or Windows – openUTM host under Solaris, Linux or Windows (via UPIC-R)



In this example, the parameters for the openUTM application were entered in the system object attributes. Alternatively, you can use the `upicfile`, see [page 134](#). In this case, you must provide the system object attribute, `SYM_DEST`.

**WebTransactions under BS2000/OSD – openUTM host under BS2000/OSD
(usual case: without BCMAP entries)**



BCMAP entries are only necessary in exceptional cases (see [section “BCMAP entries \(BS2000/OSD\)” on page 142](#)).

In this example, the parameters for the openUTM application were entered in the system object attributes. Alternatively, you can use the `upicfile`, see [page 134](#). In this case, you must provide the system object attribute, `SYM_DEST`.

7.2 Configuring the WebTransactions side

On the WebTransactions side, you must configure the following:

- The local WebTransactions application by making entries in the `localapps` file
- The openUTM application. Here there are two possibilities:
 - via system attributes, see [page 133](#)
 - via the upicfile, see [page 134](#)

In some configurations, it is also necessary to set the outgoing port number of the local socket connection in the `LOCAL_PROT` attribute, see also [page 149](#).

7.2.1 localapps file

The WTHolder program logs on to the communication system under a local name. The communication system converts this name into a name which is known to the underlying transport system.

If multiple instances of the WTHolder program are to run simultaneously, then a separate local name is required for each parallel connection. The communication system converts each of these names into a T selector.

WebTransactions controls the assignment of local names to the WTHolder programs via the file `localapps`. This file must be located in the WebTransactions base directory and must contain the following entries:

- one entry for each simultaneous program run
- the text “FREE” as of column 1 (the session ID is entered here while an entry is in use)
- local name as of column 16

The file can also contain comments. These start with an asterisk (*) in column 1.



`localapps` is created automatically when a base directory is generated and already contains default entries.

Example localapps

```
*<Session-ID>  <Name>
FREE           LOCAL1
FREE           LOCAL2
```

WebTransactions under BS2000/OSD: automatic mapping

In the case of BCAM, configuration entries for the BCAM communication system (BCMAP entries) are now only necessary in exceptional cases (for more information see [section “BCMAP entries \(BS2000/OSD\)” on page 142](#)).

Local names for UPIC-L (Solaris, Linux)

In the case of local connections via UPIC-L, the local names defined in `localapps` must either correspond to a PTERM name of the openUTM application or KDCDEF statement TPOOL with PTYPE=UPIC-L must be specified on openUTM generation.

7.2.2 Addressing the openUTM application using system attributes

The openUTM application service can also be addressed directly using system attributes, without using an entry in the `upicfile`, which the SYM_DEST system attribute refers to. An entry which is preconfigured by the current WebTransactions versions is then necessary in the `upicfile`. This entry could miss if it is transferred from an old WebTransactions installation.

In the BS2000/OSD, the line is:

```
HD.DEFAULT NOCONN;
```

Also refer to the notes in section [“Preparing upicfile in the DMS user ID \(BS2000/OSD\)” on page 73](#).

In Windows/Linux/Solaris, the line is:

```
SD.DEFAULT NOCONN
```

The following attributes are available for addressing directly:

APPLICATION_NAME

Name of the openUTM application

HOST_NAME or HOST_IP_ADDRESS

Name or IP address of the computer on which the openUTM application is running.

HOST_PORT

Port number on which the openUTM application listens. The port number must only be specified if the openUTM application is not listening on port 102.

TAC Transaction code with which openUTM starts the service.

UPIC_CODE_CONVERSION

Indicates whether a code conversion should be carried out.

LOCAL_PORT

Port number used in the client computer as outgoing port, only necessary in special configurations.

For a detailed description of these attributes, see [page 146](#).

7.2.3 upicfile

The `upicfile` contains the data required for communication with the openUTM application. The `SYM_DEST` systemattribute (see [page 151](#)) is used determine the entry in the `upicfile`.

The `upicfile` entry provides UPIC with the actual target for the symbolic target entry in the `SYM_DEST` system attribute by assigning a host application name, a host name and a transaction code.

In order for UPIC to be able to access the `upicfile`, it must be in the base directory under Solaris, Linux and Windows, for BS2000/OSD it must be copied to the DVS (see [page 73](#)).



The `upicfile` is automatically generated when a base directory is created and contains example entries, including the entry, `SD.DEFAULT NOCONN`.

The `upicfile` is always evaluated if it exists in the base directory, even if a host application is addressed directly using system object attributes.

Contents of upicfile

The `upicfile` must contain the following entries in uppercase:

- identifier for ASCII/EBCDIC conversion (HD or SD)
- symbolic destination name via which UPIC accesses the entries in the `upicfile`
- name of the host application and the host on which the openUTM application is running.
- transaction code of the first program unit of a openUTM conversation (conversation TAC)
- where applicable, the port number on which the openUTM application listens.

Under BS2000/OSD, each line in the `upicfile` must end with a semi-colon and must not contain any comments.

Under Windows, each line of the `upicfile`, including the last, must end with `CR LF` (Return/Enter key). Under Solaris, Linux and Windows, the `upicfile` may contain comment lines. These always begin with the character `*` in the first column.

A communication destination is entered as follows:

```
{HD|SD}sym_dest utmappl[.host] tac [PORT=portnumber] [PROTOCOL={34|40}]
```

Meaning

{HD|SD} sets the automatic code conversion between ASCII and EBCDIC for communication with the host.

HD The data is sent by the host in EBCDIC and expected as such on receipt.

SD The data is sent by the host in ASCII and expected as such on receipt.



Conversion of data is only carried out as required, i.e. only when the client system uses a code that differs from that of the host system.

sym_dest Symbolic name via which UPIC accesses the entries in the `upicfile`. UPIC always uses the entry whose *sym_dest* name corresponds to the current value of the SYM_DEST system attribute. *sym_dest* must be exactly 8 characters in length and must be appended to the conversion identifier without any intervening space.

utmappl.utmhost identifies the openUTM application.

When using UPIC-R, the name must be given in two sections separated by a period (dot). *utmappl* is the name of the openUTM application and *utmhost* is the name of the computer on which the openUTM application is running. *utmappl.utmhost* must be preceded by a space.

When using UPIC-L, only the first section of the name must be given (without *utmhost*). *utmappl* can have up to 8 characters.

tac Transaction code with which openUTM starts the service. This transaction code must have been created in the openUTM configuration, either by the KDCDEF statement TAC or by means of dynamic configuration.

PORT=*portnumber* Port number over which the openUTM application is reached. This operand is only required if the openUTM application cannot be reached on port 102.

PROTOCOL={34|40} In PROTOCOL you specify whether communications should be performed via the extended V4.0 UPIC protocol (PROTOCOL=40) or the Version 3.4 UPIC protocol (PROTOCOL=34). The PROTOCOL specification is optional.

If you omit to specify PROTOCOL, UPIC first attempts to establish a conversation on the basis of the extended protocol (40). If this is not possible, UPIC then tries to establish the conversation using the V3.4 protocol (34).

Example upicfile

```
HDSERVICE4 UTMAPPLI.HOST0001 TAC4
```

UPIC uses this entry if the current value of the `WT_SYSTEM.SYM_DEST` attribute is `SERVICE4`. UPIC uses the name `UTMAPPLI.HOST0001` to identify the host application. `HOST0001` is the host name, that is registered in the TCP/IP Name Service, see also [section "Declaring the server computer name" on page 137](#).

`UTMAPPLI` stands for the host application name. If the openUTM application is working without TNS (Solaris, Linux and Windows) or without BCAMP entries (BS2000/OSD), then `UTMAPPLI` is the name that is generated in openUTM using `BCAMAPPL`, see [page 138](#). If the openUTM application uses TNS or BCAMP, then `UTMAPPLI` is the name in the corresponding TNS or BCAMP entry.

In this case, port 102 is used, as `PORT=` was not specified.

Since `PROTOCOL` is not specified in this example, UPIC first tries to establish a connection via the Version 4.0 protocol. If this attempt is unsuccessful, the connection is established using the Version 3.4 protocol.

7.2.4 Declaring the server computer name

The client and server computers must be declared to one another. There are two ways of doing this:

- Both computers are configured for the DNS (Domain Name Service). In this case, no further entries are necessary.
- Otherwise, the server computer on which the openUTM application is running must be entered at the client computer together with its Internet address:

- For Solaris, Linux and Windows, in the `hosts` file:

IP address UTM host

example: 123.4.5.6 HOST0001

- In the case of BS2000/OSD, the symbolic name must be entered either statically in the BCAM-RDF (resource definition file) or dynamically via BCIN.
- Alternatively you can use the connection specific system object attribute `HOST_IP_ADDRESS` (see [section “Addressing the openUTM application using system attributes” on page 133](#)).

7.3 Configuring the openUTM (host) side

Client computers and applications must be declared on the host computer on which the openUTM application is running. This is performed using KDCDEF control statements during openUTM generation.

If the openUTM application is running under Windows, Solaris or Linux, TNS entries on the host side may be required for communication with WebTransactions. These can be generated automatically during the KDCDEF run (see openUTM manual “[Generating Applications](#)”).

If the openUTM application is running on a BS2000/OSD platform, corresponding host-side BCMAP entries are required only in exceptional cases (see [section “BCMAP entries \(BS2000/OSD\)” on page 142](#)).

7.3.1 Adapting the openUTM generation

You declare the local names of the WebTransactions applications by means of the following KDCDEF control statements during KDCDEF generation.

BCAMAPPL statement

The BCAMAPPL statement defines the local name of the openUTM host application.

```
BCAMAPPL utmappl, T-PROT=RFC1006
```

utmappl local name of the openUTM host application.

Under Solaris, Linux and Windows, there must be a TNS entry for *utmappl* that maps *utmappl* to a T selector. Under BS2000/OSD and when working without BCMAP entries, *utmappl* must directly match the agreed partner name in the `upicfile` or the `APPLICATION_NAME` attribute.



When working without TNS on a Solaris, Linux or Windows host platform, the operand `LISTENER-PORT=number` must also be defined. *number* is the number of the port on which the openUTM application can be reached. If *number* \neq 102, this port number must also be entered on the client in the `upicfile` or the `HOST_PORT` attribute.

Individual access points using PTERM-/LTERM statements

You define an access point for a WTHolder program by means of a pair of PTERM and LTERM statements. You must therefore specify a separate PTERM/LTERM- pair for each parallel connection.

- PTERM statement

```
PTERM upic_client, PTYPE=UPIC-R, LTERM=lterm_name, BCAMAPPL=utmappl,
      PRONAM=client_processor, ...
```

upic_client Name of the client. Under Solaris, Linux and Windows, there must be a TNS entry for *upic_client* which maps *upic_client* to a T selector. If you are working without BCMAP entries - correspond directly to one of the local names declared in the file `localapps`.

lterm_name Name of an LTERM partner (see LTERM statement).

utmappl Local name of the openUTM host application.

client_processor Symbolic name of the client computer. The symbolic name is mapped to the Internet address (e.g. via the Domain Name Service DNS). The PRONAM operand is only obligatory for Solaris, Linux and Windows if *client_processor* is specified in the TNS entry for the remote client (as name part 4). In the case of BS2000/OSD, the PRONAM operand is always obligatory.

Under Solaris, Linux and Windows UPIC is also available in the UPIC-L version (for local connections). Select this if you set PTYPE=UPIC-L in the PTERM statement.

PTERM statement without TNS (Solaris, Linux, Windows)

When working under Solaris, Linux or Windows without TNS, the PTERM statement must be in the following form:

```
PTERM upic_client, PTYPE=UPIC-R, LTERM=lterm_name, BCAMAPPL=utmappl,
      LISTENER-PORT=number, TPROT=RFC1006, PRONAM=client_processor
```

LISTENER-PORT=*number* defines the port number used as outgoing port at the client computer. This port number must also be defined for WebTransactions in the LOCAL_PORT attribute. The PRONAM operand is obligatory in this case.

- LTERM statement

```
LTERM lterm_name, [operands]
```

lterm_name

Name of an LTERM partner (=logical access point for the openUTM application). Any *lterm_name* can be selected. openUTM requires this symbolic name for internal management purposes.

[*operands*]

In the LTERM statement, you can also define properties for this access point by specifying additional optional operands. For example, you can set special access rights.

Example (BS2000/OSD)

```
BCAMAPPL UTMAPPL1, T-PROT=RFC1006
```

```
PTERM CLIENT1, PTYPE=UPIC-R, LTERM=lterm_name1,  
BCAMAPPL=UTMAPPL1, PRONAM=HOST002, ...
```

```
LTERM lterm_name1 [operands]
```

```
PTERM CLIENT2, PTYPE=UPIC-R, LTERM=lterm_name2,  
BCAMAPPL=UTMAPPL1, PRONAM=HOST002, ...
```

```
LTERM lterm_name2 [operands]
```

```
PTERM upic_client, PTYPE=UPIC-R, LTERM=lterm_name, BCAMAPPL=utmapp1,  
PRONAM=client_processor, ...
```

```
LTERM lterm_name, [operands]
```

Access point pool using the TPOOL statement

As an alternative to using multiple PTERM and LTERM statements, you can also use the TPOOL statement to define a limited number of access points. In this case, there is no fixed name assignment between the entries created on KDCDEF generation and the TNS entries.

```
TPOOL BCAMAPPL=utmapp1, PTYPE=UPIC-R, LTERM=prefix, NUMBER=1000, PRONAM=*ANY
```

The operand LTERM=*prefix* defines a prefix from which openUTM creates internal LTERM names for the individual access points in the pool in order to differentiate between the WTHolder programs which make parallel access attempts. The internally formed LTERM name is a maximum of eight characters in length and consists of the specified prefix followed by a serial number (e.g. UPIC0001, UPIC0002,....).

Example

```
TPOOL BCAMAPPL=UTMAPPL1, LTERM=CLNT, NUMBER=1000, PTYPE=UPIC-R, PRONAM=*ANY
```

- The BCAMAPPL operand is optional for Solaris, Linux and Windows. It is obligatory for BS2000/OSD. The specified name must be defined in a BCAMAPPL statement.
- LTERM and NUMBER generate permissible LTERM names in the form CLNT0001, CLNT0002, ...
- PTYPE=UPIC-R: the physical partner is a UPIC client (under Solaris and Linux it is also possible to specify UPIC-L)
- PRONAM=*ANY: any computer which knows the partner name (UTMAPPL1) can communicate with this host application via UPIC.

7.3.2 Declaring the client computer

The client and server computers must be declared to one another. There are two ways of doing this:

- Both computers are configured for the DNS (Domain Name Service). In this case, no further entries are necessary.
- Otherwise, the client computer on which WebTransactions is running must be entered at the server computer together with its Internet address:
 - For Solaris, Linux and Windows, in the `hosts` file:

IP address client computer name

example: 123.4.5.6 WTCOMPUTER

- In the case of BS2000/OSD, the symbolic name of the client computer must be entered statically in the BCAM-RDF (resource definition file) or dynamically via BCIN.

BCIN example:

```
/BCIN HOST002, INI=ALL, ACTIVE=ALL, IPA=(123,4,5,6), PROT=(TCP,IP)
```

The Internet address of the client computer is specified by means of the IPA parameter. Commas are used as separators.

7.4 BCPMAP entries (BS2000/OSD)

Global BCPMAP entries are only required in cases in which automatic mapping is not possible (e.g. because the names used are longer than eight characters).

An overview of the relations between the BCPMAP entries which may be necessary is presented on the following page.

- Entries on the WebTransactions side:

Make the following local entries for the WebTransactions application:

```
/BCMAP FU=DEFINE, SU=LOCAL, APPL=(OSI, LOCAL1), TSEL-I=(8, C'CLIENT1'),  
TSEL-N=CLIENT1
```

```
/BCMAP FU=DEFINE, SU=LOCAL, APPL=(OSI, LOCAL2), TSEL-I=(8, C'CLIENT2'),  
TSEL-N=CLIENT2
```

...

Make the following global entries for the remote openUTM application:

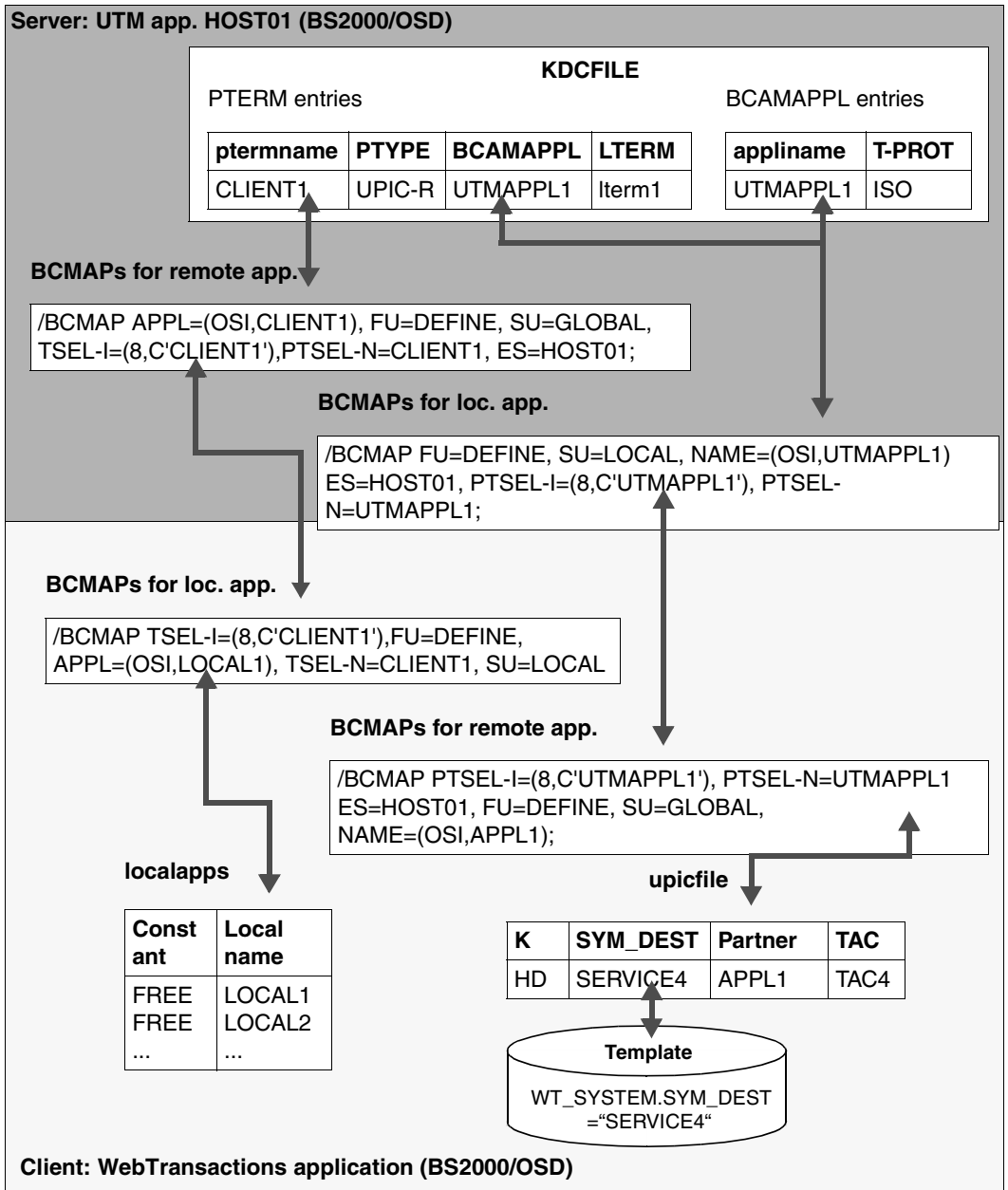
```
/BCMAP FU=DEFINE, SU=GLOBAL, NAME=(OSI, SERVER1), ES=HOST0001,  
PTSEL-I=(8, C'UTMAPPL1'), PTSEL-N=UTMAPPL1
```

This mapping is only necessary if the openUTM application and WebTransactions are not running on the same computer.

The precise BCPMAP command specification may vary depending on the transport system.

- Entries on the host side:

BCMAP entries are only necessary here if default mapping (i.e. names correspond, the transport system transparently forwards the host application name, etc.) fails to find the host application.



8 Controlling communication

8.1 openUTM-specific attributes of the system object

Certain system object attributes can be used to control communication between WebTransactions and the openUTM application.

If there is a `WT_SYSTEM` object (“connection-specific system object”) in the communication object used, then these attributes must be defined in this object. Otherwise they should be declared as attributes of the global system object `WT_SYSTEM`. For basic information on connection-specific and global system objects, refer to the WebTransactions manual “Concepts and Functions”.

When you start WebTransactions you can set these attributes in the first template (start template) and then retain them for the entire session or control them actively during the session (see the WebTransactions manual “Concepts and Functions”, keyword “active dialog”).

Here we describe only those attributes which exist specifically for openUTM or which have a special significance for openUTM. System object attributes which have the same meaning for all WebTransactions product variants are described in the WebTransactions manual “Concepts and Functions”.

8.1.1 Overview

The table below provides an overview of the attributes and their effect.



The system object attributes are divided into the following categories:

- o (**open**)
Attributes that are used with open
- t (**temporary**)
Attributes used during communication which can be modified in the templates at any time
- r (**read only**)
Attributes used during communication which cannot be modified in the templates
- c (**communication module**)
Attributes set automatically by the communication module


The category to which each attribute belongs is indicated in the right-hand column of the table below.

Attribute name	Meaning	Description/category	
APPLICATION_NAME	Name of the openUTM application	This attribute is used to establish communication with the openUTM application. If this attribute is set, the upicfile is ignored.	o
APPLICATION_PREFIX	Prefix for the host application name	This prefix makes it possible to identify FLD and template files which possess the same “format names” but belong to different applications. These FLD and template files must be saved in the following form: <i>application_prefix@formatname.fl</i> d or <i>application_prefix@formatname.htm</i> The host adapter evaluates the attribute when reading the FLD file. Evaluation of the format is template-controlled.	o
BADTAC	Specifies a BADTAC transaction code	If an attempt is made to start a openUTM conversation with an invalid transaction code, a conversation with the transaction code specified in the attribute BADTAC is started automatically (see section “BADTAC - simulating the BADTAC event service” on page 200).	t

Attribute name	Meaning	Description/category	
COMMUNICATION_FILE_NAME	File name for incoming or outgoing data	Using this attribute, a file can be specified to which the data fetched by <code>receive</code> are written or from which the data to be sent are read. Once used, the attribute is reset. The file must be located in the base directory; it is permissible to place it in the sub-directory, <code>wwdocs</code> . Default value: empty, i.e. no file will be used.	t
COMMUNICATION_INTERFACE_VERSION	Interface version, used for compatibility	Global WT_SYSTEM attribute: <ul style="list-style-type: none"> – If this variable contains a value < "3.0", then the name of the host format is entered in the global system attribute <code>FORMAT</code> on reception of a message from the host (<code>receive</code>). – If this variable contains the value "3.0" or higher, then <u>no</u> value is entered for <code>FORMAT</code> since the choice of the next page (format) is made by the templates themselves (generally by evaluating the <code>FLD</code> attribute). Default value: 7.5 	t
CONVERSATION_TAC	Controls conversation connection	With this attribute, a conversation connection is attempted via the openUTM control field or the first 8 Bytes of the format (see section "Automatic conversation chaining" on page 201).	o
CUT_TAC_FIELD	Removes the TAC field from the first message	In the case of *formats and +formats which a program unit sends on the start of a conversation, openUTM removes the transaction code in terminal mode: in the case of *formats the first 8 characters (TAC), and for +formats the first 10 characters (attribute field plus TAC). This behavior is simulated by WebTransactions by default. If the transaction code in the first message of a conversation is not to be removed (e.g. because the openUTM application inserts an Input Exit and the transaction code is not therefore allocated at the start of the message), then this attribute must be set to <code>NO</code> . In all other cases, the transaction code is removed.	t
DISPLAY_EURO	Display Euro symbol	If this attribute is set to <code>Yes</code> then the character that corresponds to code <code>X'A4'</code> of the ISO-8859 code table is output as the Euro character. If <code>DISPLAY_EURO=No</code> (default) then the currency symbol (€) is displayed for <code>X'A4'</code> .	t

Attribute name	Meaning	Description/category
EPILOG	Epilog	<p>This attribute contains the name of a template (without the suffix '.htm'). If the attribute is defined then the corresponding template is included at the end of the generated template. Default: No inclusion</p> <p> The attribute is only evaluated by the generated standard template and not by the host adapter. See also PROLOG and FORMTPL</p>
FLD	Name of the current field file	<p>The value of this attribute is set by the <code>receive</code> call and should not be changed. It points to the field file that contains the structure of the message just received from the host application. The same value is expected by the following <code>send</code> call. If the openUTM application does not supply any format names, FLD is set to the default names from the line mode template <code>wt1nmode.htm</code>. See also FORMAT and APPLICATION_PREFIX</p>
FORMAT_SEQ	Number of a partial format	<p>This attribute is set by the <code>receive</code> call. If a partial format has been received, its number is stored in this attribute: 1,2,3, In the case of the last partial format of a message, the value of this attribute is LAST. If a full format has been received, this attribute contains an empty string.</p>
FORMTPL	Form field	<p>This attribute contains the name of a template (without the suffix '.htm'). If the attribute is defined then the corresponding template is included at the start of the <code>wtDataForm</code> in the generated templates. Default: No inclusion</p> <p> The attribute is only evaluated by the generated standard template and not by the host adapter. See also PROLOG and EPILOG.</p>

Attribute name	Meaning	Description/category	
HOST_CHAR_CODE	Character coding used	This attribute specifies whether communication with the host application is to be carried out in ASCII ("A") or EBCDIC ("E"). You can also specify the name of a file that contains a conversion table, ("T[ABLE]: <i>asciifilename</i> "). This table is searched for under the base directory. A standard version is supplied in <i>/install_dir/lib/ASCII.EBCDIC.G</i> . In the case of FHS + formats or # formats with attribute combination, the conversion in UPIC must be disabled (see <i>upicfile</i> and <i>UPIC_CODE_CONVERSION</i> attribute) and <i>HOST_CHAR_CODE</i> must be used. By default, no value is set for <i>HOST_CHAR_CODE</i> , i.e. no conversion is performed.	t
HOST_IP_ADDRESS	IP address of the computer on which the openUTM application is running	This attribute is used to establish communication with the openUTM application.	o
HOST_NAME	Name of the host computer	This attribute is used to establish communication with the openUTM application.	o
HOST_PORT	Port number of the openUTM application	This attribute is used to establish communication with the openUTM application. It must only be set if the partner application is not reachable on port 102. Default setting: 102	o
LOCAL_APPLICATION	Name used by WebTransactions to log onto the transport system	If a fixed application name is to be used to establish the connection instead of the first free entry in the <i>localapps</i> file, this name must be set in <i>LOCAL_APPLICATION</i> (see section "Targeted logon via specific LTERMs" on page 204). The <i>localapps</i> file is then ignored. Default value: empty	o
LOCAL_PORT	Number of the local port	If this attribute is set, this port will be used by the socket connection as the output port on the local system.	o
PASSWORD	Password	WebTransactions uses this password for user logons to openUTM. This is only useful in combination with the <i>USER</i> attribute and <i>SECURITY_TYPE=PASSWORD</i> .	o

Attribute name	Meaning	Description/category	
NEW_PASSWORD	New password	This attribute can be used to allocate a new password for the openUTM user recognition. This is only useful in combination with the USER attribute and SECURITY_TYPE=PASSWORD. The new password can only be allocated if the old password has expired and provided that the openUTM application has been appropriately generated (grace sign-on). After a subsequent receive call, it should be checked by using the RECEIVE_ERROR attribute whether the password was successfully changed, see also page 156 .	o
PROLOG	Prolog	This attribute contains the name of a template (without the suffix '.htm'). If the attribute is defined then the corresponding template is included at the start of the generated template. Default: No inclusion  The attribute is only evaluated by the generated standard template and not by the host adapter. See also EPILOG and FORMTPL	t
RECEIVE_ERROR	UPIC return code after receive	The UPIC return code is passed in this attribute following a receive call. This permits improved error handling in the template. For possible values, see page 156 .	t
RECEIVE_SECONDARY_INFORMATION	Secondary UPIC return code after receive	The secondary UPIC return code gives information on the precise cause of error. This permits detailed error handling.	t
RESTART	Conversation restart	This attribute specifies a start with or without automatic restart. Possible values: YES, NO. Default value is NO. The value YES is only useful if SECURITY_TYPE≠NONE.	o
RETRY	Retry repeats	If this attribute is set at start time, it specifies the number of times WebTransactions should repeat an attempt to call the open method if it does not succeed - e.g. because of a lack of openUTM resources.	o
SECURITY_TYPE	openUTM security level	This attribute is set on start. The possible values are: – NONE (preset value: neither the user name nor password are used) – USER (logon under user name but without password) – PASSWORD (logon with user name and password)	o
SPECIAL_KEY	Special keys	This attribute can be used to transfer F and K keys to the openUTM application. Possible values are F1-F24 and K1-K14 (K1-K14 only for openUTM on BS2000/OSD)	t

Attribute name	Meaning	Description/category	
SYM_DEST	Symbolic Destination	This attribute must only be set if the parameter for the connection to the openUTM application is read from the <code>upicfile</code> file. In this case, the attribute must be set before the <code>open</code> call. With active dialog control, you must provide the attribute before a renewed <code>open</code> call. <code>SYM_DEST</code> must be exactly 8 characters long. Default setting: undefined, i.e. the connection parameters must be set individually using the appropriate system attributes.	t
TAC	Transaction code	This attribute is used to address the service that is to be called in the openUTM application.	o
TIMEOUT	Maximum waiting time for <code>receive</code>	This attribute controls the maximum time the system waits for a response from the host during a <code>receive</code> . If <code>TIMEOUT</code> is greater than the global system object attribute <code>TIMEOUT_APPLICATION</code> or is not set, a value derived from the <code>TIMEOUT_APPLICATION</code> is used: <ul style="list-style-type: none"> - <code>TIMEOUT_APPLICATION > 10</code>: <code>TIMEOUT</code> is <code>TIMEOUT_APPLICATION - 5</code> - <code>TIMEOUT_APPLICATION > 1</code>: <code>TIMEOUT</code> is <code>TIMEOUT_APPLICATION - 1</code> - <code>TIMEOUT_APPLICATION = 1</code>: <code>TIMEOUT</code> is <code>TIMEOUT_APPLICATION</code> The value must be specified in complete seconds.	t
UPIC_CODE_CONVERSION	Controls the UPIC code conversion	If this attribute is set to "Yes", code conversion (ASCII - EBCDIC) will be forced. In this case, the <code>HOST_CHAR_CODE</code> attribute must not be set. "No" means no conversion. This is required for FHS <code>+format</code> or <code>#format</code> with attribute combination. In this case, <code>HOST_CHAR_CODE_</code> must be set, see page 149 . Default setting: "Yes"	o
UPIC_LIB	Library name including path of the dynamically linked UPIC library (only relevant for Unix systems with UPIC-L)	This attribute supplies the complete file name of the dynamically linked UPIC library. It is evaluated each time a link is established to the openUTM application provided that the value is not an empty string.	o
UPIC_TRACE	Switch for enabling UPIC trace (does not apply for Windows)	This attribute is evaluated when the connection is established. If the value of this attribute is not 'empty string', the UPIC trace is enabled and stored in the directory, <code>BASEDIR/tmp/SESSION</code> . On the BS2000 system, the trace is stored in the same place that the <code>upicfile</code> must be stored, namely under the ID under which the Web server is started (see page 73).	o

Attribute name	Meaning	Description/category	
USER	User name	WebTransactions uses the user name specified here for user logon to openUTM. This is only of use together with SECURITY_TYPE≠NONE.	o
UTM_PATH	Path under which openUTM is installed; (only relevant for Unix platforms when using UPIC-L)	This attribute is evaluated each time a connection is established to the openUTM application if a library for UPIC-L is specified in UPIC_LIB.	o

8.1.2 Interaction between system object attributes and actions/methods

This section provides information on which openUTM-specific attributes of the communication-specific system object are relevant for which method calls.

Establishing a connection - open

A call to the `open` method opens a connection to the openUTM application. The following attributes of the system object control the `open` call:

Attribute	Meaning
APPLICATION_PREFIX	Prefix for the host application name. This prefix makes it possible to identify FLD files which possess the same "format names" but belong to different host applications. These FLD files must be saved in the following form: <i>application_prefix@formatname.fld</i>
APPLICATION_NAME	Name of the openUTM application. If this attribute is set, the <code>upicfile</code> (SYM_DEST entry) is ignored.
HOST_NAME	Name of the computer on which the openUTM application is running
HOST_IP_ADDRESS	IP address of the computer on which the openUTM application is running. If this attribute is set, any name set by means of HOST_NAME is ignored.
HOST_PORT	Port number of the openUTM application; this attribute must only be set if the openUTM application is not reachable on port 102. Default setting: 102
LOCAL_PORT	Number of the local port
TAC	Transaction code of the service that is to be called in the openUTM application.
UPIC_CODE_CONVERSION	If this attribute is set to "Yes", code conversion (ASCII - EBCDIC) will be forced. "No" means no conversion.
SYM_DEST	Specifies the symbolic destination name from the file, <code>upicfile</code> . This attribute must only be set if the service of the openUTM application is not addressed using the APPLICATION_NAME, HOST_NAME...,TAC attribute named above. The addressing parameters required must then be stored in the <code>upicfile</code> . The symbolic destination name is 8 characters long.
SECURITY_TYPE	Security level: NONE, USER or PASSWORD (see page 195)
USER	openUTM user name (see page 195)
PASSWORD	openUTM user password (see page 196)
NEW_PASSWORD	New openUTM user password (see page 196)
RESTART	Restart: Yes or NO (see page 197)

Attribute	Meaning
ERROR	Error message. If an error occurs in the execution of <code>open</code> , a corresponding message is passed to the system object attribute <code>ERROR</code> . If the call is successful, this attribute remains empty.
RECEIVE_ERROR	If <code>SECURITY_TYPE</code> \neq <code>NONE</code> but the <code>USER</code> attribute is empty, an error message (<code>CM_SECURITY_NOT_VALID</code>) is passed to this attribute as soon as an attempt is made to establish a connection.
RECEIVE_SECONDARY_INFORMATION	Secondary UPIC return code, giving information on the precise cause of error.
LOCAL_APPLICATION	Logon via fixed host application name.

In addition, all attributes marked “o” in the table in [section “Overview” on page 146](#) are evaluated on connection setup.

Sending data - send

A call to the `send` method sends a message to the openUTM application. The following system object attributes are evaluated or set:

Attribute	Meaning
FLD	Name of the field file. WebTransactions uses the <code>FLD</code> attribute to determine the corresponding field file for the interpretation of the data sent to the openUTM application. The value of the attribute is generally set correctly by an earlier <code>receive</code> call. The current host data objects are now sent as a message to the openUTM application.
ERROR	Error message. If an error occurs in the execution of <code>send</code> , a corresponding message is passed to the system object attribute, <code>ERROR</code> . If the call is successful, this attribute remains empty.
COMMUNICATION_FILE_NAME	Name of the file whose data is to be sent. The file must be located in the base directory; it can be stored in the sub-directory, <code>wwdocs</code> .

Receiving data - receive

The call to the `receive` method receives a message from the openUTM application and sets the following system object attributes:

Attribute	Meaning
FLD	Name of the received format / partial format. The corresponding field file receives the same name with the extension, <code>.fld</code> . WebTransactions uses this file to determine the structure of the received data. Following a <code>receive</code> call, the data can be accessed in the form of host data objects.
FORMAT_SEQ	Serial number of the received partial format (starting with 1). On the last partial format, <code>FORMAT_SEQ</code> is assigned the value, <code>LAST</code> . When partial formats are used, each <code>receive</code> call receives one partial format. You must therefore issue multiple calls in order to receive all the partial formats (see section "Special characteristics of FHS/FORMANT partial formats" on page 104). In the case of full formats, this attribute is not relevant and contains
COMMUNICATION_FILE_NAME	Name of the file to be used in place of the host data objects for <code>send</code> or <code>receive</code> . The file must be located in the base directory; it can be stored in the sub-directory, <code>wwdocs</code> .
ERROR	Error message. If an error occurs in the execution of <code>receive</code> , a corresponding message is passed to the system object attribute, <code>ERROR</code> . If the call is successful, this attribute remains empty.
RECEIVE_ERROR	UPIC return code. The UPIC return code is passed in this attribute following a <code>receive</code> call. This permits improved error handling in the template (possible values, see following section).
RECEIVE_SECONDARY_INFORMATION	Secondary UPIC return code, giving information on the precise cause of error.

UPIC return codes after receive

The `RECEIVE_ERROR` attribute passes the UPIC return code following a `receive` call. The following values are possible:

`CM_OK`

The call was successful

`CM_SECURITY_NOT_VALID`

Possible causes:

- invalid openUTM user name
- invalid password
- the openUTM application was generated without `USER`
- the user cannot log onto the openUTM application because of a resource bottleneck

In the `RECEIVE_SECONDARY_INFORMATION` attribute is output the precise error reason, see [page 157](#).

`CM_TPN_NOT_RECOGNIZED`

Possible causes:

- Invalid transaction code (TAC) in the `upicfile`, e.g.:
 - TAC not generated
 - no authorization to call this TAC
 - TAC only permitted as follow-up TAC
 - TAC is not a dialog TAC
- Conversation restart rejected since no openUTM user name has been generated with `RESTART=YES`.

`CM_TP_NOT_AVAILABLE_NO_RETRY`

Conversation restart is not possible since the openUTM application has been regenerated.

`CM_TP_NOT_AVAILABLE_RETRY`

Conversation restart was rejected since the openUTM application is being terminated.

`CM_DEALLOCATED_ABEND`

Possible causes:

- Abnormal termination of the openUTM conversation
- End of openUTM application
- Disconnection by openUTM administration
- Disconnection by the transport system
- Disconnection by openUTM because maximum permitted number of users exceeded (`MAX statement, CONN-USERS=`). The cause may also lie in the fact that an administrator `USER` has been passed even though the user assigned in the `LTERM ...USER= statement` is not an administrator `USER`.

CM_DEALLOCATED_NORMAL

A PENDING call has been executed in the openUTM conversation.

CM_RESOURCE_FAILURE_RETRY

A temporary resource bottleneck has resulted in the termination of the conversation. Possibly no more data can be buffered in the openUTM page pool.

CM_RESOURCE_FAILURE_NO_RETRY

An error has occurred which resulted in the premature termination of the conversation (e.g. a protocol error or premature loss of the network connection).

CM_MAP_ROUTINE_ERROR

Only occurs with openUTM-V3.4 applications which WebTransactions accesses with `PROTOCOL=UTM_V4`.

CM_PROTOCOL_ERR

Only occurs with openUTM-V4.0 applications which WebTransactions accesses with `PROTOCOL=UTM`.

CM_UNKNOWN_ERR

Programming error.

CM_PROGRAM_PARAMETER_CHECK

Programming error.

Secondary UPIC return codes in RECEIVE_SECONDARY_INFORMATION

After a receive call with `RECEIVE_ERROR=CM_SECURITY_NOT_VALID`, the secondary UPIC return code is passed to the `RECEIVE_SECONDARY_INFORMATION` attribute. The following values are possible:

CM_SECURITY_USER_UNKNOWN

The specified user name is not generated in the openUTM application.

CM_SECURITY_STA_OFF

The specified user name is locked out at present.

CM_SECURITY_USER_IS_WORKING

The specified user name is currently being used by another user.

CM_SECURITY_OLD_PASSWORD_WRONG

The old password entered is incorrect (when changing a password).

CM_SECURITY_NEW_PASSWORD_WRONG

The new password entered does not match (e.g. error on repeating password).

CM_SECURITY_PASSWORD_EXPIRED_NO_RETRY

The password has expired and can no longer be changed by the user. Please inform the administrator of the openUTM application.

CM_SECURITY_PASSWORD_EXPIRED_RETRY

The password has expired but can still be changed by the user. After changing the password, the user can log onto the openUTM application again.

CM_SECURITY_COMPLEXITY_ERROR

The new password is too simple. Enter a more complex password, e.g. with special characters, digits and fewer repeated characters.

CM_SECURITY_PASSWORD_TOO_SHORT

The new password is too short.

CM_SECURITY_UPD_PASSWORD_WRONG

The changed password is too simple. Enter a more complex password.

CM_SECURITY_TA_RECOVERY

The user must restart the transaction.

CM_SECURITY_PROTOCOL_CHANGED

The LTERM may not continue the open transaction

CM_SECURITY_SHUT_WARN

Warning from the administrator: The openUTM application is about to be shut down. Please log off immediately.

CM_SECURITY_ENC_LEVEL_TOO_HIGH

Data encryption error: the encryption mechanism required to continue the open conversation is not available on the connection.

CM_SECURITY_NO_CARD_READER

No card reader was found.

CM_SECURITY_CARD_INFO_WRONG

The data on the card are incorrect.

CM_SECURITY_NO_RESOURCES

No more resources available

CM_SECURITY_TAC_KEY_MISSING

The LTERM may not continue the open conversation

8.2 Host objects and attributes

Two types of host objects are involved in UPIC-based communications between WebTransactions and the openUTM application:

- Host data objects

These are objects which correspond to the fields in a format:

`WT_HOST.handle.fieldname` (see [section “Host objects for the individual format fields \(host data objects\)” on page 159](#))

For binary transmission, there is a separate host data object to represent the data.

- Host control objects:

These are objects which manage global data and have reserved names:

`WT_HOST.handle.WT_HOST_MESSAGE` (see [section “Host control object WT_HOST_MESSAGE” on page 165](#))

`WT_HOST.handle.WT_HOST_GLOBALS` (see [section “Host control object WT_HOST_GLOBALS” on page 169](#))

`WT_HOST.handle.$FIRST` and `WT_HOST.handle.$NEXT` (see [section “Host control objects \\$FIRST and \\$NEXT” on page 170](#))

If necessary, WebTransactions also creates a connection-specific `WT_SYSTEM` object under the root `WT_HOST.handle` in which the openUTM-specific system object attributes are created. This is described in [section “openUTM-specific attributes of the system object” on page 145](#).

8.2.1 Host objects for the individual format fields (host data objects)

Host data objects correspond to the fields of a format. Their names are determined from the field file (see [page 85](#)).

In the case of # formats, the attribute names correspond to the FHS attribute names. In the case of + formats, WebTransactions maps the attribute bits to the attribute names. If, for example, the bit for “Blinking” is set in a + format, then WebTransactions sets `VISIBILITY=S` for the corresponding host data object.

For these host data object attributes, the following types of access are possible: read (r) or read/write (w).

Unlike other objects and attributes, the attribute names of host objects are not case-sensitive and no distinction is made between upper and lower case.

Attribute	Description	Access
<i>Attributes for all host data objects</i>		
Value	Field contents in 8-bit characters (see page 163)	w
HTMLValue	Field contents in 7-bit characters in HTML notation (see page 163)	r
RawValue	Field contents as a sequence of 8-bit characters which are not converted, apart from binary zeros which are converted to blanks (see page 163)	w
HexStringValue	Field content in the form of printable half-bytes (see page 163).	w
<i>Static attributes from the IFG format definition, valid for all host objects</i>		
Align	Static field attribute <code>Align</code> (used in class templates)	r
AutoInput	Static field attribute <code>AutoInput</code> (Y / N) (used in class templates)	r
Blink	Static field attribute <code>Blink</code> (used in class templates)	r
Case	Static field attribute <code>Case</code> (used in class templates)	r
DataType	Static field attribute <code>DataType</code> (used in class templates)	r
DefaultCursor	Static field attribute <code>DefaultCursor</code>	r
FloatSign	Static field attribute <code>FloatSign</code>	r
GroupDigit	Static field attribute <code>GroupDigit</code>	r
IOType	Static field attribute <code>IOType</code> (used in class templates)	r
Length	Static field attribute <code>Length</code> (used in class templates)	r
Name	Name of the field, e.g. in order to address the same field again after <code>\$NEXT</code>	r
NumDecimals	Static field attribute <code>NumDecimals</code>	r
Signed	Static field attribute <code>Signed</code>	r
StartColumn	Static field attribute <code>Column</code> : column in which the format field begins	r
StartLine	Static field attribute <code>Line</code> : line in which the format field begins	r
SuppressZero	Static field attribute <code>SuppressZero</code>	r

Attribute	Description	Access
Unicode	<p>Static field attribute Unicode (Y / N)</p> <p>Y (YES): The host data in the UPIC buffer is converted to and from UTF-8 for forwarding to and from the browser for all fields marked Unicode == 'Y'. No ASCII-to-EBCDIC conversion is performed.</p> <p>N (NO): The host data is not converted to UTF-8. ASCII-to-EBCDIC conversion is carried out in dependence on the system object attribute HOST_CHAR_CODE.</p>	r
<i>Linking of static and dynamic attributes (# and + formats only)</i>		
Blinking	Result of the static field attribute (see FLD file, Section <fieldname>, "Blink" and possibly the dynamic field attribute (Visibility = S). Possible values: Y (Yes), N (No)	r
Detectable	Result of the static field attribute (see FLD file, <Fieldname> section, " Detectable " on page 87) and if necessary of the dynamic field attribute (Protection=D): Y (Yes), N (No)	r
Mandatory	Result of the static field attribute (see FLD file, <Fieldname> section, " Mandatory " on page 88) and if necessary of the dynamic field attribute (Input Control=M): Y (Yes), N (No)	r
Protected	Result of the static field attribute (see FLD file, <Fieldname> section, " Protection " on page 88) and if necessary of the dynamic field attribute (Protection=U/D): Y (Yes), N (No)	r
Underlined	Result of the static field attribute (see FLD file, <Fieldname> section, " Underline " on page 88) and if necessary of the dynamic field attribute (Underline=Y/N): Y (Yes), N (No)	r
Visible	Result of the static field attribute (see FLD file, <Fieldname> section, " Visibility " on page 88) and if necessary of the dynamic field attribute (Visibility=Y/N): Y (Yes), N (No)	r
<i>Dynamic attributes for # formats if selected via the IFG profile</i>		
Color	Dynamic field attribute Color, possible values: 1=red, 2=green, 3=yellow, 4=blue, 5=magenta, 6=cyan, 7=white, N=no color	w
Cursor	Dynamic field attribute Cursor, possible values: Y (Yes), N (No), H (Hold)	w
EditState	Dynamic field attribute EditState, possible values: V (Valid), I (Invalid), M (Must error), ' ' (blank means 'not checked')	w
InputControl	Dynamic field attribute InputControl, possible values: N (Normal), M (Must), P (Potmust), A (Autoret)	w

Attribute	Description	Access
InputState	Dynamic field attribute InputState, possible values: M (Modified), C (Cleared), D (Detected), U (Undefined), ' ' (blank means "not touched")	w
InputStateAct	Dynamic field attribute InputStateAct, see InputState	w
Intensity	Dynamic field attribute Intensity, possible values: N (Normal), H (High), D (Dark)	w
Inverse	Dynamic field attribute Inverse, possible values: Y (Yes), N (No)	w
OutputControl	Dynamic field attribute OutputControl, possible values: I (Init), D (Data), U (Undefined)	w
Protection	Dynamic field attribute Protection, possible values: A (Askip), P (Protected), U (Unprotected), D (Detectable)	w
Underline	Dynamic field attribute Underline, possible values: Y (Yes), N (No)	w
Visibility	Dynamic field attribute Visibility, possible values: V (Visible), S (Signaling), I (Invisible)	w
<i>Dynamic attributes which are sometimes mapped for *- and + formats in the same way as for # formats</i>		
InputState	Dynamic field attribute InputState, possible values: M (Modified), D (Detected), ' ' (blanks mean 'not touched')	w
<i>Dynamic attributes which are sometimes mapped for # formats with the field attribute group 'Attribute combination' and for + formats in the same way as for # formats. Write access is ignored and no message is output.</i>		
Cursor	Dynamic field attribute Cursor, possible values: Y (Yes), N (No) (mapped from Cursor position in this field)	r
InputControl	Dynamic field attribute InputControl, possible values: N (Normal), A (Autoret) (mapped from autoret)	r
Intensity	Dynamic field attribute Intensity, possible values: N (Normal), H (High), D (Dark) (mapped from bright, normal, dark)	r
Inverse	Dynamic field attribute Inverse, possible values: Y (Yes), N (No) (mapped from inverse)	r
Protection	Dynamic field attribute Protection, possible values: A (Askip), P (Protected), U (Unprotected), D (Detectable) (mapped from protected, unprotected, detectable)	r
Underline	Dynamic field attribute Underline, possible values: Y (Yes), N (No) (mapped from underline (italic))	r

Attribute	Description	Access
Visibility	Dynamic field attribute <code>Visibility</code> , possible values: V (Visible), S (Signaling), I (Invisible) (mapped from <code>blinking</code> , <code>dark</code>)	r

Attributes Value, HTMLValue, RawValue and HexStringValue

There are three options for accessing the contents of a data field. In all cases, zeros (binary zero, \0) are converted into blanks:

- as the "Value" attribute:

The contents of the data field are returned without appended blanks. Double quotes, single quotes and ampersands (&) are replaced by the corresponding HTML hex notation (&#nn;). All other characters (including umlauts) are output as 8-bit characters, i.e. in the form in which they are output and understood by the host application. This attribute is therefore suitable for presetting values in HTML tags of type "Input".

You can also write to this attribute, if the corresponding field is not write protected.

In the case of host objects whose `IOType` is `INPUT` or `OUTPUT`, the corresponding value is read from the message buffer. If `IOType` is `TEXT` or `FIXTEXT`, the text defined with the IFG is transferred.

- as the "HTMLValue" attribute:

The contents of the data field are returned untruncated. Double quotes, single quotes and ampersands (&) are replaced by the corresponding HTML hex notation (&#nn;). In addition, certain special characters are converted for subsequent output in HTML: <, >, ä, ö, ü, Ä, Ö, Ü, ß. This attribute is suitable for constant HTML body texts. It is a read-only attribute.

- as the "RawValue" attribute:

The contents of the data field are returned unchanged (apart from the conversion of binary zeros into blanks).

- as attribute "HexStringValue":

The content of the data field is returned in the form of printable half-Bytes. In this way, binary data and control characters can be made readable.

Example

hello<CR>world (<CR> = Windows carriage return) is represented as 68656B6B6F0A776F726B64 (ISO 8859 coding)

Unicode support

Operations on strings

Since the WebTransactions kernel itself does not support Unicode, you should take care when applying operations to strings.

String operations can return incorrect results if the string contains UTF-8 characters. The `length` attribute does not return the number of characters, but rather the number of bytes. When comparing and manipulating such strings, you must take account of the representation of the UTF-8 characters.

Such operations do not occur in the templates generated by WebTransactions.

Diagnostics –

- Unicode characters are represented by replacement characters in traces and in online displays of host objects.
- In WebLab, host objects can only be overwritten by 8-bit characters.

Setting data contents (with # formats)

When you set data contents, the attributes `Inputstate` and `InputStateAct` as well as the global attribute `FieldsMod` are set to `M` (modified). `EditState` is preset to `V`.

IObjectType attribute - class templates

You can evaluate the data fields in accordance with their `IObjectType` class. Host data objects at the UPIC interface belong to the class `INPUT` (text boxes/output fields), to the class `OUTPUT`, or to the class `TEXT` or `FIXTEXT` (text fields). For instance, if you address a class template with the name `INPUT.c1t` via the evaluation operator, all `INPUT` fields of this class template are evaluated. A class template `OUTPUT.c1t` is used for the evaluation of all `OUTPUT` fields. Further information on class templates can be found in the WebTransactions manual “Template Language”.

8.2.2 Host control object WT_HOST_MESSAGE

In addition to the host data objects for the individual fields, an additional host object in the form of the host control object WT_HOST_MESSAGE is always available for UPIC connections to openUTM. This is used to manage global data. The object's attributes can be accessed as follows: read (r) or read and write (w).

Unlike other objects and attributes, the attribute names of host objects are not case-sensitive, i.e. no distinction is made between upper and lower case.

Attribute	Contents	Format type	Access
BackgroundColor	Background color	#	r
Contents	Contents of the entire host message	all	w
Content_<offset>_<length>	Content of the UPIC buffer	all	r
CursorControl	Global attribute CursorControl	#	r
CursorField	Position of cursor (for details see next page)	all	w
CursorPosition	Global attribute CursorPosition	#	r
DateFormat	Static field attribute DateFormat	all	w
DecimalSeparator	Static field attribute DecimalSeparator	all	r
DigitSeparator	Static field attribute DigitSeparator	all	r
FieldsDetect	Global attribute FieldsDetect	#	r
FieldsMod	Global attribute FieldsMod	#	w
FieldsValid	Global attribute FieldsValid	#	w
FormatLength	Length of message	all	w
FormatName	Name of format	all	r
FormattingSystem	Formatting system: FHS, FORMANT	all	r
FormatType	Type of format: #, +, *	all	r
Hex_Content_<offset>_<length>	Content of the UPIC buffer	all	r
InputKeyClass	Global attribute InputKeyClass	#	r
InputKeyNumber	Global attribute InputKeyNumber	#	r
Level_Selection	Global attribute Level_Selection	#	r
P_Key_Set	Global attribute P_Key_Set	#	r
TimeFormat	Static field attribute TimeFormat	all	w
UserexitRc	Global attribute Userexit Rc	#	r
UndefinedValues	Global attribute Undefined Values	#	w

Attribute	Contents	Format type	Access
Unicode	Indicates that at least one field where Unicode == 'Y' is contained in the current message. For details see page 168 .	all	r
Version	Version of converter	all	r

WT_HOST_MESSAGE provides read access to a number of attributes which describe the properties of the format. In the case of # formats, you can use the WT_HOST_MESSAGE object to obtain read and write access to all of the format's global attributes.

The Contents attribute

You can use the Contents attribute to address the entire host message in order to store this in a template or system object attribute before being subsequently copied back. This functionality can, for example, be extremely useful when processing partial formats (see [section "Special characteristics of FHS/FORMANT partial formats" on page 104](#)).

The Content_<offset>_<length> attribute

Makes the contents of the UPIC buffer available with no preparation as of <offset> at the length <length>. This means that if the buffer contains binary zeros, the returned string terminates at the first binary zero.

The CursorField attribute

This WT_HOST_MESSAGE attribute supplies the name of the format field in which the cursor is located:

- In the case of # formats, the Cursor field attribute is considered if WT_HOST_MESSAGE.CursorControl has the value F (Field Cursor) or R (Relative Cursor). In this case, WT_HOST_MESSAGE.CursorField contains the name of the first unprotected field whose Cursor field attribute has the value Y or H. If the Cursor field attribute of this field has the value Y, it is implicitly set to blanks before the format is sent.

If WT_HOST_MESSAGE.CursorControl has a value other than {F/R} or if none of the Cursor field attributes is set in an unprotected field then the system responds as follows: if a field is assigned the DefaultCursor attribute, then WT_HOST_MESSAGE.CursorField contains the name of this field. Otherwise, it contains the name of the first unprotected input field.

- In the case of + formats, `WT_HOST_MESSAGE.CursorField` contains the name of the first unprotected field whose `Cursor` field attribute is set. If no such field exists but a field has been assigned the `DefaultCursor` attribute, then `WT_HOST_MESSAGE.CursorField` contains the name of this field. Otherwise, it contains the name of the first unprotected input field.
- In the case of * formats, `WT_HOST_MESSAGE.CursorField` contains the name of the field which is assigned the `DefaultCursor` attribute. Otherwise, it contains the name of the first unprotected input field.
- In the case of partial formats, a blank string is returned if the field containing the cursor is not located in the current partial format.



WebTransactions provides the `wtBrowserFunctions.htm` template for setting the cursor. This is included in the template generation as standard.

The `Hex_Content_<offset>_<length>` attribute

See the section [“The `Content_<offset>_<length>` attribute”](#) on page 166.

The contents of the buffer are, however, converted to a string, i.e. each byte is represented by two characters.

Example

Hexadecimal in the UPIC buffer 000102030405...

`Hex_Content_0_3` returns "000102" as character string.

The `Level_Selection` attribute

The `Level_Selection` attribute usually does not have any significance for openUTM dialogs with WebTransactions. In conjunction with the `P_Key_Set` attribute, the value "P" can be evaluated as "P_Key_Set is valid", see the manual [“Format-Handling System for UTM, TIAM, DCAM”](#).

The `P_Key_Set` attribute

Indicates which definitions of programmable keys are to be loaded for terminal operation via FHS.

- If the openUTM application is being operated via UPIC with WebTransactions, this attribute has no direct effect, because a terminal emulation is not participating.
- When you are mapping programmable keys with client-side Java script, you use the attribute with WebTransactions.

See the manual [“Format-Handling System for UTM, TIAM, DCAM”](#).

Attribut Unicode

If the attribute `Unicode` is assigned to the host control object `WT_HOST_MESSAGE`, this indicates that at least one field where `Unicode == 'Y'` is contained in the current message.

The global system object attribute `CHARSET` can be set to the value `UTF-8` depending on this attribute. In this event, the field `Content-Type` thus has the correct contents in the HTTP header and the browser can interpret the data correctly.

All templates that WebTransactions generates for this host adapter automatically include this assignment. This assignment must be manually inserted into existing templates which are not to be re-generated.

For further information refer to section [“Unicode support” on page 126](#).

8.2.3 Host control object WT_HOST_GLOBALS

WT_HOST_GLOBALS is a host object for the control of communication with the host.

This object is only relevant for + formats and * formats. All its attributes can be overwritten. You should set the values of these attributes once in the start template after a call to the open method.

For information on the possible and preset values of these attributes, refer to the FHS manual “[Format-Handling System for UTM, TIAM, DCAM](#)”, chapter “FHS for openUTM Users”, section “Start Parameter” (however, WT_HOST_GLOBALS does not contain a corresponding attribute for all the openUTM start parameters described in this manual):

Attribute	Description
Padding Asterisk, Padding PlusAttr, Padding PlusData	<p>These specify the character used to fill a field before changes are entered. The default setting is OUTMSG (the field content as sent by the host application). Any value that differs from this must be entered as a decimal value or as a hexadecimal value ('#xx'). This value corresponds to the start parameter, PADDING. For Unix platforms and Windows platforms, it should be noted that the value may be converted on transmission to the openUTM application (see system object attributes, UPIC_CODE_CONVERSION and HOST_CHAR_CODE, or the conversion identifier in the upicfile).</p> <p>PaddingAsterisk affects the data fields of *formats PaddingPlusData affects the data fields of +formats PaddingPlusAttr affects the attribute fields of +formats These three attributes replace the Padding attribute. Padding continues to be supported for compatibility. If Padding is set, all of the three new attributes are assigned the transferred value. The meaning is basically the same.</p> <p>In the FHS start parameters, there is only a distinction between the formats + and *: <i>Example</i> .FHS PADDING=(FORM*=' ',FORM=X'00')</p> <p>The additional distinction in WebTransactions can be used to, for example, configure situations in which the attribute fields of the + formats are filled with binary zeros by FHS, and the data field of an FHS format exit is filled with SPACE.</p>
Cursor	<p>Specifies the method used to position the cursor. Possible values: A: Cursor set using attribute fields (ATTR), default value N: Cursor set using KDCSCUR (NATTR)</p>
Detect	<p>Specifies the character used to fill a field that is selected. Default value is 255. The value can be derived from the start parameter, MAPDET. For Unix platforms and Windows platforms, it should be noted that the value may be converted on transmission to the openUTM application (see system object attributes, UPIC_CODE_CONVERSION and HOST_CHAR_CODE, or the conversion identifier in the upicfile).</p>

Attribute	Description
UnDetect	Specifies the character used to fill a field which is not selected. Default value is 0. The value can be derived from the start parameter, MAPDET. For Unix platforms and Windows platforms, it should be noted that the value may be converted on transmission to the openUTM application (see system object attributes, UPIC_CODE_CONVERSION and HOST_CHAR_CODE, or the conversion identifier in the upicfile).
Read	Specifies the mode in which the host application expects the data on a read operation. Possible values: M (Modified) or U (Unprotected). The value can be derived from the start parameter ISTD.
FieldLength	Specifies the values to be set for the fields' attribute field/length field before a message is sent to the host. Possible values: E (Effective length), D (Defined length), N (Not modified). The value can be derived from the start parameter EFFLEN.
Update	<ul style="list-style-type: none"> – If the attribute has the value 0 (for ONLY), then, if possible, the value is taken over from the previous message for all fields whose contents are binary zero. 0 is the default value. – In the value is P (for PSTN), then such fields are filled with output fill characters. Restriction: Mode 0 does not function with partial formats.

8.2.4 Host control objects \$FIRST and \$NEXT

These two objects are used to edit host objects.

Object name	Attribute	Meaning
\$FIRST	Name	Full name of the first field in the current format. If no such field exists, the name \$END is returned.
		plus all attributes of dynamic host data objects (see section “Host objects for the individual format fields (host data objects)” on page 159)
\$NEXT	Name	Full name of the next field in the current format starting from the last field accessed. This object enables you to work step-by-step through all fields in the format. If no such field exists, the name \$END is returned.
		plus all attributes of dynamic host data objects (see section “Host objects for the individual format fields (host data objects)” on page 159).

8.3 Terminal functions supported by the browser

In WebTransactions for openUTM you can display host application formats in the browser without any post-editing (1:1 conversion). In order to be able to use the terminal functions, you need to use the master template `UTM.wmt`. The templates that you have generated via the master template include the templates `wtBrowserFunctions.htm` and `wtKeysUTM.htm` which provide the functions required.

`wtBrowserFunctions.htm` in turn includes the following Javascript files:

`wtCommonBrowserFunctions.js`

contains the Javascript code that will be run for all browsers.

`wt<browser>BrowserFunctions.js`

contains the Javascript code for the current browser.

`wtKeysUTM.htm` contains the openUTM-specific buttons for the standard keys and include the Javascript files `wtKeysUTMFHS.js` and `wtKeysUTMFormant.js` which contain the FHS-/Formant-specific special key mapping for WebTransactions for openUTM. In these files you can adapt the key mapping to your needs and also extend it (see [section “Mapping keys in wtKeysUTMFHS.js and wtKeysUTMFormant.js” on page 174](#)).

8.3.1 Terminal functions supported

The following terminal functions are provided:

- Pixel-precise layout of text and entry fields with the help of style sheets.
- Support for terminal special keys sent to WebTransactions. For some of these keys there are equivalents on the PC keyboard (e.g. the “F” keys). In some cases key combinations are be used to start terminal functions.
- Support for terminal special keys which work directly in the browser form (e.g. cursor positioning keys). For some of these keys there are equivalents on the PC keyboard. In other cases, terminal functions are started using key combinations.
- Autotab:
When the maximum length of an entry field is reached the cursor automatically moves to the next entry field.

- **Overwriting fields:**
Like a terminal, the browser overwrites the characters already present in the entry field and does not insert text between the characters as per the browser default settings.
- **Transfer of the cursor position from the browser to the host application:**
Depending on the browser functions available, the browser transfers the exact cursor position or only the corresponding entry field to WebTransactions.
- **Tabulator remains inside the form:**
The entry focus does not leave the form generated by WebTransactions. Using the tabulator key in the browser also automatically moves the focus onto the browsers controls.

Which of the terminal functions (F keys, cursor positioning keys ...) is actually displayed on the browser will depend on the type and version of the browser. The tables below show the terminal functions supported by the various browser types.

Terminal function	Browser support		
	Non specialized browser	Netscape V6.0 or higher or Mozilla Firefox	Internet Explorer V4.0 or higher
Layout of text and entry fields	no	yes	yes
Support for terminal special keys sent to WebTransactions	only via a pick list or button	by individual configurable mapping via keys or pick lists or buttons	
Support for terminal special keys which work directly in the browser form	no	by individual configurable mapping via a key	
Autotab	no	yes	yes
Overwriting fields	yes (simulated in the browser by automatic selection of field content)		yes
Transmits the cursor position	Only the position at the start of the last entry field used	Position at the start of the last entry field used and the exact position in protected fields.	Exact position in protected fields and in entry fields (V5.0 or higher)
Tabulator remains inside the form	no	yes	

Key support by Internet Explorer V4.0 or higher or by a Gecko-based browser

If you use Internet Explorer V4.0 or higher or a Gecko-based browser (Netscape V6 or higher), then the browser terminal keys are mapped as follows:

Key¹ used in the browser	Corresponding key at the terminal²
ENTER	ENTER
F1 ... F12	F1 ... F12
Shift+F1 ... Shift+F12	F13 ... F24
STRG+F1 ... STRG+F12	K1 ... K12
STRG+Shift+F1 STRG+Shift+F2	K13 K14
STRG+Shift+F12	MAR
INS	INS

¹ Here, '+' means that the keys specified must be pressed together at the same time. On some keyboards the STRG key is marked with CTRL.

² K keys are only possible with openUTM applications on BS2000/OSD.

8.3.2 Mapping keys in wtKeysUTMFHS.js and wtKeysUTMFormant.js

The browser used will accept all keyboard entries. For the application-defined mapping of special function keys, WebTransactions provides an interface with the files `wtKeysUTMFHS.js` (for FHS formats) and `wtKeysUTMFormant.js` (for Formant formats). You can use this interface, no special knowledge of browser templates is required.

After creation of the base directory, these two files are in the directory `<basedir>/wwwdocs/javascript`. The interface to be used for WebTransactions application mapping of the keys is the table (array) `wtKeyMappingTableInput` given in the corresponding file.

The `wtKeyMappingTableInput` table defines an object with several attributes for each of the key maps (see table in [page 175](#)). These attributes describe:

- the key or key combination
- the action to be triggered when the key (or combination) is pressed
- if this function is also available on a selection list.

Example

```
wtKeyMappingTableInput = [
  { sl:'title of my select list'},
  { la:'Select', ac:doToggleMark, kc:VK_F12, mk:MK_CTRL+MK_SHIFT },
  { la:'F1', ac:'F1' },
  { la:'K1', 'ac':'K1', kc:VK_F1, mk:MK_CTRL }
];
```

In this definition the mapping is as follows:

- CTRL+SHIFT+F12 calls the function `doToggleMark()`
- CTRL+F1 sends the function code `K1` to WebTransactions

This definition creates a selection list with the following content:

title of my select list	no function
Select	calls up the function <code>doToggleMark()</code>
F1	sends the function code to WebTransactions (F1 or K1)
K1	

In the `wtKeyMappingTableInput` table you can enter the following attributes:

Description	Attribute	Meaning
la	label	Label, e.g. for entry in the selection list. If the attribute is not specified, no entry will be generated for the list. The corresponding key will, however, be mapped on a function.
co	comment	Comment. This attribute is not evaluated. This attribute has been provided as an alternative to the attribute <code>la</code> ; by changing the attribute from <code>la</code> to <code>co</code> you can, for example, remove a key from the selection list.
ac	action	<p>Action to be executed when the mapped key is pressed or when the action is selected from a list.</p> <p>If this attribute is a <code>string</code> type, the content will be transferred to <code>wt_special_key.value</code> and sent to WebTransactions. This means that the form is transferred to WebTransactions and as a special function is given the value of <code>ac</code> (e.g. "@1" for the PF1 key).</p> <p>If this attribute is a <code>function</code> type, a client-side function with this name will be called. This function must be defined.</p> <p>The Javascript files <code>wt<browser>BrowserFunctions.js</code> provide the following functions:</p> <ul style="list-style-type: none"> – doCursorHome – doCursorUp – doCursorDown – doCursorLeft – doCursorRight – doTab – doBackTab – doToggleMark – doToggleInsert. <p>The implementation of these functions can be empty; this depends on the browser capabilities (see the section “Callback functions in key mapping” on page 180).</p> <p>If this attribute is not defined, no action can be executed. Editing of the keyboard entries is left to the browser.</p>
kc	key code	Number assigned to the pressed key in the keyboard driver. The script <code>wtCommonBrowserFunctions.js</code> has a symbol for many of the keys; the symbol name begins with <code>VK_</code> . For key combinations there is also the modifier key (<code>mk</code>).

Description	Attribute	Meaning
mk	modifier key	<p>Additional modifier key pressed (see definition in <code>wtCommonBrowserFunctions.js</code>):</p> <ul style="list-style-type: none"> - 0 = MK_NONE (= no modifier key pressed) - 1 = MK_CTRL - 2 = MK_ALT - 4 = MK_SHIFT <p>In key combinations the corresponding values are added:</p> <ul style="list-style-type: none"> - 3 = MK_CTRL + MK_ALT - 5 = MK_CTRL + MK_SHIFT - etc. <p>If no <code>mk</code> is specified then the value 0 = MK_NONE is used.</p>
s1	select list	<p>At the start of each selection list to be generated, a component with the index 0 will be generated as a header. The component has no function. The text for this "0" component is specified in the attribute <code>s1</code>.</p> <p>Any selection lists created previously will be closed when the attribute <code>s1</code> occurs.</p> <p>For improved readability, <code>s1</code> can be made to be the only attribute in the table object.</p>

Structure of wtKeysUTMFHS.js and wtKeysUTMFormant.js

The following section describes the `wtKeyMappingTableInput` table from the `wtKeysUTMFHS.js` and the `wtKeysUTMFormant.js` file supplied with WebTransactions:

The object `wtKeyMappingTableInput` is created as literal.

```
wtKeyMappingTableInput = [
```

The attribute `s1` indicates the start of the selection list with the label `more`.

```
{ s1:'more'},
```

The attribute `co` indicates a comment for better readability. There is no `la` attribute for the following entries. The entries should not appear in the selection list. Use `kc` and `mk` to find the mapping for a PC key. With `ac` JavaScript functions are specified which are to be processed when the appropriate key or key combination is pressed.

```
{ co:'MAR', ac:doToggleMark, kc:VK_F12, mk:MK_CTRL+MK_SHIFT },
{ co:'MAR', ac:doToggleMark, kc:VK_MAR, mk:0 },
{ co:'Insert', ac:doToggleInsert, kc:VK_INS },
{ co:'CursorUP', ac:doCursorUp, kc:VK_UP },
{ co:'CursorDOWN', ac:doCursorDown, kc:VK_DOWN },
{ co:'CursorLEFT', ac:doCursorLeft, kc:VK_LEFT },
{ co:'CursorRIGHT', ac:doCursorRight, kc:VK_RIGHT },
{ co:'HOME', ac:doCursorHome, kc:VK_HOME },
{ co:'TAB', ac:doTab, kc:VK_TAB },
{ co:'BACKTAB', ac:doBackTab, kc:VK_TAB, mk:MK_SHIFT },
```

The function keys appear in the selection list. With FHS, these are the keys K1 to K14 and F1 to F24 (see below), with Formant only the keys F1 to F24.

```
{ la:'K1', ac:'K1', kc:VK_F1, mk:MK_CTRL },
{ la:'K2', ac:'K2', kc:VK_F2, mk:MK_CTRL },
{ la:'K3', ac:'K3', kc:VK_F3, mk:MK_CTRL },
{ la:'K4', ac:'K4', kc:VK_F4, mk:MK_CTRL },
{ la:'K5', ac:'K5', kc:VK_F5, mk:MK_CTRL },
{ la:'K6', ac:'K6', kc:VK_F6, mk:MK_CTRL },
{ la:'K7', ac:'K7', kc:VK_F7, mk:MK_CTRL },
{ la:'K8', ac:'K8', kc:VK_F8, mk:MK_CTRL },
{ la:'K9', ac:'K9', kc:VK_F9, mk:MK_CTRL },
{ la:'K10', ac:'K10', kc:VK_F10, mk:MK_CTRL },
{ la:'K11', ac:'K11', kc:VK_F11, mk:MK_CTRL },
{ la:'K12', ac:'K12', kc:VK_F12, mk:MK_CTRL },

{ la:'K13', ac:'K13', kc:VK_F1, mk:MK_CTRL+MK_SHIFT },
```

```

{ la:'K14', ac:'K14', kc:VK_F2, mk:MK_CTRL+MK_SHIFT },

{ la:'F1', ac:'F1', kc:VK_F1, mk:0 },
{ la:'F2', ac:'F2', kc:VK_F2 },
{ la:'F3', ac:'F3', kc:VK_F3 },
{ la:'F4', ac:'F4', kc:VK_F4 },
{ la:'F5', ac:'F5', kc:VK_F5 },
{ la:'F6', ac:'F6', kc:VK_F6 },
{ la:'F7', ac:'F7', kc:VK_F7 },
{ la:'F8', ac:'F8', kc:VK_F8 },
{ la:'F9', ac:'F9', kc:VK_F9 },
{ la:'F10', ac:'F10', kc:VK_F10 },
{ la:'F11', ac:'F11', kc:VK_F11 },
{ la:'F12', ac:'F12', kc:VK_F12 },

{ la:'F13', ac:'F13', kc:VK_F1, mk:MK_SHIFT },
{ la:'F14', ac:'F14', kc:VK_F2, mk:MK_SHIFT },
{ la:'F15', ac:'F15', kc:VK_F3, mk:MK_SHIFT },
{ la:'F16', ac:'F16', kc:VK_F4, mk:MK_SHIFT },
{ la:'F17', ac:'F17', kc:VK_F5, mk:MK_SHIFT },
{ la:'F18', ac:'F18', kc:VK_F6, mk:MK_SHIFT },
{ la:'F19', ac:'F19', kc:VK_F7, mk:MK_SHIFT },
{ la:'F20', ac:'F20', kc:VK_F8, mk:MK_SHIFT },
{ la:'F21', ac:'F21', kc:VK_F9, mk:MK_SHIFT },
{ la:'F22', ac:'F22', kc:VK_F10, mk:MK_SHIFT },
{ la:'F23', ac:'F23', kc:VK_F11, mk:MK_SHIFT },
{ la:'F24', ac:'F24', kc:VK_F12, mk:MK_SHIFT },

```

Do not close the last entry in the table with a comma. If you do close the entry with a comma, the program will wait for a further entry before the literal end (]).

```

{ la:'InsClip', ac:doInsertClipboard, kc:VK_V, mk:MK_CTRL+MK_SHIFT }
];
// END_OF_KEY_TABLE

```

8.3.3 Interaction between wtCommonBrowserFunctions.js and wt<browser>BrowserFunctions.js

The file `wtCommonBrowserFunctions.js` contains the Javascript code which will be run for all browsers. The `wt<browser>BrowserFunctions.js` files contain the Javascript code which will be run depending on the current browser. For example, `wtGeckoBrowserFunctions.js` contains the Javascript code for Gecko-based browsers.

After creation of the base directory, the files are located in the directory `<basedir>/wwwdocs/javascript`.

If you want to adapt the Javascript code in these files you will need specialist knowledge of browser behavior and browser interaction with WebTransactions. The following text describes the interaction between the functions and data structures as supplied with the product.

Symbols

The file `wtCommonBrowserFunctions.js` will be called before the files `wt<browser>BrowserFunctions.js` and `wtKeysUTMFHS.js` and `wtKeysUTMFormant.js`. This file contains the definition of the variables for symbolically invoking the keys in the other *.js files.

```
// some symbolic keycodes //////////
VK_TAB    = 9;
VK_RETURN= 13;
VK_SHIFT  = 16;
VK_CTRL   = 17;
VK_ALT    = 18;
VK_PAUSE  = 19;
VK_ESC    = 27;
VK_PGUP   = 33;
VK_PGDN   = 34;
VK_END    = 35;
VK_HOME   = 36;
VK_LEFT   = 37;
VK_UP     = 38;
VK_RIGHT  = 39;
VK_DOWN   = 40;
VK_INS    = 45;
VK_O      = 48;
VK_1      = 49;
...
MK_NONE   = 0;
MK_CTRL   = 1;
MK_ALT    = 2;
MK_SHIFT  = 4;
```

Key mapping functions

```
function wtCreateKeyMap()
```

Generates, from the `wtKeyMappingTableInput` table, a structure which is simpler and quicker to access at runtime. The call is made from `wtKeysUTM.htm`. The call is absolutely necessary; without this call, mapping cannot take place.

```
function wtCreateKeySelectList()
```

Generates, from the `wtKeyMappingTableInput` table, one or more selection lists. The call is made from `wtKeysUTM.htm`. It is possible to suppress the list by leaving out the call for this function in `wtKeysUTM.htm`; the function keys will remain operative.

Following this example it is easy to describe other functions. You can, for example, generate a key or a table component for each function.

```
function wtHandleKeyboard( modifier, keyCode )
```

Called from `wt<browser>BrowserFunctions.js` when a key is pressed. `wtHandleKeyboard()`, on the basis of the structure generated by `wtCreateKeyMap()`, can now establish if an action has been assigned to this key: If an action has been assigned, it will be run. If no action has been assigned, the keyboard event will be left to the browser.

Callback functions in key mapping

The file `wtCommonBrowserFunctions.js` also provides functions used by the table `wtKeyMappingTableInput` (see the attribute `ac` on [page 175](#)).

Most of these functions return `false` as a result in order to indicate that no general mapping for these keyboard entries is available. In this case you should use the default behavior of the current browser. This default behavior will be uploaded to the `wt<browser>BrowserFunctions` files where required by functions with the same names (see [page 182](#)).

Procedure

The behavior described above is obtained as follows:

1. Whenever a PC key is pressed, the browser calls the function `onKeyDown` from the file `wt<browser>BrowserFunctions.js`.
2. The function `onKeyDown` transmits the `modifier` key and the `key` code (see the `wtKeyMappingTableInput` table on [page 175](#)), and then calls the function `wtHandleKeyboard` (if this is present) in the file `wtCommonBrowserFunctions.js`.
3. The function `wtHandleKeyboard` recognizes if an action is defined for this key in the table `wtKeyMappingTableInput` under `ac` (see [page 175](#)).

If there is a function pointer under `ac`, the procedure continues as follows:

4. `wtHandleKeyboard` calls the function and then returns the callback value at `onKeyDown`.
This occurs with actions such as `HOME`, `TAB` or `CursorDown`. The callback function is used at this stage to process actions on the client PC with the aid of the browser.
5. The function `onKeyDown` signals to the browser that the key has just been pressed (callback value `true`). In this case the browser will no longer react to the key. If this is not the case, the browser will run its standard reaction for the current keyboard entry.

If there is a character string under `ac`, the procedure continues as follows:

6. The content of the `string` is transferred to the `SPECIAL_KEY` attribute (see [page 150](#)). The form is transferred to WebTransactions together with the value of `ac` (e.g. `F1` as function key) as a special function.
7. In this case the callback value to the browser is always `true` (the key is processed immediately. The browser no longer reacts to the key).

If the attribute `ac` is not defined (i.e. no action has been assigned to the key pressed), the callback value `false` will be signalled to indicate that the browser will handle the keyboard entry.

WebTransactions-specific callback functions

WebTransactions provides a series of special implementations of the callback functions designed for individual browser types.

Some of the following functions are uploaded by `wtGeckoBrowserFunctions.js` depending on the capabilities of the Gecko browser. `wtExplorerBrowserFunctions.js` will upload all these functions (most of the possibilities are recognized by Internet Explorer) and then run the functions described below.



You can also develop customized callback functions in order to extend the user interface. In this case, you should ensure that a function invoked in the `tablewtKeyMappingTableInput` (see [page 175](#)) is also defined in the file `wtCommonBrowserFunctions.js` and in the corresponding file `wt<browser>BrowserFunctions.js`.

```
function doCursorUp()
```

Positions the cursor in the entry field above the current cursor position.

```
function doCursorDown()
```

Positions the cursor in the entry field below the current cursor position.

```
function doCursorLeft()
```

If the cursor is at the start of an entry field, moves the cursor to the end of the previous entry field. Otherwise, the browser will react to the key entry (moving the cursor inside the field).

```
function doCursorRight()
```

If the cursor is at the end of an entry field, moves the cursor to the start of the next entry field. Otherwise, the browser will react to the key entry (moving the cursor inside the field).

```
function doCursorHome()
```

Positions the cursor at the start of the first entry field.

```
function doTab()
```

Skips to the start of the next entry field.

```
function doBackTab()
```

Skips to the start of the previous entry field.

```
function doToggleMark()
```

The marking of the entry field where the focus is located, is toggled.

```
function doToggleInsert()
```

Toggles between the Insert and Overwrite modes.

8.3.4 Using the WT_BROWSER object

In order to avoid having to transmit the browser properties and font sizes many times during a session, WebTransactions creates the object `WT_BROWSER` at the beginning of each session. This object is then available globally throughout the entire session.

The `WT_BROWSER` object contains the following attributes:

- Browser ID
- Browser version
- Browser properties
- Font size to be used

These attributes are used in the following templates:

- All templates generated with the master templates `UTM.wmt`.
- `wtBrowserFunctions.htm`
`wtBrowserFunctions.htm` includes `wt<browser>BrowserFunctions.js` and gives the font size (and other properties).

Font size in the attribute `WT_BROWSER.charSize`

In the `WT_BROWSER` object the attribute `WT_BROWSER.charSize` has the default setting 14 (previous static value).

If the attribute `WT_POSTED.wtCharSize` already exists at the start of a session then its value will automatically be taken over by `WT_BROWSER.charSize`. This feature makes it possible for individual users to set their own font sizes (depending on the screen resolution setting).

The value of `WT_BROWSER.charSize` can also be set while a session is running by using the method `WT_BROWSER.setCharSize`.



You should not try to edit the attribute `WT_BROWSER.charSize` directly because other attributes depend on this value.

You can re-initialize the object `WT_BROWSER` using the method `WT_BROWSER.refresh`. This will also refresh the attributes `WT_SYSTEM.CGI.HTTP_USER_AGENT` and `WT_POSTED.wtCharSize`. This procedure would make sense, for example, when a running session in a roaming session is taken over by another browser (for details on roaming sessions, see the WebTransactions manual “Concepts and Functions”).

Example application of WT_BROWSER.charSize

Allows a user on the call page of a WebTransactions application to select the font size to be used for displaying the application (e.g. via a selection list):

```
Font Size:
<select name="wtcharSize">
  <option value="12">12
  <option value="14" SELECTED>14
  <option value="17">17
  <option value="20">20
</select>
```

At the start of the session, these entries will automatically be taken over when the attribute WT_POSTED.wtCharSize is evaluated. All the size settings in the generated templates will depend on this value.

You can also use Javascript to make the entry field for wtcharSize dependent on the screen width. You can do this, for example, when you call up a page via a Submit button with an entry field for the font size:

```
<body onload="document.forms.wtaform.wtCharSize.value =
  Math.round(screen.width/75)">
<form method="post" name="wtaform"
  action="/scripts/WTPublish.exe/D:/webta/basedir?Start">
<input type="submit" value="Start">
Font Size:
<input type="text" name="wtCharSize">
...
</form>
</body>
```


8.4 Start templates for openUTM

Once a WebTransactions application has been started (via an entry page or the direct specification of a URL) the parameters for the connection with the host application must be set in a start template.

WebTransactions provides you with ready-to-run start templates which you can use as a basis for the development of your own start templates. There are two different possibilities:

- **Start template set**

This start template set is ready for immediate use. The necessary parameters are re-entered each time the application is started, though most are preset to appropriate values. It is suitable both for the start of an individual host application and for the start of multiple host applications that are integrated in a WebTransactions application. The set consists of the general start template, `wtstart.htm`, which enables you, for example, to create communication objects and to switch between different parallel host connections, and of specific start templates for the individual host adapters. The start template, `wtstartUMTV4.htm` is supplied especially for “WebTransactions for openUTM”. This openUTM-specific start template is presented in [section “The openUTM-specific start template in the start template set \(wtstartUMTV4.htm\)” on page 186](#). For an illustration of the general start template, refer to the WebTransactions manual “Concepts and Functions”.

- **WTBean for the generation of a start template**

To connect an individual openUTM application, you should use a specially generated start template. The WTBean `wtcStartUTM.wtc` helps you to generate such templates.


8.4.1 The openUTM-specific start template in the start template set (wtstartUTMV4.htm)

If you have selected the UTMV4 protocol and created a new communication object in the general start template `wtstart.htm` (described in the WebTransactions manual “Concepts and Functions”), then processing switches to the template `wtstartUTMV4.htm`. This template makes it possible to set the openUTM-specific parameters in two successive steps:

1. In the first step, you can define the connection parameters and open a connection to a openUTM application.
2. Once this connection has been opened, the template is redisplayed. It now possesses additional input possibilities (`WT_HOST_GLOBALS` settings) and contains buttons for communications with the openUTM application.

Both steps (and therefore both pages) are implemented internally by means of the same template, namely `wtstartUTMV4.htm`. In either case, an IF structure ensures that processing is switched to the correct template.

Step 1: Setting the connection parameters and opening the connection

 UTMV4 communication	
status	communication object: WT_HOST.UTM_0
	connection: down
workflow	destination: main menu <input type="button" value="go to"/>
	access host: <input type="button" value="open"/> <input type="button" value="open in linemode"/>
	parameters: <input type="button" value="update"/> <input type="button" value="reset"/>
connection parameters directly	APPLICATION_NAME: VTV10TRV
	HOST: <input type="text" value="NAME: HOST001"/> or IP_ADDRESS: <input type="text"/> with HOST_PORT: <input type="text"/>
	TAC: MMENUE
	UPIC_CODE_CONVERSION: <input checked="" type="checkbox"/>
... or via upicfile	SYM_DEST: <input type="text"/>
additional connection parameters	APPLICATION_PREFIX: <input type="checkbox"/>
	CONVERSATION_TAC: <input type="text"/>
	CUT_TAC_FIELD: <input checked="" type="checkbox"/>
	HOST_CHAR_CODE: ASCII <input type="text"/>
	DISPLAY_EURO: <input type="checkbox"/>
	FLD: TRAV0 <input type="text"/>
	BADTAC: <input type="text"/>
	LOCAL_APPLICATION: <input type="text"/>
	UPIC_TRACE: <input type="checkbox"/>
	UPIC_LIB: <input type="text"/>
	UTM_PATH: <input type="text"/>
	SECURITY_TYPE: NONE <input type="text"/>
	USER: <input type="text"/>
	PASSWORD: <input type="text"/>
NEW_PASSWORD: <input type="text"/>	
RESTART: <input type="checkbox"/>	

In the **connection parameters directly** section the following attributes with which the service in the openUTM application is addressed can be defined:

APPLICATION_NAME

Name of the openUTM application

HOST

Name (NAME) or IP address (IP_ADDRESS) with port number (HOST_PORT) of the computer on which the openUTM application is running.

TAC

Transaction code of the service that is to be called in the openUTM application.

UPIC_CODE_CONVERSION

If this option is selected, a code conversion (ASCII-EBCDIC) is forced. In this case the option HOST_CHAR_CODE may not be set.

Alternatively, in the section, ... or via **upicfile**, you can enter:

SYM_DEST

Here a value can be selected that specifies the desired service and is stored in the SYM_DEST attribute. The selection is automatically determined from the contents of the upicfile.

In the **additional connection parameters** section further optional parameters can be entered:

APPLICATION_PREFIX

If this option is selected, the APPLICATION_PREFIX attribute is set to the name of the communication object.

CONVERSATION_TAC, CUT_TAC_FIELD

Sets the attribute of the same name.

HOST_CHAR_CODE

Here you must specify whether messages are to be exchanged with the server in **ASCII** or **EBCDIC** code. If **TABLE** is selected, then you must specify the name of your conversion table in the right-hand text box. The contents of your selection are stored in the attribute **HOST_CHAR_CODE**.

This option must not be used in conjunction with **UPIC_CODE_CONVERSION**.

FLD

Here you can select one of the installed field files. The selection is stored in the FLD attribute and is subsequently important if communications start with the sending of a message.

BADTAC, LOCAL_APPLICATION, UPIC_TRACE, UPIC_LIB, UTM_PATH, SECURITY_TYPE, USER, PASSWORD, NEW_PASSWORD, RESTART

set the attributes of the same names.

The **workflow** section allows you to determine the next action.

destination

Here you can select the template with which you wish to continue. Click on **go to** to go to the selected page. **main menu** is proposed by default: This makes it possible to return to the general entry page `wtstart.htm`. If multiple connections are open, they are also proposed as possible selections; processing then branches to the appropriate host adapter-specific start templates for this connection.

access host

The actions which can currently be performed in the session are proposed here. If no connection has as yet been opened, **open** and **open in linemode** are available. These buttons can be used to open the connection.

For more details about **open in linemode**, please refer to [section “Support for openUTM line mode” on page 120](#).

parameters

reset is used to reset all the parameters to the state they were in when received by the browser.

update is used to send the page values to WebTransactions without initiating communications with the host. This may, for example, be of use, if the `FLD` field was not set prior to the establishment of the connection and **enter dialog** is not therefore possible. If a value is then entered for `FLD`, the **enter dialog** button becomes visible once **update** has been clicked.

Step 2: Beginning communication and setting host attributes

As soon as a connection is opened, the following buttons for communication with the host application are provided in the **workflow** area under **access host**. In addition, a new section, **global host attributes**, is available in which you can specify the attributes `Padding`, `Detect`, `Undetect`, `Read`, `FieldLength`, `Update` and `Cursor` of the `WT_HOST_GLOBALS` host control object.

If you selected **open** in step 1, the buttons **send**, **receive**, **close** and **enter dialog** (as the case may be) are available:

receive / send

The buttons **receive** and **send** are shown alternately.

receive retrieves the next message from the host application and extends the start template for setting the host attributes. **send** sends a message to the host application. If you wish to send a screen containing modified data to the host application, you should select **enter dialog**.

close

This button closes the connection to the host application and returns to the page displayed in step 1. There you can select and open a new connection.

enter dialog

This button branches directly to the next host application screen. You can then complete this screen and send it to the host application.

If you return to this page from an active host application by selecting the **suspend** button, the **resume dialog** button appears in place of the **enter dialog** button.

Starting a conversation which expects neither data nor a basic format

If the first program unit of a conversation requires no data and does not expect any basic format, and instead outputs the first screen itself, then you can start the conversation as follows:

First page: Select the conversation and define the message coding (via `connection parameters directly` or by `SYM_DEST` and/or `HOST_CHAR_CODE`), any other start parameters, select **open**.

Second page: If necessary, modify the attributes of `WT_HOST_GLOBALS`, select `receive`. The page is then redisplayed. Now select **enter dialog**.

Starting a conversation which expects data or a basic format

If the first program unit of a conversation requires defined data or expects a basic format, then you can start the first conversation as follows:

First page: Select the conversation, define the message coding (via `connection parameters directly` or by `SYM_DEST` and/or `HOST_CHAR_CODE`), determine the message (`FLD`) and any other start parameters, select **open**.

Second page: If necessary, modify the attributes of `WT_HOST_GLOBALS`, Select **enter dialog**.

8.4.2 WtBean `wtcStartUPIC.wtc` for the generation of a start template

To connect an individual openUTM application you can generate an application-specific start template. To do this, you use the WtBean `wtcStartUPIC.wtc`. This is a standalone WtBean.

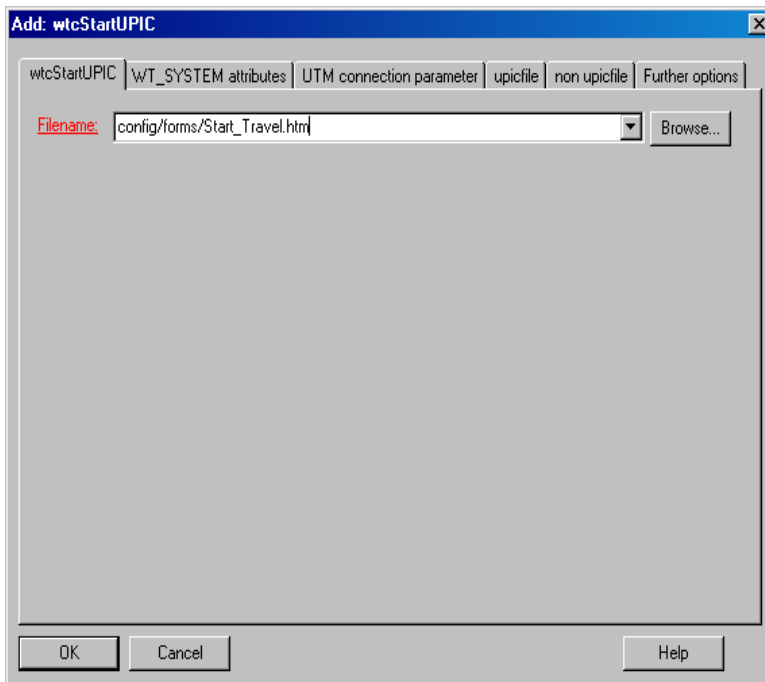
`wtcStartUPIC.wtc` contains the inline WtBean `wtcUPIC.wtc` which is used to create a new openUTM communication object and thus establish a connection to an openUTM application (see the section “[Creating a new openUTM communication object \(wtcUPIC\)](#)” on [page 193](#)).



Before you can access WtBeans there must be a connection to a WebTransactions application.

You use the **File/New/wtcStartUPIC** command to open the WtBean for editing. WebLab uses the source file to generate the **Add:wtcStartUPIC** dialog box which contains six tabs:

- In the **wtcStartUPIC** tab you specify the name of the start template you want to generate.
- In the **WT_SYSTEM attributes** tab you specify the most important system object attributes.
- In the tabs **UTM connection parameter**, **upicfile** and **non upicfile** you specify the most important connection parameters.
- In the **Further options** tab you can edit all the parameters for the connection to the openUTM application within a tree structure.



The generated start template itself does not generate any pages in the browser. When WebTransactions is started, the template corresponding to the first format received from the host application is displayed. This is due to the `wtinclude` tag at the end of the start template.

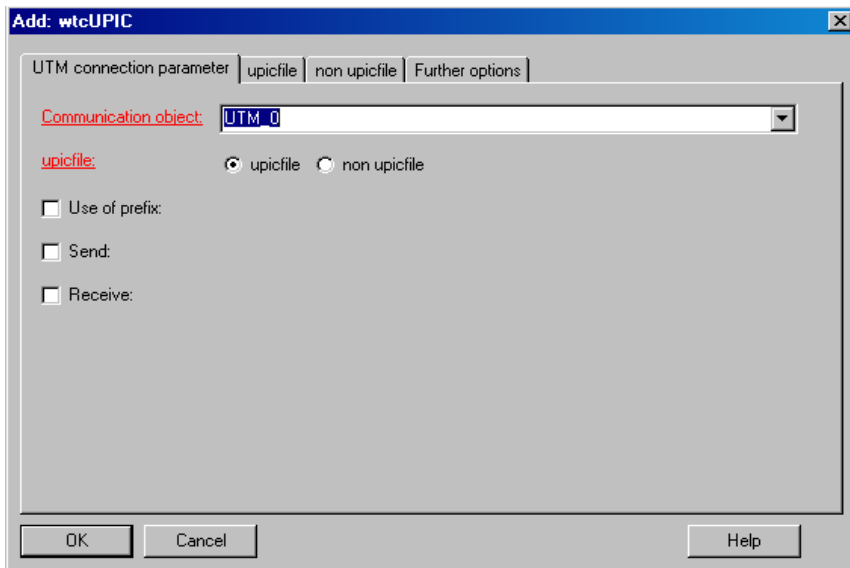
8.5 Creating a new openUTM communication object (wtcUPIC)

The WTBean `wtcUTM` is supplied in order to enable you to create a new openUTM communication object in a template and thus establish a connection to an openUTM application. You can also use this WTBean to open multiple connections in parallel. `wtcUTM` is an inline WTBean. For more information, refer to the WebTransactions manual “Concepts and Functions”.



Before you can access inline WTBeans, there must be a connection to the WebTransactions application and the template in which you want to insert the WTBean must be open.

You use the **Add/WTBean/wtcUPIC** command to open the WTBean for editing. WebLab generates the **Add:wtcUPIC** dialog box:



In this dialog box you can edit the parameters for the new communication object.

- ▶ On the tab **UTM connection parameters**, you must enter the name of the communication object and select, whether the connection parameters are to be read from the upicfile (**upicfile** option) or entered directly (**non upicfile** option).
- ▶ Depending on the selection, either the **upicfile** tab or the **non upicfile** tab is to be filled in.
- ▶ You can edit all the other parameters in a tree structure in the **Further options** tab.

Once you have entered values for the parameters and clicked **OK** to confirm your settings, the code of the WTBean is generated from the parameters and the description file and is inserted at the cursor position in the opened template.

The WTBean is made up of protected and unprotected code areas. The protected areas are grayed out. These areas can only be accessed via the interface of the WTBean. In the context menu of the start line of the WTBean (pink), select the **Edit WTBean** command (see the WebTransactions manual “Concepts and Functions”).

8.6 Security through the openUTM user concept

WebTransactions allows you to employ the openUTM user concept. This means that you can use openUTM's system and data access control mechanisms even when performing Web accesses. You can also use scalable authorizations by implementing openUTM's lock/keycode concept.

Logging on under an openUTM user name

For the use of the openUTM user concept, there are special system object attributes, `SECURITY_TYPE`, `USER`, `PASSWORD` and `NEW_PASSWORD`. With each `open` call, WebTransactions checks the `SECURITY_TYPE` attribute and, if required, returns `USER`, `PASSWORD` and `NEW_PASSWORD`.



A security check is therefore performed on each conversation start.

`SECURITY_TYPE`

Specifies the security level. Possible values:

`NONE` On logon, WebTransactions does not pass the name of a openUTM user name or a openUTM password to the openUTM application. An empty string in `SECURITY_TYPE` has the same effect.

`USER` The name of a user name stored in the `USER` attribute is passed to the openUTM application on logon. However, no password is passed.

`PASSWORD`

When logging on to the openUTM application, the name of a user name stored in `USER` is used together with the password stored in `PASSWORD`.

`USER` Name of the openUTM user name which may sometimes be passed to the openUTM application for an authorization check.

`PASSWORD`

Password of the transferred openUTM user name (permissible characters as in openUTM, i.e. only printing characters are permitted).

`NEW_PASSWORD`

New password for the openUTM user name returned (character set as for openUTM, i.e. only printable characters are permitted).



The event service SIGNON is supported as of UPIC V5.0 in conjunction with openUTM V5.0 or later.

However, with openUTM applications < V5.0 the SIGNON service cannot be used via WebTransactions. If you connect terminal host applications which use the SIGNON event service to the Web, you must integrate the SIGNON processing steps in a separate program unit and call this from the start template.

Invalid user name or invalid password

If, when logging on, an invalid user name or invalid password is used, WebTransactions sets the `RECEIVE_ERROR` attribute after the first `receive` with the value `CM_SECURITY_NOT_VALID`. The error can then be handled individually in the template by evaluating the attribute, `RECEIVE_SECONDARY_INFORMATION`. The values of this attribute are described on [page 157](#).

If the `USER` attribute is empty but `SECURITY_TYPE` is set to `USER` or `PASSWORD`, WebTransactions sets the `RECEIVE_ERROR` attribute to `CM_SECURITY_NOT_VALID` as soon as a connection is established (`open`).

Change password

The password for an openUTM user name can be changed by setting the `NEW_PASSWORD` attribute in addition to the `PASSWORD` attribute. In `PASSWORD` you must enter the existing password and in `NEW_PASSWORD` the new password.

If the openUTM application is generated with grace sign-on, the password can still be changed after it has expired.

Logon with no restart information requested

If restart information for the employed user name is available at logon time even though no such information has been requested for this user name, then WebTransactions determines whether or not a message was sent to the openUTM application in the previously rolled back transaction:

- If **no** attempt has been made to send a message (`open + receive`) in the reset transaction, WebTransactions automatically reopens the connection, discarding the restart information.
- If an attempt has been made to send a message (`open + send + receive`) in the reset transaction, WebTransactions sets the `RECEIVE_ERROR` attribute to `CM_DEALLOCATED_ABEND`. The error can now be handled individually in the template.

8.7 RESTART - automatic restart

In the case of openUTM applications based on an openUTM version V3.4 or higher (for UPIC under BS2000/OSD: V4.0 or higher), the openUTM restart functions are also available in client/server environments and can therefore be used by WebTransactions.

“Restart” means that a openUTM conversation which has been rolled back to the last synchronization point because of an error or malfunction (e.g. loss of connection due to a reboot) can be continued as of this point when the client logs on again.

Since openUTM always performs user-specific back-ups of conversation contexts, a restart is not possible unless you are working with user names (`SECURITY_TYPE=USER/PASSWORD`). The `RESTART` attribute must also be set to `YES`.

Restart on start

If the `WT_SYSTEM.RESTART` attribute is set to `YES` in the WebTransactions application start template and a user name is specified, WebTransactions first internally uses the restart transaction code `KDCDISP` for an automatic restart without regard to `WT_SYSTEM.TAC` or `WT_SYSTEM.SYM_DEST` attribute. Only if this fails the system continues with the transaction code specified by the `WT_SYSTEM.TAC` attribute (direct entry) or by `WT_SYSTEM.SYM_DEST` in the `upicfile`.

If `WT_SYSTEM.RESTART` does not have the value `YES` or if no user name is given, WebTransactions starts immediately with the transaction code specified in `WT_SYSTEM.TAC` or `WT_SYSTEM.SYM_DEST`.

Reaction if no restart is possible

It is often not possible to perform a restart. The way in which WebTransactions reacts to this type of situation depends on the cause and on whether an attempt was made to send a message to the openUTM application in the rolled back transaction.

- If no restart is possible because the openUTM application has been regenerated since the connection was lost or because the user was previously logged on via a terminal:
 - If **no** attempt has been made to send a message (only `open + receive`) in the reset transaction, the system will start with the transaction code that is identified in the `TAC` attribute or by the `SYM_DEST` attribute in the `upicfile`.
 - If an attempt has been made to send a message (`open + send + receive`) in the reset transaction, WebTransactions sets the `RECEIVE_ERROR` attribute to `CM_TP_NOT_AVAILABLE_NO_RETRY`. The error can now be handled individually in the template.

- If no restart is possible because `RESTART=NO` is set for this user name in the openUTM configuration, then WebTransactions reacts in exactly the same way as described above (only difference: different `RECEIVE_ERROR` value):
 - If **no** attempt has been made to send a message (only `open + receive`) in the reset transaction, the system will start with the transaction code that is identified in the `TAC` attribute or by the `SYM_DEST` attribute in the `upicfile`.
 - If an attempt has been made to send a message (`open + send + receive`) in the reset transaction, WebTransactions sets the `RECEIVE_ERROR` attribute to `CM_TPN_NOT_RECOGNIZED`.
The error can now be handled individually in the template.

Reaction if restart reports end of conversation (only for connections via UPIC)

If the restart returns a conversation end, WebTransactions writes the value `CONVERSATION_END` in the system attribute `RESTART`. Only after the next `receive` call is `RESTART` reset to empty string.

This makes it possible to recognize a restart in the “logoff screen” and process it accordingly in the corresponding template.

Example

The following example shows a typical template for a restart logon using the `upicfile`.

```
<wtIf (host_system.RESTART == "CONVERSATION_END")>
```

```
<wtOnCreateScript> (1)
```

```
  host_system.CONVERSATION_TAC = "SYM_DEST";
  host_system.SYM_DEST = "MAINENTR";
```

```
  host.open();
```

```
  host.receive(); (2)
  host.system.CONVERSATION_TAC = "";
```

```
</wtOnCreateScript>
```

```
<wtInclude name="##wt_system.FORMAT#>
```

```
<wtElse>
```

```
<html> (3)
<head>
<title>TRSGNOF</title>
</head>
  bye bye ...
<body bgColor="#C0C0C0">
</body>
</html>

<wtOnCreateScript> (4)
  host.close();
</wtOnCreateScript>

<wtEndIf>
```

The first branch of the `<wtif>` handles the restart. If this end template is reached on a restart attempt then clearly no end message should be output and no disconnection operation should be performed. Instead, processing should continue with the host application's entry page. That is why a new conversation is selected in section 1. Before `open` is called, the `CONVERSATION_TAC` attribute is set to `SYM_DEST` to prevent a `openUTM` control field or the start of the screen message being interpreted as a TAC. Section 2 then starts by addressing the first TAC as on initial logon (depending on the host application, it may be necessary to enter an additional `send` here).

The `<wte1se>` branch specifies the behavior of the final page. Section 3 generates a message to the user and section 4 logs off from the host application and closes the WebTransactions session.

8.8 BADTAC - simulating the BADTAC event service

You can configure a openUTM application in such a way that a special conversation is started whenever an invalid transaction code is specified in line mode or via a format's openUTM control field. This special conversation is the BADTAC event service.

The BADTAC event service is a conversation defined by you for which the transaction code KDCBADTC has been generated.

However, in the case of communications via UPIC, the BADTAC event service cannot be used directly since the transaction code KDCBADTC cannot be started via UPIC.

Nevertheless, the system object's BADTAC attribute allows you to simulate this event service for WebTransactions. To do this, you generate any transaction code other than KDCBADTC for the program unit which is to start automatically when an invalid transaction code is generated and store this transaction code in the BADTAC attribute. This program unit should not expect any messages. If you want, you can use the same program as for the BADTAC event service. However, in this case you must generate another transaction code in addition to KDCBADTC and use this transaction code for the BADTAC attribute.

If the BADTAC attribute is **not** set then invalid transaction codes are not automatically intercepted: a call to the `receive` method following the invalid entry leads to an error.

8.9 Automatic conversation chaining

The product variant “WebTransactions for openUTM” communicates with the openUTM application using UPIC. In this communication protocol, the client (here WebTransactions) selects a openUTM conversation by giving a transaction code. The selection is controlled from the template by the `TAC` or `SYM_DEST` system attribute. During a conversation, the openUTM application now determines the sequence of the program segments to be executed (exception: active dialog). At the end of the conversation, the initiative is handed back to WebTransactions: if desired, a further conversation can be selected and started.

In the terminal mode, it is possible to start the next conversation automatically using a openUTM control field or the first 8 characters of the format. This behavior is simulated by WebTransactions. At the end of a conversation, WebTransactions determines, from the openUTM control field or the first 8 characters of the current format, the transaction code for the following conversation. This conversation is then started implicitly by the next call to the `send` method.

This means that template programmers do not have to perform any adaptations for automatic conversation chaining.

8.10 Simulating function keys

TACs and return codes for function keys can be generated in a openUTM application (F1,F2,...,F24 and for BS2000/OSD additionally K1 to K14). On openUTM generation (KDCDEF statement `SFUNC`), each function key can be assigned a specific function which openUTM performs when the user presses the corresponding key at the terminal.

As of Version 4.0, openUTM makes it possible to simulate the pressing of function keys during communications via UPIC. This facilitates the integration into client/server environments of terminal host applications which use openUTM function keys.

You can use this function in WebTransactions by setting the `SPECIAL_KEY` attribute of the system object to the values F1–F24 or K1–K14. With the next `send`, WebTransactions will then send this function key to the openUTM application. After transmission, the attribute of WebTransactions is reset to prevent continued transmission of the function key.

To facilitate the selection of a function key, WebTransactions makes the files `wtKeysUTM.htm` and `wtBrowserFunctions.htm` available, which you can integrate in a FHS or FORMANT based template using the `Include` tag. `wtBrowserFunctions.htm` includes the files `wtKeysUTMFHS.js` and `wtKeysUTMFormant.js` for browser handling, see also [page 174](#).

When generating the templates, `Include` tags for `wtKeysUTM.htm` and `wtBrowserFunctions.htm` are created automatically. These files define buttons for all F and K keys and ensure that the `SPECIAL_KEY` attribute is set to the value corresponding to the key selected. In practice, the files should be copied and reduced to only those keys which are actually generated in the openUTM application.

Assigning the corresponding global attribute values for # formats

Independently of this transmission of a function key as a openUTM return code, when # formats are used, FHS and FORMANT return information on pressed function keys in the global attributes `InputKeyClass` and `InputKeyNumber`. These attributes can be set in the template (via the corresponding attributes of the `WT_HOST_MESSAGE` object). The corresponding statements are already present in generated templates (see [section “Structure of the generated templates” on page 89](#)).

8.11 Support for KDCSCUR

If a dialog step of a openUTM program unit includes format output and the cursor is to be positioned on a field by means of a KDCSCUR call, this information is passed to UPIC. UPIC then analyzes this information and forwards it to WebTransactions.

To ensure that WebTransactions can implement this information, the `WT_HOST_GLOBALS.Cursor` attribute must be set to `N.wtBrowserFunctions.htm` evaluates `WT_HOST_GLOBALS.Cursor` and uses a Java script to set the focus in the browser window. After output on the browser, the cursor is then positioned in the field provided by the openUTM application for the relevant dialog step.

`WT_HOST_GLOBALS.Cursor` is set independently of the `wtBrowserFunctions.htm` template.

8.12 Targeted logon via specific LTERMs

openUTM allows you to configure specific properties for LTERM partners (logical access points) and, for example, integrate these in the lock/keycode concept:

In the openUTM configuration, you can define lockcodes not only for services (openUTM conversations) but also for LTERM partners. You can also define keycodes for LTERM partners in the same way as for openUTM user names:

- A client program can only log on if a keycode which matches the lockcode of the associated LTERM partner has been assigned to the specified user name.
- A client program can only call a service if the keysets of **both** the user name in question **and** the LTERM partner contain a keycode which corresponds to the lockcode of the service.

When access is performed via WebTransactions, the LTERM partner which uses the WTHolder program depends on the `localapps` name which is used. Since WebTransactions always allocates the first free `localapps` name by default at runtime, this allocation is random (free allocation). If a WTHolder program needs to connect via a specific LTERM partner, you can enter a specific local host application name in the system object's `LOCAL_APPLICATION` attribute (fixed allocation). In this case, the `localapps` file is ignored.

If you decide to use a fixed allocation, WebTransactions cannot ensure that a so far unused host application name is used for each session; this must be guaranteed in some other way when the WebTransactions application is implemented.



If a logon via the `UPIC_V4` protocol fails and the `PROTOCOL` parameter is not specified in the `upicfile`, then UPIC attempts to perform the logon using the UPIC Version 3.4 protocol. For this reason, error messages indicating a failed logon may contain an “incorrect” version specification. The correct version will be output in such messages if you specify `PROTOCOL=40` in the corresponding entry in the `upicfile`.

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.

WTML

(WebTransactions Markup Language)

Markup and programming language for WebTransactions ->*templates*. WTML uses additional ->*WTML tags* to extend ->*HTML* and the server programming language ->*WScript*, 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*.

WScript

Serverside programming language of WebTransactions. WScripts are similar to client-side Java scripts 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 scripts are to be implemented by WebTransactions and not by the ->*browser* and also indicates the time of execution. OnCreate scripts are executed before the page is sent to the browser. OnReceive scripts 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 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 ypertext M arkup L anguage
HTTP	H ypertext T ransfer P rotocol
HTTPS	H ypertext 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- T ag
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

Abbreviations

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 Xtensible M arkup L 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 OSD Applications

User Guide

WebTransactions
Connection to MVS Applications

User Guide

WebTransactions
Access to Dynamic Web Contents

User Guide

WebTransactions
Web Frontend for Web Services

User Guide

Other publications

The manuals are available as online manuals, see <http://manuals.ts.fujitsu.com>, or in printed form which must be paid and ordered separately at <http://manualshop.ts.fujitsu.com>.

openUTM
Concepts and Functions
User Guide

openUTM
Programming Applications with KDCS for COBOL, C and C++
Core Manual

openUTM
Generating Applications
User Guide

openUTM
Administering Applications
User Guide

FHS (BS2000/OSD)
Format-Handling System for UTM, TIAM, DCAM
User Guide

Index

\$FIRST (host control object) 170

\$NEXT (host control object) 170

*.fld file see field file

*.htm file see template

A

access point (LTERM) 204

active dialog 205, 208

Align (field file) 87

Align (host data object attribute) 160

APPLICATION_NAME (system object attribute) 133, 146, 153

APPLICATION_PREFIX (system object attribute) 146, 153

architecture

 WebTransactions 9

array 205

ASCII-to-EBCDIC conversion 127, 161

assistant 103

asynchronous message 205

attribute 205

 WT_HOST_GLOBALS 169

 WT_HOST_MESSAGE 165

AttributeCombination 86

AttributeLength (field attribute block) 86

AttributeLength (global attribute block) 85

AttributOffset (field file) 87

AttributOffsets section (field file) 85

Autolnput (field file) 87

Autolnput (host data object attribute) 160

automask template 206

automatic restart 150

B

BackgroundColor (field file) 85

BackgroundColor (WT_Host_Message attribute) 165

BADTAC (system object attribute) 146, 200

BADTAC event service 200

base data type 205

base directory 206

 converting to a new version 72

 creating 71

 example 36

 localapps 73

 upicfile 73

BaseVariant (field file) 85

basic format 190

BCAM application name 206

BCAMAPPL 206

binary data 124

Blink (field file) 87

Blink (host data object attribute) 160

browser 206

 terminal functions 171

browser display print 206

browser platform 206

buffer 206

C

capture database 207

capturing 206

Case (field file) 87

Case (host data object attribute) 160

CGI (Common Gateway Interface) 207

change

 password 196

character coding (openUTM) 149

- CHARSET (system object attribute) [10](#), [126](#), [168](#)
- class [207](#)
 - templates [164](#), [207](#)
- clickable image [62](#)
- client [207](#)
- cluster [207](#)
- Color (field file) [86](#), [87](#)
- Color (host data object attribute) [161](#)
- Column (field file) [87](#)
- command
 - in line mode [121](#)
- communication object [207](#)
 - connection parameters [193](#)
 - creating [193](#)
- COMMUNICATION_INTERFACE_VERSION (system object attribute) [147](#)
- COMMUNICATION_FILE_NAME (system object attribute) [147](#), [154](#), [155](#)
- connection
 - opening multiple [193](#)
- Content_ (WT_HOST_MESSAGE attribute) [165](#), [166](#)
- Contents (WT_Host_Message attribute) [165](#)
- Content-Type (HTTP header field) [168](#)
- conversation
 - chaining [201](#)
 - starting [190](#)
- CONVERSATION_TAC (system object attribute) [147](#)
- conversion tools [207](#)
- create
 - base directory [71](#)
 - project [35](#)
 - start template [64](#)
- Cursor (field file) [86](#)
- Cursor (host control object attribute) [169](#)
- Cursor (host data object attribute) [161](#), [162](#)
- CursorControl (field file) [85](#)
- CursorControl (WT_HOST_MESSAGE attribute) [165](#)
- CursorField (WT_HOST_MESSAGE attribute) [165](#), [166](#)
- CursorPosition (field file) [85](#)
- CursorPosition (WT_HOST_MESSAGE attribute) [165](#)
- CUT_TAC_FIELD (system object attribute) [147](#)
- D**
- daemon [207](#)
- data
 - dynamic [209](#)
- data access control [208](#)
- data type [208](#)
- DataOffset (field file) [87](#)
- DataType (field file) [87](#)
- DataType (host data object attribute) [160](#)
- DateFormat (field file) [85](#)
- DateFormat (host data object attribute) [165](#)
- DecimalSeparator (field file) [85](#)
- DecimalSeparator (host data object attribute) [165](#)
- DefaultCursor (field file) [87](#)
- DefaultCursor (host data object attribute) [160](#)
- define
 - epilog [148](#)
 - form field [148](#)
 - prolog [150](#)
- Detect (WT_HOST_GLOBALS attribute) [169](#)
- Detectable (field file) [87](#)
- Detectable (host data object attribute) [161](#)
- dialog [208](#)
 - active [208](#)
 - non-synchronized [208](#), [213](#)
 - passive [208](#), [214](#)
 - synchronized [208](#), [217](#)
 - types [208](#)
- dialog cycle [208](#)
- DigitSeparator (field file) [85](#)
- DigitSeparator (host data object attribute) [165](#)
- disconnection [156](#)
- display
 - picture file [124](#)
- DISPLAY_EURO (system object attribute) [147](#)
- DisplayLength (field file) [87](#)
- distinguished name [208](#)
- document directory [209](#)
- Domain Name Service (DNS) [209](#)

drop-down list [54](#)

E

edit

 template (example) [53](#)

EditRC (field file) [86](#)

EditState (field file) [86](#)

EditState (host data object attribute) [161](#), [164](#)

EHLLAPI [209](#)

EJB [209](#)

enter licenses (example) [28](#)

entry page [209](#)

EPILOG (system object attribute) [148](#)

ERROR (system object attribute)

 on open [154](#)

 on receive [154](#), [155](#)

evaluation operator [209](#)

event service

 BADTAC [200](#)

 SIGNON [196](#)

expression [209](#)

F

FHS [209](#)

FHS format

 convert to template [76](#)

FHS partial formats [104](#)

field [210](#)

field file [75](#), [210](#)

 keywords [85](#)

 sections [85](#)

 specifying [148](#)

field files [10](#), [126](#)

FieldLength (field file, offset) [86](#)

FieldLength (WT_HOST_GLOBALS
attribute) [170](#)

Fieldname section (field file) [85](#)

FieldsDetect (field file) [85](#)

FieldsDetect (WT_HOST_MESSAGE
attribute) [165](#)

FieldsMod (field file) [85](#)

FieldsMod (host data object attribute) [164](#)

FieldsMod (WT_HOST_MESSAGE
attribute) [165](#)

FieldsValid (field file) [86](#)

FieldsValid (WT_HOST_MESSAGE
attribute) [165](#)

FillCharInput (field file) [87](#)

FillCharOutput (field file) [87](#)

filter [210](#)

first template see start template

FLD (system object attribute) [148](#)

 on receive [154](#), [155](#)

FLD file [81](#)

fld file [210](#)

FLD files [126](#)

FloatSign (host data object attribute) [160](#)

font size [183](#)

FORMANT [82](#)

 partial formats [104](#)

format [210](#)

 #format [210](#)

 *format [210](#)

 +format [210](#)

 -format [210](#)

format description source [10](#), [126](#), [210](#)

format type [210](#)

FORMAT_SEQ (system object attribute) [148](#)

 on receive [155](#)

FormatLength (field file) [86](#)

FormatLength (WT_HOST_MESSAGE
attribute) [165](#)

FormatName (field file) [86](#)

FormatName (WT_HOST_MESSAGE
attribute) [165](#)

FormattingSystem (field file) [86](#)

FormattingSystem (WT_HOST_MESSAGE
attribute) [165](#)

FormatType (field file) [86](#)

FormatType (WT_HOST_MESSAGE
attribute) [165](#)

FormProperties section (field file) [85](#)

FORMTPL (system object attribute) [148](#)

function [210](#)

 doBackTab() [182](#)

 doCursorDown() [182](#)

 doCursorHome() [182](#)

 doCursorLeft() [182](#)

- doCursorRight() [182](#)
- doCursorUp() [182](#)
- doTab() [182](#)
- doToggleInsert() [182](#)
- doToggleMark() [182](#)
- wtCreateKeyMap() [180](#)
- wtCreateKeySelectList() [180](#)
- wtHandleKeyboard() [180](#)
- function key [150, 202](#)

G

- grace sign-on [150, 196](#)
- GroupDigit (field file) [87](#)
- GroupDigit (host data object attribute) [160](#)

H

- Hex_Content__ (WT_HOST_MESSAGE attribute) [165, 167](#)
- hexadecimal representation [124](#)
- HexStringValue (host data object attribute) [124, 160, 163](#)
- holder task [210](#)
- host [210](#)
- host adapter [210](#)
- host application [211](#)
- host control object [211](#)
 - WT_HOST_MESSAGE [165](#)
- host data object [159, 211](#)
- host data print [211](#)
- host platform [211](#)
- HOST_CHAR_CODE (system object attribute) [149](#)
- HOST_IP_ADDRESS (system object attribute) [133, 149, 153](#)
- HOST_NAME (system object attribute) [133, 149, 153](#)
- HOST_PORT (system object attribute) [133, 149](#)
- HTML [211](#)
- HTMLValue (host data object attribute) [160, 163](#)
- HTTP [211](#)
- HTTPS [211](#)
- hypertext [211](#)
- Hypertext Markup Language (HTML) [211](#)

I

- IFG library [10, 126](#)
- IFG2FLD [10, 126](#)
 - example [42](#)
 - using [77](#)
- image
 - clickable [62](#)
- include tag [101](#)
- inline WtBean [221](#)
- INPUT field [164](#)
- INPUT.clt [164](#)
- InputControl (field file) [86](#)
- InputControl (host data object attribute) [161, 162](#)
- InputKeyClass (field file) [86](#)
- InputKeyClass (WT_HOST_MESSAGE attribute) [165](#)
 - assigning values [202](#)
- InputKeyNumber (field file) [86](#)
- InputKeyNumber (WT_HOST_MESSAGE attribute) [165](#)
 - assigning values [202](#)
- InputState (field file) [86](#)
- InputState (host data object attribute) [162, 164](#)
- InputStateAct (field file) [86](#)
- InputStateAct (host data object attribute) [162, 164](#)
- insert
 - button [60](#)
 - inline WtBean [193](#)
 - standalone WtBean [191](#)
- installation
 - BS2000/OSD [23](#)
 - host adapter [17](#)
 - Linux [22](#)
 - silent [19](#)
 - Solaris [21](#)
 - WebLab [23](#)
 - WebTransactions [17](#)
- integrated UPIC protocol [9](#)
- Intensity (field file) [86, 87](#)
- Intensity (host data object attribute) [162](#)
- Invers (field file) [87](#)
- Inverse (field file) [86](#)
- Inverse (host data object attribute) [162](#)

- IObjectType (field file) 88
- IObjectType (host data object attribute) 160
- J**
- Java Bean 212
- K**
- KDCBADTC 200
- KDCDEF 212
- key mapping
 - defining 174
 - wtKeysUTMFHS.js 174
 - wtKeysUTMFormant.js 174
- key support 173
- keyword (field file) 85
- L**
- layout
 - enhancing 99, 101
- LDAP 212
- Length (field file) 88
- Length (host data object attribute) 160
- Level_Selection (WT_HOST_MESSAGE attribute) 165, 167
- licensing 24
- Line (field file) 88
- line mode 120
- line mode template 120
- literals 212
- load
 - services (example) 25
- local application name, example 46
- LOCAL_APPLICATION (system object attribute) 149, 154, 204
- LOCAL_PORT (system object attribute) 133, 149, 153
- localapps (base directory) 73
- lockcode/keycode concept 195, 204
- logon (for openUTM) 195
 - via specific LTERM partner 204
- LTERM partner 204
- M**
- Mandatory (field file) 88
- Mandatory (host data object attribute) 161
- master template 212, 218
 - tag 212
 - UTM.wmt 100
 - UTMpartial.wmt 106
- message queuing 212
- metadata 84
- method 212
- module template 212
- MT tag 212
- multitier architecture 213
- N**
- Name (host data object attribute) 160
- name/value pair 213
- NEW_PASSWORD (system object attribute) 150, 195
- next SYM_DEST (line mode) 121
- non-synchronized dialog 208, 213
- NumAttributes (field file) 86
- NumDecimal (field file) 88
- NumDecimals (host data object attribute) 160
- O**
- object 213
- openUTM 213
 - application 213
 - Client 214
 - line mode 120
 - page pool 157
 - partner (line mode) 121
 - program unit 214
 - service 216
 - user concept 195
- openUTM password 195
- openUTM user name 195
 - invalid 156, 196
- openUTM version
 - earlier than V4.0, UPIC protocol for 135
- operations 208
- OUTPUT field 164
- OUTPUT.clt 164
- OutputControl (field file) 87
- OutputControl (host data object attribute) 162

P

- P_Key_Set (WT_HOST_MESSAGE attribute) 165, 167
- Padding (WT_HOST_GLOBALS attribute) 169
- PaddingAsterix (WT_HOST_GLOBALS attribute) 169
- PaddingPlusAttr (WT_HOST_GLOBALS attribute) 169
- PaddingPlusData (WT_HOST_GLOBALS attribute) 169
- parameter 214
- partial format
 - FHS/FORMANT 104
- partial format template
 - enhancement 117
 - structure 106
- passive dialog 208, 214
- password 195, 214
 - expired 150
 - invalid 156
 - new 195
- PASSWORD (system object attribute) 149, 195
- password protection (openUTM) 149
- picture file
 - display 124
- polling 214
- pool 214
- PopUp (field file) 86
- posted object 214
- posting 214
- process 215
- project 215
 - creating 35
 - example 35, 40
 - saving 40
- PROLOG (system object attribute) 150
- property 215
- Protected (host data object attribute) 161
- Protection (field file) 87, 88
- Protection (host data object attribute) 162
- protocol 215
- Protocol (field file) 86
- protocol file 215

R

- RawValue (host data object attribute) 160
- Read (WT_HOST_GLOBALS attribute) 170
- RECEIVE_ERROR (system object attribute) 150
 - on receive 155
- RECEIVE_SECONDARY_INFORMATION (system object attribute) 150
 - possible values 157
- recognition criteria 215
- record 215
- record structure 210
- restart 150
- RESTART (system object attribute) 150, 197
- RETRY (system object attribute) 150

S

- save
 - picture file 124
 - project 40
- scalar 215
- ScreenDimensions (field file) 86
- SD.DEFAULT 134
- secondary UPIC return code 157
- sections
 - field file 85
- security functions
 - openUTM user concept 195
- SECURITY_TYPE (system object attribute) 150, 195
- service (openUTM) 216
- service node 216
- session 216
 - start templates 185
 - WebTransactions 216
- SFUNC (KDCDEF statement) 202
- Sign (field file) 88
- Sign (host data object attribute) 160
- SignFloat (field file) 88
- SIGNON event service 196
- SOAP 216
- SPECIAL_KEY (system object attribute) 150
- specify partial format 148
- standalone WTBean 221
 - inserting 191

- start
 - start template 185
 - WebTransactions 64
 - start template 185, 218
 - creating custom 64
 - setting system object attributes 187
 - setting WT_HOST_GLOBALS attributes 189
 - StartColumn (host data object attribute) 160
 - StartLine (host data object attribute) 160
 - Startpage button (line mode) 122
 - Stop Conversation (line mode) 121
 - string operations 127, 164
 - style 217
 - SuppressZero (field file) 88
 - SuppressZero (host data object attribute) 160
 - SYM_DEST (system object attribute) 151
 - on open 153
 - symbolic destination name 151
 - upicfile 134
 - synchronized dialog 208, 217
 - system access control 217
 - system object 217
 - interaction between attributes and calls 153
 - openUTM-specific attributes 145
 - set attributes in the start template 187
 - system object attribute
 - CHARSET 10, 126, 168
- T**
- TAC 219
 - TAC (system object attribute) 133, 151, 153
 - TAC in line mode 121
 - tag 217
 - TCP/IP 217
 - template 75, 218
 - class 207
 - edit 99, 101
 - for openUTM line mode 121
 - generate from FHS format 42, 76
 - master 218
 - object 218
 - start 218
 - templates 10, 126
 - terminal application 218
 - terminal functions 171
 - terminal hardcopy printing 218
 - Terminal screen (line mode) 121
 - terminate
 - session (line mode) 122
 - Text (field file) 88
 - Thread 210
 - TimeFormat (field file) 86
 - TimeFormat (host data object attribute) 165
 - traces 164
 - transaction 218
 - transaction code
 - deleting 147
 - invalid 156
 - transaction code/TAC 219
- U**
- UDDI 219
 - umlauts 127
 - UndefinedValues (field file) 86
 - UndefinedValues (WT_Host_Message attribute) 165
 - Underline (field file) 87, 88
 - Underline (host data object attribute) 162
 - Underlined (host data object attribute) 161
 - UnDetect (WT_HOST_GLOBALS attribute) 170
 - Unicode 219
 - diagnostics 164
 - operations on strings 164
 - Unicode (host data object attribute) 161
 - Unicode (WT_HOST_MESSAGE attribute) 166, 168
 - Unicode support 126, 164
 - update
 - base directory 72
 - Update (WT_HOST_GLOBALS attribute) 170
 - UPIC 219
 - UPIC interface 10, 126
 - UPIC return code 156
 - secondary 157
 - UPIC_CODE_CONVERSION (system object attribute) 133
 - UPIC_LIB (system object attribute) 151
 - UPIC_TRACE (system object attribute) 151

- upicfile [153](#)
 - base directory [73](#)
 - default entry [134](#)
- UPIC-R distribution over computers [125](#)
- URI [219](#)
- URL [219](#)
- USER (system object attribute) [152](#), [195](#)
- user exits [219](#)
- user ID [220](#)
- user name [195](#)
 - invalid [156](#), [196](#)
- UserexitRc (field file) [86](#)
- UserexitRc (WT_Host_Message attribute) [165](#)
- UTM see openUTM
- UTM user (line mode) [121](#)
- UTM.wmt [100](#)
- UTM_PATH (system object attribute) [152](#)
- UTMControl (field file) [88](#)
- UTMpartial.wmt [106](#)

- V**
- Value (host data object attribute) [160](#), [163](#)
- value range of a data type [208](#)
- variable [220](#)
- Version (field file) [86](#)
- Version (WT_Host_Message attribute) [166](#)
- visibility [220](#)
- Visibility (field file) [87](#), [88](#)
- Visibility (host data object attribute) [162](#), [163](#)
- Visible (host data object attribute) [161](#)

- W**
- web server [220](#)
- web service [220](#)
- WebLab [10](#), [164](#)
 - editor [99](#), [101](#)
 - installing [23](#)
- WebTransactions
 - architecture [9](#)
 - distribution over computers [125](#)
 - session [216](#)
 - starting [64](#)
- WebTransactions application [220](#)
- WebTransactions platform [220](#)

- WebTransactions server [220](#)
- WSDL [220](#)
- WT_Browser [183](#)
 - font size [183](#)
- WT_HOST_GLOBALS [169](#)
- WT_HOST_MESSAGE [165](#)
- WTBean [221](#)
 - wtcStartUTM [191](#)
 - wtcUTM [193](#)
- wtBrowserFunctions.htm [171](#)
- wtCommonBrowserFunctions.js [179](#)
- wtcStartUTM [191](#)
- wtcUTM [193](#)
- wtKeyMappingTableInput [174](#)
- wtKeysUTM.htm [171](#)
- wtKeysUTMFHS.js [174](#)
 - structure [177](#)
- wtKeysUTMFormant.js
 - structure [177](#)
- wtInmode.htm [120](#)
- WTML [221](#)
- WTML tag [221](#)
- WTScrip [221](#)
- wtstart.htm [47](#), [185](#)
- wtstartUMTV4.htm [185](#)
- WWW browser [206](#)
- WWW server [220](#)

- X**
- XML [221](#)
- XML schema [221](#)

- Z**
- Zeichenkettenoperationen
 - Unicode [164](#)